

NASA Contractor Report 182505

420P.

Computers in Spaceflight

The NASA Experience

James E. Tomayko

CONTRACT NASW-3714
MARCH 1988



(NASA-CR-182505) COMPUTERS IN SPACEFLIGHT:
THE NASA EXPERIENCE (Wichita State Univ.)
409 p LIMIT USGA

X88-10180

Unclas

B3/60 : 0130186

NASA

NASA Contractor Report 182505

Computers in Spaceflight

The NASA Experience

James E. Tomayko
Wichita State University
Wichita, Kansas

Prepared for
National Aeronautics and Space Administration
under Contract NASW-3714



National Aeronautics
and Space Administration

Scientific and Technical
Information Division

1988

Table of Contents

Foreword.....	vii
Preface.....	ix
Acknowledgements.....	xi
Computing and Spaceflight: An Introduction.....	1

Part I: Manned Spacecraft Computers

Introduction to Part One.....	7
Chapter One: The Gemini Digital Computer: First Machine in Orbit.....	9
Chapter Two: Computers On Board The Apollo Spacecraft.....	27
Chapter Three: The Skylab Computer System.....	65
Chapter Four: Computers in the Space Shuttle Avionics System.....	85

Part II: Computers On Board Unmanned Spacecraft

Introduction to Part Two.....	135
Chapter Five: From Sequencers to Computers: Exploring the Moon and the Inner Planets.....	139
Chapter Six: Distributed Computing On Board Voyager and Galileo.....	171

Part Three: Ground Based Computers for Space Flight Operations

Introduction to Part Three.....	205
Chapter Seven: The Evolution of Automated Launch Processing.....	207
Chapter Eight: Computers in Mission Control.....	241
Chapter Nine: Making New Reality: Computers in Simulations and Image Processing.....	269
Epilogue: Themes in NASA's Computing Experience.....	299
Source Notes.....	303
Bibliographic Note.....	363
Appendix I: Glossary of Computer Terms.....	377
Appendix II: HAL/S, A Real-Time Language for Space Flight.....	393
Appendix III: GOAL: A Language for Launch Processing.....	399
Appendix IV: Mariner Mars 1969 Flight Program.....	403

Foreword

The Editors have taken the unusual step of devoting an entire Supplement volume of the Encyclopedia to a single topic: "Computers in Spaceflight: the NASA Experience." The reason will hopefully become apparent upon reading this volume. NASA's use of computer technology has encompassed a long period starting in 1958. During this period, hardware and software developments in the computer field were progressing through successive generations. A review of spaceflight applications of these developments offers a panoramic insight into almost two decades of change in the computer industry and into NASA's role.

NASA's role is summarized at the conclusion of this volume:

"NASA never asked for anything that could not be done with the current technology. But in response, the computer industry sometimes pushed itself just a little in a number of areas. Just a little better software development practices made onboard software safe, just a little better networking made the Launch Processing System more efficient, just a little better operating system made mission control easier, just a little better chip makes image processing faster. NASA did not push the state of the art, but nudged it enough times to make a difference."

This report could not be compressed to typical article size without destroying its usefulness and interest. We trust that the readers will find this work to be as fascinating as did the editors.

Allen Kent

James G. Williams

Preface

NASA's use of computers in spaceflight operations is a very important and large topic. Any attempt to tell the complete story of the people, calculating machines, and computer programs involved in spaceflight would fill many volumes, if, in fact, it could be told at all. The book you are about to read is a subset of all that could be said. This is the explanation of why some things appear here and others do not, and why the book is organized as it is.

When Monte Wright, then director of the NASA History Office, and I first discussed the outline for this project in 1981 and 1982, it seemed that he thought NASA had had a terrific impact on the development of computer technology. Many others shared his view, reasoning that since NASA used computers more extensively than almost any other organization, the Agency must have prodded the computer industry by making challenge after challenge to its computer contractors. One good reason, then, for writing a book on NASA's use of computers was to study the impact of NASA's demands. At the time, I did not know enough to hypothesize one way or another.

Obviously, the book required limits. Since the use of computers in administrative work paralleled that of private industry, and since the chief technological advances occurred in the flight program, we agreed to limit the project to an examination of computer systems used in actual spaceflight or in close support of it. Computers and systems used in administration and in aeronautical and other research not directly related to spaceflight were ignored.

Despite these restrictions, the amount of material and the number of systems involved remained enormous. Any thought of a chronological history had to be abandoned, because keeping the various threads running in order and in parallel was too difficult. Instead, I wrote a topical history, each chapter dealing with either a specific program, such as the Gemini or Apollo onboard computers, or a closely related set of systems, such as launch processing or mission control. This episodic organization made it possible to adapt the writing of the book to the present state of the subject area, and also to NASA's structure. One disadvantage to this approach is that, at first glance, the book has the appearance of a serial description of systems with no obvious relationship to one another. In fact, the decision to order the three major parts of the book as they are was strictly arbitrary. And yet, this organization actually reflects reality. Nearly all the systems described here were developed independently, by different teams, at different sites. Continuity occurred only when a series of systems were built under the auspices of a single center, such as the Gemini, Apollo, and Shuttle systems through the Johnson Space

Flight Center. In the rare instances that some technological exchange occurred, it is highlighted. Despite the independent development of the various systems, some common problems and experiences provided threads with which to bind the chapters. These are presented in the Introduction and developed throughout the book where they apply.

By nature, the subject of computers is technically intensive. Many times things must be discussed that require concentration on the design and engineering attributes of a system. Often the main characters in this history are the machines themselves, and not their creators. A glossary of computer terms and frequent explanatory material in the text should be enough to help those not familiar with computers to understand the story. Additionally, technical material too important to be left out of this history but not crucial to following the flow of events is set apart in boxes. I have retained the technical material in an attempt to fulfill the second objective of the NASA internal history program:

Thoughtful study of NASA history can help agency managers accomplish the missions assigned to the agency. Understanding NASA's past aids in understanding its present situation and illuminates possible future directions.

Hopefully, my choice of the level of the material does not interfere with the first objective, which is the wide "dissemination of information concerning its activities and the results thereof." I believe that at this time a book on this subject that is more expository than interpretive in nature is of greater use to the agency and the historical community. No one before me had waded through this material, therefore, much of my job was the identification of the best sources and the recording of the most useful experiences. Now that this groundwork has been done, more selective and incisive histories can be written.

One final note: often in corporations and government agencies individual achievement is buried within the institution. NASA is no exception. It was exceedingly difficult to get people both in the agency and in contractor organizations to identify who did what, or even take personal credit where appropriate. Wherever I was able to assign responsibility, I did so, but, unfortunately, those instances seem less common than the times I had to credit the development to the institution. Hopefully those who are not mentioned but should have been can take pride that their collective achievements are now part of history.

James E. Tomayko

Pittsburgh, Pennsylvania

April, 1987

Acknowledgements

No author can fool himself into thinking that his work is entirely of his own making. In a project of this size and length, many people, both within NASA and outside it, contributed mightily or it would never have been either finished or of its present quality.

In the NASA History Office, Sylvia D. Fries as Director was a great help not only in accommodating several schedule changes but in actively critiquing early chapters. Her best management decision was assigning Michal McMahon as the editor. He treated the volume as his own and spent many hours turning turgidity into something resembling smooth text. Monte Wright, former Director, is to be thanked for granting the contract in the first place, and Edward C. Ezell, for his help in Houston during the proposal phase when he was head of the History Office at the Johnson Space Center.

At The Wichita State University, my department chair, Mary Edgington, tried to keep excessive demands from overwhelming me during the 3 years I was funded under the contract. Lawrence Smith of the research office took care of the paperwork. Five assistants helped at one time or another in the research or writing phase. Dana Hamit acted as keeper of the bibliography for a year, and created the initial data bases I used while writing. Kim Allen took over from her and prepared the final version of the notes from the first three chapters, as well as acting as first editor. Linda Manfull brought the bibliographic data base into final shape and did the notes for Chapter Four. Maria Dreisziger helped with the notes for Chapters Five and Six, and identified terms for the glossary. Tamera Klausmeyer typed the notes for Chapters Seven through Nine, as well as finishing identifying terms for the glossary.

In my travels during the research phase I was privileged to meet and work with a large number of NASA and contractor personnel. Those listed in the bibliographic note as granting interviews usually shared rare materials from their files as well. Some were asked to do technical reviews of individual chapters or sections of chapters to help eliminate as many errors of fact and interpretation as possible. Those who did this double duty included Bill Bailey, Ed Blizzard, Frank Byrne, Bill Chubb, Sam Deese, Dwain Deets, Bob Ernull, Jack Gorman, Ray Hartenstein, Helmut Hoelzer, Carl Johnson, Ted Kopf, Ken Mansfield, Russ Mattox, Ann Merwarth, Bob Nathan, Henry Paul, Dick Rice, Bill Stewart, Tom Taylor, Bill Tindall, Chuck Trevathan, Paul Westmoreland, John Wooddell, and John Young.

At each site individuals opened doors for me and found office space where none was available. I want to especially thank Wanda Thrower of Johnson Space Center, Bob Sheppard of Marshall Space

Flight Center, Harriet Brown and Mike Konjevich of Kennedy Space Center, and Andrew Danni of the Jet Propulsion Laboratory for their hospitality. Frank Penovich of Kennedy was especially helpful in obtaining a tour of the Shuttle facilities.

After the termination of the actual contract, I spent a year and a half at the Software Engineering Institute (SEI) located at Carnegie-Mellon University. The SEI was kind enough to permit use of their equipment to assist in preparing the final drafts of the manuscript. My assistants Katherine Harvey and Suzanne Woolf did yeoman work editing and formatting the text for laser printing.

My thanks also goes to my wife, who lovingly never let me give up.

A final, required, word from our sponsor: This work was mostly done under NASA Contract NASW-3714.

Computing and Spaceflight: An Introduction

When the National Aeronautics and Space Administration came into existence in 1958, the stereotypical computer was the "UNIVAC," a collection of spinning tape drives, noisy printers, and featureless boxes, filling a house-sized room. Expensive to purchase and operate, the giant computer needed a small army of technicians in constant attendance to keep it running. Within a decade and a half, NASA had one of the world's largest collections of such monster computers, scattered in each of its centers. Moreover, to the amazement of anyone who knew the computer field in 1958, NASA also flew computers in orbit, to the moon, and to Mars, the latter machines running unattended for months on end. Within another 10 years the giant ground-based mainframe would be supplanted by clusters of medium-sized computers in spaceflight operations, and the single on-board computer would be replaced by multiple machines. These remarkable changes mirror developments in the commercial arena. Where there were giant computers, small computers now do similar tasks. Where there were no computers, such as on aircraft or in automobiles, computers now ride along. Where once the only solution was the large, centralized computing center, distributed computers now share the load.

Since NASA is well known as an extensive user of computers—mainly because spaceflight would not be possible without them—there is a common sense that at least part of the reason for the rapid growth and innovation in the computer industry is that NASA has served as a main driver due to its requirements. Actually, the situation is not so straightforward. In most cases, because of the need for reliability and safety, NASA deliberately sought to use proven equipment and techniques. Thus, the agency often found itself in the position of having to seek computer solutions that were behind the state of the art by flight time. However, in other cases, some use of nearly leading edge technology existed, mostly for ground systems, but occasionally when no extensively proven equipment or techniques were adequate in a flight situation. This was especially true on unmanned spacecraft, because the absence of human pilots allowed greater chances to be taken. Thus generalizations cannot be made, other than that there was no conscious attempt on the part of NASA in its flight programs to improve the technology of computing. Any ways in which NASA contributed to the development of computer techniques were side effects of specific requirements.

NASA uses computers on the ground and in manned and unmanned spacecraft. These three areas have quite different requirements, and the nature of the tasks assigned to them resulted in varying types of computers and software. Thus, the impact of NASA on computing differs in extent as a result of the separate requirements for each field of computer use, which is one reason why the three fields are considered in separate parts of this volume.

Computers are an integral part of all current spacecraft. Today they are used for guidance and navigation functions such as rendez-

vous, re-entry, and mid-course corrections, as well as for system management functions, data formatting, and attitude control. However, Mercury, the first manned spacecraft, did not carry a computer. Fifteen years of unmanned earth orbital and deep space missions were carried out without general-purpose computers on board. Yet now, the manned Shuttle and the unmanned Galileo spacecraft simply could not function without computers. In fact, both carry many computers, not just one. This transition has made it possible for current spacecraft to be more versatile. Increased versatility is the result of the power of software to change the abilities of the computer in which it resides and, by extension, the hardware that it controls. As missions change and become more complex, using software to adjust for the changes is much cheaper and faster than changing the hardware.

On-board computers and ground-based computers store data and do their calculations in the same way, but they handle processes and input and output differently. A typical ground computer of the early 1960s, when the first computers flew on manned spacecraft, would process programs one at a time, right after each other. This sort of processing, in which the entire program must be loaded into memory and data must be available in discrete form, is called "batch." Over time, computer systems were changed to make them more efficient than batch computing allowed. In a batch process, if the computer is doing a calculation, the input and output devices are idle. If it is using a peripheral device, the calculating circuits are not used. One way to improve on efficiency of the batch process would be to develop an operating system for computers that could permit one program to use resources currently unneeded by another program. Another method is to limit each program to a fraction of a second running time before going on to the next program, running it for a fraction and then going on until the original program gets picked up again. This cyclic, time-sliced method permits many users to be connected to the computer or many jobs to run on the computer in such a way that it appears that the machine is processing one at a time. The computer is so fast that no one notices that his or her job is being done in small segments. Each of these methods presupposes that data for the program are available and processed, and then the program stops. So even though lots more programs are run through the system in a period of time, each is still handled as a batch process. When the computer runs through all the processes waiting for execution, it stands idle.

Spacecraft computers operate in a radically different processing environment. They are in "real-time" mode, handling essentially asynchronous inputs and outputs and continuous processing, similar to a telephone operator who does not know on which line the next call will come. For example, computers used for controlling the descending Shuttle can hardly process commands to the aerodynamic surfaces in batch mode. The spacecraft would go out of control or at least lose

4 COMPUTERS IN SPACEFLIGHT: THE NASA EXPERIENCE

track of where it was if data were only utilized in small bunches. The requirement for real-time processing leads to other requirements for spacecraft computers not normally found on earth-based systems. Software must not "crash" or have an abnormal end. If the software stops, the vehicle ceases to be controllable. Hardware must also be highly reliable, or reliability can be obtained through redundancy. If the latter course is chosen, overhead in the form of redundancy management hardware and software will be high. Memories must be nonvolatile in most applications, so if power is lost then the program in storage will not disappear. Since modern semiconductor, random-access memories are usually volatile, older technology memories such as ferrite core continue to be used on spacecraft. Weight, size and power are other considerations, just as with all components on a spacecraft.

Even though both manned and unmanned spacecraft have similar requirements, until very recently they could not use the same computers. No computer with sufficient calculating capability to control the Shuttle flew on an unmanned spacecraft. Conversely, the Shuttle computers are so large and power hungry they would overwhelm the power supply of a deep space probe. Modern powerful microprocessors make it possible to overcome these deficiencies, but systems described herein predate most microprocessor technology. Also, computers on manned spacecraft are oriented toward relatively short-term missions lasting up to a few weeks (which will change in the Space Station and Mars Mission eras). Computers on unmanned earth orbital missions and deep space probes need to run reliably for years, yet must have low power requirements. Even though both need to be trustworthy, the different mission conditions dictate how reliability is to be attained.

NASA's challenge in the 1960s and 1970s was to develop computer systems for spacecraft that could survive the stress of a rocket launch, operate in the space environment, and thus provide payloads with the increased power and sophistication needed to achieve increasingly ambitious mission objectives. NASA found itself both encouraging new technology and adapting proven equipment. In manned spacecraft the tendency was to use what was available. On unmanned spacecraft innovation had a freer hand.

In contrast, NASA's ground computer systems reflected the need for large-scale data processing similar to many commercial applications, but in a real-time environment, until recently not normally a requirement of business computing. Therefore, commercially available computers could be procured for most of the ground-based processing, with any innovation confined to software that handled the real-time needs. Preflight checkout, mission control, simulations, and image processing all have used varying combinations of standard mainframe and minicomputers. So NASA's impact on computing driven by ground support requirements was largely in the area of operating sys-

tems and other software and not as much in hardware, whereas many of the on-board computers had to be custom built. Some of the software innovations needed on the ground have naturally had greater impact on the wider world than those made for on-board computers. The techniques of software development learned by NASA while doing both flight and support programming have advanced the state of the art of software engineering, which comprise the management and technical principles that make it possible to build large, reliable software systems.

Even though the requirements and solutions to computing problems in the manned on-board, unmanned on-board, and ground arenas are different, several common themes bind the three together. In nearly all cases, NASA managers failed to adequately allow for system growth, often causing expensive software and hardware additions to be made to meet scaled-down objectives. More positively, recent developments are designed to enable proven computer systems and techniques to fly or support more than one mission, reducing the costs associated with customized solutions. Also, there is a continuing reliance on multiple smaller computers operating in a network as opposed to large single computers, enabling task distribution and more economical means of ensuring reliability. This last trend also underscores the dependence on communications that has characterized NASA's far-flung flight operations since the beginning. These themes appear in varying strengths throughout the stories of the individual projects.

Regardless of NASA's impact on computing, its many uses of computing technology from 1958 on provide valuable examples of the growth in power, diversity, and effectiveness of the applications of computers. The late 1950s marked the beginning of the computer industry as an indispensable contributor to American science and business. NASA's insatiable desire to make the most of what the industry could offer resulted in many interesting and innovative applications of the ever-improving technology of computing.



Figure A: The first manned spaceflight program to use computers continuously in all mission phases was Apollo. Here mission controllers watch computer-driven displays while astronauts explore the lunar surface after a computer-controlled descent.

PRECEDING PAGE BLANK NOT FILMED

Part One:

Manned Spacecraft Computers

In the first 25 years of its existence, NASA conducted five manned spaceflight programs: Mercury, Gemini, Apollo, Skylab, and Shuttle. The latter four programs produced spacecraft that had on-board digital computers. The Gemini computer was a single unit dedicated to guidance and navigation functions. Apollo used computers in the command module and lunar excursion module, again primarily for guidance and navigation. Skylab had a dual computer system for attitude control of the laboratory and pointing of the solar telescope. NASA's Space Shuttle is the most computerized spacecraft built to date, with five general-purpose computers as the heart of the avionics system and twin computers on each of the main engines. The Shuttle computers dominate all checkout, guidance, navigation, systems management, payload, and powered flight functions.

NASA's manned spacecraft computers are characterized by increasing power and complexity. Without them, the rendezvous techniques developed in the Gemini program, the complex mission profiles followed in Apollo, the survival of the damaged Skylab, and the reliability of the Shuttle avionics system would not have been possible.

When NASA began to develop systems for manned spacecraft, general-purpose computers small and powerful enough to meet the requirements did not exist. Their development involved both commercial and academic organizations in repackaging computer technology for spaceflight.

1

The Gemini Computer: First Machine in Space

Project Mercury was America's first man-in-space effort. The McDonnell-Douglas Corporation developed the Mercury spacecraft in the familiar bell shape. It was barely large enough for its single occupant and had no independent maneuvering capability save attitude control jets. Its orbital path was completely dependent on the accuracy of the guidance of the Atlas booster rocket. Re-entry was calculated by a real-time computing center on the ground, with retrofire times and firing attitude transmitted to the spacecraft while in flight. Therefore, it was unnecessary for the Mercury spacecraft to have a computer, as all functions required for its limited flight objectives were handled by other systems.

Gemini both continued the objectives of the Mercury program and served as a test bed for the development of rendezvous techniques critical to lunar missions¹. At first glance, the Mercury and Gemini spacecraft are quite similar. They share the bell shape and other characteristics, partially because Gemini was designed as an enlarged Mercury and because the prime contractor was the same for both craft. The obvious difference is the presence of a second crew member and an orbital maneuvering system attached to the rear of the main cabin. The presence of a second crewman meant that more instrumentation could be placed in Gemini and that more experiments could be performed, as an extra set of eyes and hands would be available. Gemini's maneuvering capability made it possible to practice rendezvous techniques. The main rendezvous target was planned to be the Agena, an upper stage rocket with a restartable liquid-propellant engine that could be launched by an Atlas booster. After rendezvous with an Agena, the Gemini would have greatly increased maneuvering capability because it could use the rocket on the Agena to raise its orbit.

Successful rendezvous required accurate orbital insertion, complex catch-up maneuvering, finely tuned movements while making the final approach to the target, and guidance during maneuvers with the Agena. Safety during the critical powered ascent phase demanded some backup to the ascent guidance system on the Titan II booster vehicle. The Gemini designers also wanted to add accuracy to re-entry and to automate some of the preflight checkout functions. These varied requirements dictated that the spacecraft carry some sort of active, on-board computing capability. The resulting device was the Gemini digital computer.

The Gemini computer functioned in six mission phases: prelaunch, ascent backup, insertion, catch-up, rendezvous, and re-entry. These requirements demanded a very reliable, fairly sophisticated digital computer with simple crew interfaces. IBM built such a machine for the Gemini spacecraft.

By the early 1960s, engineers were searching for ways to automate checkout procedures and reduce the number of discrete test lines connected to launch vehicles and spacecraft. Gemini's computer

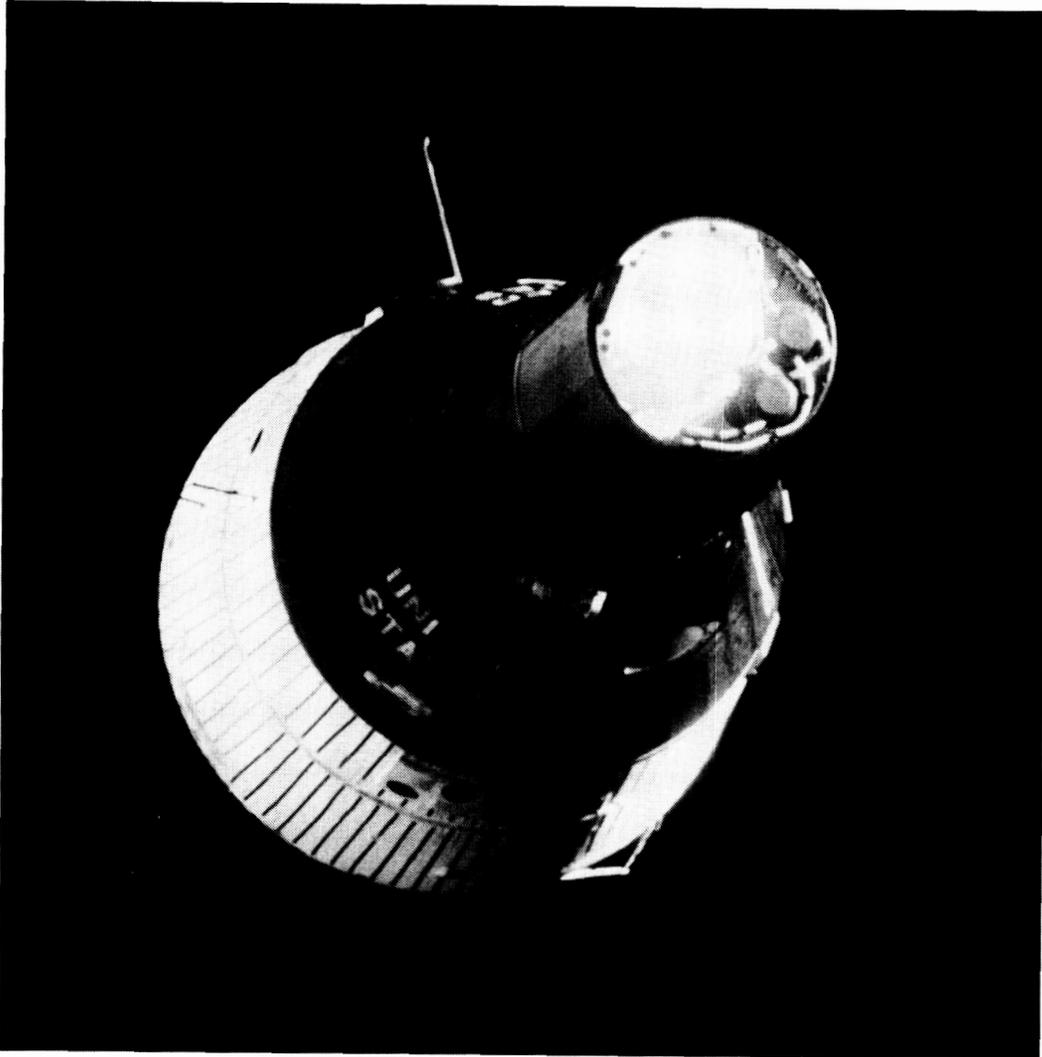


Figure 1-1. First orbital rendezvous: Gemini VI keeps station after using its on-board computer to maneuver to position near Gemini VII. (NASA photo S-65-63175)

did its own self checks under software control during the prelaunch phase. It also accepted parameters needed for the flight during the last 150 minutes before launch². During ascent, the computer received information about the velocity and course of the booster so that it would be ready to take over from the Titan's computers if they failed. Switch-over could either be automatic or manual. The computer could then issue steering and booster cutoff commands to the Titan³. Even if the updated parameters were not necessary to boost guidance, they were useful in the calculation of additional velocity needed after the Titan's second-stage cutoff to achieve the proper orbit. That velocity difference was displayed to the crew so that they could use the spacecraft's own propulsion system to reach insertion velocity⁴.

Rendezvous operations required an on-board computer because the ground tracking network did not "cover" all parts of the Gemini orbital paths. Thus, it would be impossible to provide the sort of continuous updates needed for rendezvous maneuvers. For example, Gemini XI was planned as a first-orbit Agena rendezvous, with some of the critical maneuvers conducted outside of telemetry range⁵. That same mission also featured a fully computer-controlled re-entry, which resulted in a splashdown 4.6 kilometers from the target⁶. In computer-controlled descents, the roll attitude and rate are handled by the computer to affect the point of touchdown and re-entry heating. The Gemini spacecraft had sufficient lift capability to adjust the landing point up to 500 miles on the line of flight and 40 miles laterally respective to the line of flight. Five minutes before retrofire, the computer was placed in re-entry mode and began to collect data. It displayed velocity changes during and after the retrofire. During the time the spacecraft traveled from an altitude of 400,000 feet to when it reached 90,000 feet, the computer controlled actual attitude⁷.

HARDWARE

IBM Corporation received the contract for the Gemini digital computer on April 19, 1962, amounting to \$26.6 million. It provided for the construction of the on-board computer and for integration with other spacecraft systems⁸. The first machine was in its final testing phase by August 31, 1963, and IBM delivered the last of 20 such machines by December 1965⁹. Engineers at IBM believe that the main reason why their company received the contract was the successful development of a core memory used on the Orbiting Astronautical Observatory¹⁰. One of them, John J. Lenz, said that the contract for Gemini came just at the right time. The best of the engineering teams of the IBM Federal Systems Division plant in Owego, New York were between assignments and were put on the project, increasing its chance for success.

Restrictions on size, power, and weight influenced the final form of the computer in terms of its components, speed, and type of memory. The shape and size of the computer was dictated by the design of the spacecraft. It was contained in a box measuring 18.9 inches high by 14.5 inches wide by 12.75 inches deep, weighing 58.98 pounds¹¹. An unpressurized equipment bay to the left of the Gemini commander's seat held the computer, as well as the inertial guidance system power supply and the computer auxiliary power supply. The machine consisted of discrete components, not integrated circuits¹². However, circuit modules that held the components were somewhat interchangeable. They were plugged into one of five interconnection

ORIGINAL PAGE IS
OF POOR QUALITY

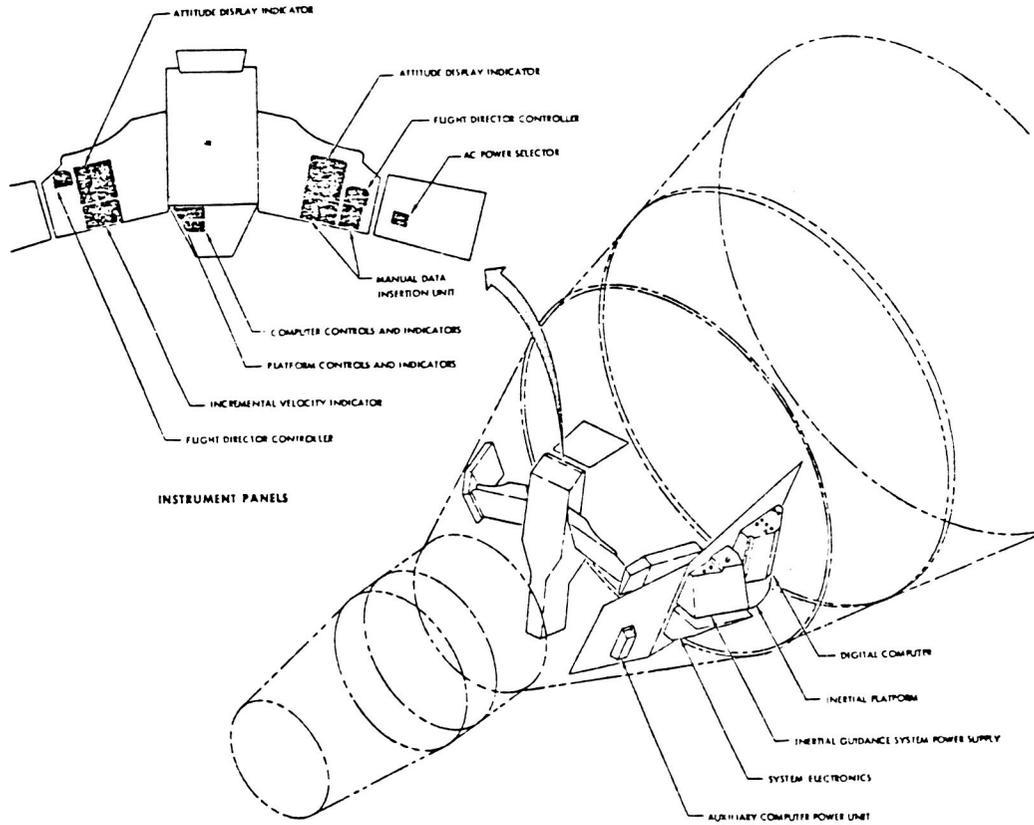


Figure 1-2. Locations of key components of the Gemini Guidance System. (From McDonnell Corp., *Gemini Familiarization Manual*)

boards, and it took 510 of the modules to build the logic section alone¹³. The computer had no redundant circuits, which meant that failures in the computer canceled whatever activity needed to be controlled by it. For example, a failure in the power switch three quarters of the way through the Gemini IV mission caused cancellation of the planned computer-controlled re-entry. It was possible to fly the Gemini computer without a backup because whatever the computer did erroneously could be either abandoned (such as rendezvous) or handled, albeit more crudely, in other ways (such as re-entry using Mercury procedures).

The machine had an instruction cycle of 140 milliseconds, the time it required for an addition. Multiplication took three cycles, or

420 milliseconds, with division requiring double that time¹⁴. The arithmetic bit rate was 500 kilocycles, and the memory cycle rate half that¹⁵. The computer was serial in operation, passing bits one at a time, which explains the relatively slow processing speeds, slower than some vacuum tube computers such as the Whirlwind. Also, its fixed decimal point arithmetic unit design limited the precision of the calculations but greatly reduced complexity. The Gemini digital computer used ferrite cores for its primary memory. Core memories store one bit in each ferrite ring by magnetizing the ring in either a clockwise or counterclockwise direction. One direction means a one is stored and the opposite direction is a zero. Each core is mounted at a perpendicular crossing of two wires. Thousands of such crossings are in each core plane, consisting of rows of wires running up and down (the x wires) and others running left and right (the y wires). Therefore, to change the value of a bit at a specific location, half the voltage required for the change is sent on each of two wires, one in the x direction and one in the y direction. This way only the core at the intersection of the two wires is selected for change. All the others on the same wires would have received only half the required voltage. By the use of a third wire it is possible to "sense" whether a selected core is a one or a zero. In this way, each individual core can be read.

The ferrite core memory in the Gemini computer had a unique design. It consisted of 39 planes of 64 by 64-bit arrays, resulting in 4,096 addresses, each containing 39 bits. A word was considered to be 39 bits in length, but it was divided into three syllables of 13 bits. The memory itself divided into 18 sectors. Therefore, it was necessary to specify sector and syllable to make a complete address. Instructions used 13 bits of the word, with data representations of 26 bits. Data words were always stored in syllables 0 and 1 of a full word, but instructions could be in any syllable. This means that up to three instructions could be placed in any full word, but only one data item could be in a full word¹⁶.

The arithmetic and logic circuit boards and the core memory made up the main part of the Gemini computer. These components interfaced to a plethora of spacecraft systems, most of which were concerned with guidance and navigation functions. This system was the Gemini digital computer through the Gemini VII mission. Beginning with Gemini VIII, the computer included a secondary storage system, which had impact on the spacecraft computer systems built by IBM and flown on the Skylab and Shuttle.

During the 1950s and well into the 1960s, the most ubiquitous method of providing large secondary storage for computers was the use of high-speed, high-density magnetic tape. By 1980, tape was used mainly to store large blocks of data unneeded on a regular basis or to mail programs and data between sites. Disk systems have considerably faster access times and have rapidly increased in storage



Figure 1-3. Cores like these were used in Gemini's memory. (IBM photo)

capacity, rivaling or even exceeding tape, and thus supplanting it in common use. In 1962, disk systems were large, expensive, and far from fully reliable. When the software for the Gemini computer threatened to exceed the storage capacity of the core memory, IBM proposed an Auxiliary Tape Memory to store software modules that did not need to be in the computer at lift-off. For example, programs that provided backup booster guidance and insertion assistance would be in the core memory for the early part of the flight. The re-entry program could be loaded into the core shortly before it was needed, thus writing over the programs already there. This concept, fairly common in earth-bound computer usage, was a first for aerospace computing.

IBM chose the Raymond Engineering Laboratory of Middletown, Connecticut to build the device¹⁷. The unit weighed 26 pounds and filled about 700 cubic inches of space in the adapter module of the Gemini spacecraft¹⁸. The tape memory increased the available storage of the Gemini computer by seven and one-half times with its capacity of 1,170,000 bits. Programs loaded from the tape would fill syllables 0

16 COMPUTERS IN SPACEFLIGHT: THE NASA EXPERIENCE

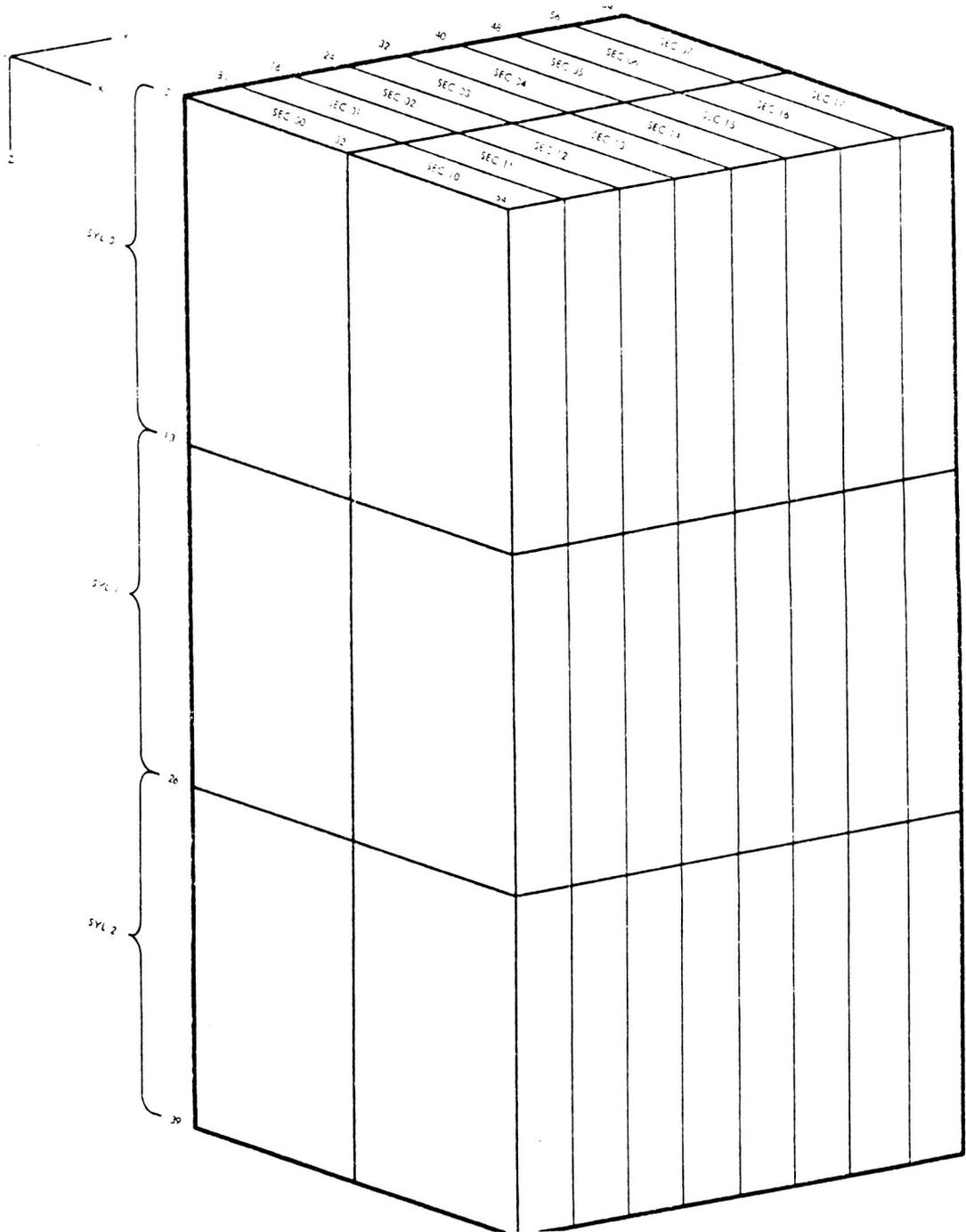


Figure 1-4. Layout of the Gemini Digital Computer core memory. (From McDonnell Corp., *Gemini Familiarization Manual*)

ORIGINAL PAGE IS
OF POOR QUALITY

and 1 of the core memory locations¹⁹. It took 6 minutes to load a program from the tape drive into core²⁰.

NASA's natural insistence on high reliability in manned spaceflight operations challenged the computer industry of the early 1960s. Tape error rates were 1 bit in 100,000 and IBM wanted to raise this rate to 1 bit in 1,000,000,000²¹. The method used was to triple record each program on the tape, pass each set of three corresponding bits through a voter circuit, and send the result of the vote to the core memory²². This scheme was later used on the Shuttle.

Gemini VIII was the first mission with the Auxiliary Tape Memory on board. Shortly after a successful rendezvous with an Agena, the combined spacecraft began to spin out of control. Mission Control decided to disengage the Agena and bring the Gemini down, as large amounts of attitude control thruster fuel had been wasted trying to regain control of the spacecraft. Thus, the first attempt to load a program from the tape was made while the spacecraft was spinning. Even though the Auxiliary Tape Memory design parameters specified low vibration levels,²³ the re-entry program was successfully loaded and used in the subsequent descent.

IBM obtained this sort of reliability beyond the original specifications as a result of an extensive testing program. For example, the Auxiliary Tape Memory had failed prequalification vibration tests, so IBM added a brass flywheel and weights on the tape reels to increase stabilization²⁴. This ensured a successful program load under adverse conditions. There were also problems with transistors shorting out due to loose particles too small to be seen on x-rays but which shook loose during acceleration²⁵. Increased cleanliness in manufacturing was one solution to this problem.

The only in-flight failure of a computer component was on the 48th revolution of the Gemini IV mission, when astronaut James McDivitt tried to update the computer in preparation for re-entry. The machine would not turn off, and it could not be used for the planned "lifting bank" re-entry²⁶. IBM could not duplicate the failure on the ground, but the manufacturers did install a manual switch that bypassed the failure for Gemini V²⁷.

SOFTWARE

In 1962, hardware was still the pacing factor in computer applications. Everything associated with computers—processors, memories and input/output (I/O) peripherals—was expensive. Many considered software development an incidental part of the overall applications of computing. Specialists wrote most of the software, usually in arcane assembly languages. FORTRAN, a high-level language, had only

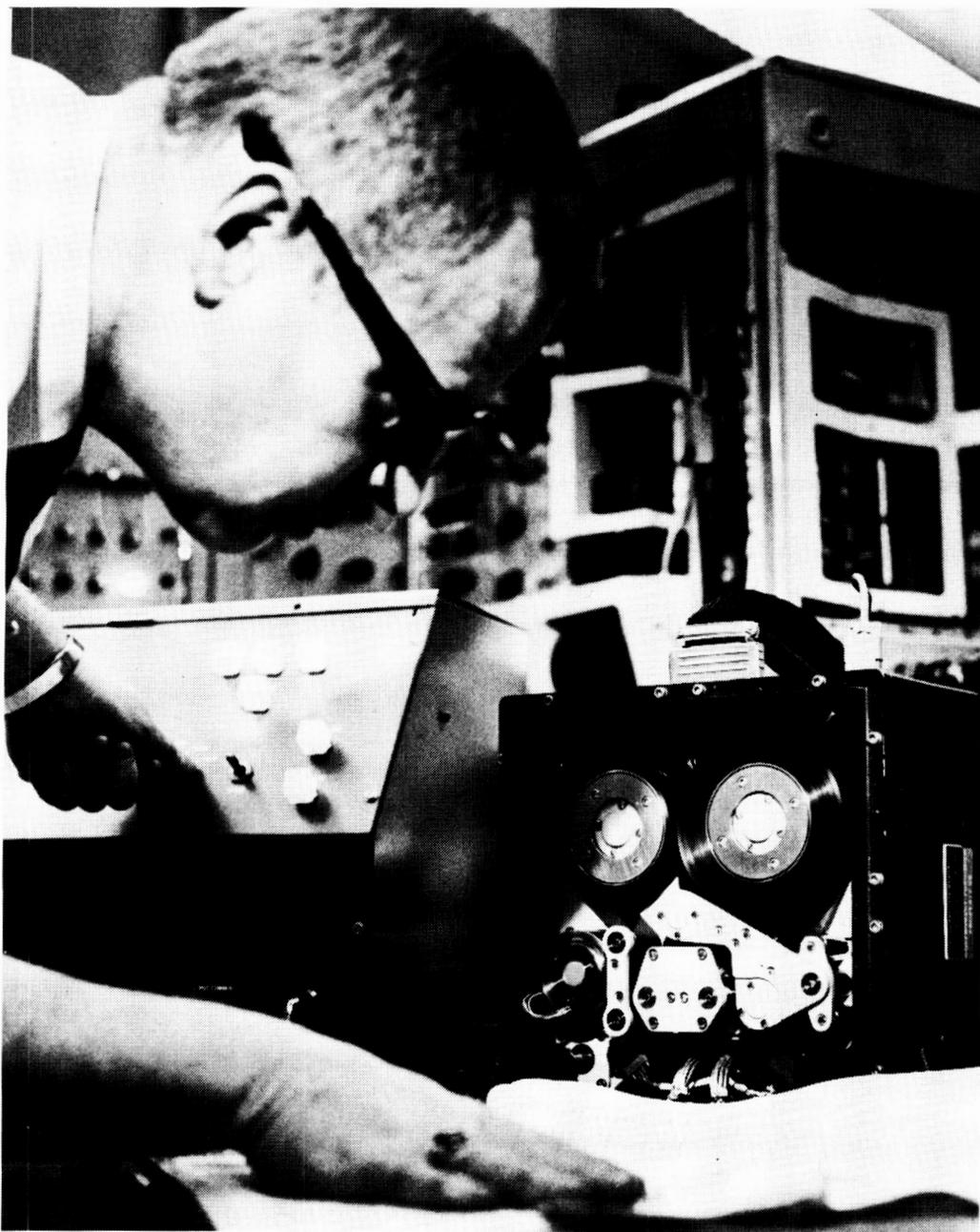


Figure 1-5. Auxiliary Tape Memory in test. (IBM photo)

ORIGINAL PAGE IS
OF POOR QUALITY

been available for a few years. Although its use in technical applications was rapidly spreading, it was still considered too inefficient for use on computers like the Gemini digital computer. Many thought its compiler-produced machine code to be less effective in utilizing machine resources than machine language programs written by humans. Experts therefore developed applications programs for Gemini using the tiny set of 16 instructions that the computer could execute²⁸. This sort of programming was considered to be more of an art than a science. Whereas the design and construction of computer hardware followed conventional engineering principles, software development was largely haphazard, undocumented, and highly idiosyncratic. Many managers considered software developers to be a different breed and best left alone. This concept of software is a myth, and although it persists in some companies and with some people today, by and large software is now considered as an engineered product, little different from a rocket engine or computer.

Although the term "software engineering" did not come into common use until 1968, programmers had applied its basic tenets to both large and small software projects for at least 15 years. Software engineering has evolved as programmers learned which techniques worked, which did not, and what actually occurred in the development of software products. The SAGE (Semi-Automatic Ground Environment) air defense system²⁹, the IBM 360 operating system³⁰, and NASA's requirements for both spacecraft software and ground-based software were instances of major software projects that directly contributed to the evolution of software engineering.

Software engineers recognize that software follows a specific development cycle, from formal specification of the product, through the design and coding of the actual program, and then to testing of the product and postdelivery maintenance. This cycle lasts for many years in the case of programs such as operating systems, or a short period of time in the case of specialized, single-use programs. During this development process, strict standards of documentation, configuration control, and managing changes and the correction of errors must be maintained. Also, breaking down the application into smaller, potentially interchangeable parts, or modules, is a primary technique. Communication between programming teams working on different but interconnected modules must be kept clear and unambiguous. It is in these areas that NASA has had the greatest impact on software engineering.

Development of the Gemini software was a learning experience for both NASA and IBM. It was, of course, the first on-board software for a manned spacecraft and was certainly a more sophisticated system than any that had flown on unmanned spacecraft to that point. When the time came to write the software for Gemini, programmers envisioned a single software load containing all the code for the flight,

with new unique programs to be developed for each mission³¹. Soon it became obvious that certain parts of the program were relatively unchanged from mission to mission, such as the ascent guidance backup. Designers then introduced modularization, with some modules becoming parts of several software loads.

Another reason for modularization is the fact that the programs developed for Gemini quickly exceeded the memory available for them. Some were stored on the Auxiliary Tape Memory until needed. The problem of poor estimation of total memory requirements has plagued each manned spacecraft computer system. In the case of Gemini, changed requirements and attempts to squeeze the programs into the allotted space resulted in nine different versions of the software³². The different versions were referred to by the name "Gemini Math Flow."

Tracing the development of the math flows shows how identifying new functions caused initial memory estimates to be wrong and how the project handled changes. Math Flow One consisted of just four modules: Ascent, Catch-up, Rendezvous, and Re-entry. Math Flow Two was proposed to add orbital navigation and re-entry initialization, but it caused the overall load to exceed the memory size and the Gemini program office canceled the additions³³. This version of the software flew on spacecraft II in January 1965. By Math Flow Four, the re-entry initialization program had been successfully added, but the load took up 12,150 of 12,288 available words. The plan had been to use this program on spacecraft III and others, but a NASA directive of February, 1964 changed the guidance logic of the re-entry mode to a constant bank angle rather than a proportional bank angle and constant roll rate. Math Flow Five incorporated this change, but it filled the memory and was scrubbed in favor of a modified Math Flow Three on spacecraft III and IV, followed by Math Flow Six containing some changes on spacecraft V through VII³⁴. The final version, Math Flow Seven, was used on spacecraft VIII through XII, all of which incorporated the Auxiliary Tape Memory. It had six program modules with nine operational modes. The six program modules of Math Flow Seven were Executor, Prelaunch, Ascent, Catch-Up, Rendezvous, and Re-Entry³⁵. The Executor routine selected other routines depending upon mission phases.

The specification procedure for the software required McDonnell-Douglas to prepare the Specification Control Document (SCD). This was forwarded to the IBM Space Guidance Center in Owego, which developed a FORTRAN program to validate the guidance equations. The use of simulations such as the FORTRAN program was endemic to the Gemini software effort and was later applied to software development for other spacecraft computers.

Gemini used three levels of simulations, beginning with the equation-validation system. The second was a man-in-the-loop

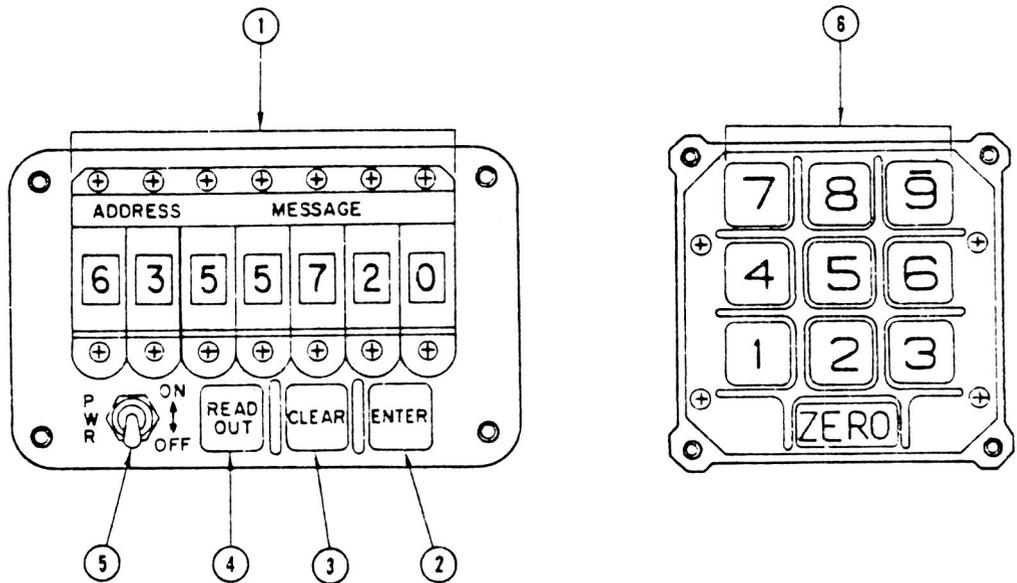
simulation to help define I/O requirements, procedures, and displays. The third level was a refined digital simulation to determine the performance characteristics of the software, useful in error analysis. This third level was carried out in the Configuration Control Test System (CCTS) laboratory, which contained a Gemini computer and crew interfaces. This Mission Verification Simulation (MVS) ensured that the guidance system worked with the operational mission program. Further tests of the software were done at McDonnell–Douglas and at the pad³⁶. NASA and IBM emphasized program verification because there was no backup computer or backup software. The verification process and the tools developed for it were later applied to military projects in which IBM became involved³⁷.

Even if the software is perfect, errors may occur because of transient hardware or software failures during operation due to power fluctuations or unforeseen demands on real-time programs. Some of these can be spotted by diagnostic subroutines interleaved in the software and used for fault detection³⁸. Such routines were put in the Gemini software and are now a part of all IBM computer systems.

The software produced during the Gemini program was highly reliable and successful. Techniques of specification development, verification, and simulations developed for Gemini were later applied to other IBM and NASA projects. NASA was certainly better prepared to monitor software development for the much more difficult Apollo program.

CREW INTERFACES TO THE GEMINI DIGITAL COMPUTER

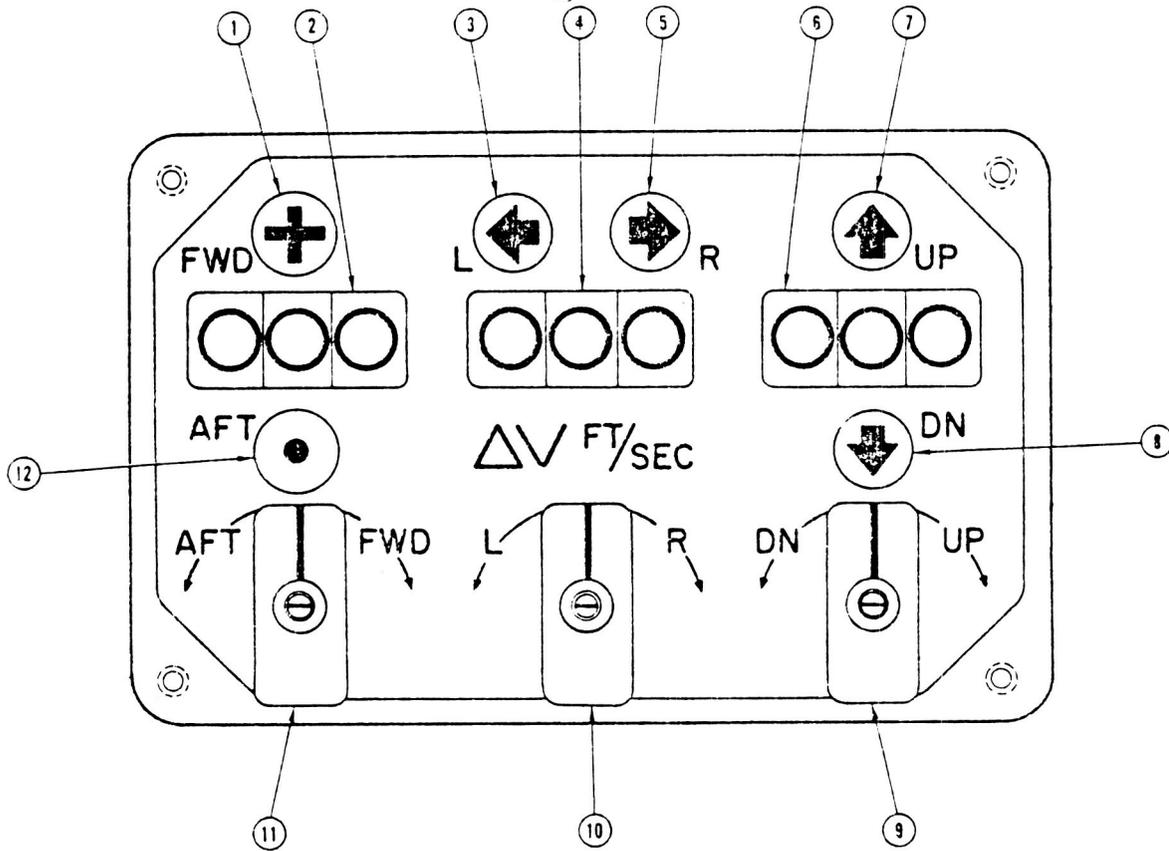
Gemini's digital computer had three sets of interfaces: the computer's controls, the Manual Data Insertion Unit (MDIU), and the Incremental Velocity Indicator (IVI). The controls consisted of a mode switch, a start button, a malfunction light, a computation light, and a reset switch. The mode switch had seven positions for selection of one of the measurement or computation programs. The start button caused the computer to run the selected program loaded in its memory. The reset switch caused the computer to execute its start-up diagnostics and prepare itself for action. The MDIU consisted of two parts: a 10-digit keyboard and a 7-digit register. The first two digits of the register, a simple odometerlike rotary display, were used to indicate a memory address. Up to 99 such logical addresses could be accessed. The remaining five digits displayed data. Errors caused all zeroes to appear. Negative numbers were inserted by making the first digit a nine; the other digits contained the value. The IVI displayed velocity increments required for, or as a result of, a powered maneuver. It had three-digit feet-per-second displays for each of forward-and-back, up-and-down, and left-to-right axes³⁹.



LEGEND		
ITEM	NOMENCLATURE	PURPOSE
①	ADDRESS AND MESSAGE DISPLAY DEVICES	DISPLAY ADDRESS AND MESSAGE SENT TO COMPUTER DURING ENTER OPERATION, DISPLAY ADDRESS SENT TO, AND MESSAGE RECEIVED FROM COMPUTER DURING READOUT OPERATION.
②	ENTER PUSH-BUTTON SWITCH	PROVIDES MEANS FOR CAUSING MESSAGE SENT TO COMPUTER DURING ENTER OPERATION TO BE STORED IN MEMORY.
③	CLEAR PUSH-BUTTON SWITCH	PROVIDES MEANS FOR CAUSING ADDRESS AND MESSAGE SET UP BY MDK TO BE CLEARED OR CANCELED.
④	READ OUT PUSH-BUTTON SWITCH	PROVIDES MEANS FOR CAUSING MESSAGE TO BE READ OUT OF COMPUTER AND DISPLAYED BY MESSAGE DISPLAY DEVICES.
⑤	PWR (POWER) TOGGLE SWITCH	PROVIDES MEANS FOR CONTROLLING APPLICATION OF POWER TO MDK AND MDR.
⑥	DATA INSERT PUSH-BUTTON SWITCHES	PROVIDE MEANS FOR CAUSING ADDRESS AND MESSAGE TO BE SENT TO COMPUTER AND TO BE DISPLAYED BY ADDRESS AND MESSAGE DISPLAY DEVICES.

Figure 1-6. Manual Data Insertion Unit. (From McDonnell Corp., *Gemini Familiarization Manual*)

ORIGINAL PAGE IS
OF POOR QUALITY



LEGEND		
ITEM	NOMENCLATURE	PURPOSE
①	FWD (FORWARD) DIRECTION INDICATION LAMP	INDICATES THAT PLUS X AXIS VELOCITY IS INSUFFICIENT.
②	FORWARD-AFT DISPLAY DEVICE	INDICATES AMOUNT OF INSUFFICIENT VELOCITY FOR PLUS OR MINUS X AXIS.
③	L (LEFT) DIRECTION INDICATION LAMP	INDICATES THAT MINUS Y AXIS VELOCITY IS INSUFFICIENT.
④	LEFT-RIGHT DISPLAY DEVICE	INDICATES AMOUNT OF INSUFFICIENT VELOCITY FOR PLUS OR MINUS Y AXIS.
⑤	R (RIGHT) DIRECTION INDICATION LAMP	INDICATES THAT PLUS Y AXIS VELOCITY IS INSUFFICIENT.
⑥	UP-DOWN DISPLAY DEVICE	INDICATES AMOUNT OF INSUFFICIENT VELOCITY FOR PLUS OR MINUS Z AXIS.
⑦	UP DIRECTION INDICATION LAMP	INDICATES THAT MINUS Z AXIS VELOCITY IS INSUFFICIENT.
⑧	DN (DOWN) DIRECTION INDICATION LAMP	INDICATES THAT PLUS Z AXIS VELOCITY IS INSUFFICIENT.
⑨	DN-UP ROTARY SWITCH	PROVIDES MEANS FOR MANUALLY SETTING UP Z AXIS VELOCITY ERROR ON UP-DOWN DISPLAY DEVICE.
⑩	L-R ROTARY SWITCH	PROVIDES MEANS FOR MANUALLY SETTING UP Y AXIS VELOCITY ERROR ON LEFT-RIGHT DISPLAY DEVICE.
⑪	AFT-FWD ROTARY SWITCH	PROVIDES MEANS FOR MANUALLY SETTING UP X AXIS VELOCITY ERROR ON FORWARD-AFT DISPLAY DEVICE.
⑫	AFT DIRECTION INDICATION LAMP	INDICATES THAT MINUS X AXIS VELOCITY IS INSUFFICIENT.

Figure 1-7. Incremental Velocity Indicator. (From McDonnell Corp., *Gemini Familiarization Manual*)

On a typical mission the computer would be in operation during ascent as the backup to the booster. On orbit, if no powered maneuvers were imminent, it could be shut down to save electrical power. Due to the nature of core memory, programs and data stored magnetically in the cores would not disappear when the power was off, as in present day semiconductor memories. This made it possible to load the next set of modules, if necessary, from the Auxiliary Tape Memory, enter any needed parameters, and then shut down the machine until shortly before its next use. It took 20 seconds for the machine to run its start-up diagnostics upon restoration of power. After the diagnostics ran successfully, the current program load was ready for use, all parameters intact.

GT-IV was following such a procedure in preparing for re-entry on June 7, 1965. The computer was placed in the RNTY mode, and the crew received and entered updated parameters given to them when they were in contact with the ground stations. But when they tried to turn the machine off, the manual on/off switch did not function. The power had to be cut off by another means, and the re-entry handled manually⁴⁰.

Using the computer for catch-up and rendezvous was a relatively simple task. The difference between catch-up and rendezvous is that catch-up maneuvers are executed to put the spacecraft into position to make an orbit-change maneuver. After the orbit change the spacecraft is prepared to rendezvous with the target⁴¹. Crews began the catch-up by entering the ground-calculated rendezvous angle desired into address 83. The rendezvous angle indicated how much farther along in a 360-degree orbit the rendezvous was to take place. For example, if the crew desired rendezvous one-third orbit ahead, 12000 was entered into address 83 using the MDIU. The interval at which the pilot wanted to see updates was then entered in address 93. For example, if 04000 was entered, the computer would display on the IVI any required velocity changes at 120 degrees from the rendezvous point (the start), 80 degrees to go, and 40 degrees to go. If the IVI indicated that the computer had calculated that such a rendezvous was possible within the designated fuel limits, the astronauts pressed the START button and the IVI displayed the first set of velocity differentials. The pilot then fired the thrusters until the displays were all at zero (Astronaut John Young reported that there was a tendency to "overshoot" in trying to burn to zero⁴²). After that, nothing was done unless the next update indicated a need for more velocity adjustments⁴³. The astronauts also did paper-and-pencil calculations of the velocity changes as a backup by using special nomographs based on time and angles to the target⁴⁴. These backup calculations were compared with the ground-calculated solution as well as the computer solution. However, the figures computed on-board were considered the primary solution for the terminal-phase intercept

maneuver⁴⁵. In the rendezvous mode, the radar would feed information to the computer, which used it to calculate the velocity adjustments needed for final approach⁴⁶.

These examples of the use of the computer on a typical flight demonstrate that it was a relatively straightforward assistant in guidance and navigation. It permitted the Gemini astronauts to be independent of the ground in accomplishing rendezvous from the terminal-phase intercept maneuver to station keeping, a valuable rehearsal for the lunar orbit rendezvous required for the Apollo program. The astronauts participated in both the hardware and software design of the computer and its interfaces, and they were able to go to Owego and be put in the man-in-the-loop simulations. By flight time, like everything else in the cockpit, use of the computer was second nature.

THE IMPACT OF THE GEMINI DIGITAL COMPUTER

The Gemini Digital Computer was a transitional machine. Dale F. Bachman of IBM characterized it as the "last of a dying breed. It was an airborne computer, ruggedized, special purpose, and slow"⁴⁷. Nonetheless, its designers claim an impressive list of firsts:

- The first digital computer on a manned spacecraft.
- The first use of core memory with nondestructive readout. The machine was designed in an era of rotating drum memories, its designers considered it a step forward⁴⁸.
- IBM's first completely silicon semiconductor computer⁴⁹.
- The first to use glass delay lines as registers⁵⁰.
- Technologically advanced in the area of packaging density⁵¹.
- The first airborne or spaceborne computer to use an auxiliary memory⁵².

Development of the Gemini computer helped IBM in significant ways. It contributed more than anything else to the hardware and software of the 4Pi series of computers⁵³. This series eventually produced the computer used on Skylab and the AP-101 used in the Shuttle. It also helped to develop IBM's reputation for delivering reliable and durable spaceborne hardware and software⁵⁴. One Gemini computer restarted successfully after being soaked in salt water for 2

weeks. Another used system went on to NASA's Electronics Research Laboratory in Boston for use on vertical and short takeoff and landing projects⁵⁵. Coupled with IBM's involvement in the real-time computing centers used to monitor Mercury and Gemini missions, the company established itself as a major contributor to America's space program as it had been to the military research and development effort. Out of early military work came computer systems such as the Harvard Mark I, the 701, and SAGE computers used in air defense. However, even though identification with the space program has been maintained through several high-visibility projects, no significant commercial hardware products resulted as spinoffs.

For NASA, Gemini and its on-board computer proved that a reliable guidance and navigation system could be based on digital computers. It was a valuable test bed for Apollo techniques, especially in rendezvous. However, the Gemini digital computer itself was totally unlike the machines used in Apollo. With its Auxiliary Tape Memory and core memory, the Gemini computer was more like the Skylab and Shuttle general purpose computers. It is in those systems where its impact is most apparent.

2

Computers on Board the Apollo Spacecraft

THE NEED FOR AN ON-BOARD COMPUTER

The Apollo lunar landing program presented a tremendous managerial and technical challenge to NASA. Navigating from the earth to the moon and the need for a certain amount of spacecraft autonomy dictated the use of a computer to assist in solving the navigation, guidance, and flight control problems inherent in such missions. Before President John F. Kennedy publicly committed the United States to a "national goal" of landing a man on the moon, it was necessary to determine the feasibility of guiding a spacecraft to a landing from a quarter of a million miles away. The availability of a capable computer was a key factor in making that determination.

The Instrumentation Laboratory of the Massachusetts Institute of Technology (MIT) had been working on small computers for aerospace use since the late 1950s. Dr. Raymond Alonso designed such a device in 1958–1959¹. Soon after, Eldon Hall designed a computer for an unmanned mission to photograph Mars and return². That computer could be interfaced with both inertial and optical sensors. In addition, MIT was gaining practical experience as the prime contractor for the guidance system of the Polaris missile. In early 1961, Robert G. Chilton at NASA–Langley Space Center and Milton Trageser at MIT set the basic configuration for the Apollo guidance system³. An on-board digital computer was part of the design. The existence of these preliminary studies and the confidence of C. Stark Draper, then director of the Instrumentation Lab that now bears his name, contributed to NASA's belief that the lunar landing program was possible from the guidance standpoint.

The presence of a computer in the Apollo spacecraft was justified for several reasons. Three were given early in the program: (a) to avoid hostile jamming, (b) to prepare for later long-duration (planetary) manned missions, and (c) to prevent saturation of ground stations in the event of multiple missions in space simultaneously⁴. Yet none of these became a primary justification. Rather, it was the reality of physics expressed in the 1.5-second time delay in a signal path from the earth to the moon and back that provided the motivation for a computer in the lunar landing vehicle. With the dangerous landing conditions that were expected, which would require quick decision making and feedback, NASA wanted less reliance on ground-based computing⁵. The choice, later in the program, of the lunar orbit rendezvous method over direct flight to the moon, further justified an on-board computer since the lunar orbit insertion would take place on the far side of the moon, out of contact with the earth⁶. These considerations and the consensus among MIT people that autonomy was desirable ensured the place of a computer in the Apollo vehicle.

Despite the apparent desire for autonomy expressed early in the

program, as the mission profile was refined and the realities of building the actual spacecraft and planning for its use became more immediate, the role of the computer changed. The ground computers became the prime determiners of the vehicle's position in three-dimensional space "at all times" (except during maneuvers) in the missions⁷. Planners even decided to calculate the lunar orbit insertion burn on the ground and then transmit the solution to the spacecraft computer, which somewhat negated one of the reasons for having it. Ultimately, the actual Apollo spacecraft was only autonomous in the sense it could return safely to earth without help from the ground⁸.

Even with its autonomous role reduced, the Apollo on-board computer system was integrated so fully into the spacecraft that designers called it "the fourth crew member"⁹. Not only did it have navigation functions, but also system management functions governing the guidance and navigation components. It served as the primary source of timing signals for 20 spacecraft systems¹⁰. The Apollo computer system did not have as long a list of responsibilities as later spacecraft computers, but it still handled a large number of tasks and was the object of constant attention from the crew.

MIT CHOSEN AS HARDWARE AND SOFTWARE CONTRACTOR

On August 9, 1961, NASA contracted with the MIT Instrumentation Lab for the design, development, and construction of the Apollo guidance and navigation system, including software. The project manager for this effort was Milton Trageser, and David Hoag was the technical director¹¹. MIT personnel generally agree that they were chosen because their work on Polaris proved that they could handle time, weight, and performance restrictions and because of their previous work in space navigation¹². In fact, the Polaris team was moved almost intact to Apollo¹³. Despite their experience with aerospace computers, the Apollo project turned out to be a genuine challenge for them. As there were no fixed specifications when the contract was signed, not until late 1962 did MIT have a good idea of Apollo's requirements¹⁴. One of the MIT people later recalled that

If the designers had known then [1961] what they learned later, or had a complete set of specifications been available...they would probably have concluded that there was no solution with the technology of the early 1960s¹⁵.

Fortunately, the technology improved, and the concepts of computer science applied to the problem also advanced as MIT developed the system.

NASA's relationship with MIT also proved to be educational. The Apollo computer system was one of NASA's first real-time, large scale software application contracts¹⁶. Managing such a project was completely outside the NASA experience. A short time after making the Apollo guidance contract, NASA became involved in developing the on-board software for Gemini (a much smaller and more controllable enterprise) and the software for the Integrated Mission Control Center. Different teams that started within the Space Task Group, later as part of the Manned Spacecraft Center in Houston, managed these projects with little interaction until the mid-1960s, when the two Gemini systems approached successful completion and serious problems remained with the Apollo software. Designers borrowed some concepts to assist the Apollo project. In general, NASA personnel involved with developing the Apollo software were in the same virgin territory as were MIT designers. They were to learn together the principles of software engineering as applied to real-time problems.

THE APOLLO COMPUTER SYSTEMS

The mission profile used in sending a man to the moon went through several iterations in the early 1960s. For a number of reasons, planners rejected the direct flight method of launching from the earth, flying straight to the moon, and landing directly on the surface. Besides the need for an extremely large booster, it would require flawless guidance to land in the selected spot on a moving target a quarter of a million miles away. A spacecraft with a separate lander would segment the guidance problem into manageable portions. First, the entire translunar spacecraft would be placed in earth orbit for a revolution or two to properly prepare to enter an intercept orbit with the moon. Upon arriving near the moon, the spacecraft would enter a lunar orbit. It was easier to target a lunar orbit window than a point on the surface. The lander would then detach and descend to the surface, needing only to guide itself for a relatively short time. After completion of the lunar exploration, a part of the lander would return to the spacecraft still in orbit and transfer crew and surface samples, after which the command module (CM) would leave for earth.

With a lunar orbit rendezvous mission, more than one computer would be required, since both the CM and the lunar excursion module (LEM) needed on-board computers for the guidance and navigation function. The CM's computer would handle the translunar and trans-earth navigation and the LEM's would provide for autonomous landing, ascent, and rendezvous guidance.

NASA referred to this system with its two computers, identical in design but with different software, as the Primary Guidance, Naviga-

tion, and Control System (PGNCS, pronounced "pings"). The LEM had an additional computer as part of the Abort Guidance System (AGS), according to the NASA requirement that a first failure should not jeopardize the crew. Ground systems backed up the CM computer and its associated guidance system so that if the CM system failed, the spacecraft could be guided manually based on data transmitted from the ground. If contact with the ground were lost, the CM system had autonomous return capability. Since the lunar landing did not allow the ground to act as an effective backup, the LEM had the AGS to provide backup ascent and rendezvous guidance. If the PGNCS failed during descent, the AGS would abort to lunar orbit and assist in rendezvous with the CM. It would not be capable of providing landing assistance except to monitor the performance of the PGNCS. Therefore the computer systems on the Apollo spacecraft consisted of three processors, two as part of the PGNCS and one as part of the AGS.

EVOLUTION OF THE HARDWARE:

Old Technology versus New: Block I and Block I Designs

The computer envisioned by MIT's preliminary design team in 1961 was a shadow of what actually flew to the moon in 1969. There always seem to be enough deficiencies in a final product that the designers wish they had a second chance. In some ways the Apollo guidance computer was a second chance for the MIT team since most worked on the Polaris computer. That was MIT's most ambitious attempt at an "embedded computer system," a computer that is intrinsic to a larger component, such as a guidance system. Although the Apollo computer started out to be quite similar to Polaris, it evolved into something very different. The Apollo guidance computer had two flight versions: Block I and Block II. Block I was basically the same technology as the Polaris system. Block II incorporated new technology within the original architecture.

Several factors led from the Block I design to Block II. NASA's challenges to the MIT contract and the decision to use the rendezvous method instead of a direct ascent to the moon were decisive. A third factor related to reliability. Finally, the benefits of the new technology influenced the decision to make Block II.

Before NASA let the contract to MIT, but after it was known that the Instrumentation Laboratory would be accorded "sole source" status, several NASA individuals began studying ways to consolidate flight computer development. In June 1961, Harry J. Goett of Goddard Space Flight Center recommended that the computers needed for the Orbiting Astronomical Observatory (OAO), Apollo, and the Saturn launch vehicle be the same. He cited an IBM proposal for \$5

million to do just that¹⁷. On the same day Goett's recommendation, RCA proposed the use of a 420-cubic-inch computer with only an 80-watt power consumption and 24-bit word size as a general-purpose spaceborne computer¹⁸. This proposal got nowhere and NASA's Robert G. Chilton challenged Goett's idea, showing that the expected savings would not materialize. Even though the projected cost of the Apollo computer would decrease to \$8 million from \$10 million, the OAO development costs would rise from \$1.5 million to \$5 million¹⁹. Ironically, in the same month, Ramon Alonso from MIT met with Marshall Space Flight Center personnel about the use of the Apollo computer in the Saturn.²⁰ Although MIT got the Apollo contract and IBM got the contract for the Saturn computer, the idea of a duplicate system did not die. Two years later, when the deficiencies of the Polaris-based system were obvious and the solutions offered by the new technology of the Block II version still unproved, David W. Gilbert, NASA manager for Apollo guidance and control, proposed replacing the MIT machine with the one IBM was building for Saturn²¹. It did not occur because Gilbert wanted NASA to accept the reprogramming costs, and the existing configuration of the IBM computer would not fit in the space allotted for it in the CM. Nevertheless, MIT would still have to deal with NASA misgivings about the hardware design as late as May 1964, when Maj. Gen. Samuel C. Phillips, deputy director of the Apollo Program, reported on a meeting to discuss the use of the triple modular redundant Saturn launch vehicle computer in Apollo²².

The decision to have a separate CM and the LEM influenced the transition to Block II by providing a convenient dividing point in the Apollo program. The early Apollo development flights were to use the CM only. Later flights would include the LEM. Since Block I design and production had already proceeded, planners decided to use the existing Block I in the unmanned and early manned development flights (all relatively simple earth-orbital missions) and to switch to Block II for the more complex combined CM-LEM missions²³.

Reliability was another force behind Block II. During early planning for the guidance system, redundancy was considered a solution to the basic reliability problem. Designers thought that two computers would be needed to provide the necessary backup; however, they dropped this scheme for two reasons. The ground had primary responsibility for determining the state vector (the position of the craft in three-dimensional space) in translunar, lunar orbit, and transearth flight²⁴. Moreover, none of the variations of the two-computer or other redundancy schemes could meet the power, weight, and size requirements.²⁵ One way to provide some measure of protection is to make the computer repairable in flight. The Block I design, due to its modularity, could be fixed during a mission that carried appropriate spares. At any rate, its predicted mean time between failures (MTBF)

was 4,200 hours, about 20 times longer than the longest projected mission²⁶. But Block I's repair capability became a negative factor when sealing the computer began to be considered more important to reliability than the ability to repair it²⁷. Aside from packaging, overall malfunction detection was improved in the Block II design, further increasing reliability²⁸.

The most important reason for going to Block II was the availability of new technology. The Block I design used core transistor logic. It had several disadvantages:

- It could not be complemented, a very important basic operation in computer arithmetic that changes a one to a zero or vice versa.
- It had the characteristic of "destructive readout," in which a datum read from a flip-flop using core transistor logic loses the datum; that forces the inclusion of a circuit to rewrite the datum if it is to be retained after the read cycle.
- Memory cycle time could not be fixed: in Block I it was an average of 19.5 milliseconds, which was quite slow for computers at the time, and the varying cycle caused timing problems within the machine²⁹.

These disadvantages led MIT to begin studying, as early as 1962, the possible use of integrated circuits (ICs) to replace core transistor circuits. ICs, so ubiquitous today, were only 3 years old then and thus had little reliability history. It was therefore difficult to consider their use in a manned spacecraft without convincing NASA that the advantages far outweighed the risks.

To accomplish this, the MIT team chose a direct-coupled transistor logic (DCTL) NOR gate with a three-input element,³⁰ consisting of three transistors and four resistors. NOR logic inverts the results of applying a Boolean OR operation to the three inputs. It took nearly 5,000 of these simple circuits to build an Apollo computer. Using a variety of circuits would have simplified the design since the component count would have been reduced, but by using the NOR alone, overall simplicity and reliability increased³¹. Also, the time it took the machine to cycle became fixed at 11.7 milliseconds, a double bonus in that speed increased and cycle time was consistent³².

Aside from these advantages, MIT believed that the lead time to the first flight would permit reliability to be established and the cost of the ICs to come down³³. At the time, the production of such circuits was low, and they were more expensive than building core transistor circuits. To place the production rate in perspective, MIT chose the NOR ICs in the fall of 1962 and by the summer of 1963, 60% of the

total U.S. output of microcircuits was being used in Apollo prototype construction³⁴. This is one of the few cases in which NASA's requirements acted as a direct spur to the computer industry. When MIT switched to ICs, it kept the Apollo computer as "state of the art" at least during its design stage. It would be hopelessly outdated technologically by the time of the lunar landing 7 years later, but in 1962, using the new microcircuits seemed to be a risk. This view is contested by one member of the MIT team, who later said that the decision "wasn't bold; it was just the easy thing to do to get the size and power and other requirements"³⁵.

With the ICs fully incorporated in the Apollo computer and the transition from Block I to Block II complete, NASA possessed a machine that was more up to date technologically. It had double the memory of the largest Block I, more I/O capability, was smaller, and required less power.³⁶ Besides, it was also more reliable, which was, as always, the major consideration.

THE APOLLO GUIDANCE COMPUTER: HARDWARE

Overall Configuration and Architecture

The Apollo Guidance Computer was fairly compact for a computer of its time. The CM housed the computer in a lower equipment bay, near the navigator's station. Block II measured 24 by 12.5 by 6 inches, weighed 70.1 pounds, and required 70 watts at 28 volts DC. The machine in the lunar module was identical.

Crew members could communicate with either computer using display and keyboard units (DSKY, pronounced "disky"). Two DSKYs were in the CM, one on the main control panel and one near the optical instruments at the navigator's station. In addition, a "mark" button was at the navigator's station to signal the computer when a star fix was being taken. A single DSKY was in the lunar module. The DSKYs were 8 by 8 by 7 inches and weighed 17.5 pounds. As well as the DSKYs, the computer directly hooked to the inertial measurement unit and, in the CM, to the optical units.

The choice of a 16-bit word size was a careful one. Many scientific computers of the time used 24-bit or longer word lengths and, in general, the longer the word the better the precision of the calculations. MIT considered the following factors in deciding the word length: (a) precision desired for navigation variables, (b) range of input variables, and (c) the instruction word format³⁷. Advantages of a shorter word are simpler circuits and higher speeds, and greater precision could be obtained by using multiple words.³⁸ A single precision word of data consisted of 14 bits, with the other 2 bits as a sign bit (with a one indicating negative) and a parity bit (odd parity). Two ad-

acent words yielded "double precision" and three adjacent, "triple precision." To store a three-dimensional vector required three double precision words³⁹. Data storage was as fractions (all numbers were less than one)⁴⁰. An instruction word used bits 15–13 (they were numbered descending left to right) as an octal operation code. The address used bits 12–1. Direct addressing was limited, so a "bank register" scheme (discussed below) existed to make it possible to address the entire memory⁴¹.

The Apollo computer had a simple packaging system. The computer circuits were in two trays consisting of 24 modules. Each module had two groups of 60 flat packs with 72-pin connectors. The flatpacks each held two logic gates⁴². Tray A held the logic circuits, interfaces, and the power supply, and tray B had the memory, memory electronics, analog alarm devices, and the clock, which had a speed of one megahertz⁴³. All units of the computer were hermetically sealed⁴⁴. The memory in Block II consisted of a segment of erasable core and six modules of core rope fixed memory. Both types are discussed fully below.

The Apollo computer used few flip-flop registers due to size and weight considerations⁴⁵, but seven key registers in the computer did use flip-flops:

- The accumulator, register 00000, referenced as "A".
- The lower accumulator, 000001, "L".
- The return address register, 000002, "Q".
- The erasable bank register, 000003, "EB".
- The fixed bank register, 000004, "FB".
- The next address, 000005, "Z".
- The both bank register, 000006, "BB" (data stored in EB and FB were automatically together here)⁴⁶.

The use of bank registers enabled all of the machine's memory to be addressed. The largest number that can be contained in 12 bits is 8,192. The fixed memory of the Apollo computer contained over four times that many locations. Therefore, the memory divided into "banks" of core, and the addressing could be handled by first indicating which bank and then the address within the bank. For example, taking the metaphor "address" literally, there are probably hundreds of "100 Main Street" addresses in any state, but by putting the appropriate city on an envelope, a letter can be delivered to the intended 100 Main Street without difficulty.

The computer banks were like the cities of the analogy. The erasable bank register held just 3 bits that were used to extend the direct

addressing of the erasable memory to its "upper" region, and the fixed bank register held 5 bits to indicate which core rope bank to address. In addition, for the addresses needing a total of 16 bits, a "super bank bit" could be stored and concatenated to the fixed bank data and the address bits in the instruction word⁴⁷. This scheme made it possible to handle the addressing using a 16-bit word, but it placed a greater burden on the programmers, who, in an environment short of adequate tools, had to attend to setting various bit codes in the instructions to indicate the use of the erasable bank, fixed bank, or super bank bit. Although this simplified the hardware, it increased the complexity of the software, an indication that the importance of the software was not fully recognized by the designers.

To further reduce size and weight, the Apollo computer was designed with a single adder circuit, which the computer used to update incremental inputs, advance the next address register, modify specified addresses, and do all the arithmetic⁴⁸. The adder and the 16 I/O channels were probably the busiest circuits in the machine.

Memory

The story of memory in the Apollo computer is a story of increasing size as mission requirements developed. In designing or purchasing a computer system for a specific application, the requirements for memory are among the most difficult to estimate. NASA and its computer contractors have been consistently unable to make adequate judgments in this area. Apollo's computer had both permanent and erasable memory, which grew rapidly over initial projections.

Apollo's computer used erasable memory cells to store intermediate results of calculations, data such as the location of the spacecraft, or as registers for logic operations. In Apollo, they also contained the data and routines needed to ready the computer for use when it was first turned on. Fixed memory contained programs that did not need to be changed during the course of a mission. The cycle times of the computer's memories were equal for simplicity of operation⁴⁹.

MIT's original design called for just 4K words of fixed memory and 256 words of erasable (at the time, two computers for redundancy were still under consideration)⁵⁰. By June 1963, the figures had grown to 10K of fixed and 1K of erasable⁵¹. The next jump was to 12K of fixed, with MIT still insisting that the memory requirement for an *autonomous* lunar mission could be kept under 16K⁵²! Fixed memory leapt to 24K and then finally to 36K words, and erasable memory had a final configuration of 2K words.

Lack of memory caused constant and considerable software

development problems, despite the increase of fixed memory 18 times over original estimates and erasable memory 16 times. Part of the software difficulties stemmed from functions and features that had to be dropped because of program size considerations, and part because of the already described addressing difficulties. If the original designers had known that so much memory would be needed, they might not have chosen the short word size, as a 24-bit word could easily directly address a 36K bank, with enough room for a healthy list of instruction codes.

One reason the designers underestimated the memory requirements was that NASA did not provide them with detailed specifications as to the function of the computer. NASA had established a need for the machine and had determined its general tasks, and MIT received a contract based on only a short, very general requirements statement in the request for bid. The requirements started changing immediately and continued to change throughout the program. Software was not considered a driving factor in the hardware design, and the hardware requirements were, at any rate, insufficient.

The actual composition of the memory was fairly standard in its erasable component but somewhat unique in its fixed component. The erasable memory consisted of coincident-current ferrite cores similar to those on the Gemini computer, and the fixed memory consisted of *core rope*, a high-density read-only memory using cores of similar material composition as the erasable memory but of completely different design. MIT adopted the use of core rope in the original Mars probe computer design and carried it over to the Apollo⁵³ Chief advantage of the core rope was that it could put more information in less space, with the attendant disadvantages that it was difficult to manufacture and the data stored in it were unchangeable once it left the factory (see Box 2-1).

Box 2-1: Core Rope: A Unique Data Storage Device

Each core in an erasable memory could store one bit of information, and each core in the core rope fixed memory could store four *words* of information. In the erasable memory, cores are magnetized either clockwise or counterclockwise, thus indicating the storage of either a one or a zero. In fixed memory, each core functions as a miniature transformer, and up to 64 wires (four sets of 16-bit words) could be connected to each core. If a wire passed through a particular core, a one would be read. If a particular wire bypassed the core, a zero would be read. For example, to store the data word 1001000100001111 in a core, the first, fourth, eighth, and thirteenth through sixteenth wires would pass through that core, the rest would bypass it. A 2-bit select code would identify which of the four words on a core was being read, and the indicated 16 bits would be sent to the appropriate register⁵⁴. In this way, up to 2,000 bits could be stored in a cubic inch⁵⁵.

The computer contained core rope arranged in six modules, and each module contained 6,144 16-bit words⁵⁶. The modules further divided into "banks" of 1,024 words. The first two banks were called the "fixed-fixed memory" and could be directly addressed by 12 bits in an instruction word. The remaining 34 were addressable as described in the text, using the 5-bit contents of the fixed bank register and the 10 bits in an instruction word⁵⁷.

The use of core rope constrained NASA's software developers. Software to be stored on core rope had to be delivered months before a scheduled mission so that the rope could be properly manufactured and tested. Once manufactured, it could not be altered easily since each sealed module required rewiring to change bits. The software not only had to be finished long in advance, but it had to be perfect.

Even though common sense indicates that it is advantageous to complete something as complex and important as software long before a mission so that it can be used in simulators and tested in various other ways, software is rarely either on time or perfect. Fortunately for the Apollo program, the nature of core rope put a substantial amount of pressure on MIT's programmers to do it right the first time. Unfortunately, the concept of "bug"-free software was alien to most programmers of that era. Programming was a fully iterative process of removing errors. Even so, many "bugs" would carry over into a delivered product due to unsophisticated testing techniques. Errors found before a particular system of rope was complete could be fixed at the factory⁵⁸, but most others had to be endured. Raytheon, the subcontractor that built the ropes, could eliminate hard-wiring errors introduced during manufacture by testing the rope modules against the

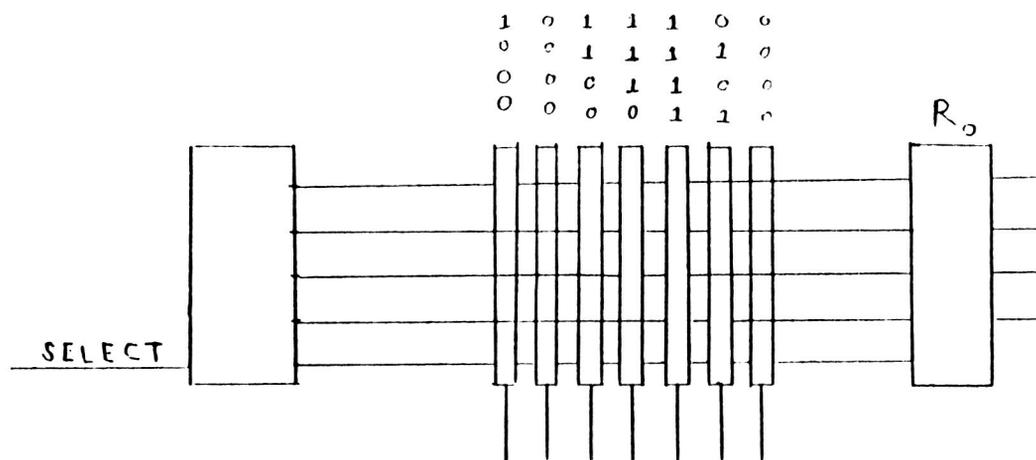


Figure 2-1. This diagram shows the principle behind core rope. Suppose that the data shown above the cores in the drawing is to be stored in the specific core. Thus 1000 is stored in the first core on the left by attaching the top wire from the select circuit to the core and bypassing it with the next three wires. When that core is selected for reading, the wire attached to the core will indicate a "one" because all cores in a rope are permanently charged as ones; the wires bypassing the core will indicate zeroes.

delivery tape of the programs. The company built a device to do this⁵⁹.

Production Problems and Testing

Development and production of the Apollo guidance, navigation, and control system reflected the overall speed of the Apollo program. Design of the system began in the second quarter of 1961, and NASA installed a Block I version in a spacecraft on September 22, 1965. Release of the original software (named CORONA) was in January 1966, with the first flight on August 25, 1966⁶⁰. Less than 3 years after that, designers achieved the final program objective. Even though fewer than two dozen spacecraft flew, NASA authorized the building of 75 computers and 138 DSKYs. Fifty-seven of the computers and 102 of the crew interfaces were of the Block II design⁶¹. This represents a considerable production for a special-purpose computer of the type used in Apollo. The need to quickly build high-quality, high-reliability computers taxed the abilities of Raytheon.

Through AC Electronic Circuits (contractor for the entire guidance system), Raytheon was chosen to build the computers MIT had designed largely because of its Polaris experience, but it had

never built a computer as complex as the one for Apollo. The Polaris machine was much simpler. Despite the use of experienced Polaris personnel, Raytheon's production division for the Apollo computer went from 800 to 2,000 employees in a year's time in order to handle the increased difficulties and speed of production⁶².

Rapid growth, underestimation of production requirements, and reliability problems dogged Raytheon throughout the program. Changes in design made by MIT in late 1962 caused the company its initial trouble. The original request for proposal had featured Polaris techniques, so Raytheon bid low, expecting to use the same tools and production line for the Apollo machine. The changes in component types and memory size caused cost estimates to nearly double, resulting in considerable friction with NASA⁶³. NASA was also worried when two computers and fully 50% of the Block I DSKYs failed vibration tests⁶⁴. These failures turned out to be largely caused by contaminated flat packs and DSKY relays. Particles would shake loose during vibration testing⁶⁵. The Block II computers would not work at first due to excessive signal propagation time in the micrologic interconnection matrix. The solution was to switch from nickel ribbon connectors to a circuit board, causing an increase of \$500,000 in production costs⁶⁶.

These sorts of problems caused the Manned Spacecraft Center to authorize a complete design review of the AGC in February 1966. The lack of adequate support documentation was found to be the most significant fault of the Block II computer⁶⁷. This sort of problem is usually the result of speeding up development to the point at which changes are not adequately documented.

Continuous and careful attention to reliability led to the discovery of problems. Builders flight-screened components lot by lot⁶⁸. Post-production hardware tests included vibration, shock, acceleration, temperature, vacuum, humidity, salt fog, and electronic noise.⁶⁹ As D.C. Fraser, an engineer on the project, later remarked, "reliability of the Apollo computer was bought with money"⁷⁰.

THE APOLLO GUIDANCE COMPUTER: SOFTWARE

Development of the on-board software for the Apollo program was an important exercise both for NASA and for the discipline of software engineering. NASA acquired considerable experience in managing a large, real-time software project that would directly influence the development of the Shuttle on-board software. Software engineering as a specific branch of computer science emerged as a result of experiences with large-size military, civilian, and spaceborne systems. As one of those systems, the Apollo software effort helped

provide examples both of failure and success that could be incorporated into the methodology of software engineering.

In the Apollo program, as well as other space programs with multiple missions, system software and some subordinate computer programs are only written once, with some modifications to help integrate new software. However, each mission generates new operational requirements for software, necessitating a design that allows for change. Since 1968, when designers first used the term software engineering, consciousness of a software life cycle that includes an extended operational maintenance period has been an integral part of proper software development.

Even during the early 1960s, the cycle of requirements definition, design, coding, testing, and maintenance was followed, if not fully appreciated, by software developers. A Bellcomm report prepared for the Apollo program and dated November 30, 1964 could serve as an excellent introduction to the concept today⁷¹. The important difference from present practice was the report's recommendation that modules of code be limited to 200 to 300 lines, about five times larger than current suggestions. The main point of the report (and the thrust of software engineering) was that software *can* be treated the same way as hardware, and the same engineering principles can apply. However, NASA was more used to hardware development than to large-scale software and, thus, initially failed adequately to control the software development. MIT, which concentrated on the overall guidance system, similarly treated software as a secondary occupation⁷². This was so even though MIT manager A.L. Hopkins had written early in the program that "upon its execution rests the efficiency and flexibility of the Apollo Guidance and Navigation System"⁷³. Combined with NASA's inexperience, MIT's non-engineering approach to software caused serious development problems that were overcome only with great effort and expense. In the end NASA and MIT produced quality software, primarily because of the small-group nature of development at MIT and the overall dedication shown by nearly everyone associated with the Apollo program⁷⁴.

Managing the Apollo Software Development Cycle

One purpose of defining the stages in the software development cycle and of providing documentation at each step is to help control the production of software. Programmers have been known to inadvertently modify a design while trying to overcome a particular coding difficulty, thus making it impossible to fulfill the specification. Eliminating communication problems and preventing variations from the designed solution are among the goals of software engineering. In

the Apollo program, with an outside organization developing the software, NASA had to provide for quality control of the product. One method was a set of standing committees; the other was the acceptance cycle.

Three boards contributed directly to the control of the Apollo software and hardware development. The Apollo Spacecraft Configuration Control Board monitored and evaluated changes requested in the design and construction of the spacecraft itself, including the guidance and control system, of which the computer was a part. The Procedures Change Control Board, chaired by Chief Astronaut Donald K. Slayton, inspected items that would affect the design of the user interfaces. Most important was the Software Configuration Control Board, established in 1967 in response to continuing problems and chaired for a long period by Christopher Kraft. It controlled the modifications made to the on-board software⁷⁵. All changes in the existing specification had to be routed through this board for resolution. NASA's Stan Mann commented that MIT "could not change a single bit without permission"⁷⁶.

NASA also developed a specific set of review points that paralleled the software development cycle. The Critical Design Review (CDR) resulted in acceptance of specifications and requirements for a given mission and placed them under configuration control. It followed the preparation of the requirements definition, guidance equation development, and engineering simulations of the equations. Next came a First Article Configuration Inspection (FACI). Following the coding and testing of programs and the production of a validation plan, it marked the completion of the development stage and placed the software code under configuration control. After testing was completed, the Customer Acceptance Readiness Review (CARR) certified that the validation process resulted in correct software. After the CARR, the code would be released for core rope manufacture. Finally the Flight Readiness Review (FRR) was the last step in clearing the software for flight⁷⁷. The acceptance process was mandatory for each mission, providing for consistent evaluation of the software and ensuring reliability. The unique characteristic of ICs of the Apollo software appeared at each stage of the software life cycle.

Requirements Definition

Defining requirements is the single most difficult part of the software development cycle. The specification is the customer's statement of what the software product is to do. Improperly prepared or poorly defined requirements mean that the resulting software will likely be incomplete and unusable. Depending on the type of project, the customer may have little or a lot to do with the preparation of the

specification. In most cases, a team from the software developers works with the customer.

MIT worked closely with NASA in preparing the Guidance and Navigation System Operations Plan (GSOP), which served as the requirements document for each mission. NASA's Mission Planning and Analysis Division at the Manned Spacecraft Center provided detailed guidance requirements right down to the equation level⁷⁸. Often these requirements were in the form of flow charts to show detailed logic⁷⁹. The division fashioned these requirements into a controlled document that contained specific mission requirements, preliminary mission profile, preliminary reference trajectory, and operational requirements for spacecraft guidance and navigation. NASA planned to review the GSOP at launch minus 18 months, 16 months, 14 months and then to baseline or "freeze" it at 13.5 months before launch. The actual programs were to be finished at launch minus 10.5 months and tested until 8 months ahead, when they were released to the manufacturer, with tapes also kept at MIT and sent to Houston, North American (CM manufacturer), and Grumman (LEM manufacturer) for use in simulations. At launch minus 4 months the core ropes were to be completed and used throughout the mission⁸⁰.

In software engineering practice today, the specification document is followed by a design document, from which the coding is done. Theoretically, the two together would enable *any* competent programmer to code the program. The GSOPs contained characteristics of both a specification and design document. But, as one of the designers of the Apollo and Shuttle software has said, "I don't think I could give you the requirements for Apollo and have you build the flight software"⁸¹. In fact, the plans varied both in what they included and in the level of detail requirements. This variety gave MIT considerable latitude when actually developing the flight software, thus reducing the chance that it would be easily verified and validated.

Coding: Contents of the Apollo Software

By 1963, designers determined that the Apollo computer software would have a long list of capabilities, including acting as backup to the Saturn booster, controlling aborts, targeting, all navigation and flight control tasks, attitude determination and control, digital autopilot tasks, and eventually all maneuvers involving velocity changes⁸². Programs for these tasks had to fit in the memories of two small computers, one in the CM and one in the LEM. Designers developed the programs using a Honeywell 1800 computer and later an IBM 360, but never with the actual flight hardware. The development computers generated binary object code and a listing⁸³. The tape

containing the object code would be tested and eventually released for core rope manufacture. The listing served as documentation of the code⁸⁴.

Operating System Architecture

The AGC was a priority-interrupt system capable of handling several jobs at one time. This type of system is quite different from a "round-robin executive." In the latter, programs have a fixed amount of time in which to run before being suspended while the computer moves on to the remaining pending jobs, thus giving each job the same amount of attention. A priority-interrupt system is always executing the one job with the highest priority; it then moves on to others of equal or lower priority in its queue.

The Apollo control programs included two related to job scheduling: the Executive and the Waitlist. The Executive could handle up to seven jobs at once while the Waitlist had a limit of nine short tasks⁸⁵. Waitlist tasks had execution times of 4 milliseconds or less. If a task ran longer than that, it would be promoted by the Waitlist to "job" status and moved to the Executive's queue⁸⁶. The Executive checked every 20 milliseconds for jobs or tasks with higher priorities than the current ones⁸⁷. It also managed the DSKY displays⁸⁸. If the Executive checked the priority list and found no other jobs waiting, it executed a program called DUMMY JOB continuously until another job came into the queue⁸⁹.

The Executive had other duties as part of controlling jobs. One solution to the tight memory in the AGC was the concept of time-sharing the erasable memory⁹⁰. No job had permanent claim to any registers in the erasable store. When a job was being executed, the Executive would assign it a "coreset" of 12 erasable memory locations. Also, when interpretive jobs were being run (the Interpreter is explained below), an additional 43 cells were allocated for vector accumulation (VAC). The final lunar landing programs had eight coresets in the LEM computer and just seven in the CM. Both had five VACs⁹¹. Moreover, memory locations were given multiple assignments where it was assured that the owning processes would never execute at the same time. This approach caused innumerable problems in testing as software evolved and memory conflicts were created due to the changes.

Programming the AGC

One can program a computer on several levels. Machine code, the actual binary language of the computer itself, is one method of specifying instructions. However, it is tedious to write and prone to error. Assembly language, which uses mnemonics for instructions (e. g., ADD in place of a 3-bit operation code) and, depending on its sophistication, handles addressing, is at a higher level. Most programmers in the early 1960s were quite familiar with assembly languages, but such programs suffered from the need to put too much responsibility in the hands of the programmer. For Apollo, MIT developed a special higher order language that translated programs into a series of subroutine linkages, which were interpreted at execution time. This was slower than a comparable assembly language program, but the language required less storage to do the same job⁹². The average instruction required two machine cycles—about 24 milliseconds—to execute⁹³.

The interpreter got a starting location in memory, retrieved the data in that location, and interpreted the data as though it were an instruction⁹⁴. Instead of having only the 11 instructions available in assembler, up to 128 pseudoinstructions were defined⁹⁵. The larger number of instructions in the interpreter meant that equations did not have to be broken down excessively⁹⁶. This increased the speed and accuracy of the coding.

The MIT staff gave the resulting computer programs a variety of imaginative names. Many, such as SUNDISK, SUNBURST, and SUNDIAL, related to the sun because Apollo was the god of the sun in the classical period. But the two major lunar flight programs were called COLOSSUS and LUMINARY. The former was chosen because it began with "C" like the CM, and the latter because it began with "L" like the LEM⁹⁷. Correspondence between NASA and MIT often shortened these program names and appended numbers. For example, SOLRUM55 was the 55th revision of SOLARIUM for the AS501 and 502 missions. BURST116 was the 116th revision of SUNBURST⁹⁸. Although these programs had many similarities, COLOSSUS and LUMINARY were the only ones capable of navigating a flight to the moon. On August 9, 1968, planners decided to put the first released version of COLOSSUS on *Apollo 8*, which made the first circumlunar flight possible on that mission⁹⁹.

Handling Restarts

One of the most significant differences between batch-type com-

puter systems and real-time systems is the fact that in the latter, an abnormal termination of a program is not acceptable. If a ground-based, non-real-time computer system suffers a software failure ("goes down") due to overloads or mismanagement of resources, it can usually be brought up again without serious damage to the users. However, a failure in a real-time system such as that in an aircraft may result in loss of life. Such systems are backed up in many ways, but considerable emphasis is still placed on making them failure proof from the start. Obviously, the AGC had to be able to recover from software failures. A worst-case example would be a failure of the computer during an engine burn. The system had to have a method of staying "up" at all times.

The solution was to provide for restarts in case of software failures. Such restarts could be caused by a number of conditions: voltage failures, clock failure, a "rupt lock" in which the system got stuck in interrupt mode, or a signal from the NIGHT WATCHMAN program, which checked to see if the NEWJOB register had not been tested by the EXECUTIVE, indicating that the operating system was hung up in some way¹⁰⁰.

An Apollo restart transferred control to a specified address, where a program would begin that consulted phase tables to see which jobs to schedule first. These jobs would then be directed to pick up from the last restart point. The restart point addresses were kept in a restart table. Programmers had to ensure that the restart table entries and phase table entries were kept up to date by the software as it executed¹⁰¹. The restart program also cleared all output channels, such as control jet commands, warning lights, and engine on and off commands, so that nothing dangerous would take place outside of computer control¹⁰².

A software failure causing restarts occurred during the Apollo 11 lunar landing. The software was designed to give counter increment requests priority over instructions¹⁰³. This meant that if some item of hardware needed to increment the count in a memory register, its request to do so would cause the operating system to interrupt current jobs, process the request, and then pick up the suspended routines. It had been projected that if 85,000 increments arrived in a second, the effect would be to completely stop all other work in the system¹⁰⁴. Even a smaller number of requests would slow the software down to the point at which a restart might occur. During the descent of *Apollo 11* to the moon, the rendezvous radar made so many increment requests that about 15% of the computer systems' resources were tied up in responding¹⁰⁵. The time spent handling the interrupts meant that the interrupted jobs did not have enough computer time to complete before they were scheduled to begin again. This situation caused restarts to occur, three of which happened in a 40-second period while program P64 of LUMINARY ran during descent¹⁰⁶. The restarts

caused a series of warnings to be displayed both in the spacecraft and in Mission Control. Steven G. Bales and John R. Garman, monitoring the computer from Mission Control, recognized the origin of the problem. After consultation, Bales, reporting to the Flight Director, called the system "GO" for landing¹⁰⁷. They were right, and the restart software successfully handled the situation. The solution to this particular problem was to correct a switch position on the rendezvous radar which, through an arcane series of circuitry, had caused the analog-to-digital conversion circuitry to race up and down¹⁰⁸. This incident proved the need for and effectiveness of built-in software recovery for unknown or unanticipated error conditions in flight software—a philosophy that has appeared deeply embedded in all NASA manned spaceflight software since then.

Verification and Validation

There could be no true certification of the Apollo software because it was impossible to simulate the actual conditions under which the software was to operate, such as zero-G. The need for reliability motivated an extensive testing program consisting of simulations that could be accomplished before flight. Three simulation systems were available for verification purposes: all-digital, hybrid, and system test labs. All-digital simulations were performed on the Honeywell 1800s and IBM 360s used for software development. Their execution rate was 10% of real time¹⁰⁹. Technicians did hybrid simulations in a lab that contained an actual AGC with a core rope simulator (as core rope would not be manufactured until after verification of the program) and an actual DSKY. Additionally, an attached Beckman analog computer and various interfaces simulated spacecraft responses to computer commands¹¹⁰. Further ad hoc verification took place in the mission trainers located in Houston and at Cape Canaveral, which would run the released programs in their interpretive simulators.

The simulations followed individual unit tests and integrated tests of portions of the software. At first, MIT left these tests to the programmers to be done on an informal basis. It was very difficult at first to get the Instrumentation Laboratory to supply test plans to NASA¹¹¹. The need for formal validation rose with the size of the software. Programs of 2,000 instructions took between 50 and 100 test runs to be fully debugged, and full-size mission loads took from 1,000 to 1,200 runs¹¹².

NASA exerted some pressure on MIT to be more consistent in testing, and it eventually adopted a four-level test structure based largely on the verification of the Gemini Mission Control Center developed by IBM in 1964¹¹³. This is important because formal

release of the program for rope manufacture was dependent on the digital simulations only. Raytheon performed the hybrid and system tests after they had the release tape in hand¹¹⁴. At that time, MIT would have released an "AGC Program Verification Document" to NASA. Aside from help from IBM, NASA also had TRW participate in developing test plans. Having an outside group do some work on verification is a sound software engineering principle, as it is less likely to have a vested interest in seeing the software quickly succeed, and it helps prevent generic errors.

Apollo Software Development Problems

Real-time flight software development on this scale was a new experience for both NASA and the MIT Instrumentation Laboratory. Memory limitations affected the software so that some features and functions had to be abandoned, whereas tricky programming techniques saved others. Quality of the initial code was sometimes poor, so verification took longer and was more expensive. Despite valiant validation efforts, software bugs remained in released programs, forcing adjustments by users. Several times, NASA administrators put pressure on MIT to reduce software complexity because there were real doubts about MIT's ability to deliver reliable software on time. Apparently, few had anticipated that software would become a pacing item for Apollo, nor did they properly anticipate solutions to the problems.

By early 1966, program requirements even exceeded the Block II computer's memory. A May software status memo stated that not only would the programs for the AS504 mission (earth orbit with a LEM) exceed the memory capacity by 11,800 words but that the delivery date for the simpler AS207/208 programs would be too late for the scheduled launch¹¹⁵. Lack of memory and the need for faster throughput resulted in complicating and delaying the program development effort¹¹⁶. One of MIT's top managers explained

If you are limited in program capacity ... you have to fix. You have to get ingenious, and as soon as you start to get ingenious you get intermeshing programs, programs that depend upon others and utilize other parts of those, and many things are going on simultaneously. So it gets difficult to assign out little task groups to program part of the computer; you have to do it with a very technical team that understands all the interactions on all these things¹¹⁷.

The development of obscure code caused problems both in understanding the programs and validating them, and this, in turn, caused delays. MIT's considerable geographic distance from Houston caused

additional problems. Thus, NASA's contract managers had to commute often. Howard W. "Bill" Tindall, newly assigned from the Gemini Project as NASA's "watchdog" for MIT software, spent 2 or 3 days a week in Boston starting in early 1966¹¹⁸.

Tindall was well known at the Manned Spacecraft Center due to his legendary "Tindallgrams"—blunt memos regarding software development for Apollo. One of the first to recognize the importance of software to mission schedules, he wrote on May 31, 1966 that "the computer programs for the Apollo spacecraft will soon become the most pacing item for the Apollo flights"¹¹⁹. MIT was about to make the standard emergency move when software was in danger of being late: to throw more bodies into the project, a tactic that often backfires. As many as 50 people were to be added to the programming staff, and the amount of interaction between programmers and, thus, the potential for miscommunication increased along with the time necessary to train newcomers. MIT tried to protect the tenure of its permanent staff by using contractors who could be easily released. The hardware effort peaked at 600 workers in June of 1965 and fell off rapidly after that, while software workers steadily increased to 400 by August of 1968. With the completion of the basic version of COLOSSUS and LUMINARY, the number of programmers quickly decreased¹²⁰. This method, although in the long-term interests of the laboratory, caused considerable waste of resources in communication and training.

Tindall's memo also detailed many of NASA's efforts to improve MIT's handling of the software development. Tindall had taken Lynwood Dunseith, then head of the computer systems in Mission Control, and Richard Hanrahan of IBM to MIT to brief the Instrumentation Laboratory on the Program Development Plan used for management of software development in the Real-Time Computing Center associated with Mission Control. The objective was to give MIT some suggestions on measuring progress and detecting problem areas early. One NASA manager pointed out that the Instrumentation Laboratory was protective of the image of MIT, and one way to control MIT was to threaten its self-esteem¹²¹. The need to call on IBM for advice was apparently a form of negative motivation. A couple of weeks later, Tindall reported that Edward Capps of MIT was leading the development of a Program Development Plan based on one done by IBM¹²². However, by July he was complaining that MIT was implementing it too slowly¹²³. In fact, some aspects of configuration control such as discrepancy reporting (when the software does not match the specification) took over a year for MIT to implement¹²⁴.

NASA had to be very careful in approving cuts in the program requirements to achieve some memory savings. Some features were obviously "frosting," and could easily be eliminated; for example, the effects of the oblate nature of the earth, formerly figured into lunar orbit

rendezvous but actually minimal enough to be ignored¹²⁵. Also cut were some attitude maneuver computations. They therefore left Reaction Control System (RCS) burns to the "feel" of the pilot, which meant slightly greater fuel expenditure¹²⁶. Overall, the cuts resulted in software that saved money and accelerated development but could not minimize fuel expenditures nor provide the close guidance tolerance that was within the capability of the computer, given more memory¹²⁷.

Flight AS-204: A Breaking Point

Despite efforts by both MIT and NASA, by the summer of 1966, flight schedules and problems in development put both organizations in a dangerous position regarding the software. A study of the problems encountered with the software for flight AS-204, which was to be the first manned Apollo mission, best demonstrates the urgency. On June 13, Tindall reported that the AS-204 program undergoing integrated tests had bugs in *every* module. Some *had not been unit tested* prior to being integrated¹²⁸. This was a serious breach of software engineering practice. If individual modules are unit tested and proven bug-free, then bugs found in integrated tests are most likely located in the interfaces or calling modules. If unit testing has not been done then bugs could be *anywhere* in the program load, and it is very difficult to identify the location properly. This vastly increases the time and, thus, the cost of debugging. It causes a much greater slip in schedule than time spent on unit tests. Even worse, Tindall said that the test results would not be formally documented to NASA but that they would be on file if needed.

The AS-204 software schedule problems affected other things. All the crew-requested changes in the programs were rejected because including them would cause even further delays¹²⁹. The AS-501 program and others began to slip because the AS-204 fixes were saturating the Honeywell 1800s used in program development¹³⁰. MIT also added another nine programmers to the team, all from AC Electronic, thus increasing communication and training problems.

The eventual result was that the flight software for the mission was of dubious quality. Tindall predicted such would be the case as early as June 1966, saying that "we have every expectation that the flight program we finally must accept will be of less than desirable quality"¹³¹. In other words, it would contain bugs, bugs that would not actually threaten the mission directly but that would have to be worked around either by the crew or by ground control. They found one such bug less than a month before the scheduled February 21, 1967, launch date. Ground computers and the Apollo guidance com-

puter calculated the time for the de-orbit burn that preceded re-entry. Simulations performed during January 1967 and reported on the 23rd indicated that there was a discrepancy between the two calculations of as much as 138 seconds! Since the core rope was already installed in the spacecraft, the only possible fix (besides a delay in the launch time) would be to have the crew ignore the Apollo computer solution. The ground would transmit the Real-Time Computing Center solution, after which an astronaut would have to key the numbers into the Apollo computer¹³². This situation, and other discrepancies, led one NASA engineer to later remark that "we were about to fly with flight software that was really suspect"¹³³.

AS-204 did not fly, so that software load was never fully tried. On January 27, 1967, during a simulation with the crew in the spacecraft on the pad, a fire destroyed the CM, killed the crew, and delayed the Apollo program for months. The changes in managing software development put into effect by NASA and MIT during 1966 had not had enough time to take effect before the fire. In the ensuing period, with manned launches on indefinite delay, MIT was under the direction of the NASA team led by Tindall and was able to catch up on its work and take steps to make the software more reliable. NASA and MIT split the effort among three programs: CM earth orbit, CM lunar orbit, and lunar module lunar landing (LM earth orbit was dropped)¹³⁴. By October 17, 1967, the SUNDISK earth orbit program was complete, verified, and ready for core rope manufacture, a year before the first manned flight¹³⁵. The time gained by the delay caused by the fire allowed for significant improvements in the Apollo software. Tindall observed at the time, "It is becoming evident that we are entering a new epoch regarding development of spacecraft computer programs." No longer would programs be declared complete in order to meet schedules, requiring the users to work around errors. Instead, quality would be the primary consideration¹³⁶.

The Guidance Software Task Force

Despite postfire improvements, Apollo software had more hurdles to clear. NASA was aware of continuing concern about Apollo's computer programs. Associate Administrator for Manned Spaceflight George E. Mueller formed a Guidance Software Task Force on December 18, 1967 to study ways of improving development and verification*. The group met 14 times at various locations before its final report in September 1968¹³⁷.

*Members of the Task Force included Richard H. Battin, MIT; Leon R. Bush, Aerospace Corp.; Donald R. Hagner, Bellcomm; Dick Hanrahan, IBM; James S. Martin, NASA-Langley; John P. Mayer, NASA-MSD; Clarence Pitman, TRW; and Ludie G. Richard, NASA-Marshall. Mueller was the chairman.

Even while the Task Force was investigating, Mueller took other steps to challenge MIT. A Software Review Board re-examined the software requirements for the lunar mission in early February 1968. The board judged the programs to be too sophisticated and complex, and Mueller requested that they aim for a 50% reduction in the programs, with increased propellant consumption allowed as a tradeoff¹³⁸. An aide reported that Mueller was convinced that MIT "might not provide a reliable, checked-out program on schedule" for the lunar landing mission¹³⁹.

The recommended 50% scrub did not occur, and the final report of the Task Force was very sympathetic to the problems involved in developing flight software. It recommended standardization of symbols, constants, and variable names used at both Houston and Huntsville to make communication and coding easier¹⁴⁰. The Task Force acknowledged that requirements would always be dynamic and that development schedules would always be accelerated, but rather than using this for an excuse for poor quality, the group recommended that software not be slighted in future manned programs. Adequate resources and personnel were to be assigned early to this "vital and underestimated area"¹⁴¹. This realization would have great effect on managing later software development for the Space Transportation System.

Mueller remained concerned about software even after the Task Force dissolved. On March 6, 1969, he wrote a letter to Robert Gilruth, NASA deputy administrator, complaining that software changes were being made too haphazardly and should receive more attention, equal to that given to hardware change requests. Gilruth replied five days later, disagreeing, noting that the Configuration Control Board and other committees formed an interlocking system adequate for change control¹⁴².

Lessons of the Apollo Software Development Process

Overcoming the problems of the Apollo software, NASA did successfully land a man on the moon using programs certifiably adequate for the purpose. No one doubted the quality of the software eventually produced by MIT nor the dedication and ability of the programmers and managers at the Instrumentation Lab. It was the *process* used in software development that caused great concern, and NASA helped to improve it¹⁴³. The lessons of this endeavor were the same learned by almost every other large system development team of the 1960s: (a) documentation is crucial, (b) verification must proceed through several levels, (c) requirements must be clearly defined and carefully managed, (d) good development plans should be created and ex-

ecuted, and (*e*) more programmers do not mean faster development. Fortunately, no software disasters occurred as a result of the rush to the moon, which is more a tribute to the ability of the individuals doing the work than to the quality of the tools they used.

USING THE AGC

The Apollo computer system made great demands on the crew. It took about 10,500 keystrokes to complete a lunar mission; not much in the life of an airline reservations clerk but still indicative of how computer centered the crew had to be¹⁴⁴. During the period in which the software was criticized for its complexity, designers attempted to reduce the number of keystrokes required to execute various programs. When possible, they also eliminated built-in halts as data were displayed for astronaut approval. However, the "fourth crew member" never abandoned center stage¹⁴⁵.

Apollo's crew employed its computer through the use of the DSKYs. In the CM one was on the main control panel opposite the commander's couch. The other was at the navigator's station in the lower equipment bay, where the computer itself was located. Block I had a different DSKY at the navigator's station than on the main panel¹⁴⁶, but they were identical in the Block II series. DSKY and computer activity could be monitored from the ground as the computer transmitted data words to drive real-time displays in Mission Control¹⁴⁷.

The crew could communicate with the computer through keys, displays, and warning lights on the DSKY. Additionally, the uplink telemetry could provide input to the machine, and so could the preflight checkout equipment¹⁴⁸. The computer, in turn, could communicate with the crew by flashing the PROGram, VERB, and NOUN displays¹⁴⁹. The DSKY displays included 10 warning lights, a computer activity light, a PROGram display, VERB and NOUN displays, three five-digit numeric displays with signs, and 19 keys including VERB, NOUN, CLEAR, KEY RELEASE, PROCEED, RESET, ENTER, PLUS, MINUS, and the digits 0-9. See Boxes 2-2 and 2-3 for functions and use.

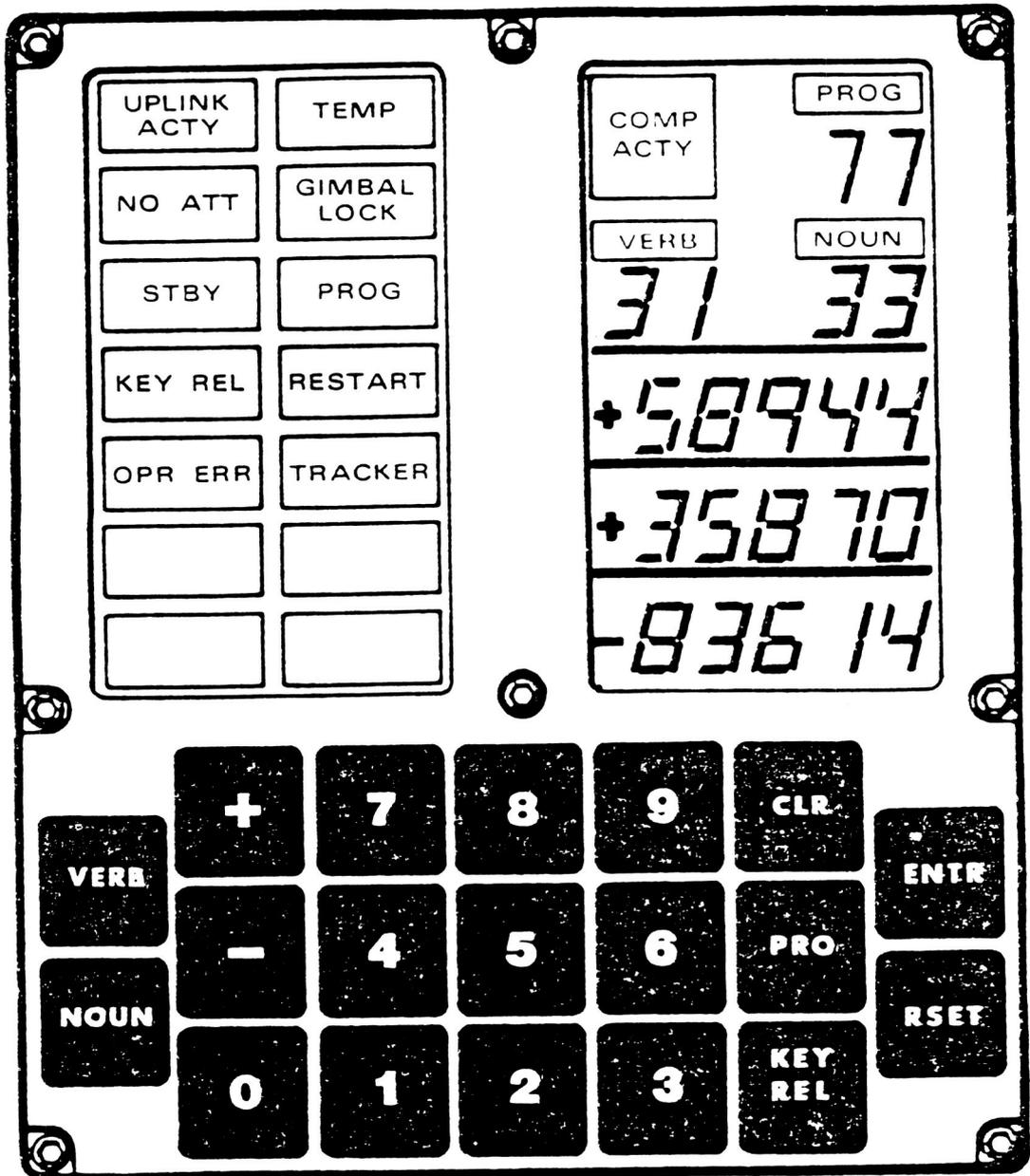


Figure 2-2. The Display and Keyboard (DSKY) of an Apollo spacecraft. (Prepared by The Wichita State University Media Services)

Box 2-2:—Apollo Display and Keyboard Lights

Ten DSKY warning lights had the following functions:

- **COMP ACTY:** This lit up when the computer was running a program.
- **UPLINK ACTY:** Lit when data was being received from the ground.
- **TEMP:** Lit when the temperature of the stable platform was out of tolerance.
- **NO ATT:** Lit when the inertial subsystem could not provide attitude reference.
- **GIMBAL LOCK:** Lit when the middle gimbal angle was greater than 70 degrees.
- **STBY:** Lit when the computer system was on standby.
- **PROG:** Lit when the computer was waiting for additional information to be entered by the crew to complete the program.
- **KEY REL:** Lit when the computer needed control of the DSKY to complete a program. Sometimes display information could be "buried" under other routines or by a priority interrupt. The crew could press the KEY REL key to release the keyboard to the requesting program¹⁵⁰. When the KEY REL light went on, that signaled the crew to press the key.
- **RESTART:** Lit when the computer was in the restart program. This was the light that kept coming on during the *Apollo 11* landing.
- **OPR ERR:** Lit when the computer detected an error on the keyboard.
- **TRACKER:** Lit when one of the optical coupling units failed.

The LEM DSKY had three additional lights: NO DAP, ALT, and VEL, which were related to failures of the digital autopilot and to warn of altitude and velocity readings outside of the predetermined limits.

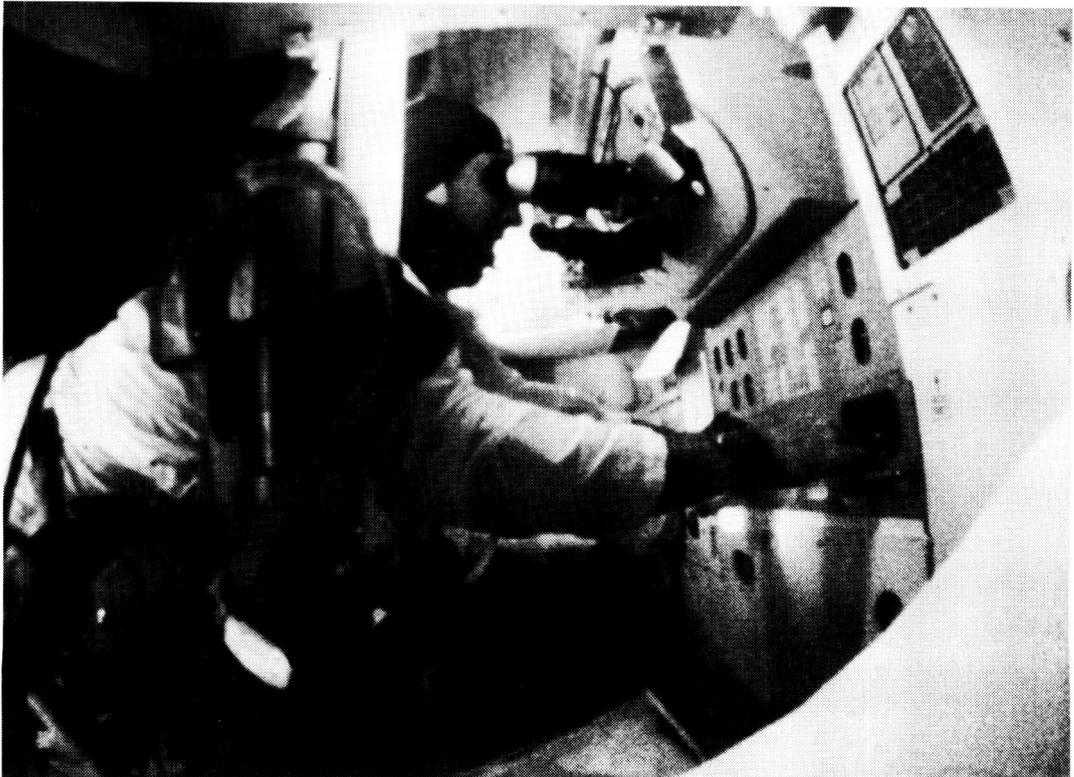


Figure 2-3. Another DSKY was located at the navigator's station in the command module. Astronaut James A. Lovell takes a star sighting during the Apollo 8 mission. (NASA photo S-69-35099)

Box 2-3:—Apollo Display and Keyboard Displays

Seven displays were available on the DSKY:

- **PROG:** This was a two-digit display indicating what numbered program the computer was currently executing.
- **VERB:** A two-digit display of the verb number being entered (the verb-noun system is discussed below).
- **NOUN:** A two-digit display of the noun number being entered.
- Three five-digit numeric displays, which showed numbers in either decimal or octal (base eight). When a sign was shown with the number, the number was decimal; otherwise, it was octal¹⁵¹.



Figure 2-4. The interior of an Apollo Command Module, showing the location of the DSKY on the main control panel at the left. Apollo 15 crewmen shown include Alfred M. Worden (center) and David R. Scott (left). James B. Irwin is mostly obscured to the right. (NASA photo S-71-29952)

Using the Keys and the Verb-Noun System

Astronauts used keys to enter information and select programs and actions. Key inputs caused automatic interrupts in the software¹⁵². The astronauts would activate a program and then interact with it by requesting and entering information; a typical software load consisted of about 40 programs and 30 simultaneous routines¹⁵³. Changing programs and making other requests involved using the verb-noun system. Those familiar with current computer keyboards

will notice the lack of alphabet keys on the DSKY. Whereas most computer commands are entered by typing in the text of the command, the Apollo computer command list specified verb and noun pairs. There were 100 two-digit numbers available for each, and most were used on any given flight. Examples of verb-noun pairs are "display velocity" and "load angle." Verb 37, for example, was "Change Prog," which enabled the crew to set up a new program for execution.

If, for example, the crew wanted to execute the rendezvous targeting program, an astronaut would first press the VERB key followed by the digits 3 and 7, and then the ENTER key. That sequence informed the computer of a request for a program change. The astronaut would then press 3, 1, and ENTER to tell the computer to execute program P31. Within the program the crew could request maneuver angles (verb 50, noun 18), monitor the changes while a maneuver was in progress (verb 06, noun 18), or request the velocity change required for the next maneuver (verb 06, noun 84), among other functions. The CSM G&C Checklist, a set of "cue cards" on three rings changed for each mission by the Crew Procedures Division in Houston, described all these sequences in detail. The document contained reference data, such as a star list, verb list, noun list, alarm codes, error handling and recovery, and the checklists for each program carried in the computer.

Despite the 100 verb-noun pairs, 70-odd programs and routines, and a very limited user interface that alternated decimal and octal and blinked for attention, the consensus is that the Apollo computer was easy to use. As with other aspects of flying space missions, hours in simulators made operating the computer second nature. NASA engineer John R. Garman commented that "it's like playing the piano--you don't have to see your fingers to know where they are"¹⁵⁴. Familiarity with the computer, remarked astronaut Eugene Cernan, meant that pressing a wrong key simply and immediately "felt" wrong¹⁵⁵. Others also confirmed that using the machine eventually became relatively natural¹⁵⁶. Apollo astronauts were also willing to adapt to design foibles that would frustrate others. There were concerns that a crewman initiating a maneuver from the navigator's station would not be able to return to his couch before the burn started. In response, Virgil Grissom was accommodating: "Well, we'll just lie down on the floor"¹⁵⁷. Astronauts also tolerated non-life-threatening software errors not cleared up before flight as merely something else to endure¹⁵⁸. They did, however, complain about the annoying number of keystrokes required during a rendezvous, so designers modified the software to make a "minkey" (minimum keystroke) option available, in which the computer could perform some functions without constant crew approval¹⁵⁹. This change contributed to an even more compact, straightforward system.

THE ABORT GUIDANCE SYSTEM

The computer in the Abort Guidance System (AGS) is probably the most obscure computing machine in the manned spaceflight program to date. The 330-page "Apollo Spacecraft News Reference" prepared for the first lunar landing mission does not contain a single reference to it, compared with several pages of description of the Primary Guidance, Navigation, and Control System (PGNCS) computer and its interfaces. The invisibility of the AGS is a tribute to PGNCS, since the AGS was never needed to abort a landing. It was, however, an interesting and pioneering system in its own right.

The AGS owed its existence to NASA's abort policy; an abort is ordered if one additional system failure would potentially cause loss of crew¹⁶⁰. Hence, the failure of *either* the PGNCS *or* the AGS would have resulted in an abort. The AGS operated in an open loop, parallel to the PGNCS in the LEM, and gave the crew an independent source of position, velocity, attitude, and steering information¹⁶¹. It could verify navigation data during the periods when the LEM was behind the moon and blacked out from ground control. The Apollo program first exercised this capability during Apollo 9 and Apollo 10 leading up to the first landing mission¹⁶².

The AGS was a pioneer in that it was the first "strapped-down" guidance system. The system used sensors fixed to the LEM to determine motion rather than a stable platform as in conventional inertial guidance systems¹⁶³. The entire system occupied only 3 cubic feet and consisted of three major components: (a) an Abort Electronic Assembly (AEA), which was the computer, (b) an Abort Sensor Assembly (ASA), which was the inertial sensor, and (c) a Data Entry and Display Assembly (DEDA), which was the DSKY for the AGS.

AEA and DEDA: The Computer Hardware

As with the PGNCS computer, the AGS computer went through an evolutionary period in which designers clarified and settled the requirements. The first design for the system did not include a true computer at all but rather a "programmer," a fairly straightforward sequencer of about 2,000 words fixed memory, which did not have navigation functions. Its job was simply to abort the LEM to a "clear" lunar orbit (one that would be higher than any mountain ranges) at which point the crew would wait for rescue from the CM, with its more sophisticated navigation and maneuvering system¹⁶⁴. The requirements changed in the fall of 1964. To provide more autonomy and safety, the AGS had to provide rendezvous capability without outside

sources of information¹⁶⁵. TRW, the contractor, then decided to include a computer of about 4,000 words memory. The company considered an existing Univector accumulation machine but, instead, chose a custom designed computer¹⁶⁶.

The computer built for the AGS was the MARCO 4418 (for *Man Rated Computer*). It was an 18-bit machine, with 17 magnitude bits and a sign bit. It used 5-bit op codes and 13-bit addresses. Numbers were stored in the two's complement form, fixed point, same as in the primary computer. Twenty-seven instructions were available, and the execution time varied from 10 to 70 microseconds, depending on the instruction being performed¹⁶⁷. The computer was 5 by 8 by 23.75 inches, weighed 32.7 pounds, and required 90 watts¹⁶⁸. The memory was bit serial access, which made it slower than the PGNCS computer, and it was divided into 2K of fixed cores and 2K of erasable cores¹⁶⁹. The actual cores used in the fixed and erasable portions were of the same construction, unlike those in the PGNCS computer. Therefore, the ratio of fixed memory to erasable in the MARCO 4418 was variable¹⁷⁰. TRW was obviously thinking in terms of adaptability to later applications.

The DEDA was much smaller and less versatile than the DSKY. It was 5.5 by 6 by 5.19 inches and was located on the right side of the LEM control panel in front of the pilot, about waist height¹⁷¹. Sixteen pushbutton keys were available: CLEAR, READOUT, ENTER, HOLD, PLUS, MINUS, and the digits 0–9. It had a single, nine-window readout display. Three windows showed the address (in octal), one window the sign, and five, digits¹⁷². This was similar to the readout in the Gemini spacecraft for its computer.

Software for the AGS

Since hardware in the AGS evolved as in PGNCS, software also had to be "scrubbed" (reduced in size) in the AGS. Mirroring the memory problems of PGNCS, by 1966, 2 full years before the first active mission using the LEM, only 20 words remained of the 4,000 in the AGS memory¹⁷³. Careful memory management became the focus of TRW and NASA. Tindall recalled that the changes all had to be made in the erasable portion, as the fixed portion was programmed early and remained set to save money. However, changing the erasable memory turned out to be very expensive and a real headache, the developers fighting to free up storage literally one location at a time¹⁷⁴. Also, some software decisions had to be altered in light of possible disastrous effects. The restart program for the PGNCS has been described. In it, a restart clears all engine burns. The first versions of the AGS software also caused engine shutdown and an at-

titude hold to go into effect when a restart occurred. This would be potentially dangerous if a restart began with the LEM close to the lunar surface. The solution was to give the crew responsibility to manually fire the engines during a restart if necessary¹⁷⁵.

Software development for the AGS followed a tightly controlled schedule:

1. 12.5 months before launch: NASA delivers the preliminary reference trajectory and mission requirements to TRW.
2. 11 months: Program specification and AGS performance analysis is complete.
3. 10.5 months: NASA conducts the Critical Design Review (CDR).
4. 8 months: The final mission reference trajectory is delivered.
5. 7 months: The equation test results, verification test plan, and preliminary program goes to NASA for approval.
6. 6.5 months: The First Article Configuration Inspection (FACI) conducted.
7. 5 months: The verified program and documentation is delivered to NASA.
8. 4.5 months: NASA conducts the Customer Acceptance Readiness Review (CARR).
9. 3 months: The operational flight trajectory is delivered by NASA to the contractor.
10. 2 months: The final Flight Readiness Review (FRR) is held.
11. 1.5 months: The tape containing the final program is delivered¹⁷⁶.

One method of software verification was quite unique. To simulate motion and thus provide more realistic inputs to the computer, planners used a walk-in van containing the hardware and software. Technicians drove the van around Houston with the programs running inside it¹⁷⁷.

Use of the AGS

The AGS was never used for an abort, but it did contribute to the final rendezvous and docking with the CM on the Apollo 11 mission, probably to avoid the problems encountered with the rendezvous radar during landing¹⁷⁸. It did monitor PGNC performance during all missions in which it flew. The only criticism of its performance was from astronaut John Young, who remarked that "one mistake in a rendezvous, and the whole thing quit"¹⁷⁹. Apparently, restarts occurred as part of the recovery from some operator errors. The AGS was actually like a parachute—absolutely necessary, but presumably never needed.

LESSONS

What did NASA learn from its experiences with the Apollo computer system? At the management level, NASA learned to assign experienced personnel to a project early, rather than using the start of a project for training inexperienced personnel; many NASA managers of software and hardware were learning on the job while in key positions. Also, more participation by management in the early phases of software design is necessary so that costs can be more effectively estimated and controlled.

From the standpoint of development, NASA learned that a more thorough, early effort at total systems engineering must be made so that specifications can be adequately set. NASA contractors in the Apollo program faced changing specifications long after final requirements should have been fixed. This was expensive and caused such problems as Raytheon's retooling, memory shortages, and design insufficiencies.

The realization that software is more difficult to develop than hardware is one of the most important lessons of the Apollo program. So the choice of memory should be software driven, and designers should develop software needed for manned spaceflight near the Manned Spacecraft Center. The arrangement with MIT reduced overall quality and efficiency due to lack of communication. Also, more modularization of the software was needed¹⁸⁰.

The AGC system served well on the earth-orbital missions, the six lunar landing missions, the three Skylab missions, and the *Apollo-Soyuz* test project. Even though plans existed to expand the computer to 16K of erasable memory and 65K of fixed memory, including making direct memory addressing possible for the erasable portion, no expansion occurred¹⁸¹. The Apollo computer did fly on missions other than Apollo. An F-8 research aircraft used a lunar module computer as part of a "fly-by-wire" system, in which control

surfaces moved by servos at the direction of electronic signals instead of traditional cables and hydraulics. In that way, the Apollo system made a direct research contribution to the Shuttle, which is completely a fly-by-wire craft. The most important legacy of the AGC, however, was in the way NASA applied the lessons it was beginning to learn in developing ground software to the management of flight software.

3

The Skylab Computer System

Skylab, America's first orbital workshop, carried a highly successful computer system. For much of the operating life of the space station, the computer was not just the fourth crew member but the *only* crew member. It made a large contribution to saving the mission during the 2 weeks after the troubled launch and later helped control Skylab during the last year before re-entry. The entire system functioned without error or failure for over 600 days of operation, even after a 4-year and 30-day interruption. It is significant as the first spaceborne computer system to have redundancy management software. The software development for the system followed strict engineering principles, producing a fully verified and reliable real-time program.

The record of the computer system stands in contrast to that of the workshop itself. NASA launched Skylab on May 14, 1973 on a Saturn V booster. The first two stages put the modified S-IVB third stage into orbit. The S-IVB contained the workshop, which included a solar telescope mount and living and working quarters. The plan was to launch the first crew the next day aboard a Saturn IB carrying an Apollo command and service module. However, shortly after achieving orbit, telemetry from the unmanned Skylab indicated that one of the two wings of solar panels was missing and the other had not deployed. The panels on the Apollo Telescope Mount (ATM) had opened properly but they were too small to supply power for the whole workshop. In addition, the gyros were drifting and the thermal shield was damaged. These failures caused concern that the interior of the space station would overheat and destroy the equipment. The damage was so serious that for the first 3 or 4 hours the ground controllers felt that NASA would be fortunate if the systems were to function for 1 day¹. However, by using the computer system that controlled the workshop's attitude, the ground controllers were able to keep the Skylab at angles to the sun such that the equipment would be exposed to tolerable temperatures in the laboratory in concert with generating adequate power from the remaining solar panels. At times these were conflicting requirements. This had to be done for 2 weeks while engineers prepared repair materials for the crew to fix the workshop. Controller Steven Bales remembered that time as "the hardest 2 weeks I have ever spent," since a 24-hour watch had to be maintained on the attitude and temperature².

The computer system again served as "captain" during the entire Skylab reactivation. The workshop systems were shut down on February 9, 1974, after the last crew left. NASA expected that the Skylab would stay in orbit until the mid-1980s. By that time the Space Shuttle would be operational and, it was thought, could be used to bring up rockets to boost the laboratory into a higher orbit. However, unexpected solar activity in the mid-1970s resulted in an increase in the density of the atmosphere, so the Skylab's orbit decayed at a much faster rate than projected³.

ORIGINAL PAGE IS
OF POOR QUALITY

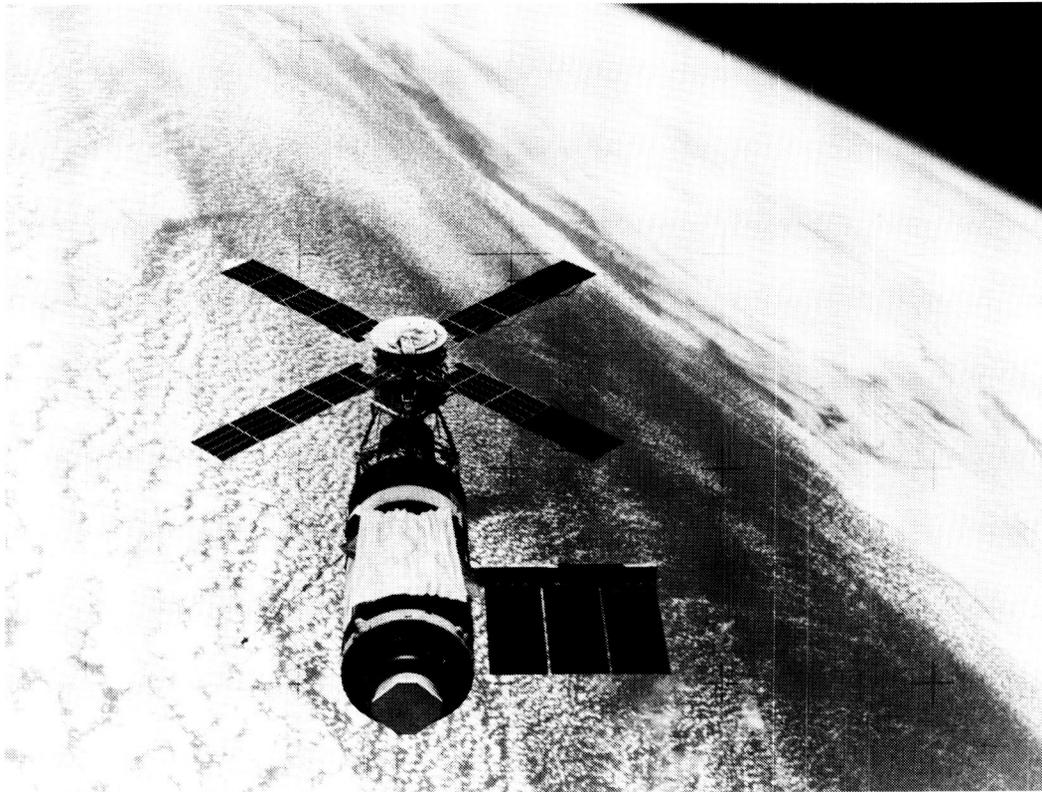


Figure 3-1. Skylab in orbit. Note the foil sun shield above the center section and the missing large solar panel. The Apollo Telescope Mount is the section with the "windmill" solar panels. (NASA photo 74-H-98)

By 1978, the predicted re-entry time was to be late that year or in early 1979. NASA decided to attempt to change the attitude of the workshop so that minimal drag would ensue. In this way, the orbit might be maintained until the Shuttle could rescue the space station. Engineers reactivated and reprogrammed the computer to maintain the proper attitude and, later, to control the re-entry when NASA abandoned the attempt to maintain orbit. They accomplished this over 4 years after the computer was shut down.

The need for the computer system that served Skylab so well was not apparent until the original "wet workshop" concept (the laboratory to be assembled in space inside of the empty propellant tanks of the last stage of the launch vehicle) had progressed through more sophisticated designs to the eventual "dry workshop"⁴. In December 1968, NASA decided to acquire a dual computer system to help control attitude while in orbit⁵. Attitude control was crucial to the success of the solar experiments. In fact, the name of the computer reflects this: Apollo Telescope Mount Digital Computer (ATMDC). Two of these computers were a part of the Skylab Attitude and Pointing Control System (APCS), which consisted of a number of other components,

such as an interface unit, magnetic tape memory, control moment gyros, the thruster attitude control system, sun sensors, a star tracker, and nine rate gyros⁶.

Marshall Space Flight Center devised this complex system—a pioneering effort because it represents the first fully digital control system on a manned spacecraft⁷. Its mission-critical status led to the use of extensive redundancy in its design, in both hardware and software. The computer system not only managed its own redundancy, but all redundant hardware on the spacecraft⁸. The uniqueness and complexity of the control laws associated with the control moment gyro attitude system led one NASA engineer to refer to it as "a crazy animal"⁹. It was up to the Skylab computer system to tame it.

HARDWARE

The choice of a central processor for the Skylab computer system marked a break from NASA's previous practice. The Gemini and Apollo computer systems were custom-built processors. Apollo did have an immediate predecessor, but the number of changes necessary before flight negated most of its resemblance to the Polaris system. To the contrary, Skylab and, later, the Shuttle, used "off-the-shelf" IBM 4Pi series processors, though they both needed the addition of a customized I/O system, a simpler and necessarily idiosyncratic component. By using existing computers, NASA avoided the serious problems associated with man-rating a new system encountered during the Apollo program.

The 4Pi descended directly from the System 360 architecture IBM developed in the early 1960s. Some 4Pis were at work in aircraft by the latter part of that decade. The top-of-the-line 4Pi is the AP-101, eventually used in the F-15, B-52, and Shuttle. The version on board Skylab was the TC-1, which used a 16-bit word, in contrast to the AP-101's 32 bits. A TC-1 processor, an interface controller, an I/O assembly, and a power supply made up an ATMDC¹⁰. Each flight computer had a memory of 16,384 words¹¹. This memory was a destructive readout core memory, which means that the bits were erased as they were read and that the memory location had to be refreshed with the contents of a buffer register, which saved a copy of the bits before they were passed on to the processor. The memory was in two modules of 8K words each¹². Addressing ranged from 0 to 8K, with a hardware switch determining which module was being accessed¹³. The redundant computer system was composed of two processors attached to a single Workshop Computer Interface Unit. The unit consisted of two I/O sections (one for each computer), a common section, and a power supply¹⁴. Only the I/O section connected to the active

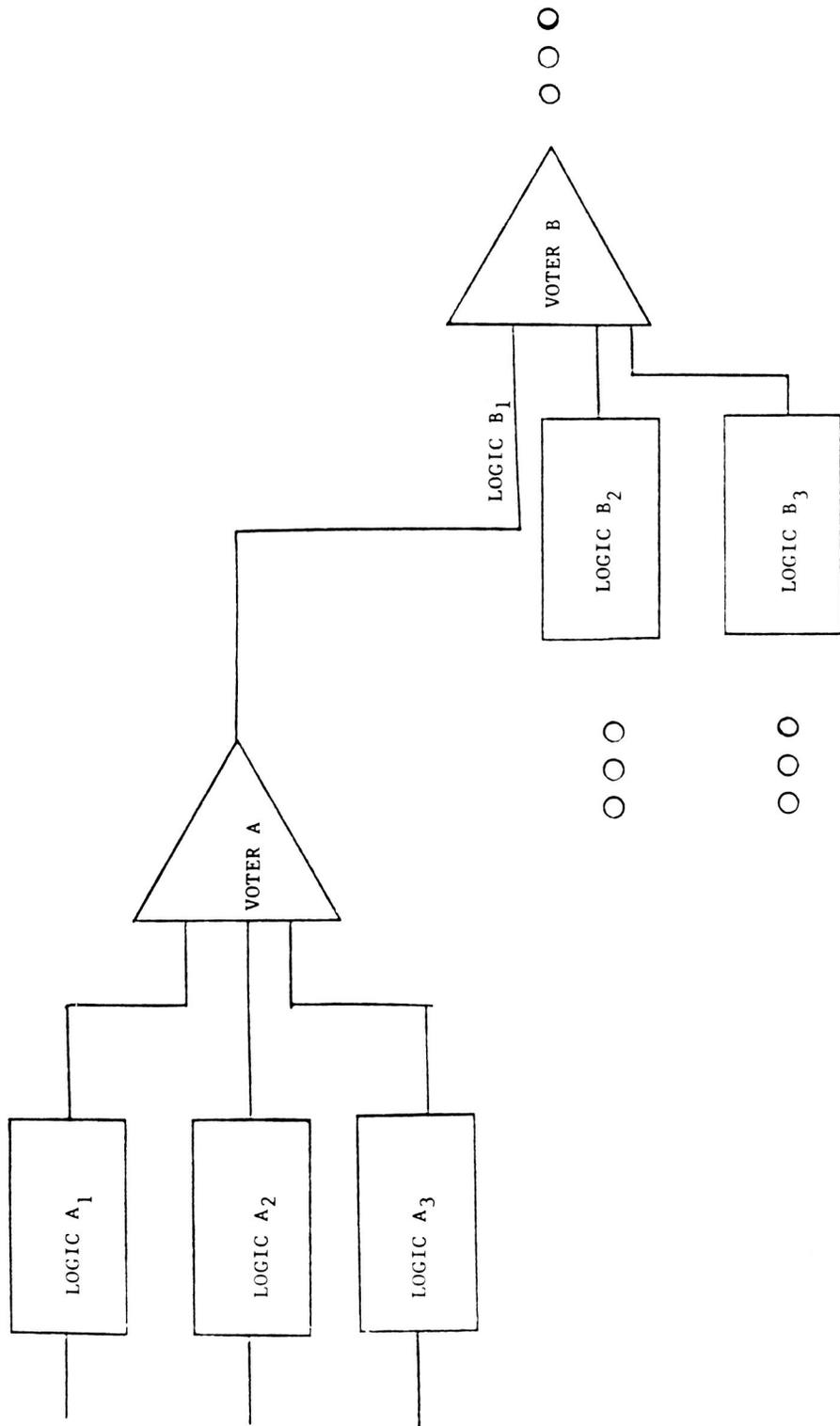
computer was powered. The inactive computer and its I/O section of the interface unit were not powered. The common section contained a 64-bit transfer register and timer associated with redundancy management¹⁵. The transfer register and timer were the only parts of Skylab that consisted of triple modular redundant (TMR) circuits¹⁶. Basically, TMR circuits sent signals in triplicate on separate channels and then voted. The single output from a TMR voter represented either two or three identical inputs.

The final component of the computer subsystem was the Memory Load Unit. The original design did not contain one, but, like the Gemini Auxiliary Tape Memory, engineers later added it. Whereas the Gemini tape unit was useful in handling memory overloads, designers included the Skylab tape unit to further increase the reliability of the system. It carried a 16K software load and an 8K load that could be written into either module of either memory of the ATDCs. If up to three modules failed, the mission could continue with reduced capabilities with an 8K program loaded into the remaining module. This raised the total reliability of the system from a factor of 0.87 to 0.97¹⁷. The tape load would take a maximum of 11 seconds¹⁸.

NASA decided to add the Memory Load Unit in the summer of 1971, when both IBM and Marshall realized that a Borg-Warner tape unit, like the two already used as telemetry recorders, could be upgraded for program storage. IBM imposed some manufacturing changes on the recorders (primarily piece part screening) to make the process more nearly match the care taken in constructing the computers¹⁹.

NASA awarded the contract for the computer system to IBM on March 5, 1969²⁰. By October, designers froze the choice of processors and their configuration, a decision heavily influenced by the concern for redundancy and reliability²¹. The first computer was delivered on December 23, 1969. IBM eventually built 10, the final 2 being the flight versions, which went to NASA on February 11, 1972, over a year before launch. Two of the ATMDCs and an interface unit were turned over to IBM for use in testing both hardware and software, ensuring that the final verification would be on actual equipment rather than simulators²².

IBM took great pride in delivering on time without sacrificing reliability. In applying Saturn development techniques to the Skylab equipment, for example, IBM required all piece parts to exceed expected stress levels²³, and prepared the ATMDC for thermal conditions, the most dangerous stress to electronic components²⁴. A number of design problems, including thermal and vibration difficulties, analog conversion inaccuracies, and interconnection failures, had to be overcome²⁵. To make up time lost handling these problems, IBM sometimes went to a 7-day, three-shift debugging cycle²⁶.



Concept of Triple Modular Redundancy

Figure 3-2. The concept of Triple Modular Redundancy.

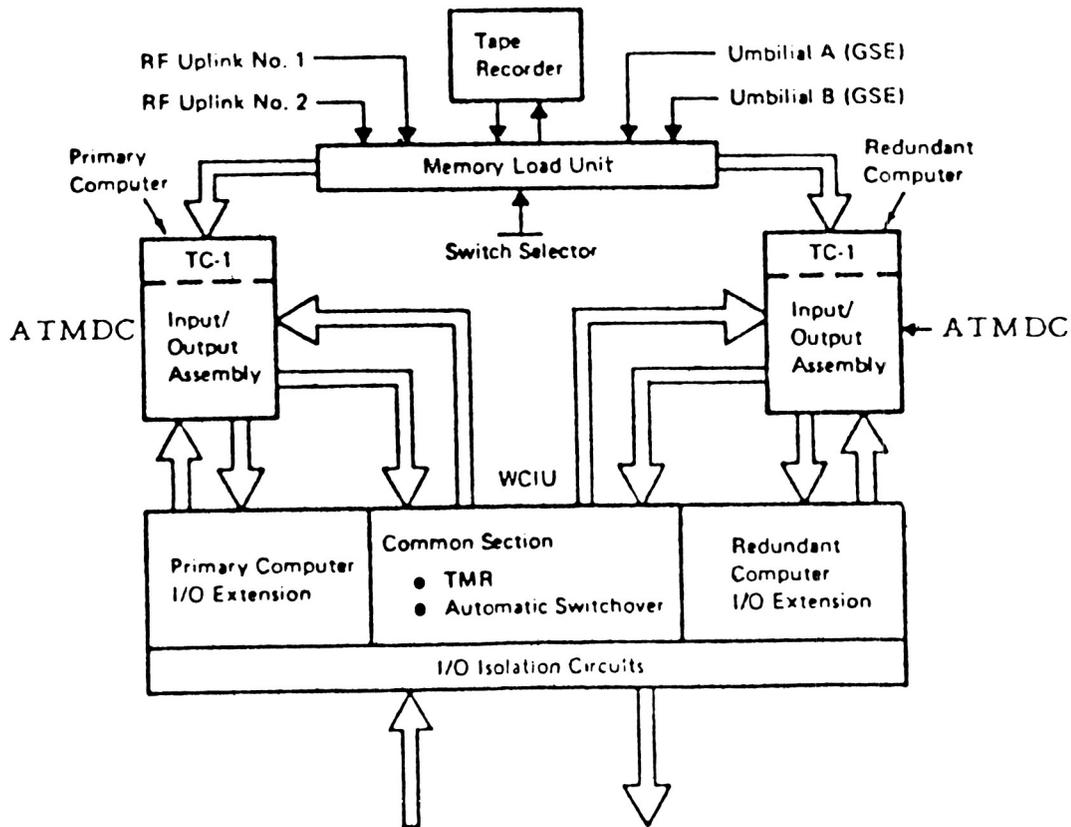


Figure 3-3. A block diagram of the Skylab Computer System with the dual ATMDCs, tape memory, and common section shown. (From IBM, *Skylab Operation Assessment, ATMDC*, 1974)

Probably due to the care taken in manufacture, the computer system had no failures. A planned ground-initiated switch-over from the primary to the secondary computer occurred after 630 hours of orbital operations. The second computer then ran the remainder of the 271-day mission²⁷. On the final day, the system did another switch-over and used the tape unit for the first time, primarily to prove that it would work. A transmission of software from the ground to the computer was also practiced. IBM's reports of the performance of the hardware are quite self-congratulatory but, based on the actual record, justified.

SOFTWARE

IBM wanted to do a careful job on the software for Skylab. In the late 1960s and early 1970s, the company internally pushed the development and implementation of software engineering techniques. IBM learned many lessons from the creation of the OS/360 operating system, and various government-related projects. Two IBM software management experts, Harlan Mills and Frederick Brooks, circulated these lessons both within IBM and to the computing public²⁸. The small size (16K) of the Skylab software and correspondingly small group of programmers assigned to write it (never more than 75 people, not all of whom were programmers, and only 5 or 6 for the reactivation software), meant that the difficulties in communication and configuration control associated with large projects were not as much of a factor. Also the IBM programmers were specialists. MIT assigned engineers to the programming of the Apollo computer, assuming that it was easier to teach an engineer to program than to teach a programmer the nuances of the system. This turned out to be a mistake, which MIT acknowledged²⁹. Thus, the stage was set for IBM to produce a superb real-time program. However, the complexity of the control moment laws, the redundancy management needs, and the inevitable memory overrun kept the development from being simple.

Requirements Definition and Design

IBM and NASA jointly defined the requirements for the Skylab software. Marshall Space Flight Center delivered the detailed requirements for the control laws, navigation, and momentum management, leaving lesser items such as I/O handling to the contractor. IBM and NASA made a parallel effort to determine if the equations actually worked³⁰. The result was the Program Requirements Document (PRD), issued July 1, 1970³¹.

The actual design, the Program Definition Document (PDD), was released later and served as the baseline for the software, which meant that the design could not be changed without formal review. The software resulting from these documents ranged from 9,000 words to nearly 20,000 words of memory. Since the memory size of the computer was just over 16,000 words, a "scrub" was necessary, continuing the NASA tradition of exceeding the memory size of an already-procured computer by the time the planners knew the final requirements. Managers had not yet learned that software needs should drive the hardware choices. Engineers changed the control moment gyro

logic to reduce core usage and made other cuts³². Memory became the prime consideration in allowing requirements changes³³.

Architecture and Coding

Skylab gave IBM an opportunity to demonstrate how to do software development right. The company carefully separated the production process into strictly designed phases. Two different flight loads resulted: one full-function program that filled the 16K memory, and an 8K version as a backup that needed only one module for storage. These two programs needed slightly different architectures, or schemes for organizing the execution of functions, which made the job tougher. Also increasing complexity was the requirement for redundancy management. An advanced development environment helped keep the complexity under control.

Production Phases

IBM developed the software load for Skylab in four baselined phases. Originally, three were planned: Phase I, Phase II, and Final, but numerous changes made during Phase I required an intermediate stage Phase IA. Crews used the software resulting from Phase IA for training in the simulators in Houston³⁴.

The PDD for Phase I was released on November 4, 1970, and coding began³⁵. The Phase I program contained most of the major components of the eventual flight load, including discrete I/O and interrupt processing, command system processing, initialization, redundancy management, attitude reference determination, attitude control, momentum desaturation, maneuvering, navigation and timing, ATM experiment control, displays, telemetry, and algorithms for utilities³⁶. IBM's programming team completed and released the Phase I program for verification on June 23, 1971. It consisted of 16,224 words, filling about 99% of the computer's memory³⁷.

It was this situation that led to the added phase, which was chiefly a memory scrub. Not only was Phase I a fairly extensive program, three modules still had to be coded and many changes would likely occur in the nearly 20 months remaining before launch³⁸. By the time IBM delivered Phase IA on February 9, 1972, it had incorporated 45 waivers and 105 software change requests (SWCR) made after the thirteenth revision of the design³⁹. This meant that nearly 40% of the original program was changed. Even with the attention to memory size, the new software amounted to 16,111 words, or 98.3% of the locations.

Phase II represented another extensive revision of the software. The baseline for it was Phase IA plus 49 approved change requests. By delivery on August 28, 1972, 102 additional changes had been incorporated and the design was up to revision 19⁴⁰. Therefore, software engineers modified about 35% of the program. The memory usage rose to 99.7%, or 16,338 locations. The final version reduced this to 16,329 words. The difference between Phase II and the flight release was only 17 additional changes. IBM made the delivery March 20, 1973, 2 months before the launch.

Architecture: The 16K Program

The ATMDC software divided into an executive and applications modules. The executive module handled the priority multitasking, interrupt processing, supporting the interval timer and also basic timekeeping chores⁴¹. Applications consisted of three major groups: time-dependent functions, asynchronous functions, and utilities. Time-dependent functions were executed in three cycles, with the possibility of higher priority jobs interrupting the currently running module. The cycles were differentiated by time: There was a "slow loop" each second, an "intermediate loop" executing five times each second, and the switch-over processor running each half second⁴². Designers grouped appropriate modules in a cycle. An exception to the cycle groupings, but nevertheless time dependent, was the output-write routine, which was run between intermediate loops in order to take more efficient advantage of the system resources. The switch-over process aided in redundancy management, as explained below. Asynchronous functions could be called at any time, one of which was telemetry, which sent 24 strings of 50 bits per second. The other was the command system, which could receive signals from either the ground or the Digital Address System (DAS, the crew interface) in the workshop. Those signals resulted in interrupts. Utility functions included such common algorithms as square root, sine, and cosine, and unique functions such as gimbal angle computations and quaternion multiplication⁴³.

Interrupt handling was quite straightforward. Each application module had a specific priority ranking. Tasks could be requested by several means, such as interrupts, discrete signals, elapsed time, or by the direct request of another program. Any current task could be interrupted when a new task was requested. The priority of the new task was immediately entered into the priority level control tables. If the new task was of a higher priority than the current task, the computer did the new one first. When telemetry or the command system requested a task, its priority was entered on the table, just like tasks

THE SKYLAB COMPUTER SYSTEM 75

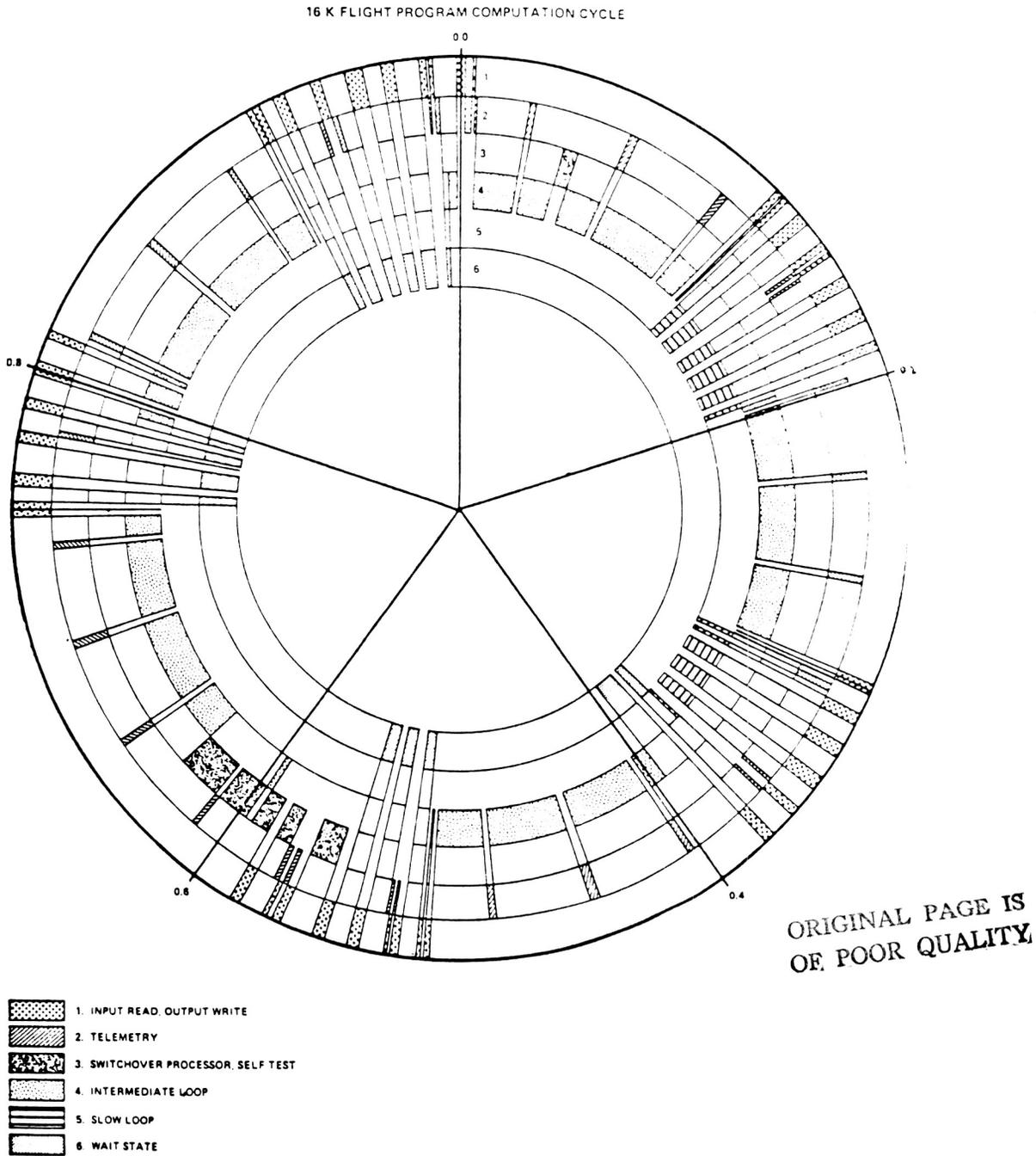


Figure 3-4. The real-time cycle of the Skylab 16K flight program. (From IBM, *Skylab Operation Assessment*, ATMDC, 1974)

called in the other ways⁴⁴. The standard telemetry signal functioning as a Digital Command System (DCS) word consisted of 35 bits. Buried in it were an enable bit, an execute bit, and 12 information bits. The enable and execute bits caused an interrupt, making it possible for the data to be stored⁴⁵.

The 16K program had a computation cycle consisting of six levels: experiment input, Control Moment Gyro gimbal rates, Workshop Computer Interface Unit tests, and the command system processor; telemetry output; the switch-over timer (reset each second) and 64-bit transfer register (refreshed about once every 17 seconds); the intermediate loop (made up of Control Moment Gyro control); the slow loop (containing timing, navigation, maneuver, momentum management, display, redundancy, self test, and experiment support functions); and the "wait" state (when all functions in a particular cycle finished, about 15% of cycle time in the flight release of the program, depending on the number and nature of interrupts⁴⁶).

The 8K Program

The 8K program was strongly related to the 16K program in that the larger version served as the model for the smaller. Its design, released April 3, 1972, developed from the Phase IA version of the software. IBM delivered the 8K program on November 14, 1972 after 10 weeks of verification activity. The functions of the short program were largely limited to attitude control and solar experiment activity and data handling⁴⁷. It was 8,001 words in length. IBM reduced the number of levels in the computation cycle of the 8K program to four: Level I handled command processing and I/O to the Gyros, Level II did telemetry, Level III consisted of the time-dependent functions from both the original intermediate loop and slow loop, and Level IV was the wait state⁴⁸.

Redundancy Management

All mission-critical systems in Skylab were redundant. The computer program contained 1,366 words of redundancy management software⁴⁹. At less than 10% of the total memory, it was a bargain. Managing redundancy with stand-alone hardware and solely mechanical switching would have added much more cost, weight, and complexity to the workshop design, with the loss of a certain amount of reliability.

The redundancy management software consisted of two parts: self

tests of the computer system and an error detection program for mission-critical hardware not in the computer system. Self tests of the computer were quite extensive: Logic tests might involve doing a Boolean OR operation on the contents of a register to see if a carry occurred; operation tests required executing EXCHANGE and LOAD instructions; and arithmetic tests meant executing an ADD and checking for planned answer⁵⁰. IBM also designed tests for memory addressing and I/O⁵¹.

The error detection program examined critical signs in several systems. If a failure was detected in attitude control hardware such as the Control Moment Gyros, rate gyros or acquisition sun sensors, then backups or reconfigurations were activated⁵². During the mission, one Control Gyro and several of the rate gyros failed. In fact, a "six-pack" of replacement rate gyros had to be brought up by the second crew.

Switch-over between the two computers was handled by the error detection program or automatically activated by the TMR timer circuits. If self tests indicated a computer hardware failure or that the software was not properly maintaining the workshop's attitude, switch-over would then be initiated. The timers were supposed to be reset about once each second during the computation cycle, after which they then counted down until reset. If two of the three reached zero, then switch-over occurred⁵³. Besides automatic switch-over, the crew or the ground could initiate it, as actually happened in mid-mission. So that the secondary computer would be properly activated, a 64-bit transfer register was kept loaded with relevant data. This register, like the timers, consisted of TMR circuits. Great care was taken to ensure that data loaded into the transfer register were uncontaminated. A write operation to the register was restricted in length to a period of 672 microseconds plus or minus 20%, which was just about how long it took to write 64 bits into a redundant circuit. This operation could only take place after 1.5 to 2.75 seconds had elapsed since the last write, so the computer would not accept transient signals as correct data and a new write could not interfere with an earlier write⁵⁴. Besides this "time-out feature," the transfer register could only be refreshed *after* a successful execution of the error detection program⁵⁵. This way, data could not be written to the register from a failed computer.

The redundancy management software was a step toward the eventual Shuttle redundancy management scheme. Previously, IBM had used TMR hardware to ensure reliability. This system, with its watchdog timer, was software based and, in effect, saved space and weight. Two ATMDCs were smaller and required less power than a single TMR computer of equal reliability.

The Development Environment and Integration

The Skylab software development was done in a programming environment that took advantage of useful software tools and proper integration techniques. Binary code for the computer was in hexadecimal (base 16) format, and loaded in that format⁵⁶. Hand coding in hex is rather tedious, so IBM prepared an assembler to translate mnemonics into it. They also provided a relocatable loader for placing separately coded modules in contiguous memory locations. Macros, blocks of frequently used code, were kept in common libraries. Listings of programs and the original source resided in an IBM System 360/75⁵⁷. This environment was small compared with the later Software Production Facility for the Shuttle, but the concept of a good tool set, promoted by IBM's Mills and Brooks, was well realized.

Integration of the Skylab software followed a top-down approach: The program was highly modular so as to keep individual functions separate for easy modification and also simple enough for a single programmer to handle. The executive and major subprocesses were coded and integrated first; then the remaining modules were added. The modules were grouped into three batches, so all the modules in a batch were added and tested, then the next batch would be added, and so on⁵⁸. This helped in the integration process.

Verification

The software for Skylab was one of the most extensively verified systems of its era. Since it was a real-time program, verification was more difficult than a corresponding batch program because it is hard to replicate test inputs when interrupts can occur at any time; thus, a combination of simulators is needed to properly verify a real-time program.

IBM used a number of different simulation configurations in the verification process. The AS-II simulator consisted of a System 360/75 used for analysis of the Skylab while it was in orbit. It could evaluate the effects of changes to the flight program. The Skylab Workshop Simulator (SWS) was an all-digital simulation used in developing the initial software, as well as verification. It ran at a 3.5/1 ratio of execution time to real time. The SWS was so effective that it once correctly identified a deficiency in the requirements relating to the Control Moment Gyro system. The Skylab Hybrid Simulator (SHS) included some analog circuits for greater fidelity. One of the most effective simulators was a System 360/44 connected to an actual ATMDC; the program in the 44 could simulate six degrees of freedom⁵⁹.

The verification process was scheduled for the final 10 weeks prior to the delivery of any software phase. The process included validation of the baseline program to the requirements, coding analysis, logic analysis, equation implementation tests, performance evaluations, and mission procedure validation. The AS-II did the logic analysis and was designed to trace all logic paths through the software. The 360/44 and ATMDC system did performance tests since it was near real time in operation⁶⁰. The digital simulators could be stopped in order to insert program changes. Tracing was also possible⁶¹. Combining simulators and software verification tools contributed to a high level of confidence that was confirmed in actual performance.

USER INTERFACES

NASA and IBM designed the computer system to operate autonomously. One crewman reported "not much interaction" with the system at all⁶², but the capability was present for significant activity if needed⁶³. The crew could enter data and actually make changes in the software through a keyboard located in the DAS on the ATM Control and Display Console.

The DAS had only 10 keys and a three-position switch. The keys were the digits 0-7 (all entries were in octal), a clear key, and an enter key. The switch could select either power bus one or two, or be off. Above the DAS was an "Orbit Phase" panel containing a digital readout of minutes and seconds to the next orbital benchmark. When the first keystroke of a five-digit command was made, the uplink DCS commands were inhibited, and the time remaining clock inputs were inhibited, so that the clock digits could be used for displaying the keystrokes. In that mode, five digits would be lit instead of four. The remaining four keystrokes were the data/command input⁶⁴. The display of the keystrokes represented an echo. If the sequence was correct, the astronaut pressed the enter key, or else he would restart the input process. Pressing the clear key brought back the digital clock. The rather limited nature of this command system indicates that it was intended for sparing use.

Besides the DAS, one other switch on the control panel related to the computer system. In the "Attitude Control" area of the panel was a three-position switch that controlled which computer was in actual use. It could be set for automatic (and usually was), in which case the redundancy management software would take care of matters. Alternately, the crew could purposely select either the primary or secondary computer. If either of these was selected, then automatic change-over was inhibited⁶⁵. The switch gave the crew protection from

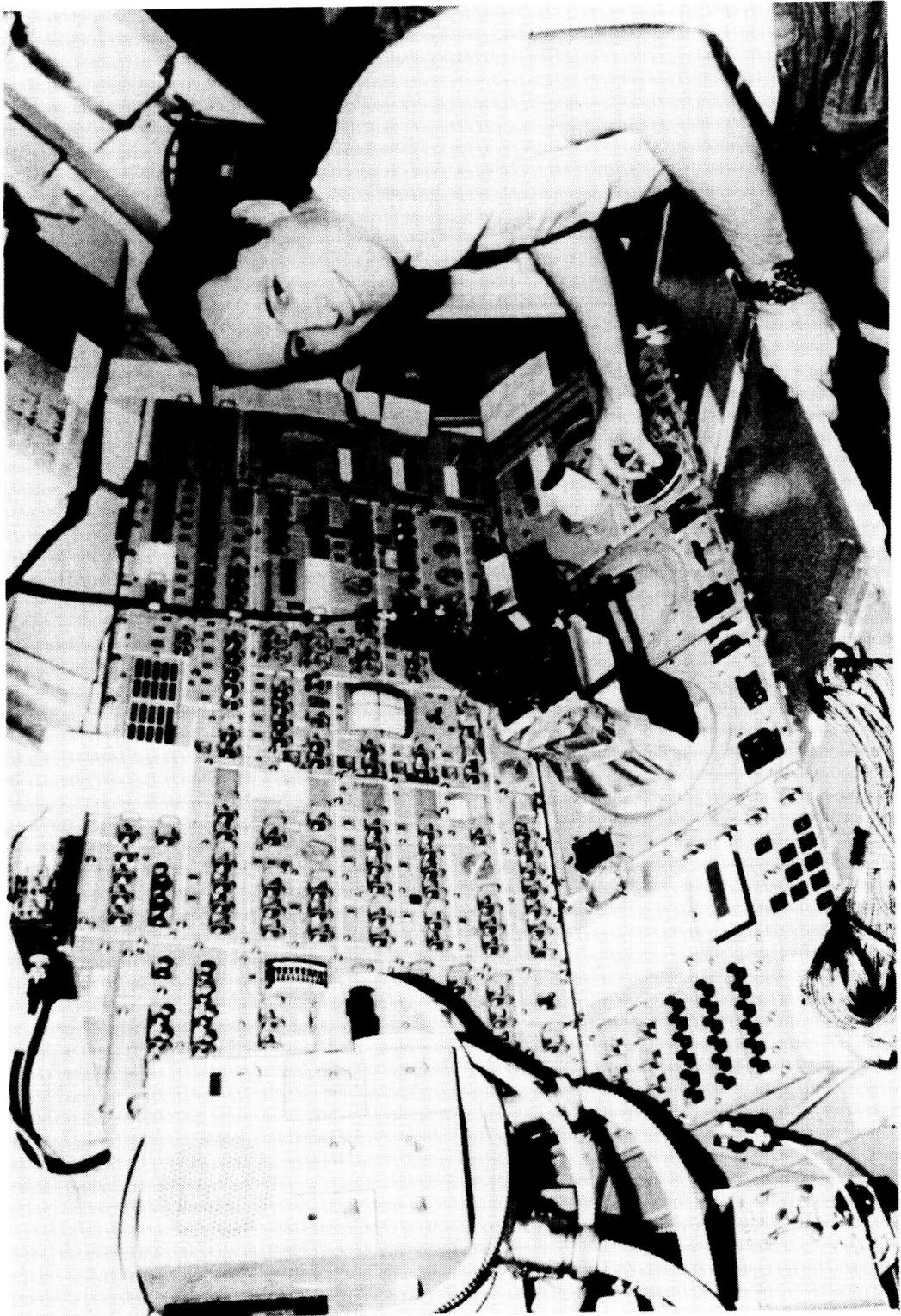


Figure 3-5. Dr. Edward Gibson at the Apollo Telescope Mount Control console. The interface to the digital computer is at lower left, on the panel immediately above the coil of cable. (NASA photo 4-60352)

failure of the redundancy management software. Incidentally, the switch was not a common three-position toggle switch but, instead, required the crew to pull out and rotate the post. This protected the crew from accidental switching.

THE REACTIVATION MISSION

The Skylab Reactivation Mission represents one of the most interesting examples of the autonomy and reliability of manned spacecraft computers. The original Skylab mission lasted 272 days with long unmanned periods. The reactivation mission, flown entirely under computer control, lasted 393 days. Therefore, the bulk of the activated life of the space laboratory fully depended on the ATMDCs.

When it was obvious that the Workshop was going to fall to the earth long before a rescue mission could be launched, NASA began studying methods of prolonging the orbital life of the spacecraft. Even though the atmosphere is very thin at the altitude Skylab was flying, the drag produced on the spacecraft was highly related to its attitude with respect to its direction of flight (velocity vector). During most of the manned mission periods Skylab flew in solar inertial (SI) mode, in which the lab was kept perpendicular to the sun to provide maximum exposure for the solar collectors. Momentum desaturation maneuvers were done on the dark side of the earth to compensate for bias momentum buildup resulting from noncyclic torques acting on the spacecraft. The SI mode was high drag, so engineers devised two new modes, end-on-velocity-vector (EOVV) and torque equilibrium attitude (TEA). EOVV pointed the narrow end of the lab in the direction of flight, minimizing the aerodynamic drag on the vehicle. TEA could control the re-entry, using the gravity gradient and gyroscopic torques to counterbalance the aerodynamic torque. Only in this way could the Workshop be controlled below 140 nautical miles altitude⁶⁶.

Use of the new modes required that they be coded and transmitted to the computers in orbit. First it was necessary to discover whether or not the computers still functioned. Since the ATMDC used destructive readout core memories, there was some concern that the software might have been destroyed during restart tests if the refreshment hardware had failed. On March 6, 1978, NASA engineers at the Bermuda tracking station ordered portions of Skylab to activate. On March 11, the ATMDC powered up for 5 minutes to obtain telemetry confirmation that it was still functioning. The software resumed the program cycle where it had left off 4 years and 30 days earlier. As far as the computer was concerned, it had suffered a temporary power transient⁶⁷!

When IBM began to make preparations to modify the software, it discovered that there was almost nothing with which to work. The

carefully constructed tools used in the original software effort were dispersed beyond recall, and, worse yet, the last of the source code for the flight programs had been deleted just weeks beforehand. This meant that changes to the software would have to be *hand coded* in hexadecimal, as the assembler could not be used—a risky venture in terms of ensuring accuracy. Eventually it became necessary to repunch the 2,516 cards of a listing of the most recent flight program, and IBM hired a subcontractor for the purpose⁶⁸.

Engineers could not test this software with the same high fidelity as during the original development. They abandoned plans for real time simulations because they could not find enough parts of any of the original simulators. Interpretive simulation could be performed because the tapes for that form of testing had been saved. However, the interpretive simulator ran 20 times slower than real time, so less testing was possible⁶⁹.

IBM approached the modification using the same principles as in the original production. The baseline software for the reactivation was Flight Program 80, including change request 3091, which was already in the second computer. Software changes for reactivation were simply handled as routine change requests. They placed the EOVV software in memory previously occupied by experiment calibration and other functions useless in the new mission. TEA replaced the command and display software⁷⁰.

When the software was ready for flight, NASA uplinked it to a reserve area of memory and then downlinked and manually verified it. If it passed the verification, engineers gave a command to activate it. The reprogramming was generally successful. The four people assigned to the software revision maintained IBM's record of quality throughout the reactivation mission⁷¹.

CONCLUSIONS

The Skylab program demonstrated that careful management of software development, including strict control of changes, extensive and preplanned verification, and the use of adequate development tools, results in quality software with high reliability. Attention to piece part quality in hardware development and the use of redundancy resulted in reliable computers. However, it must be stressed that part of the success of the software management and the hardware development was due to the small size of both. Few programmers were involved in initial program design and writing. This meant that communications between programmers and teams were relatively minimal. The fact that IBM produced just 10 computers and really needed to ensure the success of just 2 of those helped in focusing the quality assurance effort expended on the hardware.

What happened after the manned Skylab program demonstrated the need for foresight and proper attention to storage of mission-critical materials until any possibility of their use had gone away. The dispersal of the verification hardware is understandable, as it is expensive to maintain. However, some provision should have been made for retaining mission-unique capabilities such as actual flight hardware. The destruction of the flight tapes and source code for the software by unknown parties was inexcusable. A single high-density disk pack could have held all relevant material.

Skylab marked the beginning of redundant computer hardware on manned spacecraft. It was also the first project that developed software with awareness of proper engineering principles. The Shuttle continued both these concepts but on a much larger and more complex scale.

4

Computers in the Space Shuttle Avionics System

Computers are used more extensively on the Space Transportation System (STS) than on any previous aircraft or spacecraft. In conventional aircraft, mechanical linkages and cables connect pilot controls, such as the rudder pedals and stick, to hydraulic actuators at the control surfaces. However, the Shuttle contains a fully digital fly-by-wire avionics system. All connections are electrical and are routed through computers. To give the spacecraft more autonomy, system management functions (fuel levels, life support, etc.), handled on the ground during previous flight programs, are monitored on board. Software can be adjusted to suit increasingly complex and varied payloads. Subsystems, like the main engines, that had no computer assistance before use them for performance improvement. And, as in Gemini and Apollo, guidance and navigation tasks are accomplished on the Shuttle with computers. All these functions, especially flight control, are critical to mission success; therefore, the computers performing the tasks must be made fail-safe by using redundancy. Meeting these requirements has resulted in one of the most complex software systems ever produced and a computer network within the spacecraft with more powerful hardware than many ground-based computer centers in the mid-1960s.

The major differences between the Shuttle computer system and the systems used on Gemini and Apollo were the choice of an "off-the-shelf" main computer instead of a custom-made machine and the pervasiveness of the system within the spacecraft, since the main computers are the heart of any true avionics system. Avionics (*aviation plus electronics*) grew in the 1950s and 1960s as electronic devices, especially digital devices, replaced mechanical or analog equipment in aircraft. These digital devices were combined into a coherent system, rather than isolated in function and location within the aircraft. Several modern military airplanes have applied this concept to varying degrees. The FB-111, an Air Force tactical bomber, has a complex avionics system that Rockwell International built just before it was awarded the Shuttle contract¹; the F-15 fighter used an AP-1 computer in its system. A repackaged version of the F-15's computer became the AP-101 used in the shuttle². However, in no aircraft has the Shuttle's avionics system been matched as yet. For instance, its main computers have to interconnect with other computers in subsystems, such as the controllers on each main engine, whereas most aircraft systems are centered on a single set of machines.

Since the Shuttle is completely dependent on the success of its avionics system, each component must be made failure proof. The method chosen to ensure this is absolute redundancy, often to a depth of four duplicate devices. Managing this level of redundancy became a large problem in itself.

Another result of the pervasive avionics system is that the frequency and sophistication of the crew interaction with the computers exceeds any previous manned space program. A large portion of the

ORIGINAL PAGE IS
OF POOR QUALITY.

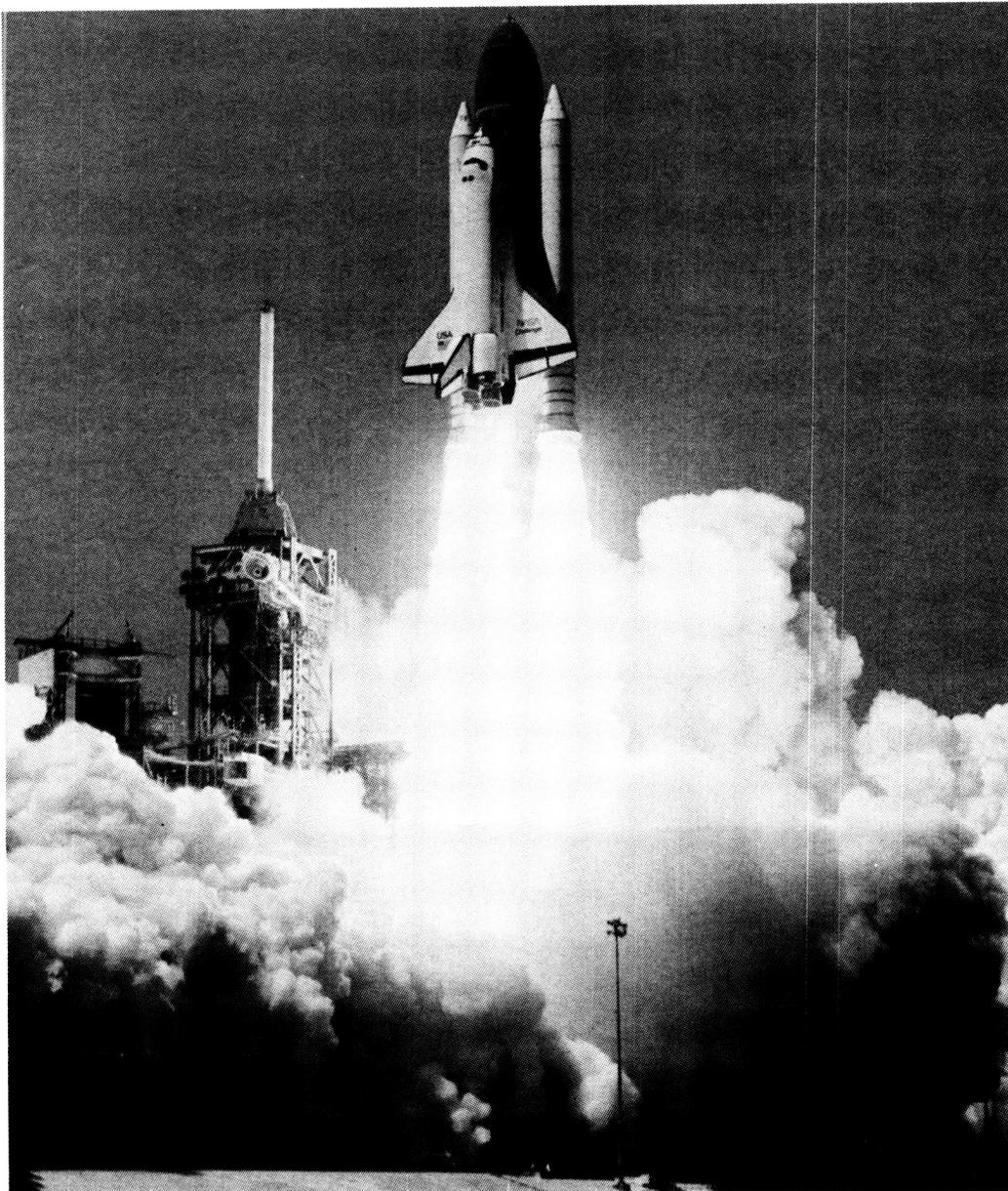


Figure 4-1. The first launch of the Shuttle *Challenger*, one of a fleet of the most computationally intensive spacecraft ever built. (NASA photo)

software is directed at easing the necessary commanding of the computers. In general, software development for the Shuttle has far outstripped any previous NASA ground or flight system in effort and cost. The combination of requirements forced the Agency to pioneer techniques in digital avionics, redundancy management, computer interconnection, and real-time software development.

EVOLUTION OF THE SHUTTLE COMPUTER SYSTEM

Planning for the STS began in the late 1960s, before the first moon landing. Yet, the concept of a winged, reusable spacecraft went back at least to World War II, when the Germans designed a sub-orbital bomber that would "skip" along the upper atmosphere, dropping bombs at low points in its flight path. In America in the late 1940's, Wernher von Braun, who transported Germany's rocket knowhow to the U.S. Army, proposed a new design that became familiar to millions in the pre-Sputnik era because Walt Disney Studios popularized it in a series of animated television programs about spaceflight. It consisted of a huge booster with dozens of upgraded V-2 engines in the first stage, many more in the second, and a single-engine third stage, topped with a Shuttle-like, delta-winged manned spacecraft.

Because the only reusable part of the von Braun rocket was the final stage, other designers proposed in its place a one-piece shuttle consisting of a very large aerospacecraft that was intended to fly on turbojets or ramjets in the atmosphere before shifting to rocket power when the atmospheric oxygen ran out. Once it returned from orbit, it would fly again under jet power. However, the first version of the reusable spacecraft to actually begin development was the Air Force Dyna-Soar, which had a lifting body orbital vehicle atop a Titan III booster. That project died in the mid-1960s, just before NASA announced a compromise design of desirable features: the expensive components (engines, solid rocket shells, the orbiter) to be reusable; the relatively inexpensive component, the external fuel tank, to be expendable; the orbiter to glide to an unpowered landing³.

The computer system inside the Shuttle vehicle underwent an evolution as well. NASA gained enough experience with on-board computers during the Gemini and Apollo programs to have a fair idea of what it wanted in the Shuttle. Drawing on this experience, a group of experts on spaceborne computer systems from the Jet Propulsion Laboratory, the Draper Laboratory (renamed during its Apollo efforts) at MIT, and elsewhere collaborated on an internal NASA publication that was a guide to help the designer of embedded spacecraft computers⁴. Individuals contributed additional papers and memos. Preliminary design proposals by potential contractors also influenced

the eventual computer system. In one, Rockwell International teamed up with IBM to submit a system⁵. Previously, in 1967, the Manned Spacecraft Center contracted with IBM for a conceptual study of spaceborne computers⁶ and two Huntsville IBM engineers did a shuttle-specific study in 1970⁷. Coupled with IBM Gemini and Saturn experience, the Rockwell/IBM team was hard to beat for technical expertise. NASA also sought further advice from Draper, as it was still heavily involved in Apollo⁸. These varied contributions shaped the final form of the Shuttle's computer system.

There were two aspects of the computer design problem: functions and components. Previous manned programs used computers only for guidance, navigation, and attitude control, but a number of factors in spacecraft design caused the list of computable functions to increase. A 1967 study projected that post-Apollo computing needs would be shaped by more complex spacecraft equipment, longer operational periods, and increased crew sizes⁹. The study suggested three approaches to handling the increased computer requirements. The first assigned a small, special-purpose computer to each task, distributing the processes so that the failure of one computer would not threaten other spacecraft systems. The second approach proposed a central computer with time-sharing capability, thus extending the concepts implemented in Gemini and Apollo. Finally, the study recommended several processors with a common memory (a combination of the features of the first two ideas). This last concept was so popular that by 1971 at least four multiprocessor systems were being developed for NASA's use¹⁰.^{*} The greater appeal of the multiprocessors, and the production of the Skylab dual computer system, replaced the idea of using simplex computer systems that could not be counted on to be 100% reliable on long-duration flights.

On a more detailed level than the overall configuration, experts also realized that increased speed and capacity were needed to effectively handle the greater number of assigned tasks¹¹. One engineer suggested that a processor 50% to 100% more powerful than first indicated be procured¹². This would provide insurance against the capacity problems encountered in Gemini and Apollo and be cheaper than software modifications later. A further requirement for a new manned spacecraft computer was that it be capable of floating-point arithmetic. Previous computers were fixed-point designs, so scaling of the calculations had to be written into the software. Thirty percent of the Apollo software development effort was spent on scaling¹³.

^{*}These were: EXAM (*Experimental Aerospace Multiprocessor*) at Johnson Space Center, the Advanced Control, Guidance, and Navigation Computer at MIT, SUMC (*Space Ultrareliable Modular Computer*) at Marshall Space Flight Center, and PULPP (*Parallel Ultra Low Power Processor*) at the Goddard Space Flight Center.

One holdover component from the Gemini, Apollo, and Skylab computers remained: core memory. Mostly replaced by semiconductor memories on IC chips, core memory was made up of doughnut-shaped ferrite rings. In the mid-1960s, core memories were determined to be the best choice for manned flight for the indefinite future, because of their reliability and nonvolatility¹⁴. Over 2,000 core memories flew in aircraft or spacecraft by 1978¹⁵. The NASA design guide for spacecraft computers recommended use of core memory and that it be large enough to hold all programs necessary for a mission¹⁶. That way, in emergencies, there would be no delay waiting for programs to be loaded, as in Gemini 8, and the memory could be powered down when unneeded without losing data.

By 1970, several concepts to be used in the Shuttle were chosen. One of these was the use of buses, which Johnson Space Center's Robert Gardiner considered for moving large amounts of data¹⁷. Instead of having a separate discrete wire for every electronic connection, components would send messages on a small number of buses on a time-shared basis. Such buses were already in use in cabling from the launch center to rockets on the launch pads. Buses were also being considered for military and commercial aircraft, which were becoming quite dependent on electronics. Additionally, there would be two redundant computer systems— though no decision had been made as to how the systems would communicate. In the LEM, the PGNCs had an active backup in the Abort Guidance System (AGS). This was not true redundancy in that the AGS contained a computer with less capacity than the AGC, and so could not complete a mission, just safely abort one. True redundancy, however, meant that each computer system would be capable of doing all mission functions.

Redundancy grew out of NASA's desire to be able to complete a mission even after a failure. In fact, early studies for the Shuttle predicated the concept of "fail operational/fail operational/fail-safe." One failure and the flight can continue, but two failures and the flight must be aborted because the next failure reduces the redundancy to three machines, the minimum necessary for voting. In the 1970 computer arrangement, one special-purpose computer handled flight control functions (the fly-by-wire system), and another general-purpose computer performed guidance, navigation, and data management functions. These two computers had twins and the entire group of four was duplicated to provide the desired layers of redundancy¹⁸.

More concrete proposals came in 1971. Draper presented a couple of plans, one fairly conservative, the other more ambitious. The less expensive version used two sets of two AGCs. These models of the AGC would contain 32K of erasable memory and magnetic tape mass memory instead of the core rope in the original¹⁹. Redundancy would be provided by a full backup that would be automatically switched into action upon failure of the primary (an idea later abandoned since

a software fault could cause a premature switch-over)²⁰. Draper's more expensive, but more robust, plan proposed a "layered collaborative computer system," to provide "significant total, modest individual computing power"²¹. A relatively large multiprocessor was at the heart of this system, with local processors at the subsystem level. This had the potential effect of insulating the central computer from subsystem changes.

Unlike Gemini and Apollo, NASA wanted an off-the-shelf computer system for the Shuttle. If "space rating" a system involved a stricter set of requirements than a military standard²², starting with a military-rated computer made the next step in the certification process a lot cheaper. Five systems were actively considered in the early 1970s: The IBM 4Pi AP-1, the Autonetics D232, the Control Data Corporation Alpha, the Raytheon RAC-251, and the Honeywell HDC-701²³. The basic profile of the computer system evolved to the point where expectations included 32-bit word size for accurate calculations, at least 64K of memory, and microprogramming capability²⁴. Microprograms are called firmware and contain control programs otherwise realized as hardware. Firmware can be changed to match evolving requirements or circumstances. Thus, a computer could be adapted to a number of functions by revising its instruction set through microcoding.

Despite the fact that Draper Laboratory favored the Autonetics machine, and a NASA engineer estimated that the load on the Shuttle computers would "be heavier than the 4Pi [could] support," the IBM machine was still chosen²⁵. The 4Pi AP-1's advantages lay in its history and architecture. Already used in aircraft applications, it was also related to the 4Pi computers on Skylab, which were members of the same architectural family as the IBM System 360 mainframe series. Since the instruction set for the AP-1 and 360 were very similar, experienced 360 programmers would need little retraining. Additionally, a number of software development tools existed for the AP-1 on the 360. As in the other spacecraft computers, no compilers or other program development tools would be carried on-board. All flight programs were developed and tested in ground-based systems, with the binary object code of the programs loaded into the flight computer. Simulators and assemblers for the AP-1 ran on the 360, which could be used to produce code for the target machine. In both the Gemini and Apollo programs, such tools had to be developed from scratch and were expensive.

One further aspect of the evolution of the Shuttle computer systems is that previous manned spacecraft computers were programmed using assembly language or something close to that level. Assembly language is very powerful because use of memory and registers can be strictly controlled. But it is expensive to develop assembly language programs since doing the original coding and verifying that the

programs work properly involve extra care. These programs are neither as readable nor as easily tested as programs written in FORTRAN or other higher-level computer languages. The delays and expense of the Apollo software development, along with the realization that the Shuttle software would be many times as complex, led NASA to encourage development of a language that would be optimal for real-time computing. Estimates were that the software development cycle time for the Shuttle could be reduced 10% to 15% by using such a language²⁶.

The result was HAL/S, a high-level language that supports vector arithmetic and schedules tasks according to programmer-defined priority levels.** No other early 1970s language adequately provided either capability. Intermetrics, Inc., a Cambridge firm, wrote the compiler for HAL. Ex-Draper Lab people who worked on the Apollo software formed the company in 1969²⁷.

The proposal to use HAL met vigorous opposition from managers used to assembly language systems. Many employed the same argument mounted against FORTRAN a decade earlier: The compiler would produce code significantly slower or with less efficiency than hand-coded assemblers. High-level languages are strictly for the convenience of programmers. Machines still need their instructions delivered at the binary level. Thus, high-level languages use compilers that translate the language to the point where the machine receives instructions in its own instruction set (excepting certain recently developed LISP machines, in which LISP *is* the native code). Compilers generally do not produce code as well as humans. They simply do it faster and more accurately. However, many engineers felt that optimization of flight code was more important than the gains of using a high-level language. To forestall possible criticism, Richard Parten, the first chief of Johnson's Spacecraft Software Division, ordered a series of benchmark tests. Parten had IBM pick its best assembly language programmers to code a set of test programs. The same functions were also written in HAL and then raced against each other. The running times were sufficiently close to quiet objectors to high-level languages on spacecraft (roughly a 10% to 15% performance difference)²⁸.

**The origins of the name of the language are unclear. Stanley Kubrick's classic film *2001: A Space Odyssey* (1968) was playing in theaters at about the time the language was being defined. A chief "character" in the film was a murderous computer named HAL. NASA officials deny any relationship between the names. John R. Garman of Johnson Space Center, one of the principals in Shuttle on-board software development, said it may have come from a fellow involved in the early development whose name was Hal. Others suggest it is an acronym for Higher Avionics Language. For a full description of the language and sample programs, see Appendix II.

By 1973, work could begin on the software necessary for the shuttle, as hardware decisions were complete. Conceptually, the shuttle software and hardware came to be known as the Data Processing System (DPS).

THE DPS HARDWARE CONFIGURATION

The DPS hardware in the shuttle avionics system consists of four major components: general-purpose computers, the data bus network, the multifunction cathode ray tube display system, and the mass memory units. Each is a substantial improvement over similar systems in any previous spacecraft. Together, they are a model for future avionics developments.

General-Purpose Computers

NASA uses five general-purpose computers in the Shuttle. Each one is an IBM AP-101 central processing unit (CPU) coupled with a custom-built input/output processor (IOP). The AP-101 has the same type of registers and architecture used in the IBM System 360 and throughout the 4Pi series²⁹. IBM announced the 4Pi in 1966, so by the early 1970s, when Shuttle procurement was complete, the machine had had extensive operational use³⁰. The AP-101 version, which is an upgraded AP-1, has since been used in the B-52 and B-1B military aircraft and the F-8 digital fly-by-wire experimental aircraft. The central processor in each case is the same, but the IOP is adapted to the particular application.

Although one of the reasons for choosing the AP-101 was its familiar instruction set, some modifications were necessary for the Shuttle version. Since the execution of instructions is dependent on microcode, rather than hardware only, the instruction set could be changed somewhat. Microcode is a set of primitives that can be combined to create new logic paths in the hardware. The AP-101 has room for up to 2,048 microinstructions, 48 bits in length³¹.

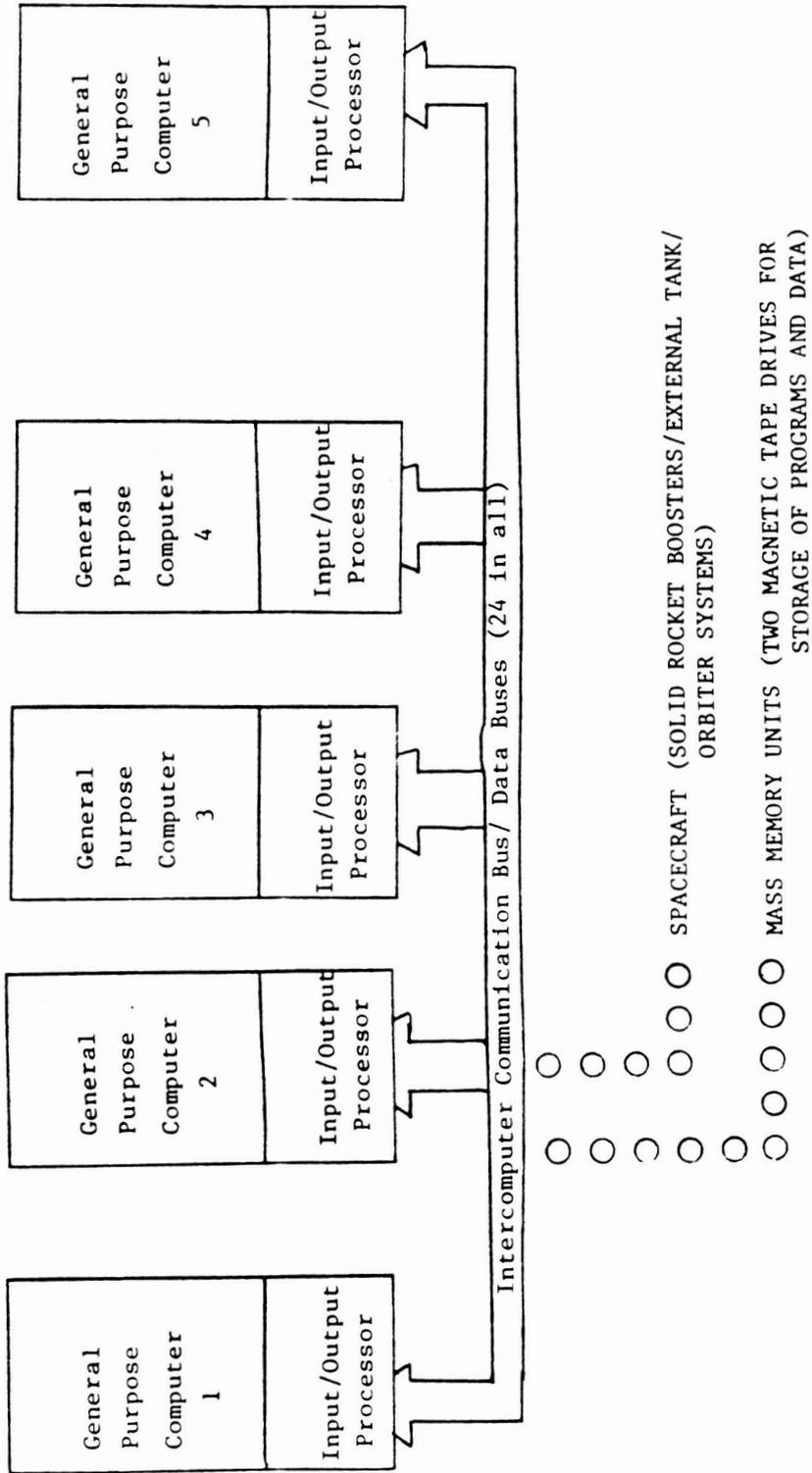


Figure 4-2. A block diagram of the hardware that makes up the Shuttle Data Processing System. The fifth computer is the Backup Flight System computer.

Box 4-1: IBM AP-101 Central Processor and Memory Hardware

Shuttle computers make extensive use of standard ICs. The AP-101 is built using transistor-transistor logic (TTL) semiconductor circuits as the basic building block. The TTL gates are arranged in medium-scale integration (MSI) and large-scale integration (LSI) configurations³². The circuits are on boards that can be interchanged as units.

The AP-101 uses a variety of word sizes. Instructions can be either 16 or 32 bits in length. Fixed-point arithmetic, done using fractional numbers stored in two's complement form, also uses 16- and 32-bit lengths. Floating-point arithmetic is done with 32-, 40- and 64-bit words, although the latter are limited to addition and subtraction³³. Instructions using floating-point take longer to execute than fixed-point arithmetic, and adding is still faster than multiplying; but average speed for the machine is 480,000 instructions per second, compared with 7,000 instructions per second in the Gemini computer³⁴.

The CPU registers are in three groups. Two sets of eight 32-bit registers are available for fixed-point arithmetic. One set of eight 32-bit registers is for floating-point operations³⁵. Semiconductor memories are used in the registers instead of discrete components. As a result, the registers are faster than those used on Gemini and Apollo and, since they are available in large sets, can be used to store intermediate results of calculations without having to access core memory. Thus, processing is accelerated and achieves the performance noted above³⁶.

A program status word (PSW), 64 bits in length, is used to help control interrupts. The PSW contains information such as the next instruction address, current condition code, and any system masks for interrupts³⁷. It has to be updated every instruction to stay current³⁸. Since the AP-101 allows 61 different interrupt conditions divided into 20 priority levels, it is necessary to have an accurate indication of where a program left off when interrupted³⁹. At any given time, several programs are likely to be in a suspended state.

The processor has more than one level of addressing. The common 16-bit address can only directly address 64K words, which was the original memory size of the AP-101. The addressing is extended by replacing the highest order bit with 4 bits from the program status word that indicate which sector of memory to access⁴⁰. This is similar to the scheme used in the AGC when its memory had to be expanded. This configuration allows 131,072 full words (32-bit words) to be addressed. The architecture permits addressing up to 262,144 full words, so memory can be expanded without affecting the processor's design⁴¹.

Box 4-1 (Continued)

Due to packaging considerations, the core memory is located partly in the central processor and partly in the IOP (they are boxed separately). However, it is still considered as a single unit for addressing and access. The entire memory is shared, not just the portion located in the individual boxes. Originally, 40K of core were in the CPU and 24K in the IOP. The memory is organized into modules with 18-bit half words. These contain 16 bits of data, a parity bit, and a storage protect bit to prevent unintentional alteration of the data⁴². The original memory modules contained 8K half words, so 6 were needed in the IOP and 10 in the CPU to store 64K full words. Later memory expansion consisted of replacing the CPU memory modules with double-density modules, in which twice the cores are in the same size container as a single-density module⁴³. So by the first flight, the Shuttle computer memories were 104K words or 106,496 full words of 32 bits. The memory access time is 400 nanoseconds, quite fast for core.

The eventual Shuttle instruction set contained 154 instructions defined within that 2K memory. However, the expected advantages of the flexibility of microcoding, which influenced the decision to select the AP-101, were lessened by the fact that at least six of the new instructions either did not work properly or performed insufficiently⁴⁴. One NASA manager said that the microcoding was bungled by "the ones and zeroes artists" (referring to the binary numbered nature of microprograms) who apparently tried to do things the tricky way⁴⁵.

NASA tried to correct its tendency to underestimate memory size, but was disappointed again on the Shuttle program. One requirement for memory was that it be large enough to contain all the programs necessary for a mission. Therefore, memory estimates became a regular part of preliminary design studies. Most estimates in the 1969 to 1971 period ranged around 28K words⁴⁶. Rockwell International settled on 32K in its bid and won the contract partially because of that estimate⁴⁷. NASA, trying to save itself from later difficulties, bought 64K of memory for each computer, hoping that doubling the estimate would be enough (despite memory increases in previous programs of several hundred percent)⁴⁸. Unfortunately, the software grew to over 700K, requiring not only more computer memory, but the addition of mass memory units to hold programs that would not fit into the extended core. Parten said after this, "I don't know how you ensure proper memory size ahead of time, unless you're incredibly lucky"⁴⁹.

From the standpoint of a spacecraft designer worried about power requirements, an interesting feature of the AP-101 memory is that only the module currently being accessed is at full power. If a memory module is used, it remains at full power for 20 microseconds.

If no further accesses are made in that interval, it automatically goes to medium power. If the entire computer is in standby mode, it goes to low power. An estimated 136 watts are saved by doing this switching⁵⁰.

The memory can be altered in flight. The ground can uplink bursts of 64 16-bit halfwords at a time, which can replace data already in the specified addresses. The crew can also change up to six 32-bit words simultaneously by using their displays and keyboards. However, those changes must be hand keyed in hexadecimal.

The Shuttle's AP-101 contains one of the most extensive sets of self-testing hardware and software ever used in a flight computer. Its self-test hardware resides in the BITE, or *built-in test equipment*. When this is coupled with the self-test software, 95% of hardware failures can be detected by the machine itself⁵¹, whereas the other 5% and potential software failures require the use of redundancy.

As evidenced by the component description given here, the IBM AP-101 is a fairly common computer architecture, easily understandable and programmable by anyone familiar with IBM's large commercial mainframes. The IOPS, bus system, and displays contain the characteristics that make the Shuttle DPS unique.

The IOPs and the Bus System

It is difficult to discuss the Shuttle's IOPs without also talking about the data bus network, because the former are designed to manage the latter. All subsystems on the spacecraft are connected redundantly to at least a pair of data buses. There are 24 of these buses, and the subsystems share them, using multiplexers to control the sharing. Eight of the 24 are "flight-critical data buses" that help fly the vehicle; 5 are used for intercomputer communication among the five general-purpose computers; 4 connect to the four display units; 2 run to the twin mass memory units; 2 more are "launch data buses," and connect to the Launch Processing System; 2 are used for payloads, and the final pair for instrumentation⁵². Each bus is individually controlled by a microprogrammed processor, essentially a small special-purpose computer, called a bus control element (BCE). The BCE can access memory and execute independent programs⁵³. A twenty-fifth computer, the Master Sequence Controller, is used to control I/O flow on the 24 BCEs⁵⁴. Thus, each IOP contains 25 dedicated computers. In addition, the IOP itself is basically a programmable processor with multiple functions. It shares main memory with the central processor. If a program affecting the IOP is initiated by the central processor, a direct memory access channel is opened to speed up reading core. That, however, creates contention for the memory

with the central processor and may have the effect of actually slowing down the system as a whole⁵⁵.

One reason an IOP is needed is that the Shuttle computers transfer data internally in parallel along 18-bit buses. This means that one half word and its associated parity bit are moved from memory to the operation registers and back again all at once. However, data are transferred from orbiter subsystems to the IOP in serial form, one bit at a time. Of course, the serial data are at a high rate (1 megahertz), so transfer speed is not a concern. The conversion of serial data to parallel data is the function of the Multiplexer Interface Adapters in the IOP⁵⁶. The Shuttle DPS also has 16 multiplexer/demultiplexers that convert parallel data to serial for output to the buses⁵⁷.

Input and output to each computer is ultimately controlled in two modes: command and listen. In command mode (CM), signals sent from the host processor to subsystems connected to a bus controlled by a commanding BCE will actually effect the commands. In listen mode, the subsystems will ignore the command signals. In both cases input to the computer from any bus is listened to, but the computer's orders are obeyed only by the systems on the buses for which it is the commander. This moding capability means that a single computer can be assigned a set of buses different from another computer, thus spreading out the responsibilities and protecting against failure. It also means that each computer receives all input data all the time, so that it can take over from a failed computer immediately. This is especially important to the backup flight system. The set of controlled buses is called a "string." A typical string for a single computer might be a pair of flight critical buses, one intercomputer bus (always), a display bus, and a bus from the mass memory unit (MMU), payload, launch, and instrumentation group. The strings can be reconfigured by the crew in flight, which is done periodically as missions proceed through various phases.

Display Electronics

The Shuttle's display system, built by the Norden Division of United Technologies Corporation, is the most complex ever used on a flying machine and contains computers of its own. For the first time in a spacecraft, cathode ray tubes (CRTs) are used as the primary display medium, although a wealth of warning lights that supplement the displays still dot the cockpit. The CRTs hold 26 lines of 51 characters on a 5- by 7-inch screen. That screen size is fairly common on portable computers. However, the number of characters per line is smaller (51 vs. the more common 80) and the number of lines larger (26 vs. the usual 24). The net effect is that the individual characters appear slightly larger on the Shuttle's screens, necessary because although

the user of a portable computer is usually about 16 inches from the screen, on the Shuttle the distance between user and screen is well over 2 feet. Information on the Shuttle's screens appears green on black, and characters can be selectively highlighted. Three of these screens are mounted in the forward cockpit between the pilots. A fourth is aft at the mission specialist station. Keyboards, built by Ebonex, are used for crew input. Two are between the pilots, with a third adjacent to the mission specialist's CRT.

Displays placed on the CRTs are controlled by a special-purpose computer with a 16-bit word size and 8K of memory. This computer provides display control and can create circles, lines, intensity changes (highlighting), and flashing messages. The display software is stored on the MMUs until the computer is powered up. The CRT and its associated processor is referred to as the display electronics unit (DEU)⁵⁸.

Mass Memory Unit: A Late Addition

The final component of the Shuttle's DPS hardware is the mass memory unit (MMU). Originally acquired only to provide initial loading of the orbiter's computers, the MMU, built by Odetics, Inc., has been used extensively to help resolve the memory growth problem. Two of these units are installed on the orbiter, each capable of containing 8 million 16-bit words, enough for three times the Shuttle software. The tape can be addressed in 512 word blocks, and the crew can alter its contents in flight using a special display⁵⁹. The MMU stores all the Primary Avionics Software System and *all* the software for the Backup Flight System, the DEUs, and the engine controllers. Thus, the Shuttle continues the same computer/mass memory configuration as the Gemini spacecraft.

This complex network of computer hardware on the orbiter has many possible points of failure. Also, the 700K of flight software may contain undiscovered bugs that could emerge at critical mission times, and self-testing might not be sufficient to protect the spacecraft from such failures. Other schemes for preventing a fatal failure need to be developed if the Shuttle is to fly with the confidence of its crew, passengers, and potential paying customers. Exactly what those schemes would be has occupied many researchers for several years.

COMPUTER SYNCHRONIZATION AND REDUNDANCY MANAGEMENT

One key goal shaping the design of the Shuttle was "autonomy." Multiple missions might be in space at the same time, and large crews, many with nonpilot passengers, were to travel in space in craft much more self-sufficient than ever before. These circumstances, the desire for swift turnaround time between launches, and the need to sustain mission success through several levels of component failure meant that the Shuttle had to incorporate a large measure of fault tolerance in its design. As a result, NASA could do what would have been unthinkable 20 years earlier: put men on the Shuttle's *first* test flight. The key factor in enabling NASA to take such a risk was the redundancy built into the orbiter⁶⁰.

Fault tolerance on the Shuttle is achieved through a combination of redundancy and backup. Its five general-purpose computers have reliability through redundancy, rather than the expensive quality control employed in the Apollo program⁶¹. Four of the computers, each loaded with identical software, operate in what is termed the "redundant set" during critical mission phases such as ascent and descent. The fifth, since it only contains software to accomplish a "no frills" ascent and descent, is a backup. The four actuators that drive the hydraulics at each of the aerodynamic surfaces are also redundant, as are the pairs of computers that control each of the three main engines.

Management of redundancy raised several difficult questions. How are failures detected and certified? Should the system be static or dynamic? Should the computers run separately without communication and be used to replace the primary computer one by one as failures occur? Could the computers, if running together, stay in step? Should redundancy management of the actuators be at the computer or subsystem level? Fortunately, NASA experience on other aircraft and spacecraft programs could provide data for making the final decisions.

Redundant Precursors

Several systems that incorporated redundancy preceded the Shuttle. The computer used in the Saturn booster instrument unit that contained the rocket's guidance system used triple modular redundant (TMR) circuits, which means that there was one computer with redundant components. Disadvantages to using such circuits in larger com-

puters are that they are expensive to produce, and an event such as the explosion on Apollo 13 could damage enough of the computer that it ceases to function. By spreading redundancy among several simplex circuit computers scattered in various parts of the spacecraft, the effects of such catastrophic failures are minimized⁶².

Skylab's two computers each could perform all the functions required on its mission. If one failed, the other would automatically take over, but both computers were not up and running simultaneously. The computer taking over would have to find out where the other had left off by using the contents of the 64-bit transfer register located in the common section built with TMR circuits. The Skylab computers were able to have such a relatively leisurely switch-over system because they were not responsible for navigation or high-frequency flight control functions. If there were a failure, it would be possible for the Skylab to drift in its attitude without serious danger; the Shuttle would have no such margin of safety.



Figure 4-3. The F-8 aircraft that proved the redundant set configuration planned for the Shuttle would work. (NASA photo ECN-6988)

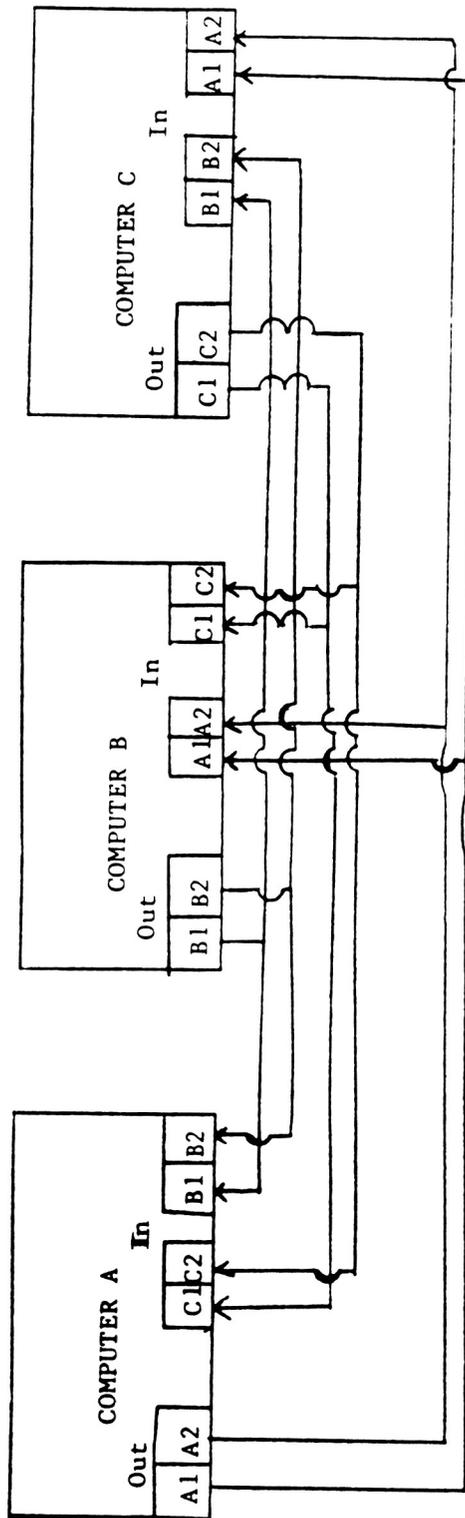
The need for the redundant computers on the Shuttle to process information simultaneously, while still staying closely synchronized for rapid switch-over, seriously challenged the designers of the system. Such a close synchronization between computers had not been done before, and its feasibility would have to be proven before NASA could make a full commitment to a particular design. Most of the

necessary confidence resulted from a digital fly-by-wire testing program NASA started at the Dryden Flight Research Center in the early 1970s⁶³. The first computer used in the F-8 "Crusader" aircraft chosen for the program was a surplus AGC in simplex, with an electronic analog backup. Later, the project engineers wanted a duplex system using a more advanced computer. Johnson Space Center avionics people noted the similarities between the digital fly-by-wire program and the Shuttle. Dr. Kenneth Cox of JSC suggested that Dryden go with a triplex system to move beyond simple one-for-one redundancy. By coordinating procurement, NASA outfitted both the F-8 aircraft and the Shuttle with AP-101 processors. Draper Laboratory produced software for the F-8, and its flight tests proved the feasibility of computers operating in synchronization, as it suffered several single point computer failures but successfully flew on without loss of control. This flight program did much to convince NASA of the viability of the synchronization and redundancy management schemes developed for the Shuttle.

How Many Computers?

One key question in redundancy planning is how many computers are required to achieve the level of safety desired. Using the concept of fail operational/fail operational/fail-safe, five computers are needed. If one fails, normal operations are still maintained. Two failures result in a fail-safe situation, since the three remaining prevent the feared standoff possible in dual computer systems (one is wrong, but which?). Due to cost considerations of both equipment and time, NASA decided to lower the requirement to fail operational/fail-safe, which allowed the number of computers to be reduced to four. Since five were already procured and designed into the system, the fifth computer evolved into a backup system, providing reduced but adequate functions for both ascent and descent in a single memory load. NASA's decision to use four computers has a basis in reliability projections done for fly-by-wire aircraft. Triplex computer system failures were expected to cause loss of aircraft three times in a million flights, whereas quadruple computer system failures would cause loss of aircraft only four times in a *thousand million* flights⁶⁴.

At first the backup flight system computer was not considered to be a permanent fixture. When safety level requirements were lowered, some IBM and NASA people expected the fifth computer to be removed after the Approach and Landing Test phase of the Shuttle program and certainly after the flight test phase (STS-1 through 4)⁶⁵. However, the utility of the backup system as insurance against a generic software error in the primary system outweighed considerations of the savings in weight, power, and complexity to be made by



F-8 Intercomputer Communication

Figure 4-4. The intercommunication system used in the F-8 triplex computer system.

eliminating it⁶⁶. In fact, as the first Shuttle flights approached, Arnold Aldrich, Director of the Shuttle Office at Johnson Space Center, circulated a memo arguing for a sixth computer to be carried along as a spare⁶⁷! He pointed out that since 90% of avionics component failures were expected to be computer failures and that since a minimum of three computers and the backup should exist for a nominal re-entry, aborts would then have to take place after one failure. By carrying a spare computer preloaded with the entry software, the primary system could be brought back to full strength. The sixth computer was indeed carried on the first few flights. In contrast with this "suspenders and belt" approach, John R. Garman of the Johnson Space Center Spacecraft Software Division said that "we probably did more damage to the system as a whole by putting in the backup"⁶⁸. He felt that the institution of the backup took much of the pressure off the developers of the primary system. No longer was their software solely responsible for survival of the crew. Also, integrating the priority-interrupt-driven operating system of the primary computers with the time-slice system of the backup caused compromises to be made in the primary.

Synchronization

Computer synchronization proved to be the most difficult task in producing the Shuttle's avionics. Synchronizing redundant computers and comparing their current states is the best way to decide if a failure has occurred. There are two types of synchronization used by the Shuttle's computers in determining which of them has failed: one for the redundant set of computers established for ascent to orbit and descent from orbit, and one for synchronizing a common set while in orbit. It took several years in the early 1970s to discover a way to accomplish these two synchronizations.

The essence of Shuttle redundancy is that each computer in the redundant set could do all the functions necessary at a particular mission phase. For true redundancy to take place, all computers must listen to all traffic on all buses, even though they might be commanding just a few. That way they know about all the data generated in the current phase. They must also be processing that data at the same time the other computers do. If there is a failure, then the failed computer could drop out of the set without any functional degradation whatever. At the start, the Shuttle's designers thought it would be possible to run the redundant computers separately and then just compare answers periodically to make sure that the data and calculations matched⁶⁹. As it turned out, small differences in the oscillators that acted as clocks within the computers caused the computers to get out of step fairly

quickly. The Spacecraft Software Division formed a committee, headed by Garman, made up of representatives from Johnson Space Center, Rockwell International, Draper Laboratory and IBM Corporation, to study the problem caused by oscillator drift⁷⁰. Draper's people made the suggestion that the computers be synchronized at input and output points⁷¹. This concept was later expanded to also place synchronization points at process changes, when the system makes a transition from one software module to another. The decision to put in the synchronization points "settled everyone's mind" on the issue⁷².

Intercomputer communication is what makes the Shuttle's avionics system uniquely advanced over other forms of parallel computing. The software required for redundancy management uses just 3K of memory and around 5% or 6% of each central processor's resources, which is a good trade for the results obtained⁷⁸. An increasing need for redundancy and fault tolerance in non-avionics systems such as banks, using automatic tellers and nationwide computer networks, proves the usefulness of this system. But this type of synchronization is so little known or understood by people outside the Shuttle program that carryover applications will be delayed.

One reason why the redundancy management software was able to be kept to a minimum is that NASA decided to move voting to the actuators, rather than to do it before commands are sent on buses. Each actuator is quadruple redundant. If a single computer fails, it continues to send commands to an actuator until the crew takes it out of the redundant set. Since the Shuttle's other three computers are sending apparently correct commands to their actuators, the failed computer's commands are physically out-voted⁷⁹. Theoretically, the only serious possibility is that three computers would fail simultaneously, thus negating the effects of the voting. If that occurs, and if the proper warnings are given, the crew can then engage the backup system simply by pressing a button located on each of the forward rotational hand controllers.

Does the redundant set synchronization work? As described, the F-8 version, with redundancy management identical to the Shuttle, survived several in-flight computer failures without mishap. On the first Shuttle Approach and Landing Test flight, a computer failed just as the *Enterprise* was released from the Boeing 747 carrier; yet the landing was still successful. That incident did a lot to convince the astronaut pilots of the viability of the concept.

Synchronization and redundancy together were the methods chosen to ensure the reliability of the Shuttle avionics hardware. With the key hardware problems solved, NASA turned to the task of specifying the most complex flight software ever conceived.

Box 4-2: Redundant Set Synchronization: Key to Reliability

Synchronization of the redundant set works like this: When the software accepts an input, delivers an output, or branches to a new process, it sends a 3-bit discrete signal on the intercomputer communication (ICC) buses, then waits up to 4 milliseconds for similar discretely from the other computers to arrive. The discretely are coded for certain messages. For example, 010 means an I/O is complete without error, but 011 means that an I/O is complete with error⁷³. This allows more information other than just "here I am" to be sent. If another computer either sends the wrong synchronization code, or is late the computer detecting either of these conditions concludes that the delinquent computer has failed, and refuses from then on to listen to it or acknowledge its presence. Under normal circumstances, all three good computers should have detected the single computer's error. The bad computer is announced to the crew with warning lights, audio signals, and CRT messages. The crew must purposely kill the power to the failed computer, as there is no provision for automatic powerdown. This prevents a generic software failure causing all the computers to be automatically shut off.

This form of synchronization creates a tightly coupled group of computers constantly certifying that they are at the same place in the software. To certify that they are achieving the same solutions, a "sumword" is used. While computers are in a redundant set, a sumword is exchanged 6.25 times every second on the ICC buses⁷⁴. A sumword typically consists of a 64 bits of data, usually the least significant bits of the last outputs to the solid rocket boosters, orbital maneuvering engines, main engines, body flap, speed brake, rudder, elevons, throttle, the system discretely, and the reaction control system⁷⁵. If there are three straight miscomparisons of a sumword, the detecting computers declare the computer involved to be failed⁷⁶.

Both the 3-bit synchronization code and sumword comparison are characteristics of the redundant set operations. During noncritical mission phases such as on-orbit, the computers are reconfigured. Two might be left in the redundant set to handle guidance and navigation functions, such as maintaining the state vector. A third would run the systems management software that controls life support, power, and the payload. The fourth would be loaded with the descent software and powered down, or "freeze dried," to be instantly ready to descend in an emergency and to protect against a failure of the two MMUs. The fifth contains the backup flight system. This configuration of computers is not tightly coupled, as in the redundant set. All active computers, however, do continue the 6.25/second exchange of sumwords, called the common set synchronization⁷⁷.

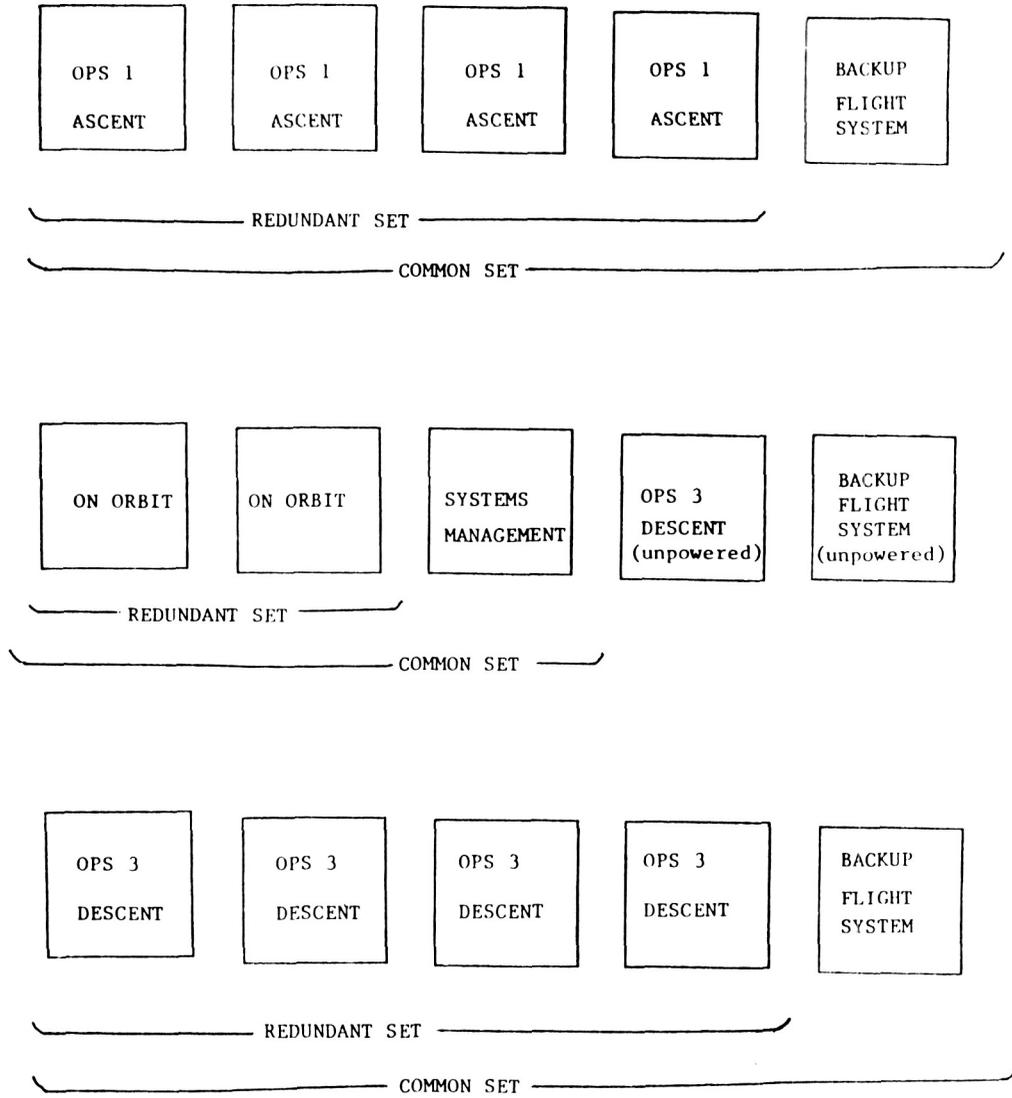


Figure 4-5. The various computer configurations used during a Shuttle mission. The names of the operational sequences loaded into the machines are shown.

DEVELOPING SOFTWARE FOR THE SPACE SHUTTLE

During 1973 and 1974 the first requirements began to be specified for what has become one of the most interesting software systems ever designed. It was obvious from the very beginning that developing the Shuttle's software would be a complicated job. Even though NASA engineers estimated the size of the flight software to be smaller than that on Apollo, the ubiquitous functions of the Shuttle computers meant that no one group of engineers and no one company could do the software on its own. This increased the size of the task because of the communication necessary between the working groups. It also increased the complexity of a spacecraft already made complex by flight requirements and redundancy. Besides these realities, no one could foresee the final form that the software for this pioneering vehicle would take, even after years of development work had elapsed, since there continued to be both minor and major changes. NASA and its contractors made over 2,000 requirements changes between 1975 and the first flight in 1981⁸⁰. As a result, about \$200 million was spent on software, as opposed to an initial estimate of \$20 million. Even so, NASA lessened the difficulties by making several early decisions that were crucial for the program's success. NASA separated the software contract from the hardware contract, closely managed the contractors and their methods, chose a high-level language, and maintained conceptual integrity.

NASA awarded IBM Corporation the first independent Shuttle software contract on March 10, 1973. IBM and Rockwell International had worked together during the period of competition for the orbiter contract⁸¹. Rockwell bid on the entire aerospacecraft, intending to subcontract the computer hardware and software to IBM. But to Rockwell's dismay, NASA decided to separate the software contract from the orbiter contract. As a result, Rockwell still subcontracted with IBM for the computers, but IBM had a separate software contract monitored closely by the Spacecraft Software Division of the Johnson Space Center. There are several reasons why this division of labor occurred. Since software does not weigh anything in and of itself, it is used to overcome hardware problems that would require extra systems and components (such as a mechanical control system)⁸². Thus software is in many ways the most critical component of the Shuttle, as it ties the other components together. Its importance to the overall program alone justified a separate contract, since it made the contractor directly accountable to NASA. Moreover, during the operations phase, software underwent the most changes, the hardware being essentially fixed⁸³. As time went on, Rockwell's responsibilities as

prime hardware contractor were phased out, and the shuttles were turned over to an operations group. In late 1983, Lockheed Corporation, not Rockwell, won the competition for the operations contract. By keeping the software contract separate, NASA could develop the code on a continuing basis. There is a considerable difference between changing maintenance mechanics on an existing hardware system and changing software companies on a not-yet-perfect system because to date the relationships between components in software are much harder to define than those in hardware. Personnel experienced with a specific software system are the best people to maintain it. Lastly, Christopher Kraft of Johnson Space Center and George Low of NASA Headquarters, both highly influential in the manned spacecraft program during the early 1970's, felt that Johnson had the software management expertise to handle the contract directly⁸⁴.

One of the lessons learned from monitoring Draper Laboratory in the Apollo era was that by having the software development at a remote site (like Cambridge), the synergism of informally exchanged ideas is lost; sometimes it took 3 to 4 weeks for new concepts to filter over⁸⁵. IBM had a building and several hundred personnel near Johnson because of its Mission Control Center contracts. When IBM won the Shuttle contract, it simply increased its local force.

The closeness of IBM to Johnson Space Center also facilitated the ability of NASA to manage the project. The first chief of the Shuttle's software, Richard Parten, observed that the experience of NASA managers made a significant contribution to the success of the programming effort⁸⁶. Although IBM was a giant in the data processing industry, a pioneer in real-time systems, and capable of putting very bright people on a project, the company had little direct experience with avionics software. As a consequence, Rockwell had to supply a lot of information relating to flight control. Conversely, even though Rockwell projects used computers, software development on the scale needed for the Shuttle was outside its experience. NASA Shuttle managers provided the initial requirements for the software and facilitated the exchange of information between the principal contractors. This situation was similar to that during the 1960s when NASA had the best rendezvous calculations people in the world and had to contribute that expertise to IBM during the Gemini software development. Furthermore, the lessons of Apollo inspired the NASA managers to push IBM for quality at every point⁸⁷.

The choice of a high-level language for doing the majority of the coding was important because, as Parten noted, with all the changes, "we'd still be trying to get the thing off the ground if we'd used assembly language"⁸⁸. Programs written in high-level languages are far easier to modify. Most of the operating system software, which is rarely changed, is in assembler, but all applications software and some of the interfaces and redundancy management code is in HAL/S⁸⁹.

Although the decision to program in a high-level language meant that a large amount of support software and development tools had to be written, the high-level language nonetheless proved advantageous, especially since it has specific statements created for real-time programming.

Defining the Shuttle Software

In the end, the success of the Shuttle's software development was due to the conceptual integrity established by using rigorously maintained requirements documents. The requirements phase is the beginning of the software life cycle, when the actual functions, goals, and user interfaces of the eventual software are determined in full detail. If a team of a thousand workers was asked to set software requirements, chaos would result⁹⁰. On the other hand, if few do the requirements but many can alter them later, then chaos would reign again. The strategy of using a few minds to create the software architecture and interfaces and then ensuring that their ideas and theirs alone are implemented, is termed "maintaining conceptual integrity," which is well explained in Frederick C. Brooks' *The Mythical Man-Month*⁹¹. As for other possible solutions, Parten says, "the only right answer is the one you pick and make to work"⁹².

Shuttle requirements documents were arranged in three Levels: A, B, and C, the first two written by Johnson Space Center engineers. John R. Garman prepared the Level A document, which is comprised of a comprehensive description of the operating system, applications programs, keyboards, displays, and other components of the software system and its interfaces. William Sullivan wrote the guidance, navigation and control requirements, and John Aaron, the system management and payload specifications of Level B. They were assisted by James Broadfoot and Robert Ernull⁹³. Level B requirements are different in that they are more detailed in terms of what functions are executed when and what parameters are needed⁹⁴. The Level Bs also define what information is to be kept in COMPOOLS, which are HAL/S structures for maintaining common data accessed by different tasks⁹⁵. The Level C requirements were more of a design document, forming a set with Level B requirements, since each end item at Level C must be traceable to a Level B requirement⁹⁶. Rockwell International was responsible for the development of the Level C requirements as, technically, this is where the contractors take over from the customer, NASA, in developing the software.

Early in the program, however, Draper Laboratory had significant influence on the software and hardware systems for the Shuttle. Draper was retained as a consultant by NASA and contributed two

key items to the software development process. The first was a document that "taught" NASA and other contractors how to write requirements for software, how to develop test plans, and how to use functional flow diagrams, among other tools⁹⁷. It seems ironic that Draper was instructing NASA and IBM on such things considering its difficulties in the mid-1960s with the development of the Apollo flight software. It was likely those difficult experiences that helped motivate the MIT engineers to seriously study software techniques and to become, within a very short time, one of the leading centers of software engineering theory. The Draper tutorial included the concept of highly modular software, software that could be "plugged into" the main circuits of the Shuttle. This concept, an application of the idea of interchangeable parts to software, is used in many software systems today, one example being the UNIX^{***} operating system developed at Bell Laboratories in the 1970s, under which single function software tools can be combined to perform a large variety of functions.

Draper's second contribution was the actual writing of some early Level C requirements as a model⁹⁸. This version of the Level C documents contained the same components as in the later versions delivered by Rockwell to IBM for coding. Rockwell's editions, however, were much more detailed and complete, reflecting their practical, rather than theoretical, purpose and have been an irritation for IBM. IBM and NASA managers suspect that Rockwell, miffed when the software contract was taken away from them, may have delivered incredibly precise and detailed specifications to the software contractor. These include descriptions of flight events for each major portion of the software, a structure chart of tasks to be done by the software during that major segment, a functional data flowchart, and, for each module, its name, calculations, and operations to be performed, and input and output lists of parameters, the latter already named and accompanied by a short definition, source, precision, and what units each are in. This is why one NASA manager said that "you can't see the forest for the trees" in Level C, oriented as it is to the production of individual modules⁹⁹. One IBM engineer claimed that Rockwell went "way too far" in the Level C documents, that they told IBM too much about *how* to do things rather than just *what* to do¹⁰⁰. He further claimed that the early portion of the Shuttle development was "underengineered" and that Rockwell and Draper included some requirements that were not passed on by NASA. Parten, though, said that all requirements documents were subject to regular review by joint teams from NASA and Rockwell¹⁰¹.

The impression one gains from documents and interviews is that both Rockwell and IBM fell victim to the "not invented here"

***UNIX is a trademark of AT&T.

syndrome: If we didn't do it, it wasn't done right. For example, Rockwell delivered the ascent requirements, and IBM coded them to the letter, thereby exceeding the available memory by two and a third times and demonstrating that the requirements for ascent were excessive. Rockwell, in return, argued for 2 years about the nature of the operating system, calling for a strict time-sliced system, which allocates predefined periods of time for the execution of each task and then suspends tasks unfinished in that time period and moves on to the next one. The system thus cycles through all scheduled tasks in a fixed period of time, working on each in turn. Rockwell's original proposal was for a 40-millisecond cycle with synchronization points at the end of each¹⁰². IBM, at NASA's urging, countered with a priority-interrupt-driven system similar to the one on Apollo. Rockwell, experienced with time-slice systems, fought this from 1973 to 1975, convinced it would never work¹⁰³.

The requirements specifications for the Shuttle eventually contained in their three levels what is in both the specification and design stage of the software life cycle. In this sense, they represent a fairly complete picture of the software at an early date. This level of detail at least permitted NASA and its contractors to have a starting point in the development process. IBM constantly points to the number of changes and alterations as a continuing problem, partially ameliorated by implementing the most mature requirements first¹⁰⁴. Without the attempt to provide detail at an early date, IBM would not have had any mature requirements when the time came to code. Even now, requirements are being changed to reflect the actual software, so they continue to be in a process of maturation. But early development of specifications became the means by which NASA could enforce conceptual integrity in the shuttle software.

Architecture of the Primary Avionics Software System

The Primary Avionics Software System, or PASS, is the software that runs in all the Shuttle's four primary computers. PASS is divided into two parts: system software and applications software. The system software is the Flight Computer Operating System (FCOS), the user interface programming, and the system control programs, whereas the applications software is divided into guidance, navigation and control, orbiter systems management, payload and checkout programs. Further divisions are explained in Box 4-3.

The most critical part of the system software is the FCOS. NASA, Rockwell, and IBM solved most of the grand conceptual problems, such as the nature of the operating system and the redundancy management scheme, by 1975. The first task was to convert the FCOS from the proposed 40-millisecond loop operating system to a priority-

Box 4-3: Structure of PASS Applications Software

The PASS guidance and navigation software is divided into major functions, dictated by mission phases, the most obvious of which are preflight, ascent, on-orbit, and descent. The requirements state that these major functions be called OPS, or operational sequences. (e.g., OPS-1 is ascent; OPS-3, descent.) Within the OPS are major modes. In OPS-1, the first-stage burn, second-stage burn, first orbital insertion burn, second orbital insertion burn, and the initial on-orbit coast are major modes; transition between major modes is automatic. Since the total mission software exceeds the capacity of the memory, OPS transitions are normally initiated by the crew and require the OPS to be loaded from the MMU. This caused considerable management concern over the preservation of data, such as the state vector, needed in more than one OPS¹⁰⁵. NASA's solution is to keep common data in a major function base, which resides in memory continuously and is not overlaid by new OPS being read into the computers.

Within each OPS, there are special functions (SPECs) and display functions (DISPs). These are available to the crew as a supplement to the functions being performed by the current OPS. For example, the descent software incorporates a SPEC display showing the horizontal situation as a supplement to the OPS display showing the vertical situation. This SPEC is obviously not available in the on-orbit OPS. A DISP for the on-orbit OPS may show fuel cell output levels, fuel reserves in the orbital maneuvering system, and other such information. SPECs usually contain items that can be selected by the crew for execution. DISPs are just what their name means, displays and not action items. Since SPECs and DISPs have lower priority than OPS, when a big OPS is in memory they have to be kept on the tape and rolled in when requested¹⁰⁶. The actual format of the SPECs, DISPs, OPS displays, and the software that interprets crew entries on the keyboard is in the user interface portion of the system software.

driven system¹⁰⁷. Priority interrupt systems are superior to time-slice systems because they degrade gracefully when overloaded¹⁰⁸. In a time-slice system, if the tasks scheduled in the current cycle get bogged down by excessive I/O operations, they tend to slow down the total time of execution of processes. IBM's version of the FCOS actually has cycles, but they are similar to the ones in the Skylab system described in the previous chapter. The minor cycle is the high-frequency cycle; tasks within it are scheduled every 40 milliseconds. Typical tasks in this cycle are those related to active flight control in the atmosphere. The major cycle is 960 milliseconds, and many monitoring and system management tasks are scheduled at that frequency¹⁰⁹. If a process is still running when its time to restart

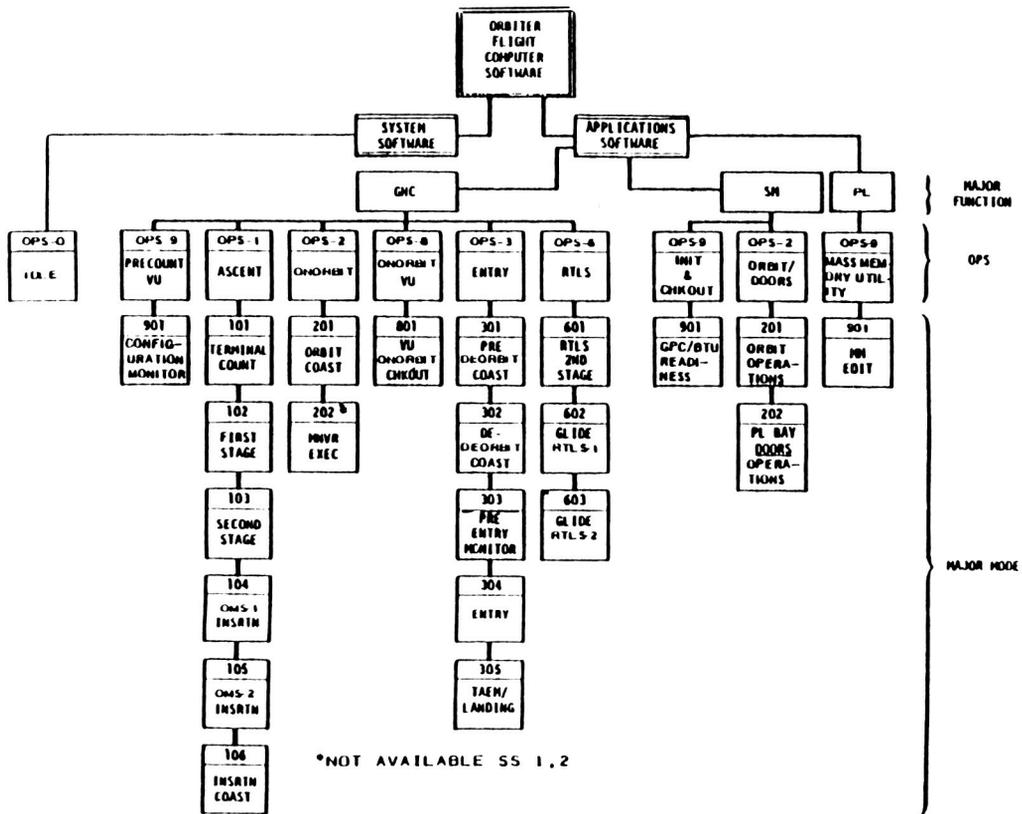


Figure 4-6. A block diagram of the Shuttle flight computer software architecture. (From NASA, *Data Processing System Workbook*)

comes up due to excessive I/O or because it was interrupted, it cancels its next cycle and finishes up¹¹⁰. If a higher priority process is called when another process is running, then the current process is interrupted and a program status word (PSW) containing such items as the address of the next instruction to be executed is stored until the interruption is satisfied. The last instruction of an interrupt is to restore the old PSW as the current PSW so that the interrupted process can continue¹¹¹. The ability to cancel processes and to interrupt them asynchronously provides flexibility that a strict time-slice system does not.

A key requirement of the FCOS is to handle the real-time statements in the HAL/S language. The most important of these are SCHEDULE, which establishes and controls the frequency of execution of processes; TERMINATE and CANCEL, which are the opposite of SCHEDULE; and WAIT, which conditionally suspends execution¹¹². The method of implementing these statements is con-

trolled by a separate interface control document¹¹³. SCHEDULE is generally programmed at the beginning of each operational sequence to set up which tasks are to be done in that software segment and how often they are to be done. The syntax of SCHEDULE permits the programmer to assign a frequency and priority to each task. TERMINATE and CANCEL are used at the end of software phases or to stop an unneeded process while others continue. For example, after the solid rocket boosters burn out and separate, tasks monitoring them can cease while tasks monitoring the main engines continue to run. WAIT, although handy, is avoided by IBM because of the possibility of the software being "hung up" while waiting for the I/O or other condition required to continue the process¹¹⁴. This is called a race condition or "deadly embrace" and is the bane of all shared resource computer operating systems.

The FCOS and displays occupy 35K of memory at all times¹¹⁵. Add the major function base and other resident items, and about 60K of the 106K of core remains available for the applications programs. Of the required applications programs, ascent and descent proved the most troublesome. Fully 75% of the software effort went into those two programs¹¹⁶. After the first attempts at preparing the ascent software resulted in a 140K load, serious code reduction began. By 1978, IBM reduced the size of the ascent program to 116K, but NASA Headquarters demanded it be further knocked down to 80K¹¹⁷. The lowest it ever got was 98,840 words (including the system software), but its size has since crept back up to nearly the full capacity of the memory. IBM accomplished the reduction by moving functions that could wait until later operational sequences¹¹⁸. The actual figures for the test flight series programs are in Table 4-1¹¹⁹. The total size of the flight test software was 500,000 words of code. Producing it and modifying it for later missions required the development of a complete production facility.

TABLE 4-1: Sizes of Software Loads in PASS

NAME	K WORDS
Preflight initialization	72.4
Preflight checkout	81.4
Ascent and abort	105.2
On-orbit	83.1
On-orbit checkout	80.3
On-orbit system management	84.1
Entry	101.1
Mass memory utility	70.1

Note: Payload and rendezvous software was added later during the operations phase.

Implementing PASS

NASA planned that PASS would be a continuing development process. After the first flight programs were produced, new functions needed to be added and adapted to changing payload and mission requirements. For instance, over 50% of PASS modules changed during the first 12 flights in response to requested enhancements¹²⁰. To do this work, NASA established a Software Development Laboratory at Johnson Space Center in 1972 to prepare for the implementation of the Shuttle programs and to make the software tools needed for efficient coding and maintenance. The Laboratory evolved into the Software Production Facility (SPF) in which the software development is carried on in the operations era. Both the facilities were equipped and managed by NASA but used largely by contractors.

The concept of a facility dedicated to the production of onboard software surfaced in a Rand Corporation memo in early 1970¹²¹. The memo summarized a study of software requirements for Air Force space missions during the decade of the 1970s. One reason for a government-owned and operated software factory was that it would be easier to establish and maintain security. Most modules developed for

the Shuttle, such as the general flight control software and memory displays, would be unclassified. However, Department of Defense (DoD) payloads require system management and payload management software, plus occasional special maneuvering modules. These were expected to be classified. Also, if the software maintenance contract moved from the original prime contractor to some different operations contractor, it would be considerably simpler to accomplish the transfer if the software library and development computers were government owned and on government property. Lastly, having such close control over existing software and new development would eliminate some of the problems in communication, verification, and maintenance encountered in the three previous manned programs.

Developing the SPF turned out to be as large a task as developing the flight software itself. During the mid-1970s, IBM had as many people doing software for the development lab as they had working on PASS¹²². The ultimate purpose of the facility is to provide a programming team with sufficient tools to prepare a software load for a flight. This software load is what is put on to the MMU tape that is flown on the spacecraft. In the operations era of the 1980s, over 1,000 compiled modules are available. These are fully tested, and often previously used, versions of tasks such as main engine throttling, memory modification, and screen displays that rarely change from flight to flight. New, mission-specific modules for payloads or rendezvous maneuvers are developed and tested using the SPF's programming tools, which themselves represent more than a million lines of code¹²³. The selection of existing modules and the new modules are then combined into a flight load that is subject to further testing. NASA achieved the goal of having such an efficient software production system through an 8-year development process when the SPF was still the Laboratory.

In 1972, NASA studied what sort of equipment would be required for the facility to function properly. Large mainframe computers compatible with the AP-101 instruction set were a must. Five IBM 360/75 computers, released from Apollo support functions, were available¹²⁴. These were the development machines until January of 1982¹²⁵. Another requirement was for actual flight equipment on which to test developed modules. Three AP-101 computers with associated display electronics units connected to the 360s with a flight equipment interface device (FEID) especially developed for the purpose. Other needed components, such as a 6-degree-of-freedom flight simulator, were implemented in software¹²⁶. The resulting group of equipment is capable of testing the flight software by interpreting instructions, simulating functions, and running it in the actual flight hardware¹²⁷.

In the late 1970s, NASA realized that more powerful computers were needed as the transition was made from development to operations. The 360s filled up, so NASA considered the Shuttle Mission

Simulator(SMS), the Shuttle Avionics Instrumentation Lab (SAIL), and the Shuttle Data Processing Center's computers as supplementary development sites, but this idea was rejected because they were all too busy doing their primary functions¹²⁸. In 1981, the Facility added two new IBM 3033N computers, each with 16 million bytes of primary memory. The SPF then consisted of those mainframes, the three AP-101 computers and the interface devices for each, 20 magnetic tape drives, six line printers, 66 million bytes of drum memory, 23.4 billion bytes of disk memory, and 105 terminals¹²⁹. NASA accomplished rehosting the development software to the 3033s from the 360s during the last quarter of 1981. Even this very large computer center was not enough. Plans at the time projected on-line primary memory to grow to 100 million bytes¹³⁰, disk storage to 160 billion bytes¹³¹, and two more interface units, display units, and AP-101s to handle the growing DOD business¹³². Additionally, terminals connected directly to the SPF are in Cambridge, Massachusetts, and at Goddard Space Flight Center, Marshall Space Flight Center, Kennedy Space Center, and Rockwell International in Downey, California¹³³.

Future plans for the SPF included incorporating backup system software development, then done at Rockwell, and introducing more automation. NASA managers who experienced both Apollo and the Shuttle realize that the operations software preparation is not enough to keep the brightest minds sufficiently occupied. Only a new project can do that. Therefore, the challenge facing NASA is to automate the SPF, use more existing modules, and free people to work on other tasks. Unfortunately, the Shuttle software still has bugs, some of which are no fault of the flight software developers, but rather because all the tools used in the SPF are not yet mature. One example is the compiler for HAL/S. Just days before the STS-7 flight, in June, 1983, an IBM employee discovered that the latest release of the compiler had a bug in it. A quick check revealed that over 200 flight modules had been modified and recompiled using it. All of those had to be checked for errors before the flight could go. Such problems will continue until the basic flight modules and development tools are no longer constantly subject to change. In the meantime, the accuracy of the Shuttle software is dependent on the stringent testing program conducted by IBM and NASA before each flight.

Verification and Change Management of the Shuttle Software

IBM established a separate line organization for the verification of the Shuttle software. IBM's overall Shuttle manager has two managers reporting to him, one for design and development, and one for verification and field operations. The verification group has just

less than half the members of the development group and uses 35% of the software budget¹³⁴. There are no managerial or personnel ties to the development group, so the test team can adopt an "adversary relationship" with the development team. The verifiers simply assume that the software is untested when received¹³⁵. In addition, the test team can also attempt to prove that the requirements documents are wrong in cases where the software becomes unworkable. This enables them to act as the "conscience" of the entire project¹³⁶.

IBM began planning for the software verification while the requirements were being completed. By starting verification activity as the software took shape, the test group could plan its strategy and begin to write its own books. The verification documentation consists of test specifications and test procedures including the actual inputs to be used and the outputs expected, even to the detail of showing the content of the CRT screens at various points in the test¹³⁷. The software for the first flight had to survive 1,020 of these tests¹³⁸. Future flight loads could reuse many of the test cases, but the preparation of new ones is a continuing activity to adjust to changes in the software and payloads, each of which must be handled in an orderly manner.

Suggestions for changes to improve the system are unusually welcome. Anyone, astronaut, flight trainer, IBM programmer, or NASA manager, can submit a change request¹³⁹. NASA and IBM were processing such requests at the rate of 20 per week in 1981¹⁴⁰. Even as late as 1983 IBM kept 30 to 40 people on requirements analysis, or the evaluation of requests for enhancements¹⁴¹. NASA has a corresponding change evaluation board. Early in the program, it was chaired by Howard W. Tindall, the Apollo software manager, who by then was head of the Data Systems and Analysis Directorate. This turned out to be a mistake, as he had conflicting interests¹⁴². The change control board moved to the Shuttle program office. Due to the careful review of changes, it takes an average of 2 years for a new requirement to get implemented, tested, and into the field¹⁴³. Generally, requests for extra functions that would push out current software due to memory restrictions are turned down¹⁴⁴.

Box 4-4: How IBM Verifies the Shuttle Flight Software

The Shuttle software verification process actually begins before the test group gets the software, in the sense that the development organization conducts internal code reviews and unit tests of individual modules and then integration tests of groups of modules as they are assembled into a software load. There are two levels of code inspection, or "eyeballing" the software looking for logic errors. One level of inspection is by the coders themselves and their peer reviewers. The second level is done by the outside verification team. This activity resulted in over 50% of the discrepancy reports (failures of the software to meet the specification) filed against the software, a percentage similar to the Apollo experience and reinforcing the value of the idea¹⁴⁵. When the software is assembled, it is subject to the First Article Configuration Inspection (FACI), where it is reviewed as a complete unit for the first time. It then passes to the outside verification group.

Because of the nature of the software as it is delivered, the verification team concentrates on proving that it meets the customer's requirements and that it functions at an acceptable level of performance. Consistent with the concept that the software is assumed untested, the verification group can go into as much detail as time and cost allow. Primarily, the test group concentrates on single software loads, such as ascent, on-orbit, and so forth¹⁴⁶. To facilitate this, it is divided into teams that specialize in the operating system and detail, or functional verification; teams that work on guidance, navigation, and control; and teams that certify system performance. These groups have access to the software in the SPF, which thus doubles as a site for both development and testing. Using tools available in the SPF, the verification teams can use the real flight computers for their tests (the preferred method). The testers can freeze the execution of software on those machines in order to check intermediate results, alter memory, and even get a log of what commands resulted in response to what inputs¹⁴⁷.

After the verification group has passed the software, it is given an official Configuration Inspection and turned over to NASA. At that point NASA assumes configuration control, and any changes must be approved through Agency channels. Even though NASA then has the software, IBM is not finished with it¹⁴⁸.

Box 4-4 (Continued)

The software is usually installed in the SAIL for prelaunch, ascent, and abort simulations, the Flight Simulation Lab (FSL) in Downey for orbit, de-orbit, and entry simulations, and the SMS for crew training. Although these installations are not part of the preplanned verification process, the discrepancies noted by the users of the software in the roughly 6 months before launch help complete the testing in a real environment. Due to the nature of real-time computer systems, however, the software can *never* be fully certified, and both IBM and NASA are aware of this¹⁴⁹. There are simply too many interfaces and too many opportunities for asynchronous input and output.

Discrepancy reports cause changes in software to make it match the requirements. Early in the program, the software found its way into the simulators after less verification because simulators depend on software just to be turned on. At that time, the majority of the discrepancy reports were from the field installations. Later, the majority turned up in the SPF¹⁵⁰. All discrepancy reports are formally disposed of, either by appropriate fixes to the software, or by waiver. Richard Parten said, "Sometimes it is better to put in an 'OPS Note' or waiver than to fix (the software). We are dependent on smart pilots"¹⁵¹. If the discrepancy is noted too close to a flight, if it requires too much expense to fix, it can be waived *if* there is no immediate danger to crew safety. Each Flight Data File carried on board lists the most important current exceptions of which the crew must be aware. By STS-7 in June of 1983, over 200 pages of such exceptions and their descriptions existed¹⁵². Some will never be fixed, but the majority were addressed during the Shuttle launch hiatus following the 51L accident in January 1986.

So, despite the well-planned and well-manned verification effort, software bugs exist. Part of the reason is the complexity of the real-time system, and part is because, as one IBM manager said, "we didn't do it up front enough," the "it" being thinking through the program logic and verification schemes¹⁵³. Aware that effort expended at the early part of a project on quality would be much cheaper and simpler than trying to put quality in toward the end, IBM and NASA tried to do much more at the beginning of the Shuttle software development than in any previous effort, but it still was not enough to ensure perfection.

Box 4-5: The Nature of the Backup Flight System

The Backup Flight System consists of a single computer and a software load that contains sufficient functions to handle ascent to orbit, selected aborts during ascent, and descent from orbit to a landing site. In the interest of avoiding a generic software failure, NASA kept its development separate from PASS. An engineering directorate, not the on-board software division, managed the software contract for the backup, won by Rockwell¹⁵⁴.

The major functional difference between PASS and the backup is that the latter uses a time-slice operating system rather than the asynchronous priority-driven system of PASS¹⁵⁵. This is consistent with Rockwell's opinion on how that system was to be designed. Ironically, since the backup must listen in on PASS operations so as to be ready for instant takeover, PASS had to be modified to make it more synchronous¹⁵⁶. Sixty engineers were still working on the Backup Flight System software as late as 1983¹⁵⁷.

USING THE SHUTTLE DPS

With the level of complexity present in the hardware and software just described, it is not surprising that the crew interfaces to those components are also complex. The complexity is caused not so much by the design of the interfaces but by the limited amount of memory available for graphics displays, automatic reconfiguring of the computers, and other utilities to make the system more cooperative and simpler for the users. There is some difference between the way the users of the system perceive the DPS and the way the designers, both NASA and IBM, perceive it. Some astronauts and trainers are openly critical. John Young, the Chief Astronaut in the early 1980s, complained, "What we have in the Shuttle is a disaster. We are not making computers do what we want"¹⁵⁸. Flight trainer Frank Hughes also remarked that "the PASS doesn't do anything for us"¹⁵⁹, noting that such practical items as the time from loss of ground-station signals and acquisition of new stations is not part of the primary software. Both said, "We end up working for the computer, rather than the computer working for us." This comment is something reminiscent of Apollo days, when the number of keystrokes needed to fly a mission was a concern. John Aaron, one of NASA's designers of PASS interfaces and later chief of spacecraft software, said that the Apollo experience influenced Shuttle designers to avoid excessive pilot interaction with the computers. Even so, he found the "crew

ORIGINAL PAGE IS
OF POOR QUALITY.



Figure 4-7. The forward flight deck of a Shuttle, with the three CRT screens and twin keyboards visible in the center. (NASA photo S80-35133)

interfaces...more confusing and complex than I thought they would be"¹⁶⁰. One statistic that supports his perception is that the 13,000 keystrokes used in a week-long lunar mission are matched by a Shuttle crew in a 58-hour flight¹⁶¹.

Another aspect of the "working for the computer" problem is that steps normally done by computers using preprogrammed functions are done manually on the Shuttle. The reconfiguration of PASS from the ascent redundant set to the on-orbit groupings has to be done by the crew, a process taking several minutes and needing to be reversed before descent. Aaron acknowledges that the computer interfaces are too close to machine level, but points out that management "would not buy" simple automatic reconfiguration schemes. Even if they had, there is no computer memory to store such utilities.

Tied to the computer memory problem is the fact that many func-

tions have to be displayed together on a screen because of the fact that such displays are "memory hungry." As a result, many screens are so crowded that reading them quickly is difficult, the process being further affected by the blandness and primitive nature of any graphics available. Astronaut Vance Brand claimed that after initial confusion, several hours with simulators makes things easier to find; he makes a point of checking his entries on the input line before pressing the execute key¹⁶². Young does that as well, but for additional reasons: The keyboard buffer is so small that entering data too quickly causes some to be lost, and he wants to check whether he is accessing the right screen display with the proper keyboard. This latter concern arises because there are only two keyboards for the three forward CRTs. Since both keyboards can be assigned to the same screen, two CRTs may not be currently set up for input. Even if the two keyboards are assigned to different screens, one CRT is left without capability for immediate crew input. Astronaut Henry Hartsfield termed this situation "prone to error"¹⁶³.

Since flying the Shuttle is in many ways flying the Shuttle computers (they provide the active flight control, guidance and navigation, systems management, and payload functions), the astronauts are interested in making suggestions for improving the computer system. Most revolve around more automation, more user friendliness, more color, better graphics, and more functions, such as adding a return-to-launch-site (RTLS) abort with two engines out in addition to the present version with only one engine out¹⁶⁴. Each of these enhancements is tied to increasing memory. IBM proposed a new version of the Shuttle computers with 256K of memory and software compatibility with the existing system. Johnson Space Center began testing these AP-101F computers in 1985, with the first operational use projected for the resumption of Shuttle missions in 1988.

In the meantime, the astronauts themselves pioneered efforts to use small computers to add functions and back up the primary systems. Early flights used a Hewlett-Packard HP-41C programmable calculator to determine ground-station availability, as well as carry a limited version of the calculations for time-to-retrofire. Beginning with STS-9 in December, 1983, a Grid Systems Compass portable microcomputer with graphics capabilities was carried to display ground stations and to provide functions impractical on the primary computers. Mission Specialist Terry Hart, responsible for programming the HP-41Cs, said that placing the mission documentation on the computer was also being considered.¹⁶⁵

THE SPACE SHUTTLE MAIN ENGINE CONTROLLERS

Among the many special-purpose computers on the Shuttle, the

Box 4-6: Using the Shuttle's Keyboards

The Shuttle's keyboards are different from those found on Gemini and Apollo because they are hexadecimal, or base 16, rather than decimal, so that memory locations can be altered by hex entries from the keyboard. A single hex digit represents 4 bits, so just four digits can fill a half-word memory location. The other keys perform specialized functions. The most often used are

- **ITEM:** This selects a specific function displayed on a CRT. For example, if the astronaut wishes to perform a function numbered 32 on the screen, he or she presses ITEM, 3, 2, EXEC.
- **OPS:** This, plus a four-digit number, selects the operational sequence and major mode desired by the crew. For instance, to choose the first major mode of the ascent software, OPS, 1, 1, 0, 1, and PRO is entered.
- **SPEC:** This key, plus appropriate digits and PRO, selects a specialist function or display function screen. Each OPS has associated with it a number of primary screens that reflect what is happening in the software. The ascent program has a vertical path graphic, for instance. Additionally, special functions can be called from SPEC displays that are overlaid on the primary screens when called. On-orbit, and several other OPS, have a "GPC Memory" display that can be used to read or write to individual memory locations. It cannot be called from either the ascent or descent OPS. Display function screens are just that: used to show various data such as fuel cell levels, but with no crew functions. To return to the primary screen that was on the CRT before the SPEC or DISP call, the RESUME key is used.
- **CLEAR:** Each time this key is depressed, one character is deleted from the input line on the CRT accessed. This enables an astronaut to erase an error if it is caught before EXEC or PRO is depressed.
- **'+' :** This sign can be used as a delimiter around numeric data or between a series of function selections.

main engine controllers stand out as a clear "first" in space technology. The Shuttle's three main liquid-propellant engines are the most complex and "hottest" rockets ever built. The complexity is tied to the mission requirements, which state that they be throttleable, a common characteristic of internal combustion engines and turbojets, but rare in the rocket business. They run "hotter" than any other rocket engine because at any given moment they are closer to destroying themselves than their predecessors. Previous engines were overbuilt in the sense that they were designed to burn at full thrust through their entire

FAULT SUMM	SYS SUMM	MSG RESET	ACK
GPC/ CRT	A	B	C
I/O RESET	D	E	F
ITEM	1	2	3
EXEC	4	5	6
OPS	7	8	9
SPEC	-	0	+
RESUME	CLEAR	•	PRO

Figure 4-8. Keyboard layout of the Shuttle computer system. (From NASA, *Data Processing System Workbook*)

lifetime of a few minutes with no chance that the continuous explosion of fuel and oxidizer would get out of control. To ensure this, engineers designed combustion chambers and cooling systems better than optimum, with the result that the engines weighed more than less-protected designs, thus reducing performance. Engineers also set fluid mixtures and flow rates by mechanical means at preset levels, and levels could not be changed to gain greater performance. The Shuttle engines can adjust flow levels, can sense how close to exploding they are, and can respond in such a way as to maintain maximum performance at all times. Neither the throttleability or the performance enhancements could be accomplished without a digital computer as a control device.

In 1972, NASA chose Rocketdyne as the engine contractor, with

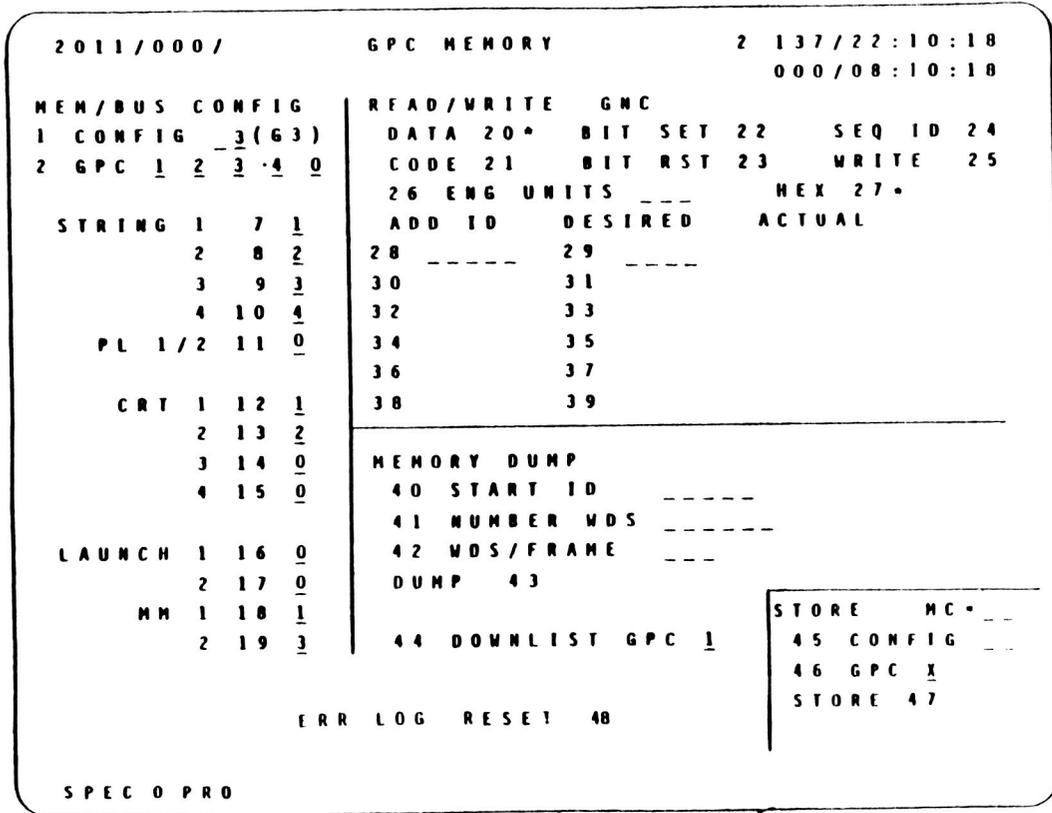


Figure 4-9. A typical display of the Primary Avionics Software System. (From NASA, *Data Processing System Workbook*)

Marshall Space Flight Center responsible for monitoring the design, production, and testing of the engines. Rocketdyne conducted a preliminary study of the engine control problem and recommended that a distributed approach be used for the solution¹⁶⁶. By placing controllers at the engines themselves, complex interfaces between the engine and vehicle could be avoided. Also, the high data rates needed for active control are best handled with a dedicated computer. Both Marshall and Rocketdyne agreed that a digital computer controller was better than an analog controller for three reasons. First, software allows for greater flexibility. Inasmuch as the control concepts for the engines were far from settled in 1972, NASA considered the ease of modifying software versus hardware a very important advantage¹⁶⁷.

Second, the digital system could respond faster. And third, the failure detection function could be simpler¹⁶⁸. Basically, the computer has only two functions: to control the engine and to do self tests.

The concept of fail operational/fail-safe is preserved with the engine controllers because each engine has a dual redundant computer attached to it. Failure of the first computer does not impede operational capability, as the second takes over instantly. Failure of the second computer causes a graceful shutdown of the affected engine¹⁶⁹. Loss of an engine does not cause any immediate danger to a Shuttle crew, as demonstrated in a 1985 mission that lost an engine and still achieved orbit. If engine loss occurs early in a flight, the mission can be aborted through a RTLS maneuver that causes the spacecraft essentially to turn around and fly back to a runway near the launch pad. Slightly later aborts may lead to a landing in Europe for Kennedy Space Center launches. If the engine fails near orbit it may be possible to achieve an orbit and then modify it using the orbital maneuvering system engines.

Controller Software and Redundancy Management

As with the main computers on the Shuttle, software is an important part of the engine controller system. NASA managers adopted a strict software engineering approach to the controller code. Marshall's Walter Mitchell said, "We try to treat the software exactly like the hardware"¹⁷⁰. In fact, the controller software is more closely married to engine hardware than in other systems under computer control. The controllers operate as a real-time system with a fixed cyclic execution schedule. Each major cycle has four 5-millisecond minor cycles for a total of 20 milliseconds. This is a high frequency, necessitated by the requirement to control a rapidly changing engine environment. Each major cycle starts and ends with a self test. It proceeds through engine control tasks, input sensor data reads, engine limit monitoring tasks, output, another round of input sensor data, a check of internal voltage, and then the second self test¹⁷¹. Some free time is built into the cycle to avoid overruns into the next cycle. So that the controller will not waste processing time handling data requests from the primary avionics system, direct memory access of engine component data can be made by the primary¹⁷².

As with the primary computers in the Shuttle, the memory of the controller cannot hold all the software originally designed for it. A set of preflight checkout programs have to be stored on the MMU and rolled in during the countdown. At T-30 hours, the engines are activated and the flight software load is read from the mass memory¹⁷³. Even this way, fewer than 500 words of the 16K are unused¹⁷⁴.

ORIGINAL PAGE IS
OF POOR QUALITY

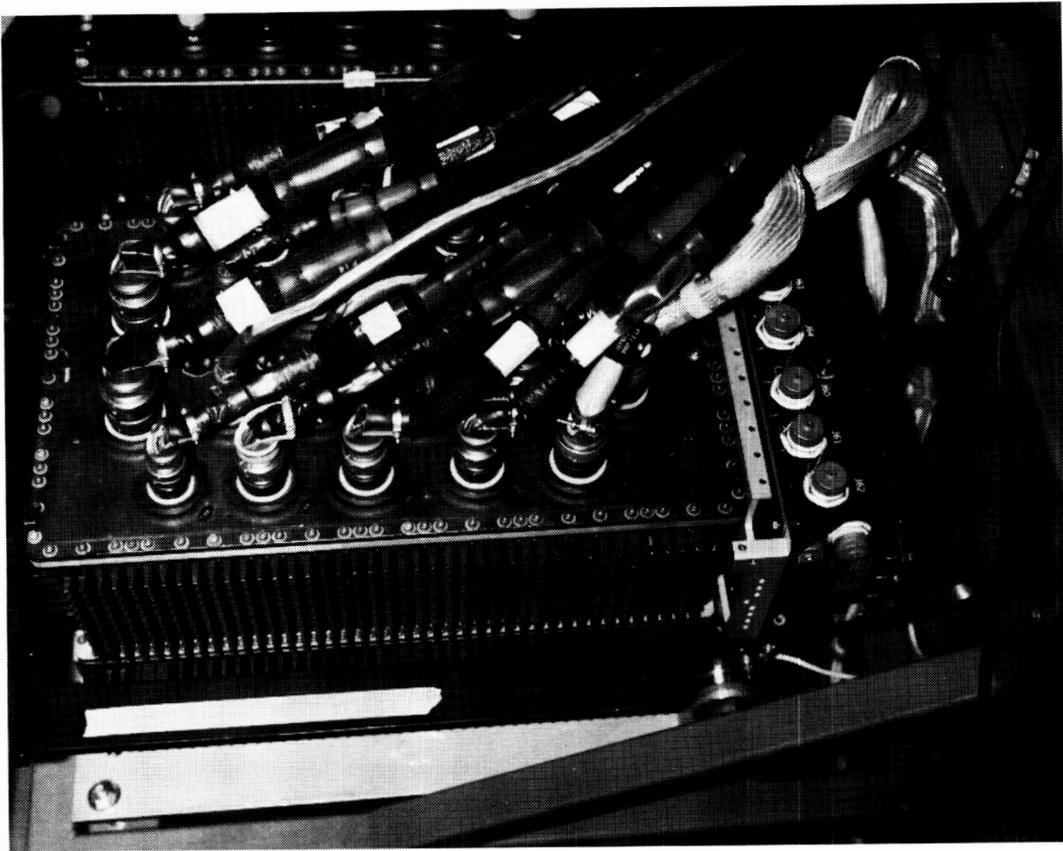


Figure 4-10. A Shuttle Main Engine Controller mounted in an engineering simulator at the Marshall Space Flight Center. (NASA photo)

Although redundant, the controllers are not synchronized like the primary computers. Marshall Space Flight Center studied active synchronization, but the additional hardware and software overhead seemed too expensive¹⁷⁵. The present system of redundancy management most closely resembles that used by the Skylab computers. Since Marshall also had responsibility for those computers and was making the decision about the controllers at the same time Skylab was operating, some influence from the ATMDC experience is possible. Two watchdog timers are used to flag failures. One is incremented by the real-time clock and the other, by a clock in the output electronics. Each has to be reset by the software. If the timers run out, the software or critical hardware of the computer responsible for resetting them is assumed failed and the Channel B computer takes over at that point. The timeout is set at 18 milliseconds, so the engine involved is "uncontrolled" by a failed computer for less than a major cycle before the redundant computer takes over¹⁷⁶.

Box 4-7: Shuttle Engine Controller Hardware

The computer chosen for the engine controllers is the Honeywell HDC-601. The Air Force was using it in 1972 when the choice was made, so operational experience existed. Additionally, the machine was software compatible with the DDP 516, a ground-based Honeywell minicomputer, so a development environment was available. Honeywell built parts of the controller in St. Petersburg, Florida and shipped those to the main plant in Minneapolis for final assembly; within a couple of years, all the construction tasks moved to St. Petersburg. By mid-1983, Honeywell completed 29 of the computers¹⁷⁷.

The HDC-601 uses a 16-bit instruction word. It can do an add in 2 microseconds, a multiply in 9. Eighty-seven instructions are available to programmers, and all software is coded in assembly language¹⁷⁸. The memory is 2-mil plated wire, which has been used widely in the military and is known for its ruggedness. It functions much like a core memory in that data are stored as a one or zero by changing the polarity in a segment of the wire. Each machine has 16K of 17 bits, the seventeenth bit used to provide even parity¹⁷⁹. Plated wire has the advantage of having nondestructive readout capability.

The controllers are arranged with power, central processor, and interfaces as independent components, but the I/O devices are cross strapped. This provides a reliability increase of 15 to 20 times, as modular failures can be isolated. The computers and associated electronics are referred to as Channel A and Channel B. With the cross strapping, if Channel A's output electronics failed, than Channel B's could be used by Channel A's computer¹⁸⁰.

Packaging is a serious consideration with engine controllers, since they are physically attached to a running rocket engine, hardly the benign environment found in most computer rooms. The use of late 1960s technology, which creates computers with larger numbers of discrete components and fewer ICs, means that the engine builders are penalized in designing appropriate packages¹⁸¹. Rocketdyne bolted early versions of the controller directly to the engine, resulting in forces of 22g rattling the omputer and causing failures. The simple addition of a rubber gasket reduced the g forces to about 3 or 4. Within the outer box, the circuit cards are held in place by foam wedges to further reduce vibration effects¹⁸².

THE FUTURE OF THE SHUTTLE'S COMPUTERS

The computers in the Shuttle were candidates for change due to the rapid progress of technology coupled with the long life of each Shuttle vehicle. First to be replaced were the engine controllers. By

ORIGINAL PAGE IS
OF POOR QUALITY.

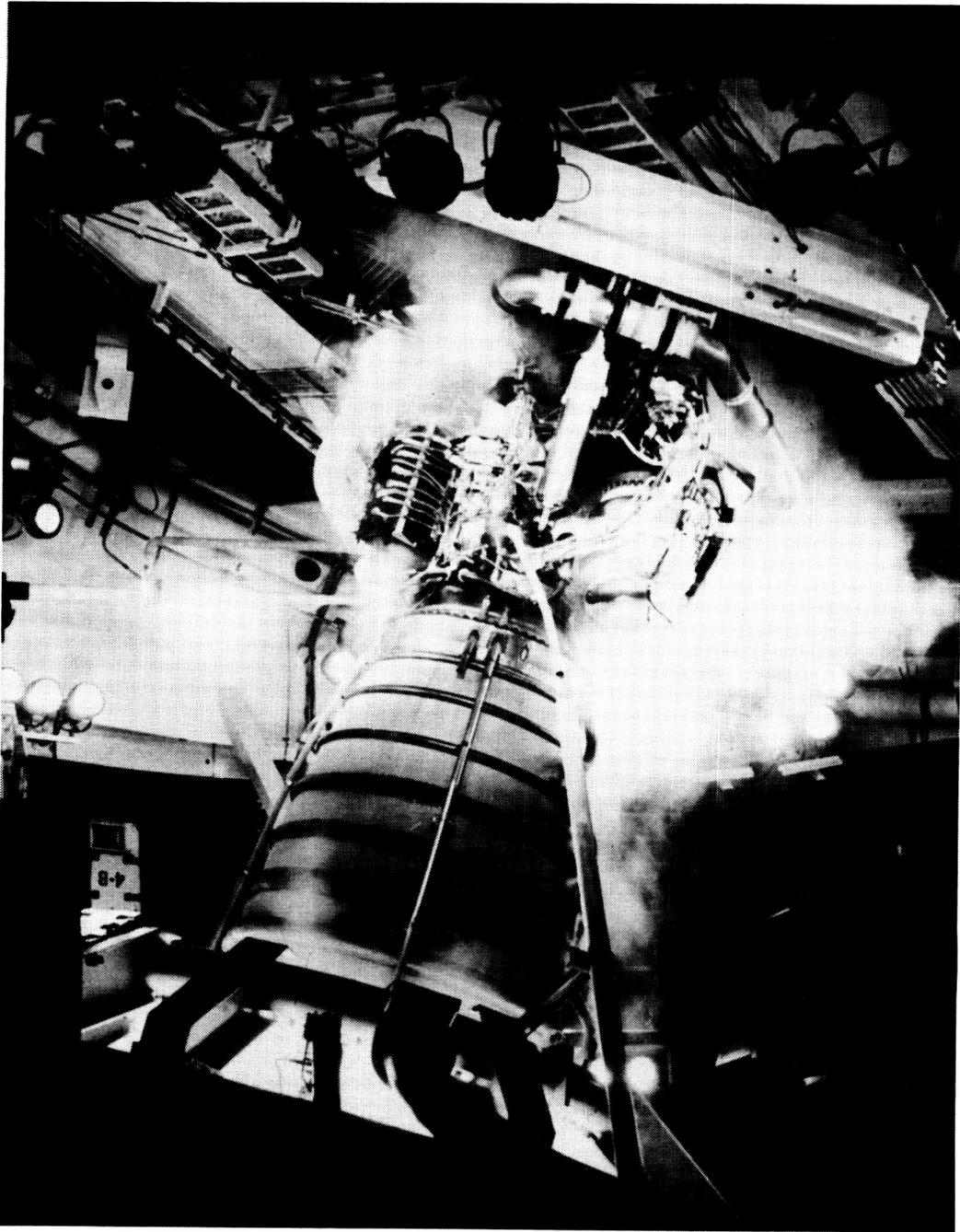


Figure 4-11. A Shuttle main engine in a ground test. The Controller can be seen mounted on the left side of the combustion chamber. (NASA photo 885338)

the early 1980s, Marshall Space Flight Center began studying a Block II controller design because it was becoming impossible to find parts and programmers for the late 1960s components of the Block I¹⁸³. The revised computer uses a Motorola 68000 32-bit microprocessor. When selected, it was clearly the state of the art. Instead of plated wire, a CMOS-type semiconductor random-access memory is used. Finally, the software is written in the high-level programming language, C. Such a computer reflects the current design and components of a ground-based, powerful digital control system. The C language is also known as an excellent tool for software systems development. In fact, the UNIX operating system is coded in it.

Aside from the processor change, the Block II's memory was increased to 64K words. Therefore, the entire controller software, including preflight routines, can be loaded at one time. Semiconductor memories have the advantages of high speed, lower power consumption, and higher density than core, but lack core memory's ability to retain data when power is shut off. Reliability of the memory in the Block II computer was assured by replicating the 64K and providing a three-tier power supply¹⁸⁴. Both Channel A and Channel B have two sets of 64K memories, each loaded with identical software. Failure in one causes a switch-over to the other. This protects against hardware failures in the memory chips. The three tiers of power protect against losing memory. The first level of power is the standard 115-volt primary supply. If it fails, a pair of 28-volt backup supplies, one for each channel, is available from other components of the system. Last, a battery backup, standard on most earth-based computer systems, can preserve memory but not run the processor.

The significance of the evolution to Block II engine controllers is that they represent the first use of semiconductor memories and microprocessors in a life-critical component of a manned spacecraft. Honeywell scheduled delivery of a breadboard version suitable for testing in mid-1985. The new controller is physically the same length and width, so it fits the old mounting. The depth is expected to be somewhat less. When the first of these computers flies on a Shuttle, NASA will have skipped from 1968 computer technology to 1982 technology in one leap.

IBM's new version of the AP-101 (the F) incorporates some of the same advantages gained by the new technology of the engine controllers. Increasing the memory to 256K words means that the ascent, on-orbit, and descent software can be fitted into the memory all at once. (This is not likely to happen, however, because of the pressing need to improve the crew interfaces and expand existing functions.) Higher component density allows the CPU and IOP to be fitted into one box roughly the size and weight of either of their predecessors. Execution speed is now accelerated to nearly 1 million operations per second, twice the original value. In essence, NASA has finally acquired the power and capability it wanted in 1972, before the software requirements showed the inadequacy of the original AP-101.

As in the engine controllers, the memory in the AP-101F is made of semiconductors. Power can be applied to the memory even when the central processor is shut down so as to keep the stored programs from disappearing. A commercially available error detection and correcting chip is included to constantly scan the memory and correct single bit errors. These precautions help eliminate the disadvantage of volatility while still preserving the size, power, and weight advantages of using semiconductors over core memories¹⁸⁵.

CONCLUSION

The DPS on the Shuttle orbiter reflects the state of software engineering in the 1970s. Even though the software was admittedly the key component of the spacecraft, NASA chose the hardware before the first software requirement was written. This is typical of practice in 1972, but less so now. NASA managers knew that time and money spent on detailed software requirements specification and the corresponding development of a test and verification program would save millions of dollars and much effort later. The establishment of a dedicated facility for development was an innovative idea and helped keep costs down by centralization and standardization. A combination of complete requirements, an aggressive test plan, a decent development facility, and the experience of NASA, Rockwell, Draper, and IBM engineers in real-time systems was enough to create a successful Shuttle DPS.

Even as the system took shape, NASA managers looked to the future of manned spacecraft software. Increased automation of code and test case generation, automated change insertion and verification, and perhaps automated requirements development are all considered future necessities if development costs are to be kept down and reliability increased. In the 1980s, a new opportunity for software development and hardware selection presents itself with NASA's long-awaited Space Station. NASA has another chance to adopt updated software engineering techniques and, perhaps, to develop others. Success in space is increasingly tied to success in the software factory.

power and complexity, paralleling the development of computers for manned spacecraft. Unlike the manned programs, however, JPL sponsored fundamental research into spacecraft computing, which was then translated into concepts that guided the development of flight systems. The result was a series of innovative and flexible on-board computers.

Part Two: Computers

On Board Unmanned Spacecraft

Unmanned spacecraft computers differ from manned spacecraft computers in that they are designed to work much longer and use much less spacecraft resources. A typical manned mission lasts a week or less; the exception was Skylab, whose computers operated for 9 months straight and again later during its reactivation mission. Unmanned missions in earth orbit or to the outer planets can last a decade or longer. Manned spacecraft usually carry large auxiliary power units based on fuel cell technology, as power requirements for life support, experiments, and computers are high. Spacecraft in earth orbit are often dependent on solar cell arrays, which are by nature low-power generators. Interplanetary probes use either solar cells or small radioisotope generators. Clearly, these circumstances cause different requirements for computers.

Of the two types of unmanned spacecraft, one is designed for earth orbit operations and the other flies to the moon, planets, or deep space. Earth orbiters usually need no navigation after achieving orbit; space probes, however, are critically dependent on proper guidance. Earth orbiters can be commanded nearly instantaneously from the ground during the roughly 10% of the time they are "visible" to ground stations. Interplanetary probes need to be autonomous, at least capable of independent routine operation, due to speed of light delays in communication and longer periods out of earth control. Multiple missions and simple geography prevent interplanetary probes from being in constant contact with the three Deep Space Network stations. Therefore, the basis of fault handling on an interplanetary probe is failure detection and repair, whereas earth orbiters concentrate on "safing" the spacecraft until the ground stations can help out. For these reasons, computers became more sophisticated on spacecraft designed to leave the gravity pull of the earth.

Moreover, the different computers have distinct origins. Many near earth spacecraft used a variant of a single machine developed at the Goddard Space Flight Center, whereas the Jet Propulsion Laboratory (JPL) of the California Institute of Technology, a long time NASA contractor, has dominated computer construction for deep space flight, designing and building an evolving series of computers for the Agency's interplanetary probes. These two lines of development represent the most fruitful of NASA's forays into computer

research. Computers on manned spacecraft were generally developed from other computers (Apollo from Polaris; 4Pi from the System 360). Computers in ground operations were adapted from commercial machines. However, computers on unmanned spacecraft were custom designed. In these cases, NASA was not only a contract monitor but was actively involved in development.

The making of the first NASA Standard Spacecraft Computer, which has controlled a number of earth-orbiting missions, has been described elsewhere* As can be inferred by its name, NASA designed this computer to fly on multiple, varying missions, which it has done to good effect. For example, both the Solar Maximum Mission and the Hubble Space telescope used the computer. Goddard Space Flight Center led development of the device over a 10-year period from the late 1960s to the late 1970s.

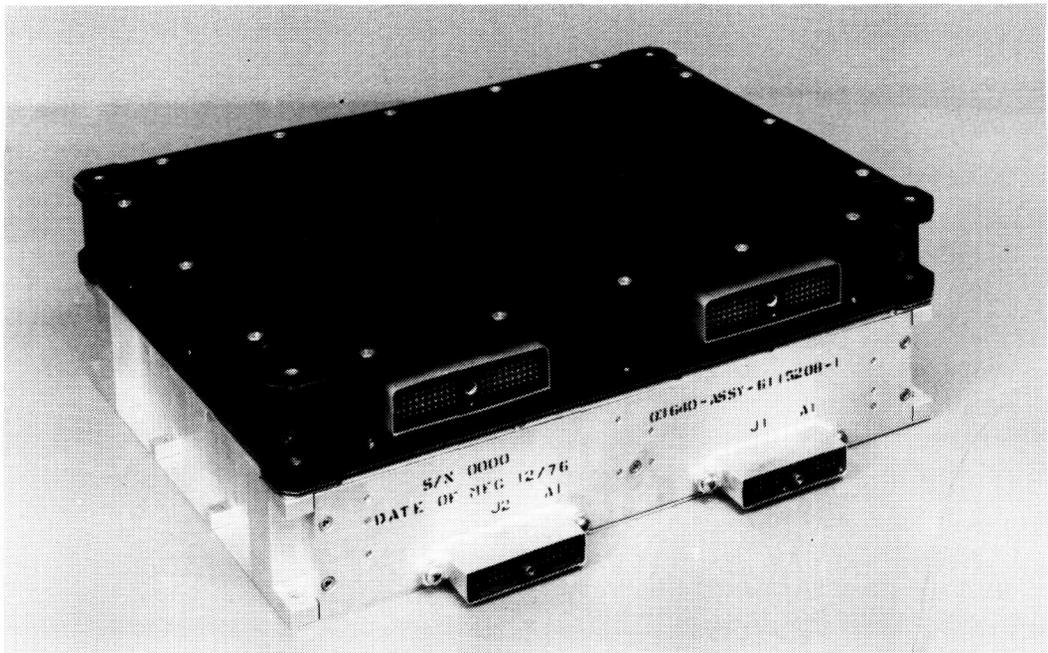


Figure B: The NASA Standard Spacecraft Computer I in its packaging. (NASA photo)

In contrast, machines built at JPL have had a longer and more related history. Although some reuse has occurred, the various space probes built at JPL carried mission-unique computers of increasing

*See Raymond G. Hartenstein, Ann C. Merwarth, William N. Stewart, Thomas D. Taylor, and Charles E. Trevathan, "Development and Application of NASA's First Standard Spacecraft Computer," *Commun. ACM*, 27(9), 902-913 (September 1984.)

5

From Sequencers to Computers: Exploring the Moon and the Inner Planets

One organization more than any other has dominated the exploration of deep space: the Jet Propulsion Laboratory (JPL) of the California Institute of Technology. JPL was responsible for the Ranger and Surveyor series of lunar exploration spacecraft, the Mariner and Viking Orbiter explorers of Mercury, Venus, and Mars, and the Voyager and Galileo probes of the outer planets. As a result, the evolution of on-board computers for deep space operations took place at JPL.

JPL's chief contribution to computing on unmanned spacecraft was in leading progress from hard-wired sequencers to programmable sequencers to digital computers. The Pioneer spacecraft developed mostly at NASA's Ames Research Center and the Lunar Orbiters used to map the moon in the 1960s did not carry on-board computers. Like their earth-orbiting cousins and the first JPL probes, they used sequencing devices to activate and command experiments. Later the Mariner spacecraft acquired more autonomy and flexibility by using machines that stored command sequences in changeable software. Finally, sophisticated spacecraft flew with special-purpose digital computers.

Unique in its relationship to NASA, JPL is *not* solely a government installation in the same way as, for example, the Johnson or Marshall Space Flight Centers. JPL's personnel receive their paychecks from Cal Tech, yet almost every piece of equipment on the site has a NASA property tag, since, for over a quarter of a century, Cal Tech has administered contracts that have paid for all research and development of the many spacecraft originated at JPL.

Another way in which JPL is unique is its products. Whereas thousands of earth-orbiting satellites have been launched, less than a dozen each of Rangers, Surveyors, and Mariners were constructed, and just two Vikings and Voyagers and one Galileo were sent into space. Not only were few spacecraft built, but the interplanetary launches were separated by years and had to be on strict deadlines due to the realities of celestial mechanics. This created a completely different development environment than that at other NASA centers. The emphasis on basic research at JPL has perhaps been stronger than at any other NASA installation. This orientation and its application in spacecraft forms a special part of the story of JPL.

JPL's computer development activities were shaped by its organizational structures. When a project is started at the Laboratory, an office is established to house the project manager, key systems managers, and staff. Offices have come and gone with the projects themselves. The Ranger office, for example, has been closed for nearly 20 years, whereas the Voyager office is likely to be open for as long as that. Most personnel are housed in divisions and sections relating to specific discipline or system functions, as, in 1984, the "Technical Divisions" contained sections on "Guidance and Control" and "Spacecraft Data Systems." When a project office needs a component or service, it "subcontracts" it to the appropriate technical sec-

tions. For instance, Spacecraft Data Systems supplies on-board computers, whereas the Navigation Systems Section does the trajectory calculations needed for a specific mission. In this way, specialists can be kept busy on a series of projects over a period of years without depending on a specific project for their jobs. Competition between sections to develop related components can also exist, as on the Voyager project, when the attitude control staff wanted to make their own computer for their system while the data systems people claimed sole domain over computer development. Within this setting, JPL has produced high quality on-board computers that have demonstrated outstanding reliability*.

FIXED SEQUENCERS: "COMPUTERS" ON RANGER, SURVEYOR AND THE EARLY MARINERS

Whether the final mission destination is as close as the moon or as far as Neptune, probe spaceflights consist of the same milestones and activities: launch, mid-course maneuver, cruise, and encounter. Spacecraft are launched in a stowed position dictated by the geometry of the booster vehicle. Most space probes look like multiarmed Hindu gods in flight due to the need to expose solar panels, point antennas, and deploy imaging equipment, but they must be folded to fit into the nose fairing of a rocket. During the launch period the spacecraft is injected into its transfer orbit to intercept the target, deploys its various appendages into their proper positions, and orients itself. A decision was made early at JPL to build spacecraft that would be stabilized in three axes during flight¹. Spacecraft would be oriented by using the sun, earth, and/or a star as a reference. If kept from tumbling they would always be pointed in a specific direction. A key advantage of this plan is that a directional antenna could be used for earth-space communications, reducing power requirements. Imaging equipment could also be more stable than on a spin-stabilized spacecraft such as a Pioneer. A disadvantage of three-axis stabilization is that a fairly sophisticated attitude control system must be carried, including a sensor system to find the sun and a guide star. Part of the launch phase, then, is spent scanning the sky for Canopus, Vega, or whatever star has been chosen for aligning the spacecraft.

The mid-course maneuver phase often comes only a day or two after initial transfer orbit insertion in order to correct relatively large

*JPL's roots and its role in NASA receive excellent treatment in Clayton Koppes' *The Jet Propulsion Lab and the American Space Program*, Yale University Press, 1982.

injection errors. Consisting of a timed burn of the spacecraft's propulsion system in each of three axes, it serves a number of purposes. Early launches could not depend upon the launch vehicle to establish an adequate flight path. Later, as booster guidance improved, probes were purposely aimed to miss the target so as to avoid contaminating planetary atmospheres with earthly bacteria hitching a ride on a spacecraft if the spacecraft ceased to function during launch and could not change its path to miss the planet. Therefore, the mid-course burn took place to correct the path of a "live" spacecraft. On long-duration missions with several targets, such as the Voyager probe to Jupiter, Saturn, Uranus, and Neptune, this maneuver might be repeated before and after each encounter. Engine firings are made before encounter to improve the accuracy of the trajectory to achieve a better gravity assist from the target planet to the new trajectory and reduce the size of the postencounter maneuvers.

Less is done on the spacecraft during the cruise period than in any other mission phase. However, recent larger and more complicated spacecraft have particle and fields experiments that run constantly and engineering calibrations that need periodic attention. If the spacecraft attitude is disturbed, reorientation may be necessary. This period of relative quiet ends when the encounter sequences begin as the spacecraft nears its target. Instruments must be turned on, calibrated and aimed. Imaging instrument pointing must be programmed and controlled. Data must be recorded and transmitted to earth. Of course, these activities are repeated during multiple encounter missions.

Initiating the functions done in each phase requires on-board control. This was unnecessary for Ranger missions to the moon, which were simple impact flights with televised imaging during the last minutes. Because maximum speed-of-light delay in radio signals to the moon is less than a second, near-real-time commanding could be done. Ground commands could fire engines, point the spacecraft, and turn on cameras. Ranger flights used a voice/manual commanding system for this. Desired instructions were developed and formatted at JPL and then delivered by telephone to the Deep Space Network station currently in contact with the spacecraft. An operator would thumb-wheel the octal codes into a panel called the "Read-Write-Verify Console," sending them to the spacecraft after verification². Such care was not always enough. On Ranger III, a guidance error caused the spacecraft to miss the moon by 23,000 miles. Although JPL flight controllers were able to get images during the flyby, a documentation discrepancy between the command set developed during the ground testing of the spacecraft and the flight set caused Ranger to point the wrong way, returning images of open space³.

Ranger carried a "Central Computer and Sequencer" to back up the direct command system. Activated before lift-off, it counted the hours, minutes, and seconds until a specified mission event was to oc-

cur and then executed a set of commands that performed the required functions. If the uplink radio channel failed, the mission would proceed according to a prepared plan. This assumed optimum performance, turning on the cameras regardless of where the spacecraft might be actually pointing. Still, it provided a bit of insurance for the mission.

At the same time that the Rangers were being built, JPL designed and flew the first Mariners. Mariner's initial mission was a Venus flyby launched in 1962. In the case of this spacecraft and its later brethren, the Central Computer and Sequencer was the prime source of commands, at least for cruise and encounter portions of the mission⁴. The time delay for commands to travel to Venus and Mars defeats real-time control from the ground. For Mariner II, at launch time minus 15 minutes, the clock was set so that the encounter sequence would begin at 12 hours from the closest approach to Venus. The sequencer's clock, a very accurate oscillator similar to computer clocks today, started at launch time minus 3 minutes⁵. Direct commanding capability was maintained. When the star tracker got confused and locked onto the wrong target, ground controllers could reinitiate a search⁶. Direct command could also be used for mid-course maneuvers. As a complement to direct command, "quantitative" commands could be sent to the sequencer for later use⁷. For instance, times such as "51 seconds of minus roll" and "795 seconds of minus pitch" or burn times could be inserted into the memory for later execution⁸. Mariners could abandon direct command and go to automatic command if a radio failure was detected. On the Mariner Mars 1964 spacecraft the sequencer contained a cyclic command that checked for such a failure at 66 2/3 hour intervals, effecting an auto switch-over⁹.

The Mariner II spacecraft to Venus (1962), Mariner IV to Mars (1964), and Mariner V to Venus (1967) carried the same Central Computer and Sequencer. Just one flew on each mission, due to space and weight restrictions, even though the machine weighed in at 11.5 pounds¹⁰. However, with the direct command capability intact, each had essentially the same level of redundancy as the Gemini and Apollo spacecraft, with their single-processor on-board computer systems and ground control computers. Plans for Mariner Mars 1969 called for a larger spacecraft and a more ambitious mission: two picture-taking flybys of different portions of the "red planet". JPL's Neil H. Herman, who had headed development of the Sequencer, saw an opportunity to improve the device for the upcoming flights¹¹. One aim was to give the new spacecraft more flexibility. If the first flyby turned up something special, it would be very useful if the second spacecraft could be reprogrammed in flight to take advantage of lessons learned on the initial pass¹². This actually happened during the missions when reprogramming was accomplished for Mariner VII's

August 5, 1969 flyby in response to Mariner VI's July 31 passage¹³. Another reason for more on board autonomy is that command sessions for the Mariners lasted as long as 8 hours! Mariner's command rate was 1 bit per second, so long sequences were expensive both in personnel time and Deep Space Network time¹⁴. The availability of more space and weight plus the desire for flexibility and greater autonomy caused JPL to change the Sequencer to make it more of a computer and less of what it really was, a fixed-program counter.

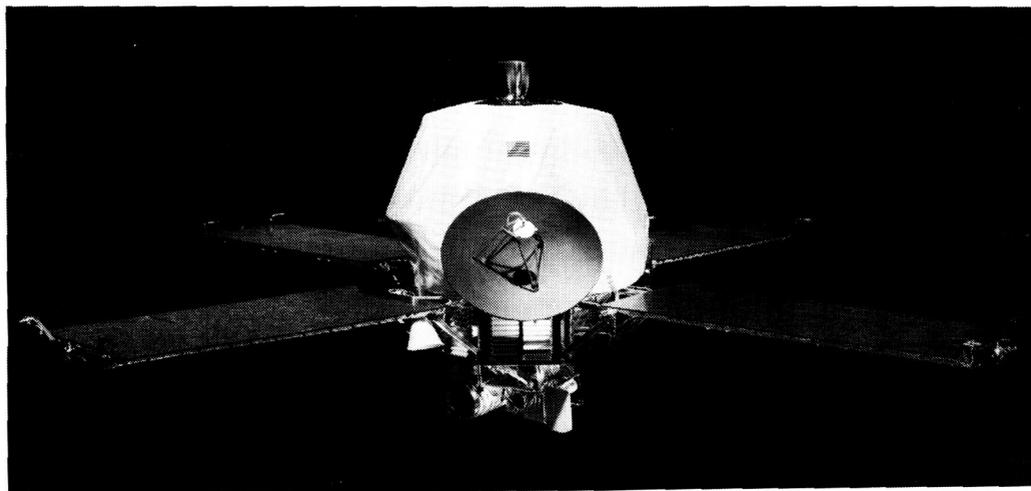


Figure 5-1. Mariner Mars 1971 carried a programmable sequencer with an expanded memory. (JPL photo P12035)

PROGRAMMABLE SEQUENCERS: MARINERS TO MARS, VENUS, AND MERCURY

Mariner Mars 1969 carried a 26-pound, programmable Central Computer and Sequencer designed by Herman and his team at JPL and built by Motorola¹⁵. The machine originated in studies done in 1964-1965 for a Mars orbiter and lander called "Voyager" and Mariner Mars 1966, neither of which flew¹⁶. The major difference between the fixed sequencers and the programmable sequencer is that it had a memory of 128 words that could be altered in flight. Although this device had far greater flexibility and capabilities than the fixed sequencers, one of the older sequencers traveled on the spacecraft as a backup. During critical maneuvers, the two sequencers would run in parallel, a disagreement causing an abort of the maneuver. The Sequencer commanded all spacecraft systems, including the Attitude and Pointing System and Flight Data System, each of which evolved to include their own computers by the time JPL designed the outer planets Voyager in the 1970s.

Original requirements for Mariner Mars 1969 called for 20 words of memory, making the 128-word version more than enough. Yet the memory was quickly exceeded, necessitating the use of "creative" programming techniques for the duration of the mission. Fortunately, the Sequencer was reprogrammable in flight. Memory locations used for terminated mission phases could be given to tasks scheduled for later. Edward Greenberg of JPL, who did most of the programming, replaced the launch and mid-course burn routines with new code after they had been executed¹⁷.

Despite the autonomy and flexibility gained by using the programmable Central Computer and Sequencer, the two Mariner Mars 1969 missions were the "most commanded" to that date. Mariner VI received 946 radio commands, Mariner VII got 957; either number exceeded the total number of commands sent to all the three previous successful missions *combined*¹⁸. One of the reasons for this was the memory restrictions; another could be that the engineers on the project downplayed the full capabilities of the Sequencer, not realizing what was possible¹⁹. However, the full capabilities of the device were more than exercised on the last Mariner missions.

Box 5-1: Programmable Central Computer and Sequencer Architecture and Software

The new Central Computer and Sequencer had no accumulator or central registers common to standard computers. Each memory location could be used as a register, and all operations began at a location, acted on the contents of another location, and ended in a third memory location²⁰. Memory consisted of 22-bit words stored in magnetic core, with destructive serial readout²¹. Three types of words could be stored. An *instruction word* used the first 4 bits for one of the 16 operation codes, the next 9 for the address of the memory location to be acted on, and the last 9 for the address of where to go afterward²². Instruction DHJ meant "decrement hours and jump," so the computer would subtract one from the time portion of the *event word* stored in the location specified by the first address in the instruction and then jump to the location specified by the second address. An event word contained a 13-bit time, scaled to hours, minutes, or seconds, and a 9-bit address of where to go to start the event being timed when the time part zeroed²³. For instance, if the event word was timing the mid-course correction burn, when the time portion reached zero, a branch would occur to the specified address, the first address of the mid-course maneuver subroutine. The last type of word was a *data word*, containing 22 bits of data.

Box 5-1 (Continued)

An instruction set of 16 operation codes contained mostly counting-type instructions: five scaled decrementing instructions (countdown hours, minutes, seconds, variables), and an incrementing instruction (count and jump). There was an ADD and a SUBtract, each requiring 27 milliseconds of machine time, by far the slowest instructions. Programmers used those very rarely, as the other instructions were better suited to sequencing. Subtraction was in one's complement form.

The sequencer executed the software by making a scan of the first seven instructions each hour²⁴. Those instructions constituted the entire executive! They contained sufficient decrement and branching instructions to check if anything needed to be done during that hour of flight. As an example, the exec might contain a counter that kept track of the time to an imaging session. The resulting routine might look like this:²⁵

1. Count 123 hours from start.
2. Count 45 minutes.
3. Activate camera and start frame count.
4. At 29th frame, start sending images.
5. At 32nd frame, rotate filter wheel to blue.
6. At 93rd frame, stop scan and stow platform.
7. Resume cruise mode counting.

After resuming cruise mode, the spacecraft clock would activate a scan at hour intervals again. Mission control could interrupt a scan, or a quiet time, and cause a jump to a specified subroutine²⁶. (The entire Mariner Mars 1969 flight program is reproduced in Appendix IV.)

A memory location could be changed by issuing two consecutive commands from earth stations. JPL called these commands CC-1 and CC-2. CC-1 sent the address and the least significant 7 bits of the new word, and it caused an interrupt in the receiving Sequencer. CC-2 relayed the most significant 15 bits and released the scan inhibit²⁷. A related command, CC-3, caused the Central Computer and Sequencer to read out the contents of a specified memory location, 1 bit per second²⁸. Input was even slower, requiring an average of 2 minutes per word, compared with a ground-loading time of all 128 words in less than a minute²⁹.

EXPANDED MEMORY AND EXPANDED FUNCTIONS

The new sequencer had a 9-bit address field, providing a 512 address limit. Expanding the memory to 512 words did not require a change in the logic. So JPL added the extra memory for the Mariner

Mars 1971 orbiter missions. Still, the old fixed sequencer remained in charge of the Mars orbit insertion burn. After the spacecraft established orbits, however, the ground control center used the new sequencer to control the imaging of Mars and its moons. The expanded memory proved sufficient. Preflight estimates for Mariner VIII specified 150 words of memory and 225 words for Mariner IX, yet both grew to over 400 words in flight³⁰.

The mission that used the sequencer to its limits was Mariner Venus Mercury 1973, or Mariner X. Mission profile called for the spacecraft to turn its imaging equipment on the earth as it flew toward deep space, do some studies of the moon in flyby, and then research in the area of Venus during a gravity assist maneuver that would send it toward Mercury, where JPL planned three separate encounters with the innermost planet.

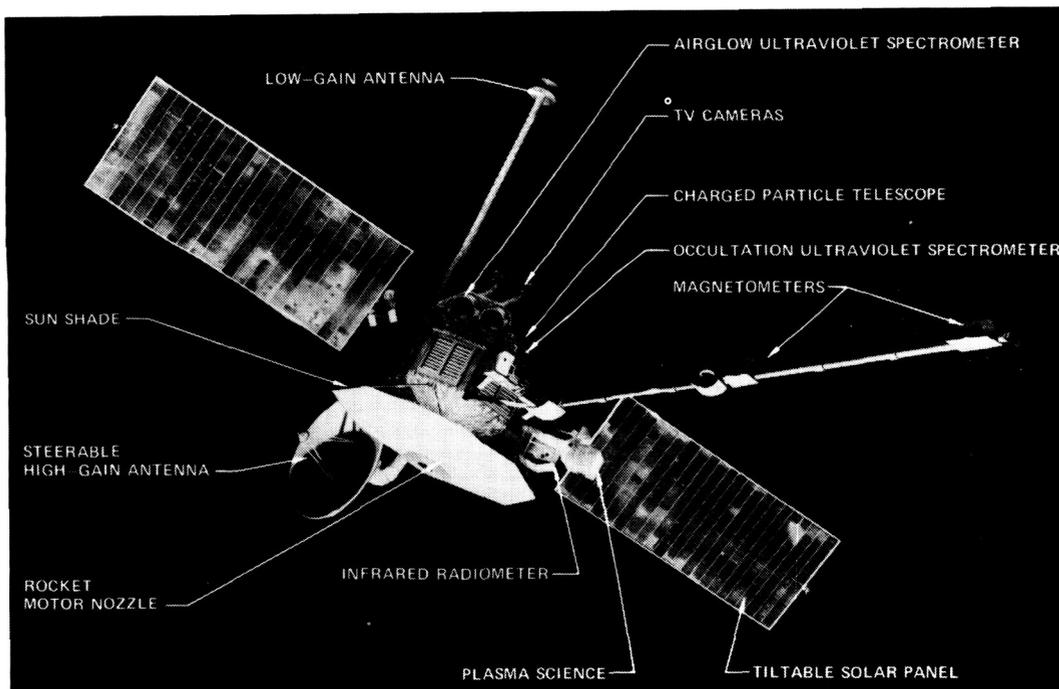


Figure 5-2. Mariner Venus/Mercury 1973 made the most use out of the programmable sequencers. (JPL photo 251-135AC)

Due to the more complex mission requirements, the design team wanted a bigger and better sequencer, but cost constraints killed any chance of building a new machine³¹. Adrian Hooke of JPL, one of the project's managers, decided to use planned memory updates at regular intervals. He also instituted a "suspenders and belt" approach to reliability. The sequencer would not only carry a detailed program for the next mission phase but also a constantly updated bare minimum program to complete the mission if the spacecraft lost contact with the ground. If a command was not received for a certain time, then the sequencer would follow whatever commands were in the backup program. Thus, software moved ahead in leap frog fashion. During the earth-moon phase the Venus backup was loaded, during the Venus encounter the backup Mercury encounter sequence was on board, and so on³². Software development was assigned to three programmers. Ronald Priestersbach of JPL wrote the near-earth and post-Mercury sequences, George Elliot of the Boeing Company did the Venus encounter, and Larry Koga of JPL wrote all three Mercury encounters³³.

During the 1969 missions, most changes and subroutines were hand-coded and used once. By 1971, the COMGEN ground computer program that produced memory loads for the Sequencer could develop blocks of commands that functioned much like subroutines in a standard computer program or macros in an assembly language program³⁴. In 1973, COMGEN resided in an IBM 360/75 computer that generated the commands and sent them via the NASA communications net to the appropriate Deep Space Network station for transmission. By this time, each station had a command computer, thus ending the voice/manual era³⁵. Another improvement to the Sequencer was that engineers could do memory checks by comparing a sumword stored in location 512 to the result of summing the first 511 locations. If a miscompare occurred, *then* a location-by-location check for error could be made³⁶.

The improvements both in the Sequencer and in programming and ground control techniques were not enough to ensure its use beyond the Mariner series of spacecraft. In spite of the success of the long and complicated mission of Mariner X, JPL's Hooke complained that memory limits were too costly due to excessive need for optimization and constant relocation of subroutines³⁷. Besides, the sequencers, regardless of their full name, were *not* computers. Spacecraft needed to do on-board computations, to have more room for software (and, thus, increased flexibility), and to use the central computer for other functions such as spacecraft health and safety monitoring done on other manned and unmanned spacecraft. Some missions intrinsically needed computers, as, for example, the Viking Mars orbiters and landers and the Voyager outer planet probes. The computer eventually designed, built, and used for the Viking Orbiter had its roots in the programmable sequencer, but it also owed some

concepts, at least in comparison, to a computer built in the research side of JPL and aimed at the long-duration, complex missions of the future. The story of that computer research project adds a necessary perspective for understanding the direction JPL's on-board computer development took in the 1970s.

THE STAR COMPUTER

Researching the Reliability Problem

In 1961 a Lithuanian-born computer scientist named Algirdas Avizienis, employed at UCLA, began research on a highly fault-tolerant computer system for use on long-duration space missions. The nonprogrammable version of the Central Computer and Sequencer would soon make its first flight on Mariner Venus 1962. Even at that early date, JPL expected to use computers on board the "Grand Tour" spacecraft planned for the 1970s. A favorable alignment of the outer planets would make possible a mission that could fly by Jupiter, Saturn, Uranus, and Neptune, thus having encounters with all the gas giants in one sweep. Such a mission would have to last for years, with the spacecraft operating autonomously for long periods of time. Inconvenient speed-of-light communications delays in the exploration of the inner planets would become crippling in an outer planets mission, requiring a spacecraft to carry its own "brain," because the earth-bound brains of its makers would be hours away in an emergency.

Avizienis' chief interest was in computer reliability. Computer failures occurred much more frequently then than in today's world of ICs. A computer entrusted with the successful completion of a deep space mission could not afford to fail before or during its long-awaited encounter, so JPL and Avizienis' interests came together at just the right time. During the period from 1961 to 1965, the Laboratory sponsored his search for a more fault-tolerant computer. In 1965 the reliability scheme was settled and construction of a prototype began. The breadboard version first ran a program in March of 1969, after a 2-year effort at software development³⁸. Avizienis named the computer STAR, for *self testing and repair*, and the name gives a clue to the architecture. JPL's Flight Computers and Sequencers Section of the Guidance and Navigation Division paid for the work. Avizienis was responsible for the concept; David A. Rennels, later a colleague at UCLA, for the hardware; John A. Rohr, for the software. F.P. Mathur did the reliability calculations, and the MIT Instrumentation Laboratory developed the read-only memory, which was basically a core rope type of memory³⁹.

Avizienis used selective redundancy to achieve reliability. On the Space Shuttle, the on-board computers are complete redundant versions of each other and are considered multiple computers. In the STAR, the computer is considered a single entity with its separate components replicated. Thus, each subsystem of STAR had several duplicate versions of itself in the computer as spares. The key advantage is that the spares were unpowered as long as the primary component ran successfully. Only when there was a failure would the spare come to life, and then power to the failed component would be cut off. Thus, the total power consumption of the STAR equaled, but did not exceed, that of a similar computer without the spares, making it attractive to power-conscious spacecraft designers⁴⁰. In the 1960s, all spacecraft computers were simplex systems. The only ultrareliable system was the Launch Vehicle Digital Computer used on the Saturn IB and Saturn V boosters. Its reliability was achieved by using triple modular redundant (TMR) circuits such as those in the Common Section of the Skylab computer system. Avizienis evaluated TMR circuitry and found that the number of independent failures a TMR system could tolerate before failing was much smaller than a component-redundant computer such as STAR could tolerate⁴¹. Also, reliability theoretically increased through dormancy⁴². Mean-time-between-failure (MTBF) figures for a component begin when the component is turned on; thus, a subsystem with a MTBF of 1,000 hours, backed up with two identical spares, yields a MTBF of 3,000 hours. That was the theory behind STAR.

Avizienis reasoned that failures were either caused by transient conditions or permanent component failures. In order to check for transient faults, STAR would repeat the program segment in which a fault was first detected. If the fault repeated itself, the affected component would be turned off and its spare activated, with the program segment repeated again. All fault detection was by hardware techniques, with error-correcting codes included in the software⁴³. Potentially, STAR could be an "automatic repairman" for the entire spacecraft, if other spacecraft systems used the same concepts⁴⁴.

Applications for STAR

In 1969, JPL began designing a Thermoelectric Outer Planet Spacecraft, or TOPS. In previous inner planet probes, the flight paths were close enough to the sun to enable the spacecraft to use solar cells for power generation. Outer planet missions ranged so far from the sun that solar cells would be inadequate. TOPS would carry radioisotope thermoelectric generators to provide electrical power.

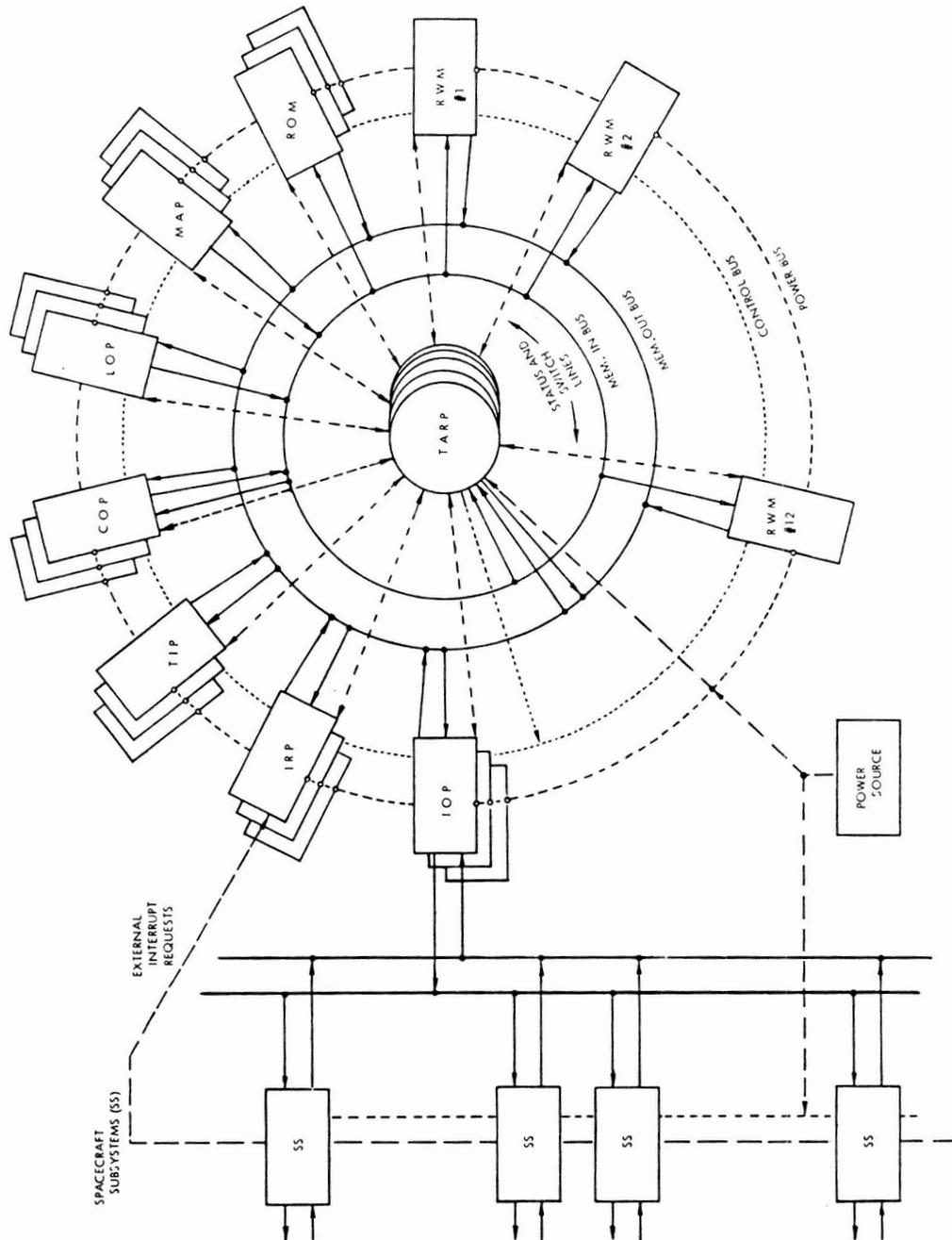


Figure 5-3. The STAR computer configuration. (From Avizienis, "Design Methods for Fault-Tolerant Navigation Computers," JPL TR-32-1409)

ORIGINAL PAGE IS
OF POOR QUALITY

Box 5-2: STAR Architecture and Software

STAR was a fixed-point machine with a 32-bit word. Using separated components for redundancy meant that they had to be connected on a bus, which had 4-bit bytes as the basic transfer block⁴⁵. There were 16K words of read-only memory, which Avizienis said consisted of a "braid" of transformers and wires⁴⁶. Since MIT built the device, the description almost certainly indicates that it was a core rope similar to that used in the Apollo Guidance Computer (AGC). The basic version used two copies of 4K of random-access memory, with up to 12 units attachable. Avizienis foresaw that the memory would have to be reprogrammed in flight on a mission like Grand Tour, so provision was made for that function⁴⁷.

Use of a large word size was not to increase arithmetic power as much as to provide space for error-checking codes. A STAR address consisted of a 16-bit field for the address and a 4-bit check field. The address would be multiplied by 15 (yielding 20 bits), and then stored or transmitted along the bus during an operation. At the receiving end, the address would be evaluated according to the following equation:

$$C(a) = 15 - 15|a$$

where $15|a$ is the modulo residue of a . Numeric data were handled similarly; the 28-bit operands multiplied by 15 to get a 32-bit word. If the result of the check operation was zero residue, the data or address was correct. If not, STAR issued a fault signal⁴⁸.

STAR had three control signals. One was the common 1-megahertz CLOCK signal. RESET indicated a return to a standard initial state. SYNC signaled the beginning of a new 10-step instruction cycle⁴⁹. If a fault was detected, the computer would return to the last SYNC point and begin executing instructions from there. If the 10 instructions after the SYNC were executed successfully, STAR sent a new SYNC signal.

STAR's read/write memory units were different in that they would recognize either their hard-wired name or an assigned name⁵⁰. In this way, if a memory unit and its backup copy failed, another memory unit could be assigned its name, loaded with the appropriate data, and then act like the original memory unit, thus avoiding the necessity of changing all the addresses in the software. When an instruction appeared on the memory in (MI) bus, the memory unit that had that address put its contents on the memory out (MO) bus, and the Arithmetic Processor or other component loaded it in for processing⁵¹.

Box 5-2 (Continued)

The heart of the STAR was the Testing and Repair Processor, or TARP. Whereas the other components of STAR had either one or two unpowered spares, the TARP had three active versions and two inactive spares. Functions of the TARP were to maintain the rollback points to which the software returned after a failure detection, to diagnose failures, and to check itself. Each time an error check was made, TARP's three units would vote. If all three or two of three indicated a failure, then the TARP issued an unconditional transfer to the rollback point. In the case of a 2-to-1 vote, the dissenting unit was considered failed, and was shut off as a spare was activated⁵². Another TARP disagreement caused the last spare to be activated. On the third TARP failure, one of the previously shut down units would be reactivated, so that there were always three TARPs in action at any given time. Avizienis thought that since most failures would be transients, it would be safe to reactivate a unit. After all, if it disagreed again, it would be shut down.

John A. Rohr's software group did not begin work until 1967. An assembler, loader, and simulator were developed on a UNIVAC 1108 mainframe computer owned by JPL⁵³. Software was all done in assembler, with a rich set of 180 single address instructions⁵⁴. The assembler did allow some types of higher level statements, mostly for arithmetic. For instance, $COMP\ Y=Y + 1$ was directly compiled into the several machine instructions necessary for execution⁵⁵. In this way, some of the tedium associated with assembly language programming was avoided. A floating-point subroutine to extend the calculating power of the machine was planned, but there is no evidence it was ever implemented⁵⁶. It would have had to have been done in software. The STARlet, a limited breadboard version, ran its first program on March 24, 1969⁵⁷. The full system, save the timing processor, was on the breadboard by April 1970⁵⁸.

STAR was considered as the on-board computer for TOPS⁵⁹. A control computer subsystem for the TOPS would use STAR technology, the full 32-bit word, but just 4K of read-only memory and 8K of the read/write memory⁶⁰. The chief physical obstacle to using STAR on a spacecraft was size. The breadboard version filled 100 cubic feet. Avizienis wanted to reduce it to 2 cubic feet and 50 watts⁶¹. By 1971, the requirements reduced to 1 cubic foot, 40 pounds weight, and 40 watts power⁶². Even though progress was made in this area, STAR never flew on a spacecraft. Components built to STAR specifications found their way into the NASA Standard Spacecraft Computer 1 (NSSC-1), used in earth orbital operations, but the concept of selective redundancy was not incorporated into flight computers to the extent desired by Avizienis.

STAR did not find its way to the outer planets for two reasons.

One was budget cuts⁶³. Even though the Voyagers were launched in the late 1970s, the original TOPS program and the Grand Tour were canceled due to budget constraints. The fact that Voyager 2 is essentially executing the Grand Tour is a bonus. On-board computers used on Voyager developed from a different line. So, even though Avizienis designed a Super-STAR with a microprogrammable processor using large-scale integration technology, which seemed certain to fulfill the requirements of size, power, and weight, he never sold it to JPL⁶⁴. A second reason STAR never flew was that engineers were concerned that the STAR's TARP and its failure switches were a weak point. The concept of a TARP, as with TMR, is always limited by the question of "who tests the tester?"⁶⁵. The actual switches entrusted with powering down a failed component and charging up another are the weakest link in the system. At one point, JPL subcontracted to the Stanford Research Institute for work on a magnetic switch, but apparently the results were not satisfying⁶⁶.

The STAR research program was not a waste even though the computer itself did not fly. It contributed to the development of new, reliable electronic components, such as those used on NSSC-1. It also provided a contrast to the development track being taken on the Mariner and Viking Orbiter spacecraft. One engineer involved in Viking Orbiter computer development said that STAR-type hardware was considered but deemed too complex. He thought that a two machine system running in parallel would be simpler and as reliable for a Mars orbiter/lander⁶⁷. Even though the technology of computers was not ready for STAR, it remains an innovative design and one of the few computer research projects funded by NASA. The principles developed remained valid for possible future applications that JPL was about to begin.

By far the most direct and far-reaching contribution of the STAR program to the future of JPL projects was John Rohr's work on the assembler/linker/loader for the software. It was the basis for the command sequence translators used through the present. Though extensively reworked and redesigned, the fundamental concepts were established by Rohr during the STAR development⁶⁸.

VIKING COMPUTER SYSTEMS

Viking missions to Mars were among the most complex ever executed by an unmanned spacecraft. Two probes were launched in 1975, with landings planned for the Bicentennial Summer of 1976. The project was controlled by the NASA Langley Space Flight Center, making it unique among deep space projects. Major work began in 1970, with a planned 1973 landing put off until 1976 because of budget cuts⁶⁹.

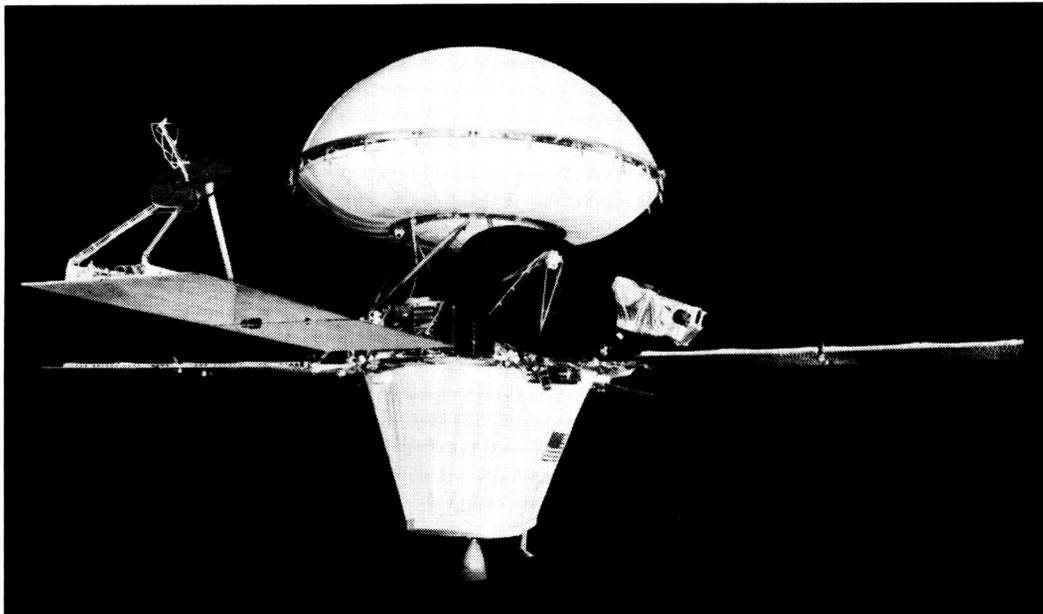


Figure 5-4. The Viking Orbiter and Lander each carried dual redundant computer systems. The Lander is in the elliptical shroud. (JPL photo 293-9157)

The Viking mission profile was a combined orbiter/lander. NASA had successfully orbited Mars with two Mariners in 1971. The Viking Orbiters were to conduct much the same science and imaging experiments as their smaller predecessors. But the Lander was a dramatic addition: it would be the first spacecraft to land on a planet that had a chance of harboring life as we could understand it. JPL got the contract to develop the Orbiter as a result of its Mariner experience. Because JPL maintained the Deep Space Network and an existing control center, it also got the mission support contract. The surprise for JPL was that the Martin Marietta Corporation's Denver division received the contract for the Lander. JPL had built the only U.S. unmanned landers, the Surveyor moon probes. Despite that experience and the difficulty of coordinating work on the Orbiter, Lander, and mission support in sites ranging from California to Denver to Virginia, Martin Marietta was chosen.**

Both the Orbiter and Lander carried dual redundant computer sys-

**Edward and Linda Ezell make the point in their book *On Mars* (NASA SP-4212) that part of the reasoning for choosing Martin Marietta was that the project management team at Langley felt that JPL would be overtaxed handling responsibilities for two spacecraft. Also, the difficulty of integrating the Lander components was greater, and a large aerospace contractor such as Martin had more extensive experience with such activity.

tems. JPL had evolved past the programmable sequencer stage and flew a device called the Command Computer Subsystem (CCS) on the Orbiter. The Lander carried the Guidance, Control, and Sequencing Computer (GCSC). On both systems JPL and Martin demonstrated exceptional competence in software engineering in the areas of documentation and configuration control. JPL was essentially programming its first flight computer. The standards and practices used during the Viking project surpassed all but the Shuttle on-board software in quality. Somehow JPL avoided the trial and error learning process Johnson Space Center went through with the Gemini and Apollo flight software. On the other hand, Martin Marietta typically used good software development practice. Along with other defense contractors such as Boeing Military Airplane Company and TRW Corporation, it was among the leading producers of software in the world. Whereas commercial computer companies such as IBM, Honeywell and Digital Equipment have generally written systems software for their own products, large-scale applications software has been the domain of vendors supplying the military services with command and control systems and embedded software in weapons. Martin Marietta is such a vendor and subscribed to military contract specifications that required the use of strict software engineering principles. That experience carried over to Viking, prompting an innovative method of developing the flight program that holds promise for future space systems.

Viking Orbiter CCS

The Viking CCS made it possible to increase the results of the Orbiter mission many times over the Mariners of 1971. According to one designer, the 512-word Central Computer and Sequencer would have returned less than a hundredth of the data received from the Orbiters⁷⁰. JPL considered several designs for the Viking computer, finally settling on the eventual Command Computer because of its simplicity. It had the least number of parts and was similar to prior systems in concept⁷¹.

Viking's CCS was the first JPL command device to be fully redundant. Mariner missions that retained the original hard-wired sequencer to back up the programmable sequencer were redundant in the same way the Apollo lunar excursion module (LEM) had computer redundancy: two different systems could accomplish some, but not all, of the other's functions. The dual redundancy of the Viking subsystem was more like Skylab's computer system, with two power supplies, two processors, two output units, two discrete command buffers, and two coded command buffers. Interrupts and level inputs to the system were split and thus delivered identically to both processors.

Processors and output units were cross strapped so that in case of failures they could be reassigned. The hardware requirements document generated by JPL called this type of redundancy "single fault tolerance," in that each component had a backup, making possible extensive redistribution of functions⁷². In practice, the two sets of computers were useful because, at times, there was too much for one computer to do⁷³. Although the designated secondary processor and memory were rarely on line, certain operating modes called for dual processing. Requirements specified three operating modes: individual, where each computer could be working on different events; parallel, where each computer worked on the same event; and tandem, where each computer worked on the same event and the output units were voted in a manner similar to that used on the Mariners when the two sequencers were in action together⁷⁴.

In general, the design of the processor was exceedingly simple, yet fairly powerful, as indicated by the use of direct addressing, a minimal set of registers, and a reasonably rich set of 64 instructions. The key is that the design placed relatively light demands on spacecraft resources while replacing both the programmable sequencer and the command decoder used in the Mariners. The fact that the processor was later adopted by the Voyager project as its Command Computer and modified for use as the attitude control computer is not only a statement of JPL's frugality but also a testament to the versatility of the design.

Software Development Practices for the Viking Command Computer Subsystem

By the time Viking was under development, JPL had over a decade of ground software experience, with resulting institutional development standards. Most space-related software done at JPL in the 1960s was for the Deep Space Network and the large computers in the mission support area. Viking was the first flight software project, so it was remarkable that effective software standards were in effect from the beginning.

JPL's project organization assigned each subsystem a Cognizant Engineer responsible for the overall development of the component. For the CCS, Wayne H. Kohl was the "Cog Engineer." Samuel G. Deese and T. K. Sorenson also signed the hardware and software requirements documents and were heavily involved in the development of the computer. Significantly, JPL also assigned a Cognizant Software Engineer, R.A. Proud. JPL's project management apparently believed that software could be engineered, like hardware. Both hardware and software had requirements documents that set forth the

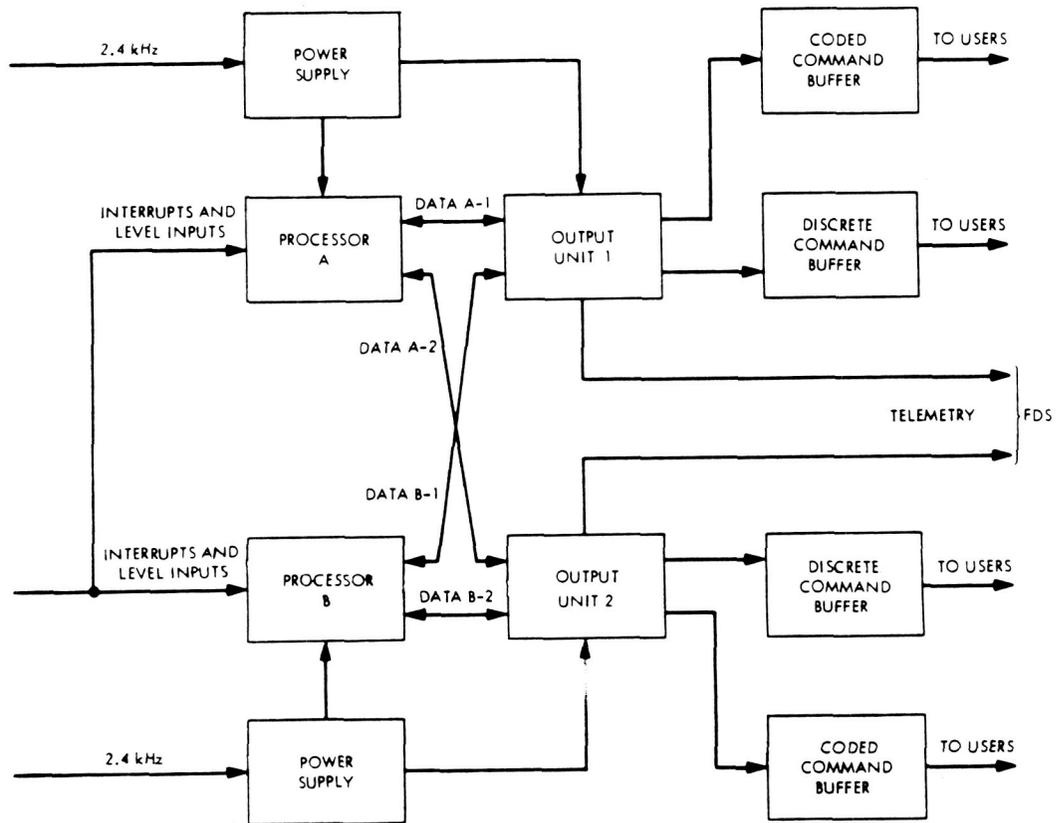


Figure 5-5. A block diagram of the Viking Orbiter Command Computer Subsystem hardware. This basic dual computer configuration was used for both Viking computers and all three Voyager computer systems. (From Kohl, *Viking Orbiter Computer Command Subsystem Hardware*)

functional specifications for the Command Computer and its software load⁷⁵. These were followed by detailed design documents.

Design documents generated for the Viking software and available to programmers were based on a software design description. That volume contained an overview to the mission and the software architecture, and, for each routine, detailed process descriptions, entry/exit points to other routines, variables and their descriptions, constants, and other relevant notes. A flowchart followed each routine's narrative description. Appendices to the document included a hardware description and a reference guide to the instruction set. Programmers were expected to use the design description as a manual. Volume two of the document contained the assembly listings of the resulting flight routines. By opening both volumes to the same routine, it was possible to easily follow the logic of the programs by reading both the narrative and the comments on the listing.

Box 5-3: CCS Hardware

The Command Computer's central processor contained the registers, data path control and instruction interpreter⁷⁶. The machine was serial in operation, thus reducing complexity, weight, and power requirements. It had 18-bit words and used the least significant 6 bits for operation codes and the most significant 12 for addresses, as numbered from right to left. This permitted 64 instructions and 4K of direct addressing, both of which were fully utilized. Data were stored in signed two's complement form, yielding an integer range from -131,072 to +131,071. Average instruction cycle time came to 88 microseconds. Thirteen registers were in the Command Computer, mostly obvious types such as an 18-bit accumulator, 12-bit program counter, 12-bit link register that pointed to the next address to be read, and a 4-bit condition code register that stored the overflow, minus, odd parity, and nonzero flags⁷⁷.

Timekeeping on the Orbiter was in three units. The clock issued interrupt pulses every hour, second, and 10 milliseconds⁷⁸, similar to the sequencer clocks used in Mariner, save that the 10-millisecond pulse provided finer timing. Pulses entered an interrupt processor that collected and interpreted them before transmission to the central processor. The interrupt processor had 32 interrupt levels, and constantly scanned for the highest priority task being requested⁷⁹. Thus, the Command Computer had the same interrupt-driven concept used in the Apollo and Shuttle manned spacecraft computers and the NSSC-1, but it was accomplished in hardware rather than software.

Viking's Command Computer used 4K of plated-wire memory⁸⁰, divided into four equal parts. The first three could be set as either read only, write protected, or read/write, but the last 1K was always read/write⁸¹. On Viking the first 2K was specified as read only, and the program instructions stored there. The second 2K was read/write, and the data resided in that segment.

Software development was guided by the "Viking 1975 Orbiter CCS and Support Equipment Software Development and Control Plan," which set the standards for production of the flight software and software for ground support and testing equipment. The Cognizant Software Engineer, Cog Engineer, Subsystem representative to the Systems Engineer, and all software design team members reviewed each routine as it was designed and coded⁸². Coding was assisted by the Orbiter Sequence Translator Program, or OSTRAN. Code produced by the programmers was verified by running it in both the CCS Breadboard and the CCS Programming System. The former was a complete hardware version of the Subsystem, and the latter a software simulation. The Command Computer Subsystem Tech-

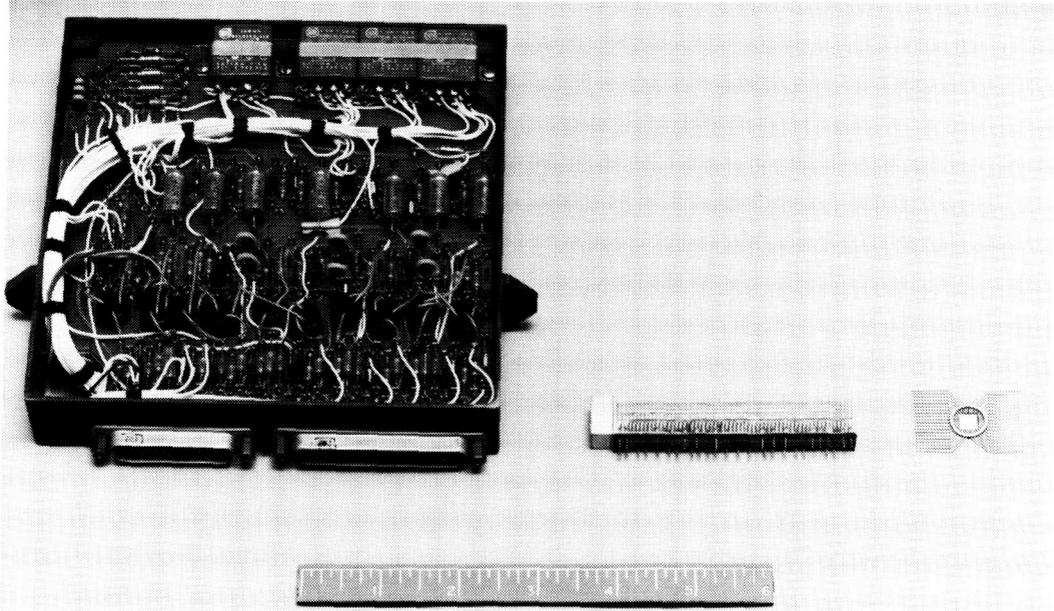


Figure 5-6. Different types of packaging used in the Viking computer system. Note the discrete components in the leftmost device. (JPL photo 360-276-AC)

nical Manager, hardware Cog Engineer, and Cog Software Engineer each compared the performance of the routines on these devices⁸³.

Testing and system integration was done from the bottom up. Programmers tested individual routines through all options at expected times, all expected branches, and all expected interaction with other routines, and then through selected failures⁸⁴. As with any real-time system, it was impossible to test for *all* possible failures. After this unit testing, the programmers integrated the routines with related code and ran it on either the breadboard or the flight hardware.

As with the most successful software development projects, JPL exercised strict configuration control. Even though the memory was eight times larger than that on the programmable sequencers, so many functions were transferred from hardware to software that memory was constrained from the beginning. Viking Orbiter Data Management Office handled changes to the documents. The Configuration Control Board, consisting of the Subsystem Technical Manager, the CCS Support Equipment Tech Manager, the hardware Cognizant Engineer, and the CCS Software Engineer, decided on software changes or what to do about discrepancies between the design and code or performance⁸⁵.

ORIGINAL PAGE IS
OF POOR QUALITY

Box 5-4: CCS Software Structure

The Viking Command Computer software structure appears different from others described in this volume because of the apparent lack of an operating system or executive program. The functional block diagram used in the CCS requirements document (reproduced here) shows that all inputs, either interrupts, or "level" inputs, enter a software block that contains conditioning routines. The TRAP routine maintains 32 memory locations that correspond to the 32 levels of the interrupt processor. Each location contains an instruction to be executed or an address to branch to if the appropriate interrupt occurs⁸⁶.

After clearing the input conditioning block, signals are either routed to the command decoding software or to the generation software. The command decoder does just what its title implies: examines the bit streams of commands routed to it for specific orders and then routes them to either the event generator, the output unit driver, or the telemetry processor.

The event generator block contains the most complex software in the system. Its chief routine is the Master Table Driver. Software requirements documents specified that the Master Table Driver handle all time sequenced events, maintaining up to 20 tables at once⁸⁷. Thus, it was the replacement for the programmable sequencer carried on previous missions. Implementation of the Master Table Driver was the TARMEX routine: Timing and Region Management Executive containing many of the common executive functions centralized in other machines. TARMEX is referred to as a "time-sharing executive" in the software documentation, but that is perhaps too ambitious a title⁸⁸. It did regularly scan through the event tables and maintain the time countdowns for a number of mission events. At 454 statements, it was one of the largest routines on the spacecraft. Functionally, it acted like any other sequencer JPL built, the difference being that it was implemented in software and thus highly flexible, which contributed to its success on the Viking mission and later on Voyager.

Other routines in the event generator were used less comprehensively than TARMEX. The Data Acquisition and Playback Routine controlled science instruments, imaging, and data storage until broadcast to earth. The accelerometer control routine was needed because for the first time an unmanned spacecraft would have active control over engine burns, rather than depending on precalculated timed firings. In the past, the maneuver and insertion firings were made based on calculations done before the flight and implemented as timed sequences in the Central Computer and Sequencer.

Box 5-4 (Continued)

Viking carried accelerometers and a computer, making it possible for the spacecraft to fire its engines and calculate when to turn them off in real time based on velocity figures uplinked in advance from navigation computers. A Launch/Hold/Reset routine handled spacecraft functions as a fixed sequence during the prelaunch, launch, and early cruise phases of the mission, with the capability to reset its timers if holds occurred in the countdown⁸⁹. This was a more robust version of the sequences carried for the first phase of previous missions. An Error Recovery routine included a programmable version of the 66 2/3-hour command loss sequence implemented in Mariner missions. The computer could be programmed to check for commands at varying times. During cruise, the command loss routine could be set to check just once a week or more, and changed to check at much closer intervals near encounter⁹⁰. Deep Space Network resources were thus less tied up during relatively dormant periods of the mission, as commands did not need to be sent just for the reason of keeping the command loss sequence from starting.

Remaining software blocks were the output driver, which transferred output signals to the appropriate output unit for distribution to the command buffer and eventually the affected systems, and the telemetry processor. The telemetry routine took over some of the functions previously done by the hard-wired Flight Data System. The Flight Data System on Viking had its own dual 1K memories of 8-bit words. Command Computer software helped manage that memory and prepare data streams for transmission. The Checksum routine was similar to that used in the Central Computer and Sequencer, except that a range of addresses could be specified, instead of the entire memory being summed.

Viking Orbiter software had to be written in an assembler, which fortunately had relocatable addresses, simplifying the maintenance task. The 64 instructions were mostly common to other computers, but there was no multiply or divide. There were two sets of loads, stores, increments, and subroutine calls: one used during independent operation and one aimed at dual operation, so that the two memories could be kept equivalent⁹¹. Even though many interrupts were available, most routines as coded had all but the internal error and counting interrupts disabled⁹². Many routines were free to run out without being interrupted, in contrast to the highly interrupted Apollo and shuttle software. Programmers avoided the memory and processing time overhead required to preserve the current accumulator and register contents during an interrupt.

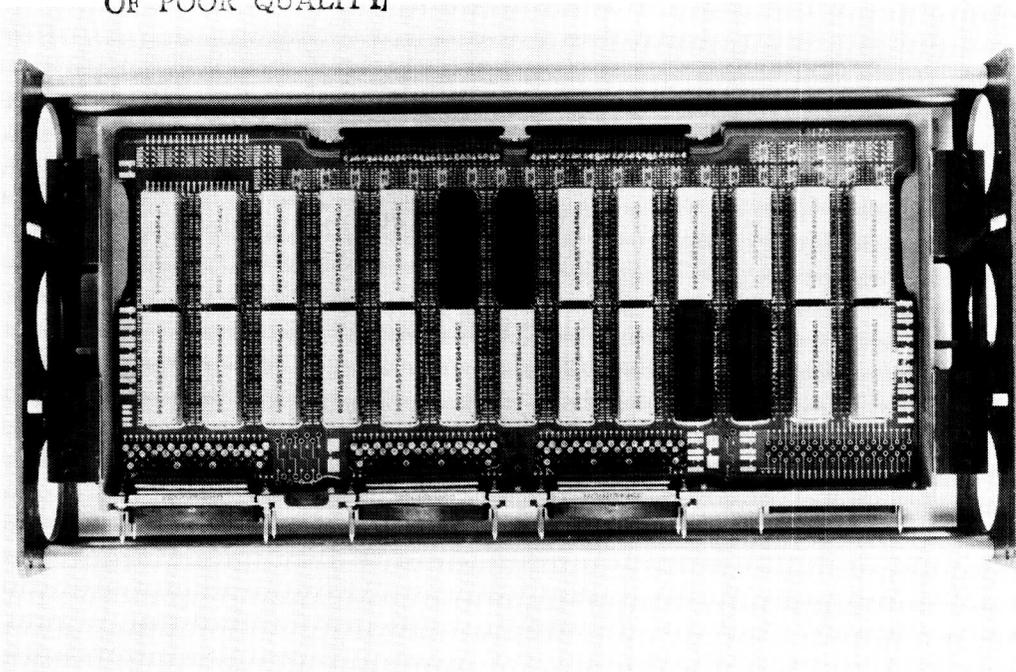


Figure 5-7. A circuit board with integrated circuits used in the Viking Orbiter computers. (JPL photo 360-371-AC)

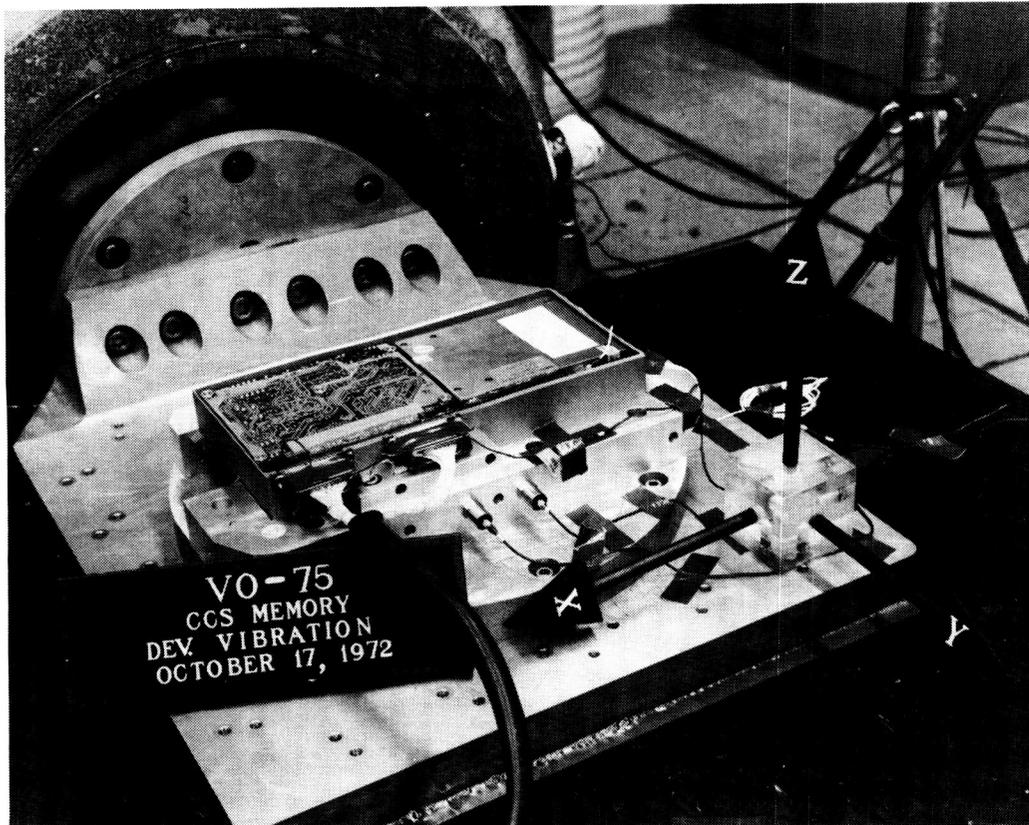


Figure 5-8. One of the Viking Orbiter plated wire memories in a vibration test device. (JPL photo 360-276-AC)

Instituting concepts of full documentation, configuration control, and engineering principles, such as modularization in software development, made it possible to have a successful flight program at launch and to remain successful throughout a long mission. Some of the people involved in Viking left the project before the Orbiters reached Mars, either to other projects, or to leave the Laboratory. With the materials and techniques available to maintain the computer software, it was possible to bring new people into the project and have them make necessary updates and upgrades to the flight program. This capability is as important to a long-term mission as the reliability of the hardware. JPL's concern for Grand-Tour-length reliability in hardware, exemplified by STAR, also extended to software. Without such an attitude the later Voyager would be much more difficult to maintain as an active project.

The Viking Lander GCSC

Martin Marietta's Denver division developed the computer system for the Viking Lander in an innovative way. To this point the stories of the development of various on-board computer systems have a similar theme: Project managers determine the expected specifications of the system, choose the hardware, and develop the software. By the 1980s, the danger of this approach became evident to computer and software engineers and to some of their customers. Choosing the hardware first and then developing the software for an embedded computer system runs the risk of the eventual software exceeding the hardware's capabilities or capacity. If the hardware is chosen before the full requirements of the mission are known, which was often the case, then the software is written in such a way as to compensate, thus exceeding the memory size because the compensating programs were not in the original software estimate. If the hardware turns out to be more powerful than needed, the software is expanded to take advantage of the additional capability, so it pushes the hardware to its limits. Either way, the development of the computer system and its software becomes more complex, expensive, and late. The Gemini, Apollo, Shuttle, NSSC-1, Mariner X, and Galileo projects all suffered because of insufficiencies in computing power or memory, largely because of poor specifications.

Martin Marietta did a number of military projects that repeated the same mistakes that the space program had made in regard to on-board computers. In 1970, when the company received the Viking Lander contract, it determined to follow a different course of development by adopting a policy of "software first"⁹³. This was one of the earliest attempts to break the paradigm of specification/ hardware selection/ software development/ reaction to changed requirements, a

ORIGINAL PAGE IS
OF POOR QUALITY

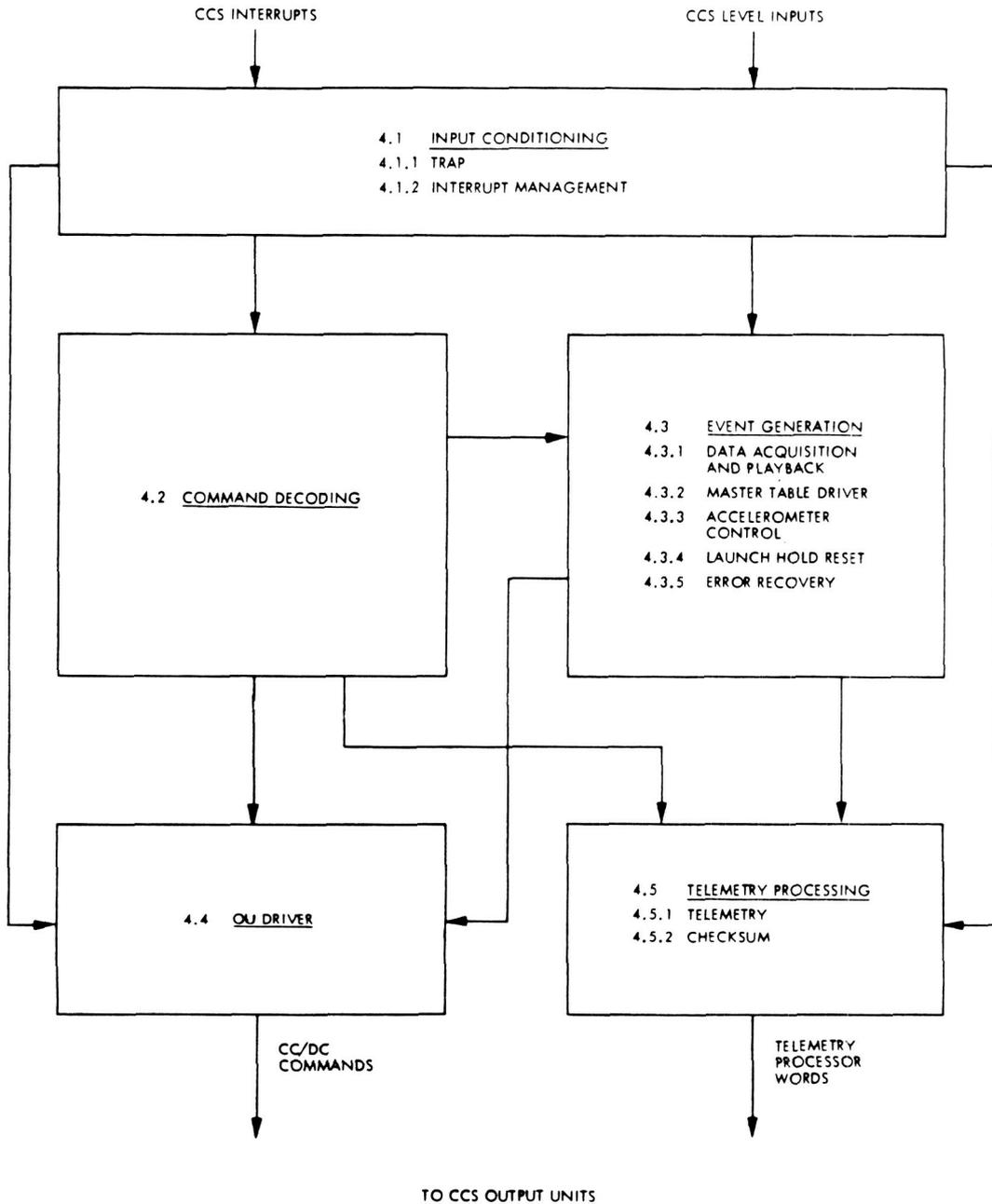


Figure 5-9. A block diagram of the Viking Orbiter software. This same software structure was used for the Voyager Command Computer Subsystem software (From Kohl, *Viking Orbiter Computer Command Subsystem Software*)

cycle that crippled many projects. The decision turned out to be highly successful. The Air Force studied the results and disseminated the technique. Thus, it contributed to a shift in attitude which, though only barely established 15 years later, is still a turning point in the history of computing.

"Software first" techniques make it possible to compensate for the severest deficiency of most projects, incomplete or incorrect requirements specification. Hardware decisions and software sizing are based on the requirements document developed early in the program. Users of the eventual product must be careful in contributing to this document. In the Apollo program, NASA gave MIT a broad statement of requirements, basically to develop a guidance system capable of navigating a spacecraft to the moon and back. Painful lessons learned as a result of the Apollo project included a greater appreciation for more detailed specifications. As a result, the requirements for the Shuttle were among the most outstanding written to that time. Still, memory estimates were far off, stretching the computer system to its limit. In contrast, the Viking Lander software developed contemporary to the Shuttle kept to its original boundaries, staying within the hardware capabilities of its computer. Both Shuttle and Viking had extremely high change traffic, and both project management teams anticipated many changes; but the software first philosophy handled change differently. Martin Marietta recognized that it is easier to change software independently of hardware than to react to revised specifications. More importantly, it recognized that if it is necessary to change hardware requirements, it is easier to change *before it is purchased* than afterwards. If the software is developed first, the hardware can be bought to fit it. Martin Marietta completed the flight software for the Lander 1 year before the hardware was delivered, which was only 2 months before the launch⁹⁴! The company accomplished this feat because detailed timing and sizing experiments gave confidence in the eventual hardware selection⁹⁵.

To implement the concept of software first, Martin developed a Viking Controls Mock-up Unit (VCMU) using two Standard Computer Corporation IC-7000 computers. IC-7000s had a two-section CPU. The Viking team microprogrammed one processor as an emulator of the proposed GCSC on the Lander; the other processor acted as a simulator of the other spacecraft systems⁹⁶. This system could be linked to an IBM 360/75 used at JPL for mission control, thus providing simulations of flight operations⁹⁷.

Differences between an emulator and a simulator are rather fine but very important in this case. A simulator imitates a computer using software that functions interpretively. For example, a simulator running a program written in the machine language of the target computer executes a set of instructions in its own machine language that has the same effect. The problem with this method is that it has vari-

able results. Even though most simulations are done on computers more powerful than the target machine, performance is far less, and a real-time simulation is virtually impossible. Even such simple instructions as an ADD, which can usually be simulated using a single instruction, run much slower because of the software overhead involved in maintaining pseudoregisters and fake memory. Programs that run in a minute on the target machine might take as long as an hour on a simulator.

An emulator runs the target machine's program in near real time. In fact, its performance on some instructions is likely to exceed that of the target due to the performance difference between the emulator and the actual hardware, but other instructions run slower, creating something of a balance. Microprogramming makes it possible to achieve these results. Older computers had the control unit that handles the flow of signals in the computer permanently hard-wired during manufacture; therefore, the way a particular computer executed instructions was fixed throughout its operating life. As early as 1950, Maurice Wilkes of Cambridge University suggested representing the control paths in the form of special software. He called these control programs "microprograms" and their instructions "microinstructions" to differentiate them from higher level programs and code⁹⁸. Such "microcode" could not be implemented in the 1950s because suitably cheap and permanent memory was not available. The IC-7000 was a microprogrammable machine, so its microcode could be changed by Martin Marietta to make the processor execute instructions like another computer. Martin started with a reasonable set of instructions and tried to write the software. If a problem or change arose that would be better handled by hardware or a new instruction or two, the microprograms for the new operation codes were installed⁹⁹. In this way, the hardware evolved along with the software, and when a fully functioning software load was complete, the hardware requirements were also complete.

Ironically, other constraints eventually thwarted Martin's plan for the computer hardware. Developers working on other subsystems of the Lander had trouble delivering hardware that could accomplish all the mission goals without increasing its weight. So when the time came to purchase the computer, the weight gains by the other systems had to be compensated for by the only system without hardware. Therefore, the computer that flew on Viking was actually the "third best" of those available, its chief deficiency being a poor instruction set, but it weighed less than the first choice¹⁰⁰. Martin changed the software affected by the less powerful computer. Even though the optimum computer did not fly, the principle of software first was demonstrated. Additionally, Martin introduced the concept of using off-the-shelf equipment for unmanned spacecraft projects. Despite the care taken to anticipate problems, some of the most common

development difficulties occurred. In an environment created to anticipate change, the lack of detailed software requirements and a large number of change requests still caused serious problems¹⁰¹. At one point, testing came to a standstill, which turned out to be fortunate in that the Systems Engineering Director began to take software seriously and to treat it like hardware, a lesson painfully repeated on project after project¹⁰². And again, memory sizing, though controlled, posed difficulties. Martin completed the prototyping for the Lander software in July 1971. Its actual size at that point was 13K. Martin engineers specified 18K, anticipating inevitable growth to accommodate new requirements and set up a control group to ensure that the memory stayed under that size. Twice during development the software exceeded memory limits, first in March–May 1973, when it topped off at 18.5K, and then in June 1974, when it hit 19K. Both times the flight program was reduced to the correct level¹⁰³. These overruns are minimal compared to those of the Apollo and Shuttle programs.

Eventually the software for the Lander reached 20,000 words and required 1,609 man–months to produce (the reason more than 18,000 words are shown here is that some routines used after landing overlaid landing software). Over 200,000 instructions of emulator and simulator software were produced, requiring just 494 man–months¹⁰⁴.

Differences in the proportion of development time to instructions are because the Lander software was hand-coded, whereas the simulators could be written with the aid of assemblers and higher level language compilers. Langley Research Center project managers determined that the flight software would be verified by an independent organization, so TRW Corporation was contracted to provide such services on site at Langley¹⁰⁵. Such completely independent verification is somewhat more useful than an "independent" quality assurance group within a company, as it has a more adversary relationship.

Box 5-5: Viking Lander Computer Characteristics

The GCSC consisted of two Honeywell HDC 402 processors, each with 18K of 2-mil plated-wire memory. These processors had the capability of eight levels of interrupt and an average 4.34-microsecond instruction cycle time¹⁰⁶. Original plans for the Viking Lander specified a single computer for the landing phase and another for on-surface operations, but when the project was delayed this changed to a dual redundant system similar to the Orbiter CCS¹⁰⁷. Honeywell's computer had a 24-bit word size, with 47 instructions, and used two's complement representation for data. Compared to the NSSC-1 and Viking Orbiter computer, it is slightly faster than the former and much faster than the latter, with better numerical precision than both.

Lander software structure reflected common short-cycle real-time control concepts such as those used in the Space Shuttle Main Engine Controllers. During descent, the software executed a 20-millisecond control loop, cycling through a set of routines¹⁰⁸. Martin claims that the executive was a "virtual machine" facility, in that each process "thought" that it had its own machine and was not sharing resources with other processes¹⁰⁹. Galileo Command and Data System software developers used the same terminology, but on that spacecraft the virtual machines resided on several microprocessors and were more truly "virtual." Martin's system is more like the cyclic time-sharing executive found in the Shuttle Backup Flight System.

One problem Lander software developers had was that no adequate assembler was ever written for the computer, perhaps because of the changing nature of the instruction set¹¹⁰. Patches had to be hand-coded in octal, with many jumps to unused memory space because of the lack of an assembler with relocatable addressing. A programmer trying to trace a routine thus had to contend with having to go back and forth on the memory map to follow the logic. JPL's Viking programmers could keep their routines in contiguous memory locations by reassembling the code after changes. The assembler would automatically move the data around to accommodate the modifications.

Lessons learned in the Viking Lander computer system development program influenced Martin Marietta's future work. After the VCMU outlived its usefulness, the organization and equipment were renamed EMULAB to reflect what takes place inside it. The Air Force requested that Martin study its software development practices during its participation in the space project, resulting in a report entitled *Viking Software Data*¹¹¹ and issued by the Rome Air Development Center at Griffith Air Force Base, New York. This report and the experience gained influenced the continuing shift from "hardware first" to "software first" among some contractors in the late 1970s and early

1980s. As microprocessors become military- and space-rated, it will become easier to adopt such a sequence because readily available microcomputers can be adapted to specialized functions. Users are also becoming more likely than before to adopt microprogramming to tailor instruction sets, as in the shuttle general-purpose computers and the Galileo attitude control computer.

ON TO THE OUTER PLANETS

Experience and the appreciation of the flexibility of computer processors are the legacy of the computer systems development for the inner planet probes. Consistent and detailed documentation, simple, reliable, and reusable hardware designs, and the practice of many missions contributed to the later and continuing success of Voyager. Just as management experience gained during Apollo applied to the shuttle, JPL's success with Viking made the concurrent development of Voyager and Galileo easier. People like Samuel Deese, who gained practical experience in the 1960s, led subsystem management in the 1970s. Viking's Wayne Kohl went on to Galileo after the Mars landings in a position similar to the one he held on the former project. Both Voyager and Galileo are better projects because of the continuity of techniques and personnel.

6

Distributed Computing On Board Voyager and Galileo

Voyager and Galileo are two outer planetary spacecraft that carry extensive computing capability. In spectacular encounters with Jupiter and Saturn, Voyagers 1 and 2 returned science data and imaging that far exceeded results of previous planetary flybys. Uranus was the successful 1986 objective of Voyager 2, nearly 10 years after launch. Galileo is designed for a Jupiter orbiter and probe mission.* Both types of spacecraft carry multiple computer systems, distributing functions among several machines, rather than using one central computer system as on the Viking Orbiter and Lander.

Distributed computing on large unmanned spacecraft developed conceptually from several sources. In 1967, Marshall Space Flight Center commissioned a study by General Electric Corporation's Missile and Space Division in Philadelphia as part of preparation for a huge "Voyager" Mars lander to be launched on a Saturn V booster in the early 1970s. Marshall asked GE to compare the advantages of a central computer configuration versus separate computers for different subsystems. General Electric used a highly mathematical approach to develop power, size, and weight comparisons of the different proposals in light of reliability considerations. Computer physical limits were set as high as 100 pounds and 300 watts due to the large size of the booster. This would allow computers such as the IBM 4Pi series, Autonetics D26J, and IBM's Saturn Launch Vehicle Digital Computer (LVDC) to be considered. Planners expected that the functions that later showed up on advanced Mariners—such as accelerometers, programmable sequencers with 512 words of memory, and telemetry registers—would be part of the proposed computer's capabilities and responsibilities. However, GE found that economies gained by a central system were outweighed by reliability advantages intrinsic to a distributed system¹.

Another approach came from Edward Greenberg, a Jet Propulsion Laboratory (JPL) engineer who programmed for the Mariner VI and VII Central Computer and Sequencer and contributed to the Viking Command Computer Subsystem (CCS) design. In December, 1972, he proposed that the Viking computer be standardized as a multimission processor². His intent was to reuse hardware and software development tools such as assemblers and simulators. Since one Viking computer could never handle all the functions needed on Voyager, several computers, each with a limited domain of functions, were needed.

Aside from the GE study and Greenberg's proposal, JPL

*Originally set for launch in the early 1980s, the mission slipped to May of 1986, but the grounding of the Shuttle fleet and cancellation of the Shuttle Centaur upper stage program in early 1986 led to an indefinite postponement and probably a change of launch vehicle.

developed an additional argument for distributed computing. Edward H. Kopf, Jr., a JPL engineer specializing in attitude control, pointed out that different sections of the Laboratory needed computers to perform their assignments on Voyager and Galileo. Each group wanted its "own" computer, so that it would not be constantly competing for resources with other groups³. Therefore, a distributed system would help keep the peace.

The attractions of distributing computing, reliability, potential reusability, and separation of tasks, proved true in the development of the Voyager and Galileo spacecraft. Each has a functionally distributed set of computers. Voyager makes use of two of the Viking machines and a third, custom-built, computer. Each concentrates on processing different functions, such as attitude control, data formatting, and commanding. Galileo has dual processors for attitude control and six in a network for command and data handling. Both spacecraft were designed for long-duration, autonomous flight, a goal difficult to attain without the use of distribution.

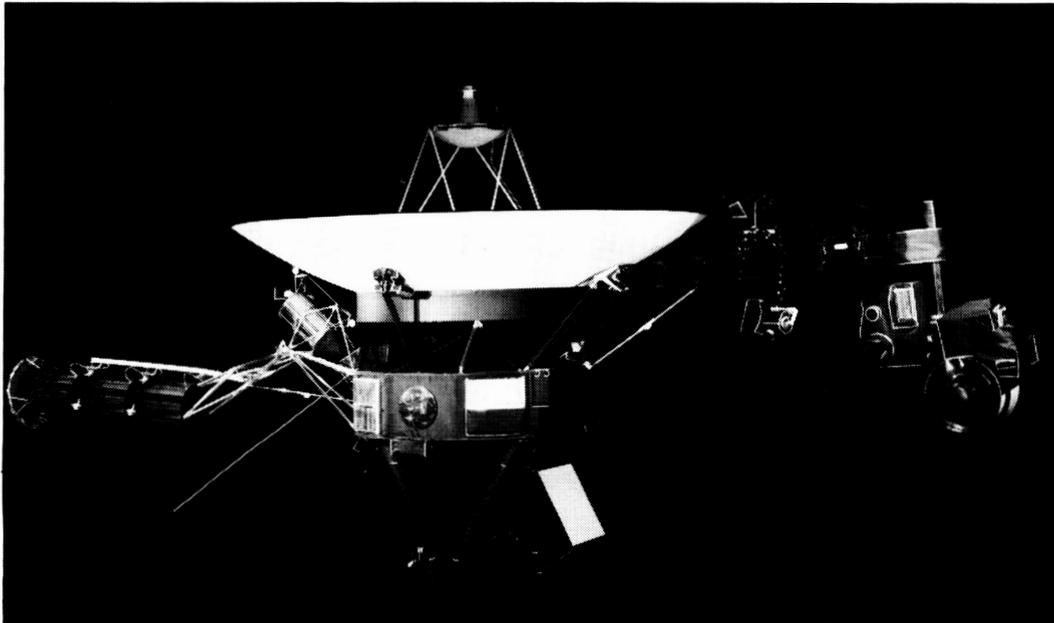


Figure 6-1. The Voyager spacecraft with the radioisotope generators on the left boom and the scan platform on the right boom. (JPL photo P10727B)

VOYAGER—THE FLYING COMPUTER CENTER

After the cancellation of the Thermoelectric Outer Planet Spacecraft (TOPS) project as such, JPL proposed, and NASA funded, a project called Mariner-Jupiter-Saturn 1977. It was given the name

Voyager in the mid-1970s. Although TOPS' original mission was to conduct the Grand Tour of the four gas giant planets, Voyager was limited to flybys of the innermost two, Jupiter and Saturn. However, favorable gravity assists and hardware longevity made it possible to plan for a Uranus flyby by the Voyager 2 spacecraft and, potentially, a Neptune encounter. After visiting Jupiter and Saturn, Voyager 1 is to travel out of the plane of the planetary orbits and leave the solar system.

Voyager employs three dual-redundant computer systems per spacecraft. The first, the CCS, is nearly identical to that flown on Viking, performing sequencing and spacecraft health functions along with new ones necessitated by the addition of the other computers. Telemetry data formatting and transmission handled by the Flight Data System are done on Voyager with the help of a custom-built computer. Attitude control and articulation of the scan platform are accomplished with the third computer system. One concept from the STAR computer proposed for the TOPS, applicable to Voyager, is dormancy. JPL's project staff believed that equipment would last longer if unpowered⁴. Although both CCSs are always powered, rarely are both Flight Data Systems running, and both attitude control computers are never turned on at the same time. Full bit-for-bit redundancy is not maintained in the dual memories. For example, "expended" algorithms, such as the deployment sequence executed shortly after separation from the booster, need not be maintained⁵. Both memories are accessed by the single active processor in each system. The Flight Data System keeps a copy of its instructions in both memories, but intermediate data and variables can be stored in either memory. This seemingly casual attitude toward memory duplication tightens up considerably near encounter periods, which is one time that both CCS processors are in tandem mode.

Since there are three computer systems on Voyager, JPL had to establish another layer of organizational control over its flight hardware and software development. Whereas Viking was assigned a single Cognizant Software Engineer, Voyager had three, managed by a Spacecraft Software Engineer. H. Kent Frewing of JPL assumed this position in early 1974 and sent out a series of organizing memos during the first half of that year.** Frewing's February 20, 1974 note set out his duties and a project time line through the summer 1977 launch dates⁶. Manpower estimates for software development ranged from one programmer in 1974 and 1977, with a peak of four full-time programmers in late 1975. The small group allowed most work to be

**He was replaced in early 1976 by Christopher P. Jones, who designed the integrated fault protection algorithms used on the mission, but Frewing laid the groundwork for management of the software.

done informally, easing communication. To provide some structure, Frewing established a Mariner–Jupiter–Saturn 1977 On-Board Software Design Team consisting of himself, Donald R. Johnson, the Flight Data System Cog Engineer, Stanley Lingon, the CCS Cog Engineer, and an Attitude and Articulation Control System representative⁷. They helped ensure the same close control of software development as on Viking, with good documentation and effective subroutine interfaces. The validation end of the software development process was handled by the Capability Demonstration Laboratory (CDL). Completed after the initial software was produced, it was a collection of either breadboard or flight surplus computer and science hardware, and its interfaces interconnected in the same way as those on the actual spacecraft. Its function is identical to that of the Shuttle Avionics Integration Laboratory (SAIL), in which both software and hardware changes could be tested to see if they functioned successfully⁸. Under this management umbrella, and with Cog Engineers constantly elucidating requirements from the science side and interpreting them to software engineers, each of the three computer systems took shape.

Voyager CCS: Parameters and Problems

NASA reeled from massive budget cuts during the 1970s. A changed political climate ended the Apollo era of near "carte blanche." Hampered by expensive Shuttle contracts as well as other factors, NASA management reduced its plans for unmanned exploration of the solar system. As Voyager developed under the new conditions, cost savings became a key ingredient in all engineering evaluations. JPL thus conducted a "CCS/CCS Memory Subsystem Design Inheritance Review" on January 17, 1974⁹. Held a year after Greenberg's proposal for standardizing the Viking computer, the Review resulted in the adoption of the Viking CCS as the Voyager CCS. The eventual hardware functional requirements document reads like a copy of the Viking document¹⁰. I/O interfaces with the new Flight Data System and Attitude Articulation and Control System computers are the major differences. Software such as the command decoder, certain fault processing routines, and others are fundamentally identical to Viking¹¹. Here again, differences are related to the new computers. All command changes and memory loads for the other computers are routed through the CCS¹². This required the addition of the routine MEMLOAD¹³. Another routine, AAC SIN, was added to evaluate power codes sent from the Attitude Control computer as a "heartbeat" to inform the CCS of its health¹⁴. The frequency of the heartbeat, roughly 30 times per minute, caused concern

that the CCS would be worn out processing it. Mission Operations estimated that the CCS would have to be active 3% to 4% of the time, whereas the Viking Orbiter computer had trouble if it was more than 0.2% active¹⁵. As it turns out, this worry was unwarranted.

Part of the reason why the more complex Voyager spacecraft could be controlled by a computer with the same size memory as Viking is the ability to change software loads. In-flight reprogramming, begun when the programmable sequencers flew on Mariners, and brought to a state of high quality on Mariner X, was a nearly routine task by the time of Voyager's launch in 1977. Both the CCS and Flight Data System computer have been reprogrammed extensively. No less than 18 loads were uplinked to Voyager 1 during its Jupiter encounter. During long-duration cruise, such as between Saturn and Uranus, new loads are spaced to every 3 months¹⁶. As pioneered on Mariner X, a disaster backup sequence was stored in the Voyager 2 CCS memory for the Uranus encounter, and later for the Neptune encounter. Required because of the loss of redundancy after the primary radio receiver developed an internal short, the backup sequence will execute minimum experiment sequences and transmit data to earth; it occupies 20% of the 4K memory¹⁷. CCS programmers are studying ways to use some bit positions in a failed Flight Data System memory to compensate for the shortened memory in their system. A readout register in the Flight Data System has a failed bit, giving the impression that the entire memory has a one stored in that position in each word. Remaining "good" areas may be assigned to the use of the CCS¹⁸.

Voyager Attitude Articulation and Control System Computer

JPL has been committed to three-axis stabilized spacecraft since it began designing probes in 1959. Attitude control systems maintain the proper pointing. The tasks assigned to the systems later expanded to include the actual operation of scanning platforms for imaging and other remote sensing instrument pointing. On the early Mariner missions the control systems consisted of analog circuits made up of hard-wired logic. By Mariner VIII, digital circuits replaced the analog electronics, and those were used on Mariner X as well as the Viking Orbiter¹⁹. Viking's Lander used the Honeywell central computer to run its independent attitude control system²⁰. A landing craft engaged in a powered descent needed far finer pointing than a spacecraft in free flight, and the bandwidth of a hard-wired system was insufficient to provide such control²¹.

Future probes, however, might need computer-controlled attitude electronics due to complex mission requirements or unusual

spacecraft configurations. NASA's Office of Aeronautics and Space Technology funded a study of extended life attitude control systems as the TOPS project wound down in 1972. The result was a combination analog and digital programmable attitude control system. Dubbed "HYPACE," for *Hybrid Programmable Attitude Control Electronics*, it was a byte-serial processor with substantial power²². Using the same 4K, 18-bit-wide plated-wire memory from the Viking Orbiter computer, HYPACE added transistor-transistor logic (TTL) medium-scale integrated circuits to create a relatively fast (28-microsecond cycle) processor with index registers for addressing. Byte-serial architecture was possible because the TTL chips were designed for 4-bit parallel operation, so the 18-bit words could be moved around in five cycles instead of the 18 a serial machine would need, increasing overall speed. Index registering meant that the same block of code could be used for all three axes, reducing memory requirements. It appeared that the attitude control systems of future spacecraft would almost certainly benefit from such a computer.

Voyager was the first to do so, due to new requirements. One difference between Voyager and Mariner and Viking is that the latter two were fairly rigid in construction. Voyager's radioisotope thermoelectric generators, however, were mounted on a boom to keep radiation leakage away from scientific instruments. In addition, the magnetometer was boom mounted to avoid interference from spacecraft magnetic fields caused by motors, actuators, power buses, and electronics. Finally, the scan platform was also on a boom to give a better field of view. The extended booms made Voyager much less rigid in flight, with thruster firings and maneuvers causing the booms to flex, complicating the attitude control problem²³. Additionally, the Titan III booster used for Voyager required a "kick stage" to successfully inject Voyager into the transfer orbit to Jupiter. Since the kick stage was kept simple, the spacecraft itself was required to do attitude control during firing, which entailed much narrower margins of control than the three-axis pointing in cruise²⁴.

JPL's Guidance and Control Section wanted to use a version of HYPACE as the computer for the Voyager. However, there was considerable pressure to build on the past and use existing equipment²⁵. Greenberg proposed using the same Viking computer in all systems on the Voyager spacecraft that needed one²⁶. A study showed that the attitude control system could use the CCS computer, but the Flight Data System could not due to high I/O requirements²⁷. Wayne Kohl, the Viking computer Cog Engineer, thought that the Voyager project could save \$300,000 by using the Viking machine for the attitude control function²⁸. His division chief, John Scull, supported that idea, possibly because of budget pressure from NASA²⁹. Raymond L. Heacock, as Spacecraft Systems Manager in the Voyager Project Office, and others from that organization were the key personnel in-

volved in making the final decision, influenced by the economy and feasibility of the idea³⁰. Money could be saved in two ways by using the existing system: avoidance of new development costs and retraining of personnel.

Guidance and Control grudgingly accepted the CCS computer on the condition it be speeded up. Requirements for active control during the kick stage burn meant that real-time control programs would have to be written to operate within a 20-millisecond cycle, roughly three times faster than the command computer³¹. An executive for the attitude control computer differed in nature from those for either the command computer or the Flight Data System computer. Basically, the attitude control computer needed to run subprograms at different rates, requiring several cycles, as in Apollo, Skylab, and the Shuttle. Guidance and Control asked for a 1-megahertz clock speed but wound up getting about three quarters of that³². The attitude control engineers also added the index registers that proved so useful during the HYPACE experiment. Documentation for the system still refers to the attitude control computer as HYPACE, even though its heart was the command computer. General Electric, which built the command computer, naturally built HYPACE, but the rest of the attitude control system was constructed by Martin-Marietta Corporation in Denver.

Teoguer A. Almaguer was the hardware Cog Engineer for the attitude control computer, whereas H. Karl Bouvier led the software development group. Bouvier actually worked on an analysis team within the Guidance and Control Section, but the team members were afraid to use the word "software" in their name because their tasks might have been taken away and given to an existing software team in another division³³. The programmers must have done an outstanding job, considering the slow processor and limited memory. At launch, only two words of free space remained in the 4K of plated wire³⁴. Tight memory is now a problem because the scan platform actuators on Voyager 2 are nearly worn out, and software has to compensate for this during Uranus and Neptune encounter periods.

Box 6-1: Voyager HYPACE Operation

HYPACE had four execution rates. Scan platform stepper motors and thruster actuators were among the routines executed during the 10-millisecond cycle. Attitude control laws and thruster logic executed in the 20-millisecond cycle. Scanning control and turn execution were placed in the 60-millisecond group, and the command interpreter and heartbeat were 240-millisecond routines³⁵. In operation, the standard 10-millisecond time interrupt would cause all 10-millisecond routines to execute. If it was time for one of the 20-, 60-, or 240-millisecond routines to run, it would be scheduled. Sometimes if the computer got too busy, the 240-millisecond cycle slipped to up to 350 milliseconds, but routines in that cycle were less critical than a routine to shut off an engine on time.

One thing needed on Voyager that did not exist when only single computers flew on unmanned spacecraft was an interface between the machines. The command computer could directly request data from either of its partners. A primary function of the command computer was to check periodically on the health of the other computers. Programmers in the Guidance and Control Section originally intended to send a "heartbeat" to the command computer each second³⁶. This was later raised to once about every 2 seconds, partly because of the command computer overload problem mentioned above. To carry the heartbeat, six direct input lines, similar to the 3-bit synchronization bus on the Shuttle, ran from the HYPACE to the command computer. A "power code" was the content of the 6 bits transmitted on those lines. For example, power code 37 was the simple heartbeat. Others related to passing information such as pointing commands. Power code 66, called "the Omen," told the command computer to save disaster parameters, because a failure was imminent³⁷. Every eight 240-millisecond cycles the heartbeat was sent. Between times, the attitude control computer conducted its self tests. If it failed, the heartbeat generator was bypassed. After about 10 seconds passed with no heartbeats, the command computer would issue a switch-over command to the backup processor.

A switch-over to the backup attitude control computer took place on Voyager 2 16 seconds after separation from the solid rocket stage³⁸. Separation was so rough that the spacecraft was sent off attitude. Simultaneously, the booms were being deployed by the command computer. A thruster configuration initialization involving the plumbing for the thrusters delayed their acting to correct the attitude error.

Box 6-1 (Continued)

Since this was one of the mission-critical times that the command computer was running in dual mode, the attitude control computer got *two* commands to initialize the plumbing. Executing the second command pushed back the attitude control recovery even farther. Soon the computer exhausted its options and voluntarily stopped the heartbeat. When the backup came on-line it had no record of the gyro readings. Not knowing how bad things were was a blessing, as it executed a simple orientation and stopped the spacecraft roll³⁹. Here is an instance where maintaining bit-for-bit identical memories would have been disastrous, as the backup computer would also have tied itself in knots.

Developing Voyager's Flight Data System Computer

Flight Data Systems handle the collection, formatting, and storage of science and engineering data on spacecraft. If the data are to be transmitted directly, a high rate of input and output is needed so that nothing is lost. If data transmission is deferred because a spacecraft is occulted from the tracking station, then the Flight Data System sends the data to a magnetic tape recorder known as the Data Storage System (DSS). As JPL progressed through Ranger to Surveyor to Mariner and to Viking, the rates of the data-handling requirements went steadily upward. This was because of increased instrumentation, greater sophistication in the spacecraft engineering systems, imaging equipment with better resolution (thus needing higher bit rates), and improved communications equipment permitting faster transmission of data. These changes led away from hard-wired Flight Data Systems. One big step was the use of a digital memory on Viking to store different sequences of data handling. It was much like the microprogram in a central processor and for a similar purpose: to save hardware⁴⁰. From there it was a short step to a full-fledged computer.

TOPS feasibility studies refer to a Measurement Processor Subsystem, the first time a separate computer was considered for flight data⁴¹. Although the command computer had been suggested as a possible Flight Data System machine, JPL engineers soon realized that even though the processing part of the job was well within the power of the computer, the I/O rates precluded its use.

JPL commissioned the development of a new computer from scratch and assigned Jack L. Wooddell to the job. Wooddell prepared an unusual document to tell the story of his work on the computer: a paper for a graduate computer science course taught by Dr. Melvin Breuer at the University of Southern California. Written around 1974, the paper includes what appears to be the flight version of the

design⁴². In it Wooddell lists the tasks he performed during the design period. He began by preparing a list of functions that the proposed Flight Data System was required to provide. These included sending control signals to sequence the science instruments, the ability to handle a wide variety of data rates and formats from the various instruments, potential for redesigning the mission in flight (as is now being done), monitoring engineering telemetry, and keeping to the reliability standard that no single failure result in loss of data from more than one scientific instrument or one-half the engineering sensors⁴³.

ORIGINAL PAGE IS
OF POOR QUALITY

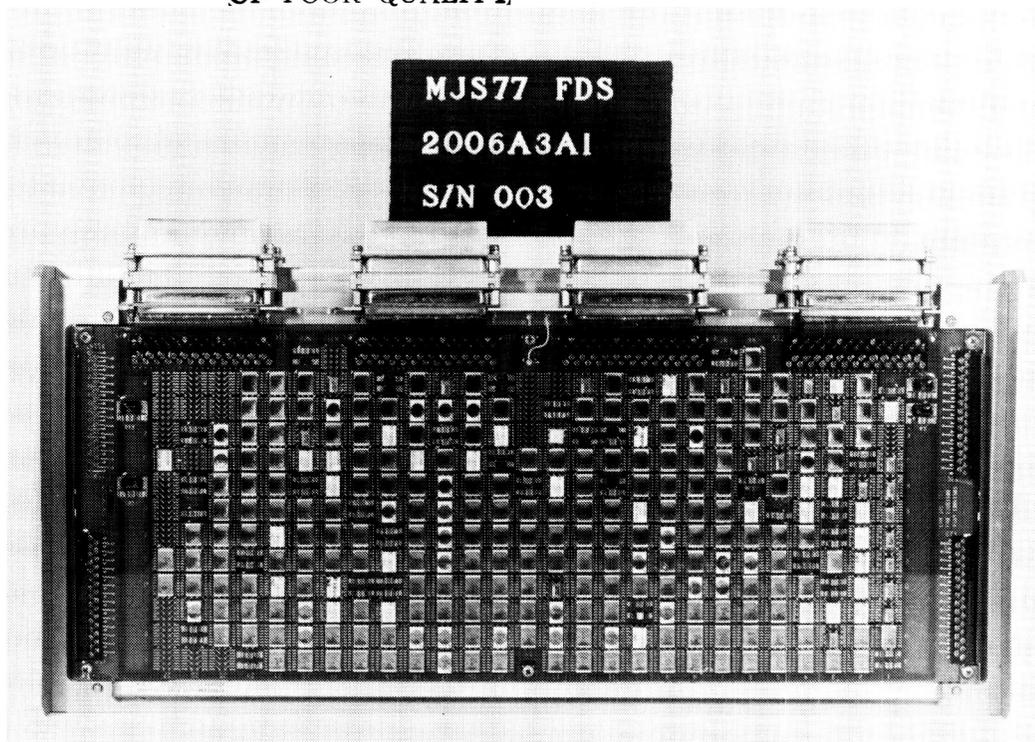


Figure 6-2. The Flight Data System hardware in its package. (JPL photo 360-751AC)

After determining requirements, Wooddell examined possible hardware and software tradeoffs. In an insightful memorandum, John Morecroft explained the concept of "soft logic" as a complement to the "hard logic" in the Flight Data System⁴⁴. Writing in 1975, when the actual flight software began to be prepared, Morecroft pointed out that the program for the computer was actually a soft representation of hard-wired circuits. Conceptually, the memo stands as an explanation of the essential meaning of firmware in general. During the second phase of his work, Wooddell determined which functions could be handled by hardware and which should be left to the flexibility of

software. With those decisions made, a preliminary instruction set and logic design could be prepared.

Uniquely, Wooddell began working with a programmer in 1973, as soon as the instructions were ready⁴⁵. Richard J. Rice of JPL began by developing software for a breadboard version of the data computer. The breadboard originally used the ubiquitous 4K memory of plated wire with 18-bit words and 150 of the same low-power TTL ICs used in other JPL machines⁴⁶. Instruction execution times for this version ranged from 12 to 24 microseconds. Rice's prototype flight program, developed on the basis of what was then known about Voyager instrumentation and previous experience, showed that the processor speed should be doubled⁴⁷.

Two significant hardware changes solved this problem. One hardware modification added direct memory access circuits and provided for using them on *each* instruction cycle. Direct memory access capability meant that some data could be sent directly to the memory without having to go through the central processor. In other computers, direct memory access is permitted as a sort of interrupt and is often referred to as "cycle stealing" because it takes time away from instruction execution. In the data computer, it would have been foolhardy to do direct memory access in that way because the data rate was so high that the instructions might never get a chance to be executed quickly enough for time-critical sequencing. Wooddell solved this by adding a direct memory access cycle to those instructions that did not already have cycles in which the memory was accessed⁴⁸. By adding that cycle all the instructions took the same time to execute regardless of direct memory access, making it easier to predict program run times and to guarantee the memory access rate⁴⁹. Rice, who suggested the change, later said that his programming job would have been impossible without it⁵⁰.

The second hardware modification to Voyager's data computer led to a first in spaceflight computing: volatile memory. After the first round of prototype programs, an intermediate hardware design evolved using CMOS ICs⁵¹. This type of circuit is very low powered, fast, and can tolerate a wide range of voltages, making it excellent for space use. Early in the 1970s, CMOS was still relatively new, so it was with some risk that JPL chose the circuits. To go along with the new CMOS processor, the data computer group fought for CMOS memories as well. Trying to drive a slow plated-wire memory with fast CMOS circuits would have negated the attempt to speed up the computer. However, CMOS memories are volatile, in that if power is cut off, the data stored in them disappear. The designers of previous manned and unmanned spacecraft avoided volatile memories, fearing that power transients would destroy the memories at critical mission times. Voyager management had to be convinced that the risk was acceptable.

James T. Kinsey, a JPL manager, was instrumental in getting the semiconductor memory accepted because a method of providing backup power was devised⁵². Voyager's primary electricity is alternating current. The radioisotope generators produce direct current, which is converted. By running a separate power line from the direct current bus fed by the generators to the CMOS memories, the only way power would be lost is if a major catastrophe destroyed the generators. If that happened there would not be any need for a data computer anyway. Enough voltage is supplied to retain the information in memory and in the registers in the processor that contain the state vector⁵³. Success with the CMOS memory led to the adoption of all CMOS circuits in both computer systems on the Galileo spacecraft. Along with the new chips, the memory changed with an expansion to 8K. Two "external" address bits were added to flag whether the top or bottom half of the memory is being accessed⁵⁴. One bit is used to select the memory half used for data access; the other, for the half used for instruction access.

Eventually, the cycle of prototyping and interaction between Rice and Wooddell stopped as a final design was accepted. Wooddell wrote that the extensive use of breadboards instead of paper designs optimized the process⁵⁵. His method, although not strictly "software first" was certainly software sensitive. Martin-Marietta's experiences with a software first philosophy as described in the previous chapter indicate that Wooddell had a clearer idea of his objective than did Martin. The job done on the Flight Data Systems computer is a good model of fine engineering practice in developing a total system.

Voyager Flight Data System Software

The original software development for the data computer has essentially been a two-man show since 1975, beginning when Edgar M. Blizzard joined Richard Rice to develop the flight version of the code. Others have been involved in testing and management, but these two JPL engineers have been the key programmers for the entire mission to date. They sit in the same area as the "Laboratory Test Set," an Interdata computer and peripherals that contain the software simulator of the data computer and the assembler and flight load generator. Across from them is the CDL, the loose conglomeration of hardware that represents the real spacecraft. From start to validation to release, their tools were within sight, and certainly hearing, since the room is filled with the constant hum of spinning disks, occasional clattering printers, and the undefinable sound of computers crunching numbers.

Rice characterized the unique nature of the data computer software this way: "We didn't worry about top-down or structured;

Box 6-2: Voyager Flight Data System Computer Architecture

Voyager's data computer is different from most small general-purpose computers in several ways. Its special registers are kept in memory, permitting a large number (128) of them. Wooddell also wrote more powerful shift and rotate instructions because of data-handling requirements. Despite its I/O rate, the arithmetic rate is quite slow, mostly due to byte-serial operation. This means 4-bit bytes are operated on in sequence. Since the word size of the machine is 16 bits, it takes six cycles to do an add, including housekeeping cycles⁵⁶. If all the arithmetic, logic and shifting were not done in the general registers, the machine would have been even slower. Reflecting its role, in addition to the usual ADD, SUB, AND, OR, and XOR instructions found on most computers, the data computer has many incrementing, decrementing, and branching instructions among the 36 defined for the flight version of the machine⁵⁷.

Overall, the Flight Data System requires 14 watts of power and weighs 16.3 kilograms⁵⁸. Its computer needs just one third of a watt and 10 volts, less than the power required for a temperature sensor⁵⁹! At first the estimated throughput required was 20,000 16-bit words per second⁶⁰. By flight time, the instruction execution rate was 80,000 per second, with data rates of 115,000 bits per second, much higher than previous Flight Data Systems⁶¹. The dual processor/dual memory architecture of the command computer and attitude control computer is repeated in the data computer. There was no provision for automatic switch-over in case of failure. A command from the ground routed by the command computer is necessary for reconfiguration⁶². Note that the attitude control computer can be switched by the command computer without ground intervention because it is much more critical to retain orientation.

we just defined functions"⁶³. One important function is the software's provision of basic timing for the entire spacecraft, not just itself. It is also required to provide the capability to read out the memories of all three computers, under orders of the command computer⁶⁴. Don Johnson, the Cog Engineer, determined other requirements and interfaces with the scientific instruments. Rice called him "Mr. FDS," claiming that Johnson often knew more about the scientific instruments than the scientists themselves: "If someone forgot something, Johnson knew it"⁶⁵. Raymond L. Heacock, Voyager Project Manager, said that Johnson was largely responsible for the overall success of the system, including the design⁶⁶. Rice said that Johnson's ebullient style and competence worked well in the informal mode in which the data computer requirements were set, which was a fully iterative process. New software needs continued to be discovered during the mission, which is one reason why a programmable machine

was chosen. For example, at one point Rice and Blizzard were asked to create software to determine where the limbs of satellites were so that imaging could be started⁶⁷. Development of some programs was deferred until after launch, such as the Saturn encounter program, when better data on the telecommunications rates and specific science requirements would be available⁶⁸.

Allowing for constant change mandated certain controls over the data computer's memory. A limit of 90% capacity was set in 1976 by Frewing, the Software Cog Engineer⁶⁹. Though later abandoned, the constraint indicated the software management's early concern about memory overruns. Also, since the machine can directly address the lower 4K of memory, programs were to be kept there, with the upper portion for transient data⁷⁰. Later, the flight configuration of the computer evolved to one processor accessing both memories. Therefore, a copy of the programs is kept in the lower portion of each memory, but both upper portions are usable by the single processor as a scratch pad⁷¹. If dual mode is required, the memories are separated. Experience has produced increased confidence in the memories. At first, complete loads had to be sent when an update was done; recently, pieces of software have been allowed to be inserted in the programs. Full redundancy between the memories is not now automatically maintained⁷².

Box 6-3: Flight Data System Computer Executive

Like the command computer, the data computer has a simple executive. Time is divided into twenty-four 2.5-millisecond intervals, called "P periods." Each 24 P periods represent one imaging system scan line. Eight hundred of those lines is a frame. At the beginning of each P period, the software automatically returns to memory location 0000, where it executes a routine that determines what functions to perform during that P period⁷³. Care is taken that the software completes all pending processes in the 2.5-millisecond period, a job made easier by the standardization of execution times once the direct memory access cycle was added.

Voyager's Future

Voyager software development continued into the late 1980s. Kohl, Wooddell, Greenberg, Deese, Johnson, Kopf, and others closely connected with the hardware of Voyager's computers were then on other projects, but Rice and Blizzard and their counterparts on the command computer and attitude control computer were still program-

ming, preparing Voyager 2 for Uranus and Voyager 1 to discover the boundary of the solar wind. An increasing problem as the spacecraft recede from the earth is the reduction in the data transmission rate. The closer a spacecraft is to earth, the higher the bandwidth possible. Computer loads that once took minutes now take hours because error checking by retransmitting to earth is slowed. In the summer of 1984, a Flight Data System software load took 4 hours, and the situation cannot improve⁷⁴. Voyager Project officials decided to use the Flight Data System in dual processor mode for the first time for the Uranus encounter to provide image data compression. Thus, the information content remained high even though the transmission rate was grossly reduced⁷⁵.

Voyager's computer system did not carry on to the next JPL project. Galileo combined the CCS and the Flight Data System into a single Command and Data System. This is logical from JPL's standpoint because both systems are the responsibility of the same Information Systems Division. Attitude control is provided by a separate computer. Whereas Voyager was a functionally distributed system with dual redundancy, Galileo's Command and Data System contains computers that do true distributed processing and use a new concept of redundancy. That system may be a model for the future, as it can impact designs aimed at complex spacecraft with extensive data processing needs, such as the Space Station and Mariner Mark II, both due in the 1990s.

GALILEO—TRUE DISTRIBUTED COMPUTING IN SPACE

Project Galileo began at JPL in the late 1970s with the objective of developing an orbiter and probe for further exploration of Jupiter. Galileo will proceed toward Jupiter, launching a probe 5 months before arrival. Plans are for the probe to enter the Jovian atmosphere at a relatively low angle, using an aeroshell braking system for entry followed by a parachute system for final braking and descent. Due to the nature of the entry, an antenna large enough to send data directly to earth cannot be carried on the probe. Instead, it has to relay the data to the orbiter, which will fly a parallel path thousands of kilometers above. At the end of the probe mission, expected to last 60 to 75 minutes until the probe is crushed by atmospheric pressure, the orbiter will execute an insertion burn. For the next 2 years, the orbiter will fly by the four Galilean satellites (hence, the mission name), using gravity assists to change its path after each encounter.

Great demands will be placed on Galileo's on-board computer systems, because of both the nature of the mission and the design of the spacecraft itself. First, Galileo is a one-shot mission. Interplanetary probes have been mostly launched in pairs for the obvious

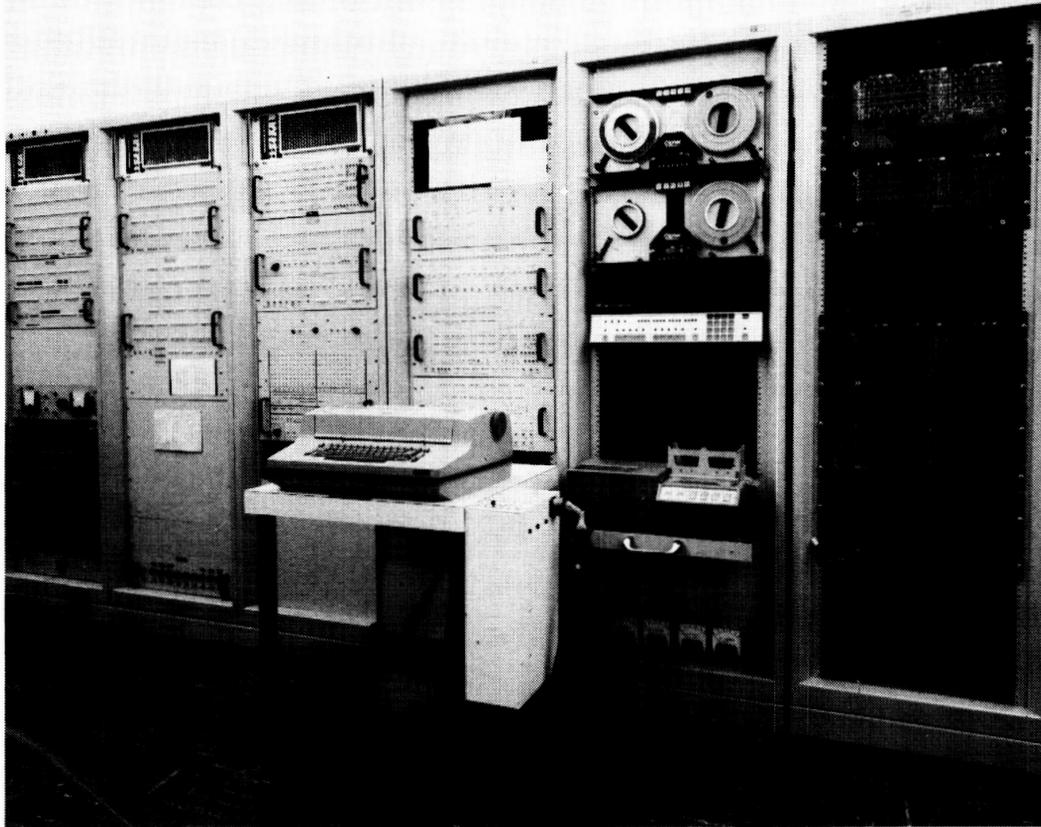


Figure 6-3. The Unified Data System: precursor to Galileo's Command and Data Subsystem. (JPL photo 360-630)

reason that a full backup then exists. Such dual launches are generally cost effective, as a second spacecraft can be obtained for 15% of the price of the first one⁷⁶. Budget constraints forced NASA to buy just one Galileo, so there was tremendous pressure to construct a highly reliable spacecraft. Additional pressure has been on the project because of changes in the launch date and booster rocket. Originally scheduled for a 1982 launch, delays in the Shuttle program and other factors caused rescheduling to 1984, then 1985 and, finally, 1986, when the grounding of the Shuttle fleet forced an indefinite postponement. At first, the Air Force's Inertial Upper Stage rocket was chosen for the booster. Later, the new "wide body" Centaur got the job. Centaur upper stages have flown on Atlas and Titan III boosters since the 1960s. The new "fat Centaur" would carry 50% more fuel than the earlier version. Other changes were made to adapt it to the Shuttle cargo bay. One JPL engineer said that it is "like Abe Lincoln's axe. The head broke and they replaced it and the handle broke and they put on a new one, but it's still Abe Lincoln's axe"⁷⁷. However, NASA canceled the Shuttle version of the new Centaur in the spring of 1986 due to safety considerations, leaving Galileo without a ride to Jupiter.

By early 1987, NASA decided to go back to using the Inertial Upper Stage, but it has significantly lower lifting capability than the Centaur. As a result, the flight path has to be changed to include a Venus flyby and two Earth flybys to gain velocity by gravity assistance. Unfortunately, the total flight time to Jupiter will nearly triple to about seven years.

Spacecraft design also caused problems for the computer designers. All previous JPL probes have been three-axis inertial to gain advantages such as easing communication and providing a stable scan platform for imaging. Galileo has a fixed attitude area, called the "despun section," and also a "spun section" that rotates three times a minute. Fields and particles experimenters required a spin to help them differentiate local fields from external fields. Aside from the obvious increase in the order of magnitude of the attitude control problem on a dual-spin spacecraft, communication between the two parts is hampered by the need to transmit serially across a rotary transformer. Four hundred milliseconds are required to send a message between the spun and despun sections⁷⁸. To overcome this time penalty and provide more real-time control, a fully distributed system of computers is mandated.

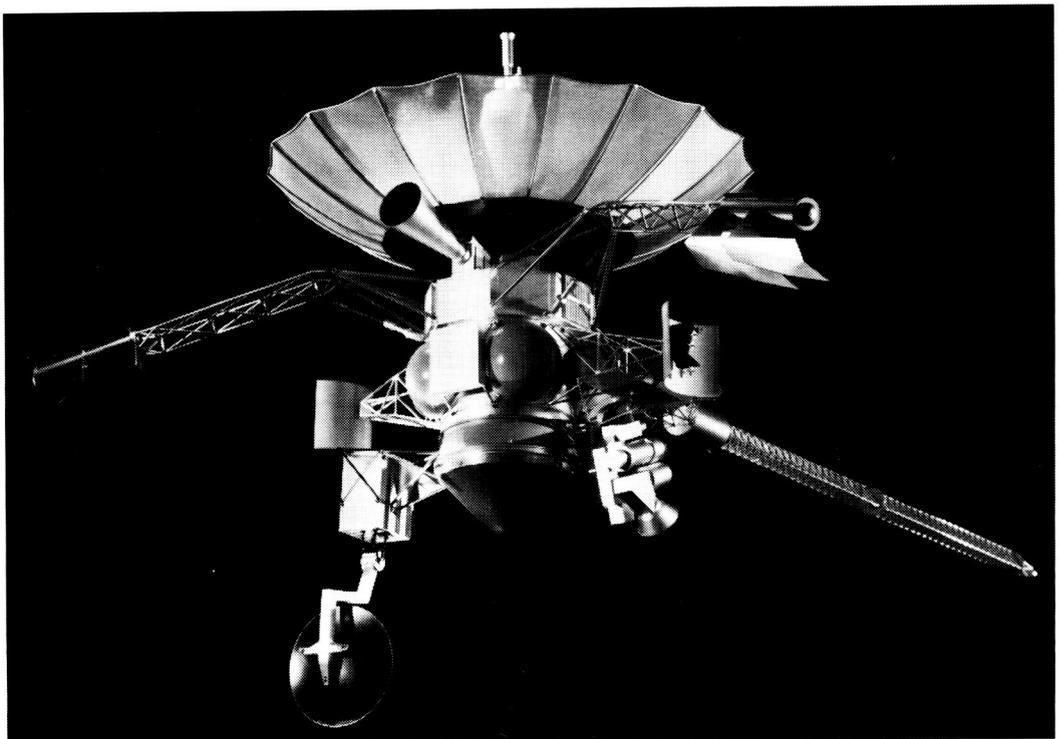


Figure 6-4. A model of the Galileo spacecraft with the probe visible at the bottom center. (JPL photo 230-222A)

ORIGINAL PAGE IS
OF POOR QUALITY

Finally, computer system complexity is further increased by the number of science experiments on board and the fact that they are largely computer controlled as well. Eight of the nine instruments have microprocessors for control and data handling⁷⁹. These have to communicate with the Command and Data System, itself containing six microprocessors. Attitude and Articulation Control has dual computers, and the probe also contains a dual microprocessor system. In all, Galileo contains 19 microprocessors with about 320K of semiconductor random access memory and 41K of read-only memory⁸⁰. No unmanned spacecraft launched to date can approach Galileo in the power and size of its on-board computer network.

JPL took great care in the selection of the computer systems for Galileo. Procurement of the systems for the Command and Data handling equipment and the attitude control equipment proceeded separately but in a somewhat coordinated fashion. In the case of the Command system, a 1977 study examined using the existing Voyager computers, the National Standard Spacecraft Computer (NSSC-1), and some form of a microprocessor distributed system like the Unified Data System (UDS), then a research program at JPL aimed at complex long-duration space missions⁸¹. Although a lot of pressure was exerted on Galileo's builders to choose either the existing equipment or the NSSC-1, cost factors favored the UDS as the basis for the Command and Data Subsystem⁸². Similarly, the attitude control group was pressured to use the NSSC-1, but the desire for floating point and greater power defeated that idea. Since the star tracker is in the spun section and thus moving, complex coordinate transformations must be calculated, and the NSSC-1 was not up to it⁸³.

With new computers needed for both major controlling subsystems, JPL carefully explored memory requirements and software development prospects. Prototype programs were written in HAL/S and FORTRAN for the command computer and the attitude control computer⁸⁴. Ideas for the content of the programs came from Voyager experience and the executive written for the NSSC-1. The project office originally specified that HAL would be used for programming all flight software. When irreducible inefficiencies appeared in the compiler bought for the command and data computers, HAL was abandoned for that system and replaced with "structured macros"⁸⁵. HAL was retained for the other computer system. Although most microprocessors in the scientific experiments are coded in assembler, one is programmed in FORTH, so high-level languages finally appeared on unmanned spacecraft. The project office set a limit of 75% memory usage at launch (later raised to 85% for the attitude control computer only) and 85% load at Jupiter insertion. Officials hoped to avoid the tight memory problems associated with earlier missions and even asked the Shuttle software office for advice in setting these limits⁸⁶.

JPL considered software crucial to the success of the overall Galileo mission. As of 1985, Neil Ausman of JPL was in charge of all software, both flight and ground support, and he reported directly to project manager John Casani. Patricia Molko, also of the project office, wrote a standards document for software development. At one point, Howard W. Tindall, first introduced in the chapter on the Apollo computer systems, was brought in to serve as a consultant. He found that in some ways JPL was "going through the same problems that we did when we developed our original large programs"⁸⁷. However, in some respects Galileo is more complex than Apollo because of the number of intercommunicating computer systems. Thus, size is not the only factor contributing to the difficulty of writing the software.

The Galileo probe mission is handled by NASA Ames Research Center, and the entry probe was assembled by Hughes Corporation. Even though it contains a dual microprocessor system, its function is primarily confined to sequencing, and its architecture is similar to systems already described on Mariner, Viking, and Voyager.

Galileo's Command and Data Subsystem Origins: The UDS

STAR was JPL's foray into ultrareliable computer research in the 1960s; the UDS was its 1970s counterpart. David Rennels, who had been instrumental in the STAR program, led in developing the system. Assisting him on the hardware side was Borge Riis-Vestergaard, a visiting scientist, and Vance C. Tyree. Frederick Lesh and Paul Lecoq did the software. One reason the UDS project started was the desire to develop a new architecture for flight computers that would reduce life cycle costs⁸⁸. Another impetus came from 1973 studies of distributed systems done in support of Voyager and by the Air Force⁸⁹. Distribution of functions among several computers on Voyager has been shown to be a natural result of requirements. The Air Force study found that avionics tasks are better handled by partitioning and using dedicated computers for specific functions. Since microprocessors became commercially available at about that time, they were recommended for use in such distributed systems.

Rennels' UDS project explored the difficulties in tying multiple computers together in a flexible manner. He defined an architecture using two levels of computers. Individual computers and associated memories at one level were called "high-level modules (HLM)." These computers controlled system-wide functions such as the data buses and fault detection. Other computers were located at specific subsystems and were called "terminal modules." They controlled one functional area such as engineering instruments or attitude and articulation. Each module had its own processor and memory. Com-

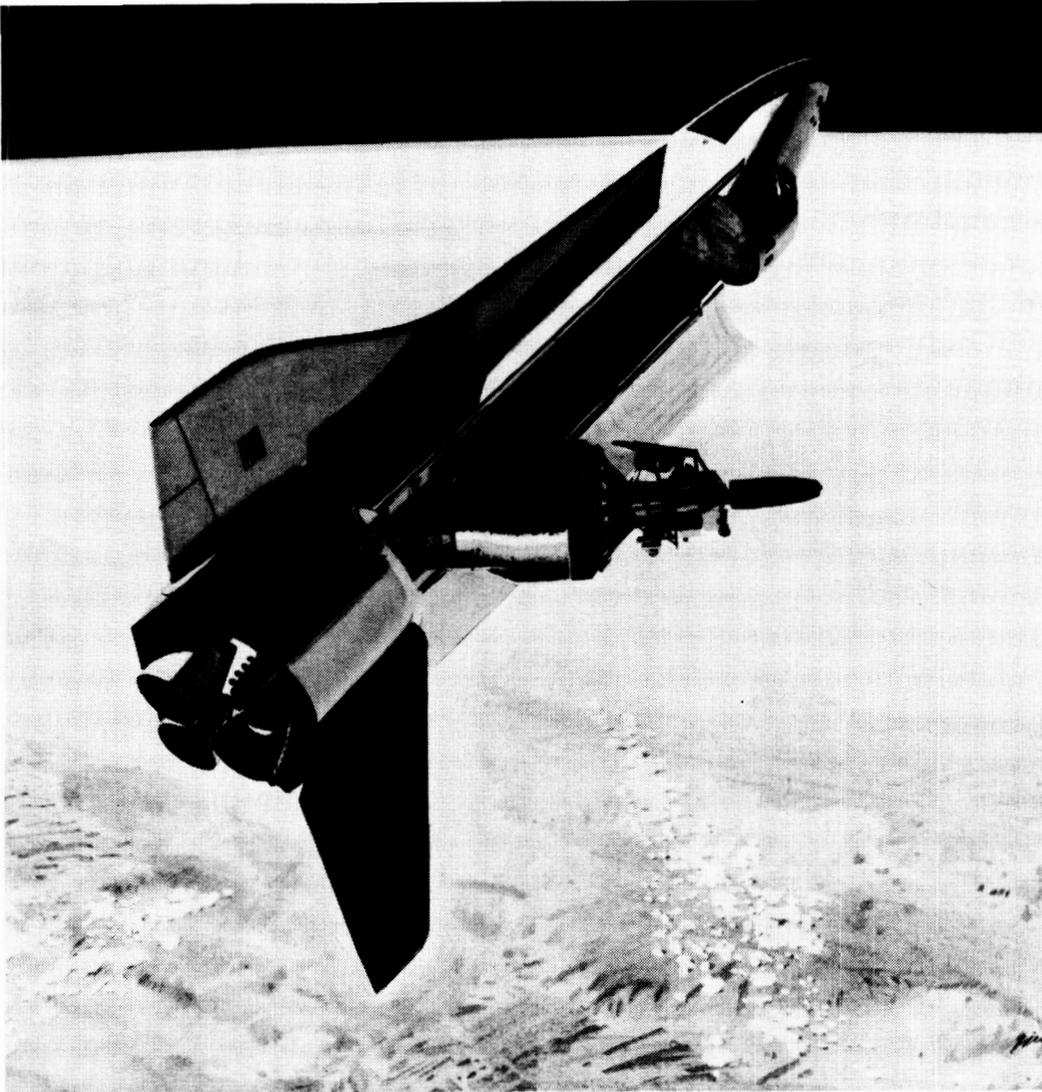


Figure 6-5. Galileo and its booster being deployed from an orbiting shuttle. (P25722AC)

munication between modules was accomplished on a bus that carried data from one memory to another. By using direct memory access for all intercommunication, processor resources other than for transfer commands were unaffected. HLMs did not have I/O capability other than to the terminal modules via the bus. All input and output to spacecraft systems and the ground was handled by the terminal modules.

Some influence from the STAR project can be noted in that Rennels kept critical functions highly redundant and simple. Reconfiguration after failures reflected STAR concepts⁹⁰. In order to avoid a potential single-point source of failure, there was no central bus controller in the UDS. Each of the HLMs controlled a separate bus, but

only one bus was needed to support all processors because any HLM could transfer data between any two memories⁹¹. The breadboard built for the UDS project had three HLMs and three terminal modules, so it had three buses as well⁹². Failure of a HLM caused its functions to be accepted by the remaining ones. Reliability is obtained by such reallocation of functions to resources, making this a highly fault-tolerant system.

One advantage of distributed systems is that interfaces can be simpler than in a system using a central computer. Each local computer is responsible for its own timing and control. Only on and off commands and data transfers need be made between machines⁹³. System-wide synchronization is accomplished by providing all processors with a real-time interrupt signal. By using a cyclic interrupt, the complexity attendant with priority interrupt systems is avoided⁹⁴. Every 2.5 milliseconds, a signal is sent to all components. Basically, every processor has to be finished with its current processes before the next interrupt occurs⁹⁵. Data transfers and scheduling of tasks can be timed using the periodic interrupt.

Key advantages of the UDS concept are that expanded requirements can be handled by adding terminal modules, software can be highly specialized and distributed, and fault tolerance is very high. Availability of flight-capable microprocessors in the mid-1970s made it possible for JPL to seriously consider the UDS as a competitor with NSSC-1 and the old Voyager equipment. Its flexibility and potential as a permanent architecture for space flight helped its case.

NASA chose RCA's 1802 microprocessor for the Galileo implementation of the UDS. A CMOS-type device, it was "nobody's favorite choice"⁹⁶ but at the time (c. 1977) was considered to be the only microprocessor suitable for spaceflight. Recall the use of CMOS components in the processors and memory in the Voyager Flight Data System. Similar advantages accrue with the use of CMOS microprocessors: low power requirements (30 milliwatts) and tolerance of a wide range of voltages⁹⁷. However, some disadvantages had to be dealt with. CMOS chips are especially susceptible to damage from electrostatic discharges⁹⁸. RCA 1802s are slow, with a 5-microsecond cycle time and an average of two cycles per instruction. In contrast, the discrete component Voyager CCS had a 1.37-microsecond cycle, making it faster for functions that did not require multiple cycles⁹⁹. Speed has been a major constraint to the software development¹⁰⁰. Carryover of the direct memory access cycle to Galileo from the Voyager Flight Data System alleviates this problem somewhat¹⁰¹. In general, making the six microprocessors "come together" has been much more difficult than originally expected¹⁰². Software is the most important component in achieving the success of the CCS.

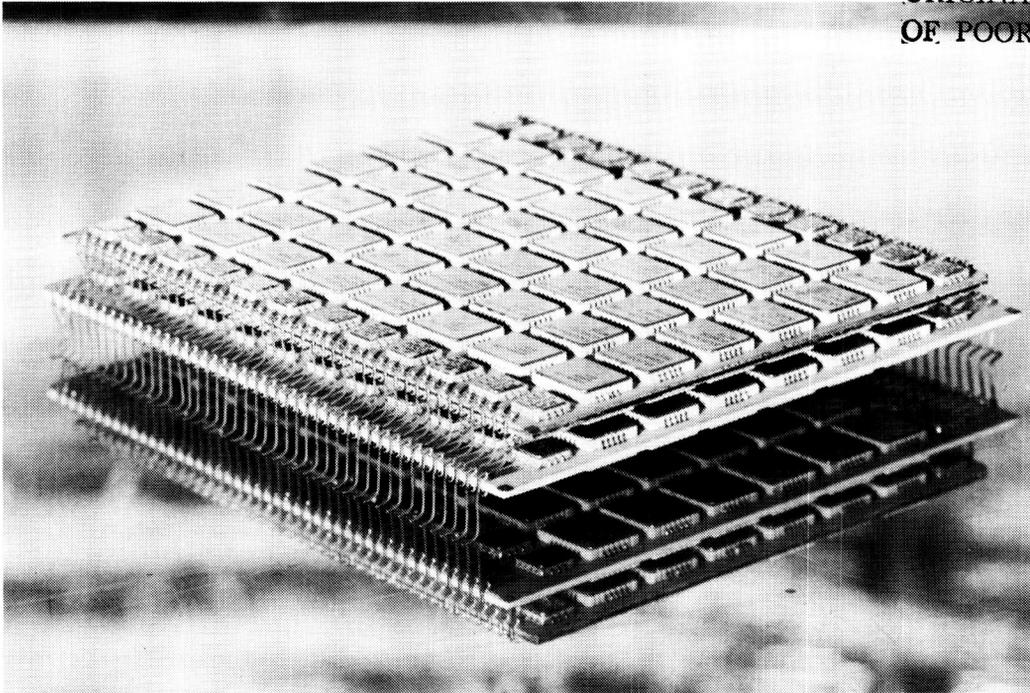
ORIGINAL PAGE IS
OF POOR QUALITY

Figure 6-6. Circuit boards for the Galileo Command and Data Subsystem. (JPL photo 360-1756)

Box 6-4: Evolution of the Command and Data Subsystem

Galileo's Command and Data Subsystem adapted UDS technology fairly directly. In 1978 designs, the Subsystem was shown as consisting of three HLMs and four low-level modules (LLMs), which were the realization of the terminal modules. Three buses were also present, each controlled by a HLM. Functionally there was one HLM dedicated to stored sequence control, one for real-time control, and the third as a spare. LLMs in that configuration handled sequencing, telemetry, status polling, and other subfunctions. Eventually one high-level processor was eliminated, but three buses remain, though one of them is used for test equipment only and will not function in flight¹⁰³. Software architecture is now much different than the UDS.

Box 6-4 (Continued)

One major difference from the UDS concept is the way the processors in the Subsystem are separated into redundant strings. Whereas reconfiguration after a failure was done by combining any of the remaining processors into a new control string in the UDS, on Galileo two basically identical strings are configured from the start, one backing up the other much like the backup processors on Viking and Voyager. Each HLM has 32K of memory and a bus controller associated with it. A LLM with a processor and 16K of memory for engineering control is connected to the string, along with a data bulk memory (DBUM) of 8K and a bulk memory (BUM) of 16K. These components are in the spun section of the spacecraft. Another LLM with 16K of memory is in the despun section, connected to the probe and the launch vehicle, among other functions¹⁰⁴. This configuration of one HLM and two LLMs, a BUM and a DBUM is repeated in string B. Therefore, the total system consists of six microprocessors with 176K of semiconductor memory. About 12K of HLM memory is write protected and used for programs¹⁰⁵. BUMs are used for auxiliary storage and buffering in that all new sets of commands are directly inserted into them from the ground and then redistributed by the software in the HLMs. Data memories serve as buffers for incoming science instrument data, again with direct memory access¹⁰⁶. Commands and data are transferred on the buses using packets with three-word headers. Headers contain the code numbers of the source and recipient, the starting address in memory of the message, and fillers for timing. More than one address can be specified for a message, but usually there is only one recipient¹⁰⁷. Data transfer is coordinated by the real-time interrupt. Odd-numbered real-time intervals are used for input; even numbered intervals for output.

As in previous missions, several operating modes are available for the Command and Data Subsystem. During cruise and other noncritical mission phases, one string is up and running and the other is in a quiet state. Otherwise both can be commanding in one of several ways. Dual string mode means that the strings are executing code concurrently and both send commands. Parallel mode is used for time-critical operations needing closer synchronization. Tandem mode is used during maneuvers. If a failure is detected in one string, the other halts the activity¹⁰⁸.

Developing the Command and Data Subsystem Software

Development of the software for this Subsystem consumed more time and labor than any previous unmanned spacecraft. Dr. John Zipse, the Cog Engineer for the Subsystem software, had an average

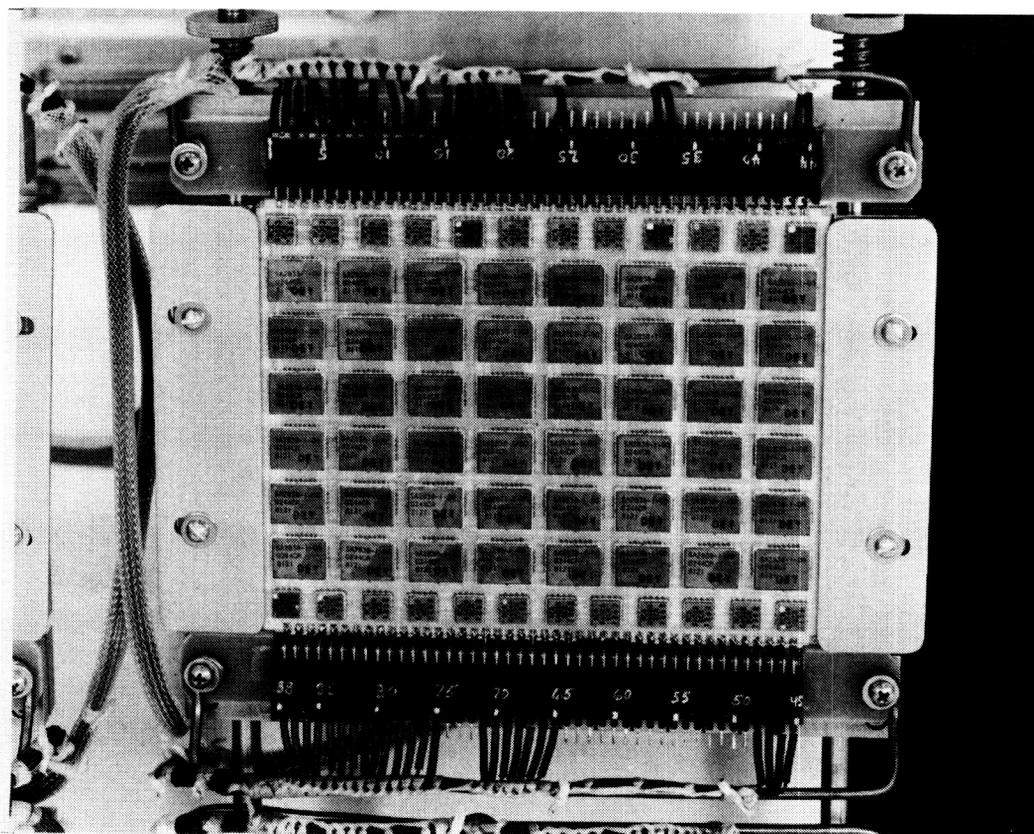
ORIGINAL PAGE IS
OF POOR QUALITY

Figure 6-7. Memory modules for the Galileo Command and Data Subsystem. (JPL photo 360-1704)

of 12 full-time software developers working under him at peak periods¹⁰⁹. They shared five terminals hooked to an IBM System 370/158 computer on which the assembler and functional commands resided. Originally, HAL/S was specified as the programming language for the Subsystem¹¹⁰. A prototype compiler for the RCA 1802s was not successful, and HAL was dropped in favor of "structured macros"¹¹¹. Called "functional commands" in the software documentation, they have names such as IF, ELSE, DO, ASSIGN, and others very similar to the statements of a high-level programming language¹¹². These functional commands make up the "Virtual Machine Language" in which most of the software was written. Each command causes the execution of a prepared block of 1802 assembly code, much like a subroutine call. Project documents recognize three layers of language associated with the Subsystem: Level A is the hardware external to the 1802s that may provide input and receive output, Level B is the 1802 assembler, and Level C is the Virtual Machine Language¹¹³.

Recognizing the complexity of the software, JPL instituted ever

ORIGINAL PAGE IS
OF POOR QUALITY

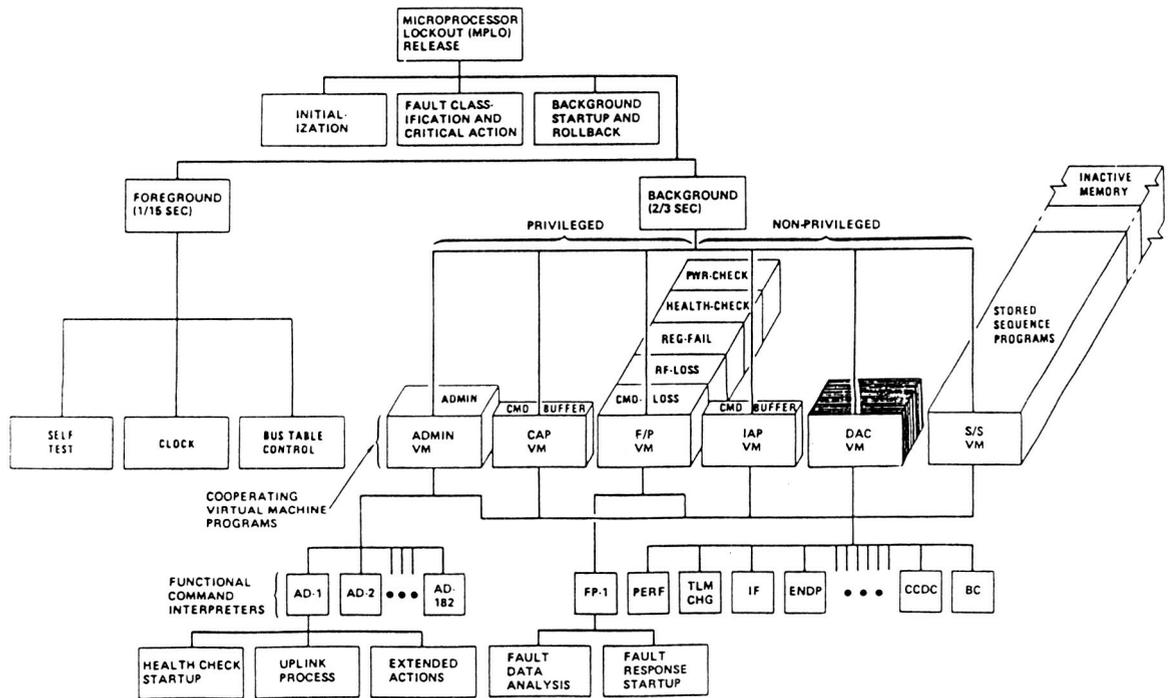


Figure 6-8. The software architecture of the Galileo Command and Data Subsystem High Level Modules. (From JPL, 625-340-006000)

more stringent development requirements. Preparation for the development process began with a "Galileo Software Thinking Group" which met in 1977-1978¹¹⁴. Programmers were ordered to keep software modules smaller than 150 assembly language statements and were reminded that simplicity was the highest priority¹¹⁵.

Box 6-5: Command and Data Subsystem Software Architecture

Galileo's Command and Data software is considered a "hierarchical software architecture" and is divided into two sets of processes, with further divisions within them. Foreground processes are executed at each real-time interrupt, or every 1/15th second. They are a self test, a clock, and bus control. Background processes begin each 2/3 second, and are much more complex. Functions done in the background have been divided into six "virtual machines," a term of many meanings. In this case, three virtual machines are considered "privileged": the administration machine, the contingency action program machine, and the fault processing machine. These three machines consist of software that is always resident in the Galileo computers and is kept in the 12K of write-protected memory¹¹⁶. They are called privileged because the non-privileged machines can be canceled if they do not complete processing before the end of the 2/3-second cycle, whereas the privileged machines can never be canceled. Nonprivileged machines include an immediate action program machine, a delayed action program machine, and a stored sequence program machine. So the nonprivileged machines are reserved for commands and sequence control, whereas the privileged machines are for executive and fault detection and correction. Nonprivileged software is to be updated about once a week in flight¹¹⁷. Originally, the immediate action programs were considered privileged, but with the addition of a contingency machine they were moved to nonprivileged status¹¹⁸. Software developers imagine that a "wall" exists between the privileged and nonprivileged machines. They consider that the non-privileged software is more error prone because it is constantly changing, whereas the privileged software should have had a thorough exercise over several years in testing before the flight.

Execution of the virtual machine software is related to the 2/3-second interrupt. In each cycle the software goes through each virtual machine pending program stack and executes what is waiting. Many programs can be running in each virtual machine in each cycle, up to 10 in the administration and fault protection machines, for example. As mentioned above, the privileged machines always get to clear their pending programs, whereas the nonprivileged machines do what they can until the time is up.

Software described so far is resident in the HLMs. LLMs have specialized software for their particular functions, such as monitoring engineering instrumentation and talking to the launch vehicle. Data from those tasks needed by the virtual machines are passed on the buses during the 1/15-second interrupt.

Design of the software was done in a JPL-developed Software Design and Development Language that used statements similar to those in high-level languages¹¹⁹. Even with excellent documentation and tools, such as a hardware-based simulator for software validation¹²⁰, it takes (according to Zipse) a new programmer three months to be effective. Until Galileo has flown, no final evaluation can be made of the virtual machine architecture. Yet, future spacecraft requiring expandability and a high degree of flexibility could probably gain from using such an architecture as a complement to the UDS type hardware structure.

Galileo Attitude and Articulation Control Computer System

In terms of tasks, the Attitude and Articulation Control System has less to do than the Command and Data Subsystem, but it must perform its jobs faster and with more critical tolerances. During the discussion of the Voyager computers, it became clear that the attitude control system needed a fast-cycle, real-time software architecture running on a high-speed computer. Galileo's control requirements are much greater than Voyager's; the dual spin problem and more complex imaging equipment indicated from the beginning a need for a completely new computer system. The computer was to provide star-based attitude determination and control an inertially referenced, target-body-tracking scan platform¹²¹. Speed was the primary criterion for the new processor¹²².

Kenneth Holmes, in charge of looking for the Galileo control computer¹²³, and the other engineers ran old Voyager attitude control programs on several processors. One of those processors soon proved itself superior: Itek's 2900 series¹²⁴. Itek, now a division of Litton Industries, built a computer known as the ATAC, or Advanced Technology Airborne Computer. Using 2900 series processors, each with 4-bit words, Itek assembled a 16-bit, low-power, flying minicomputer roughly equal in power to a Digital Equipment Corporation PDP-11/23. Navy aircraft use this computer, although its specific applications are classified¹²⁵. ATAC's basic cycle time is 250 nanoseconds, or more than five times faster than the Voyager computer's cycle. However, the memory cannot cycle faster than 2 microseconds, so operations rates average 143,000 cycles per second¹²⁶. Floating-point capability is a plus, and since it handles eight interrupts using microcode, there is no software overhead for real-time operation¹²⁷. Another good feature is that its 16 registers are general purpose; none are dedicated as accumulators, program counters, address registers, and so on. Therefore, multiprocessing is made much easier. Further advantages to the computer are that special



Figure 6-9. The central processor and input/output circuits of the Galileo Attitude and Articulation Control Subsystem. (JPL 230-1128bc)

instructions can be added by the user for specific applications. Four instructions added by the Galileo project saved over 1,500 words of code in the flight program¹²⁸.

The ATAC came with a considerable amount of software support. Target compilers were available for FORTRAN, BASIC, and HAL/S¹²⁹. Galileo project management wanted a higher order language used in the coding, so HAL was adopted for the attitude control system. Unlike the Command and Data Subsystem, HAL was successfully adapted to the ATAC. Compilers developed by Intermetrics for the ATAC are about 12% speed inefficient, but 36% memory inefficient compared with assembler¹³⁰. Apparently this was within acceptable limits, and the flight applications code is written in HAL, with the operating system in assembler. Edward H. Kopf, Jr. said, "We love HAL/S; we could never do it without HAL/S," even though he referred to it facetiously as "flight PL/I." Given the complexity of the resultant software, he was probably right that a high-level language was critical to success.

Attitude Control Electronics Software Organization

Implementing HAL/S on the ATAC involved creating a special operating system. Ted Kopf wrote GRACOS, or the Galileo Real-

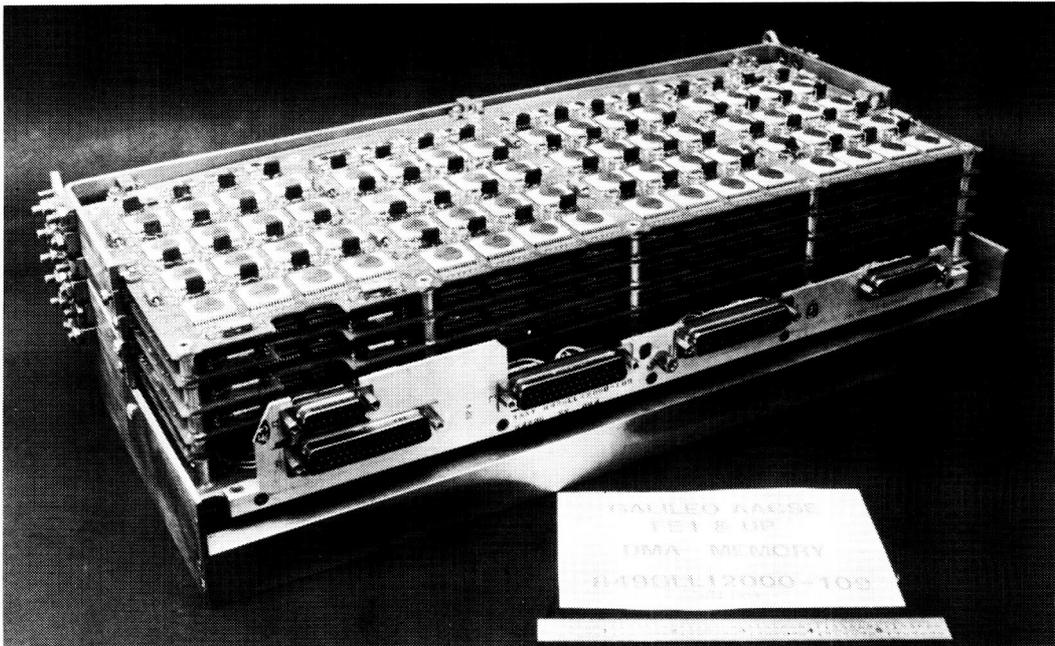


Figure 6-10. Memory modules of the Galileo attitude control computers. (JPL 230-1121bc)

time Attitude Control Operating System, to accomplish that implementation¹³⁵. GRACOS begins the operation of the software by scheduling a HAL/S module called STARTUP. Within STARTUP are statements that set up the concurrent processes necessary to do the attitude control and articulation tasks (a version of STARTUP is reproduced in the HAL/S appendix as an example of the language). Up to 17 concurrent processes may be running under GRACOS, with at least 10 up all the time¹³⁶. STARTUP is given as much time as necessary to complete, and then the established processes begin at the next 1/15-second real-time interrupt¹³⁷. Kopf wrote GRACOS to be mission independent. He avoided constraining the timing of rate groups and other things that would have been too specific¹³⁸. Interestingly, if the fault-handling routine has to come in, it restores the registers of the failed HAL/S module where the fault occurred and tries again, very similar to the "roll back and try again" scheme in STAR¹³⁹.

Sanford M. Krasner, the Software Cog Engineer for the Attitude Control Electronics, reported directly to Brian T. Larman, the Spacecraft Flight Software System Engineer. Early in the project, five people were full time on the software development, several more than on Voyager¹⁴⁰. The coding process was speeded up when HAL was used as the software design language¹⁴¹. To make development easier, Voyager structure was adopted whenever possible¹⁴².

Box 6-6: Configuration of the Attitude Control Electronics

Galileo's Attitude and Articulation Control Subsystem necessarily has parts in both the spun and despun sections of the spacecraft. Most components are in the spun section, including the two redundant processors and 64K of memory. Called the ACE, for Attitude Control Electronics, its despun partner is the DEUCE, for *Despun Section Electronics*, a long way to go for an acronym. Communication between the ACE and DEUCE is across the rotary transformer. Since the transformer is not considered fully reliable, input or output to the DEUCE is not complete until an ACKNOWLEDGE interrupt reaches the ACE¹³¹. CMOS-type memories similar to those used in the Voyager Flight Data System are in the ACE. Sixty-six 1K chips are needed, two of which are actually permanent read-only memory. Those two contain the Memory Loss Recovery Routine written in ATAC assembler¹³². A 5-volt keep-alive current directly from the generators is constantly fed to the memories, but in the case of a destructive transient the Recovery Routine can restart the software after it has been repaired or replaced.

Planning for the system included careful memory sizing. Based on actual Voyager programs and extrapolations from them to handle the new requirements, a 1978 study thought 10K of memory to be sufficient. Using a policy of 100% margin and 75% limit at launch, 32K was eventually bought¹³³. As noted above, a waiver to 85% full at launch was given. Ground computers can reprogram the ACE in flight by sending code to the off-line memory through the Command and Data Subsystem. As with its own LLMs and as pioneered in the UDS, the command computers can directly access the ACE memory. Commands can be placed in the active memory where they are "discovered" by the ACE software¹³⁴.

Ironically, the advanced nature of the new attitude control system and its control computer made it more vulnerable to space conditions than its predecessors. A potential disaster was averted when the "single event upset" was discovered and dealt with before launch.

The Single Event Upset Problem

Space environments are much harsher to electronics than the surface of the earth. Since circuitry essentially consists of hardware that moves electrons, creates and destroys magnetic fields, and emits waves of electromagnetic radiation, fields and particles of the types loose in space can affect the operation of electronic equipment. Ironically, the miniaturization of components has made electronics more sensitive to interference. One possibility that concerns computer

designers is the effect of highly charged particle impacts on memory cells. If a particle has sufficient energy to change the information stored in a bit, it can affect the software in a potentially disastrous way. Such a spurious change is called a single event upset (SEU). Sufficient numbers of particles can cause so many bits to change states that the software fails. Since primary and backup memories are equally vulnerable, simple redundancy is not a solution. Error-correcting codes that test for random bit flips exist but require storage and processing time not always available on a spacecraft. Additionally, bits in the processor can be affected during execution, so the problem is not limited to memory.

Galileo's processors and memories were chosen in 1977. Voyager had not yet reached Jupiter, so hardware decisions were based on 1973–1974 Pioneer data¹⁴³. Nothing was known about SEU vulnerability, so no space for error detection and correcting codes and no provision for special shielding was made. Some incorrect imaging commands sent by sequencers in the Pioneers were later tagged as SEUs. Voyager's clocks were slowed by Jovian radiation so that the computers were forced out of synchronization occasionally¹⁴⁴. By 1980–1981, the nature of the SEU problem became apparent. Sulphur ions from Jupiter's volcanic moon, Io, were being whipped up to high energy by the Jovian gravity. In 1982, Galileo Project Chief Engineer B. Gentry Lee was assigned the job of determining how bad the SEU problem could be and finding a solution. Lee arranged for cyclotron tests at the University of California's Berkeley campus in which computer and other electronic parts were submitted to bombardment by high-speed particles. Results indicated that the 2901 chips used in the attitude control computers were highly SEU sensitive, with 20% 50% of hits causing probable software failures¹⁴⁵. RCA 1802s used in the Command and Data Subsystem were actually much less sensitive, being of older, and thus less dense, technology.

Attitude control engineer Kopf commented, "It is not worth flying the mission if you cannot get rid of the SEU problem." Failures were most likely at the most critical part of the probe mission when the orbiter is very near Jupiter. In order to avoid possible further delays in an already much postponed mission, Lee searched for solutions along two tracks. One solution would use a radiation-hardened processor built by Trecor called the RHEC—Rad Hardened Emulating Computer—1750A. Even though it is an emulator capable of imitating the 2901, a new retargeted HAL compiler would be needed. The cost of this solution would be \$20 million. Lee's second solution was to contract with Sandia National Laboratories to custom make radiation-hardened 2901s. No software needed be changed, just new ICs were necessary, and they cost \$5 million. Due to cost considerations and the inherent attraction of retaining the already created and largely validated software, the Sandia solution was chosen¹⁴⁶. As a footnote,

it is interesting that if the Galileo had launched on time, a sufficient understanding of the SEU problem would not yet have been available, and a doomed spacecraft carrying an unknown time bomb would have been traveling toward an unfriendly Jupiter waiting to hurl ion thunderbolts at it.

FUTURE UNMANNED SPACECRAFT COMPUTERS

Distribution of computers aboard spacecraft has now been done several times. Both Voyager spacecraft inherited command computers from the Viking project. Computers for specific functions such as attitude control and data formatting were added in response to increased requirements. The result was a functionally distributed system of processors. Galileo's project managers also adopted the concept of functional distribution, assigning microprocessors to control attitude and, in the lower-level modules of the Command and Data Subsystem, to connect to engineering and other instruments, including the scientific experiments. Additional innovations on the Galileo spacecraft centered on the development of virtual machine software, which distributes functions over several processors.

Advancing microprocessor technology makes the continuation of the concept of single function computers more attractive. At JPL, plans are currently under way for the Mariner Mark II, which will be the deep space version of the Multimission Modular Spacecraft developed by Goddard Space Flight Center for earth orbital operations. Using the same concepts of a standard bus and modular equipment, JPL hopes to reduce mission costs to \$400 million each, about half the price of Galileo¹⁴⁷. The staff is exploring the use of the C programming language, a very powerful tool, for the new spacecraft. Future missions seem certain to use multicomputers, with internal networks similar to Galileo's. In the 15 years since the first primitive programmable sequencers flew with 128 word memories, JPL spacecraft have grown to carry *2,500 times* more memory. Progressing from simple counting to complex coordinate transformations in such a short time is remarkable, and the application of computer power to each spacecraft function will make for ever more remarkable gains.

Part Three:

Ground-Based Computers

For Spaceflight Operations

NASA's ground computer systems are characterized by large size, by the implementation of real-time programming, and by the use of many computers connected together. The need for these three attributes has caused NASA and its contractors to devise new techniques for computer applications, such as operating systems for mainframe computers capable of handling real-time processing and sophisticated networking. Through these developments NASA has had its largest impact on computing in the commercial world.

Differences between ground-based computers and on-board computers center on the relative ease of hardware procurement with the continued difficulty of software development. On-board computers evolved from custom-made systems to the largely off-the-shelf Skylab and Shuttle computers. Ground computers followed a more conventional line, as they could be, from the beginning, commercially available systems, though applied to noncommercial tasks. NASA examined many existing computer systems each time it needed a machine. In fact, the government's bidding process gave NASA a larger mix of different vendors' equipment than most commercial enterprises, causing occasional difficulties in connecting computers together. This problem and that of adapting business machines to real-time processing were largely solved by software. Contractors received invaluable experience in large systems development and networking in the process of achieving NASA's goals.

Ground-based computer systems are used for preflight checkout and the launching of space vehicles, controlling both unmanned and manned missions, creating simulations of rocket flight for vehicle development and of space flight for crew training, processing telemetry data from launch vehicles and space probes, and in basic research. In the following chapters these functions are grouped into launch processing, mission control, and support tasks. Chapter 7 develops the concept of launch processing from the manual era to the fully automated Shuttle flight preparation. The chief result from this effort was a large integrated network of computers that proved to be highly innovative. Chapter 8 presents computer systems in both the manned Mission Control Center in Houston and the unmanned control

centers at the Jet Propulsion Laboratory (JPL) and Goddard Space Flight Center. In Chapter 9, the uses of computers in simulations and data reduction are discussed.

7

The Evolution of Automated Launch Processing

Rocket technology is both old and new. Since the Chinese first started shooting off fireworks a millenium ago, the sight of a rocket streaking ever faster skyward, a comet's tail of fire behind, has excited even those unimpressed with machines. Fireworks rockets, and, later, military bombardment rockets through the first three decades of this century, shared the same components: casing, fuel, and payload. Construction was complete when the gunpowder fuel was loaded in the casing, warhead affixed, and a fuse planted the base. Such rockets could be stored without maintenance and fired with little preparation, needing only to assure that the fuse was still attached. The difficulty came in the area of guidance. A set of fins or a balancing stick passively guided the early rockets. Frequently they would turn on the men who launched them or shoot horizontally over the heads of fireworks watchers. Thus, the old technology of preparing rockets for flight consisted of keeping them dry, aiming them carefully, and lighting the fuse.

In Germany during the late 1930s the new technology of rockets began to mature. Increased interest in rocketry developed in Europe and the United States after World War I. Rocket societies flourished in England, Germany, and the United States. Robert Goddard flew a liquid propellant rocket, the first of its kind, in Massachusetts in 1926. Liquid fuels, with their higher specific impulse and thrust potential, soon replaced solid fuels as the primary area of propulsion research. Shortly after Hitler came to power, the German army established a rocket development program that led to the liquid-propellant A-4 (popularly known as the V-2). A-4 rockets far exceeded the capabilities of previous ones, terrorizing the populations of London and Antwerp in the latter stages of World War II. Over 14 meters tall and weighing over 12,000 kilograms, an A-4 carried nearly 1,000 kilograms of explosive payload up to 400 kilometers. Its guidance system was a radio beam-rider type with an electronic analog computer controlling vanes in the exhaust and elevons on the fins. If wind deflection caused the rocket to veer horizontally off course, the analog computer would calculate corrections and activate the vanes. Complex plumbing and turbopumps were needed to feed the engine with fuel. Experience gained in nearly 2,000 expensive failures led German technicians working on the A-4 to develop techniques of testing the many components of the rocket during manufacture and before committing it to flight. For example, the guidance system was tested at the factory by an electronic analog computer that simulated the flight of the rocket so that the system's reactions could be observed¹. On the launching pad, engineers could test various moving parts of the vehicle by activating them using actual physical connections to the firing room.

German rocket scientists who came to the United States after World War II brought this new technology with them. Eventually based in Huntsville, Alabama, at the Army's Redstone Arsenal, they

conceived an increasingly sophisticated series of rockets: Redstone, Jupiter, Juno, and Saturn. Concurrently, the Air Force chartered the Atlas, Titan, and Thor ballistic missiles. During the 1950s, each of these vehicles was developed in programs marred by frequent flight failures. Actual numbers and the complexity of components grew by several factors over the A-4. The new devices and their failures led to more testing, both at the factory and before launch. The concept of a "countdown," during which each flight-critical component of the vehicle is systematically checked, reached a high level of efficiency.

As the 1960s began, most rockets and their payloads were still being checked out by discrete connections between the components and a test panel. When the countdown reached an advanced stage, particularly after fueling, the test engineers were cloistered in a blockhouse. Through cables from the rocket to the blockhouse, the engineers could monitor the status of various components and activate tests. An engineer would flip a switch, and something would happen, either on a dial or a strip chart, that he could actually see and interpret. When the first Saturn I rockets were launched and the Mercury spacecraft made their appearance, both in 1961, it became obvious that the level of complexity of both vehicles and payloads had reached the point where manual test methods were inadequate. Individual NASA engineers and managers on different programs began to evaluate the possibility of automating some of the checkout procedures using digital computers. Eventually, this led to the Shuttle's fully automated Launch Processing System.

The heart of the Shuttle is its computer system. Without it, no component of the spacecraft could be adequately tested or monitored. When a Shuttle is being refurbished after a flight in the Kennedy Space Center's orbiter Processing Facility, a large double hangar near the landing runway, the spacecraft's computers are connected to checkout and launch computers located in a firing room in the Launch Control Center. When moved to the Vehicle Assembly Building for mating with its fuel tank and solid propellant boosters, the Shuttle is reconnected to the firing room. After being transported to the pad, the final preparations are also controlled from the firing room. Finally, countdown and launch are executed from the same firing room. This scenario came after two decades of evolution, during which the role of computers became dominant both on board spacecraft and in launch processing. The integrated techniques exemplified in the Shuttle Launch Processing System developed from separate automated systems devised for vehicle checkout, spacecraft checkout, and telemetry monitoring. Important in the evolution is the part played by on-board computers. The journey toward full automation got great impetus from the Saturn and Apollo programs.



Figure 7-1. Launch processing facilities at the Kennedy Space Center: the Shuttle Orbiter Processing Facility (left), the Vehicle Assembly Building (center), and the Launch Control Center (right). (NASA 116-KSC-377C-82/41)

LAUNCH PROCESSING IN THE SATURN ERA

A Saturn V rocket with an Apollo spacecraft on top presented a magnificent sight, which engineers nonetheless viewed with a mixture of prideful awe and dread. No earthly booster since the Skylab launch has been as large or as powerful. Shuttles being mated in the Vehicle Assembly Building originally designed for the Saturn appeared as dwarfs in houses made for giants. It looked as though there was nearly enough room to stack them two high. The dread came from the fear of failure among the thousands of components, many capable of bringing disaster and killing a crew in flight. Early in the Saturn program automation began to be introduced in the testing of the gargantuan rockets. Marshall Space Flight Center acquired computers for Saturn vehicle checkout. Marshall also had responsibility for the Launch Vehicle Digital Computer (LVDC) housed in the Instrument Unit that

was the last stage below the Apollo spacecraft on both Saturn IB and Saturn V configurations. NASA's Launch Operations Center, later renamed Kennedy Space Center, acquired computers for telemetry data reduction and display and began work on the checkout systems used for the Apollo spacecraft, a project later transferred to the Manned Spacecraft Center in Houston. Each of these computer-controlled systems contributed to the concepts and development of the Shuttle Launch Processing System, now wholly based at the Kennedy Space Center.

Checkout of the Saturn Vehicle

Marshall Space Flight Center in Huntsville had primary responsibility for the design, manufacture, and flight preparation of the Saturn vehicles. In 1951, when Marshall was still the headquarters of the Army Ballistic Missile Agency, Kurt H. Debus formed a launch team that commuted to the Air Force's Eastern Test Range in Cape Canaveral, Florida. Within a short period of time, the frequency of launches made it necessary to establish a permanent group at the Cape, called the Missile Firing Laboratory. When Marshall was established on July 1, 1960, the Laboratory was renamed the Launch Operations Directorate. By 1962, the activities at Cape Canaveral grew to the level that the Launch Operations Center was formed separately from Marshall and given status equal to other NASA centers. However, its charter stated that the centers responsible for a particular vehicle or spacecraft had to perform its checkout and test, so during the Apollo era Marshall prepared Saturns and the Manned Spacecraft Center worked on the Apollos. Personnel at the Launch Operations Center performed facilities management and provided telemetry data reduction.

Computers were used both on-board the Saturn vehicles and in preparing them for flight.* Ten Saturn I vehicles were launched be-

*For a complete description of the evolution of the Saturn and its components, see Roger Bilstein, *Stages to Saturn: A Technological History of the Apollo/Saturn Launch Vehicles*, NASA SP-4206, 1980. Chapter 8 centers on the use of computers in checkout and the development of the Instrument Unit and its flight computer. Chapter 16 of Charles D. Benson and William B. Faherty, *Moonport: A History of Apollo Launch Facilities and Operations*, NASA SP-4204, 1978, describes the development of automated launch operations. Due to these prior treatments, my account will concentrate on briefly summarizing the use of the computers to provide the necessary introduction to the section on the Launch Processing System, rather than retelling the whole story. Some new evidence is presented where applicable, but the reader is urged to consult both previous works.

tween 1961 and 1965. Each was unmanned, the series being used primarily to demonstrate that clustered-engine first stages and high-energy upper stages were feasible. The first five launches did not use a computer for guidance. Each was a suborbital mission utilizing a German-made mechanical time-tilt device for control². On the fifth flight, an ASC-15 computer, built by IBM originally for the Air Force's Titan, flew as a passenger and handled telemetry transmissions³. It guided the last five missions, several into earth orbit. When Saturn evolved into the IB and V series, an Instrument Unit containing the LVDC was mounted atop the S-IVB stage on each vehicle. Termed the "integrating element" of Saturn, IBM was not only responsible for its computer but for its construction⁴. Besides the computer, the Instrument Unit contained the Launch Vehicle Data Adapter as an I/O front end, analog control circuits and an ST-124 guidance platform. On lunar missions the LVDC guided the spacecraft until the S-IVB stage separation after the lunar trajectory insertion.

IBM's LVDC was architecturally quite similar to the Gemini guidance computer⁵. It used nearly the same instruction set, 26-bit data words and 13-bit instructions. One difference was that the memory had two-syllable locations instead of Gemini's three. Construction of the LVDC, however, was radically different. For reliability reasons, triple modular redundant (TMR) circuits were adopted. Even though the component count went up just 3.5 times, the reliability increased 35 times⁶! Three logic channels, each with seven functional modules, required 395 voters⁷. Packaging the computer used techniques developed under the Advanced Saturn Technology Program commissioned by Marshall and executed by IBM⁸. First of the "flat pack" integrated circuit series, IBM applied this silicon semiconductor technology in its System 360 commercial machines⁹.

Use of a computer in the launch vehicle led directly to using one for checkout. Marshall bought an RCA 110 to communicate with the IBM ASC-15 used in the Saturn I. Later, RCA upgraded its machine by enlarging the memory to 32K 24-bit words of core and an additional 32K on an associated magnetic drum. When the Saturn IBs began to be launched, discrete circuits for interfaces with the rest of the launch vehicle were added¹⁰. Renamed RCA 110As, these computers continued to be augmented to handle more communications circuits, so that by the time Saturn Vs appeared, the computers could maintain the status of each of 1,512 signal lines¹¹. At first the 110s simply handled communications and switching. Activating test procedures and conducting tests were still done manually. But in 1962, IBM suggested that Chrysler convert the 110s they used for stage checkout of the Saturn I to do the tests automatically¹². Even though the advantages of automating procedures seem obvious, chief among them the fact that all are done exactly alike, it was difficult to get people responsible for checkout to convert from doing things

ORIGINAL PAGE IS
OF POOR QUALITY.

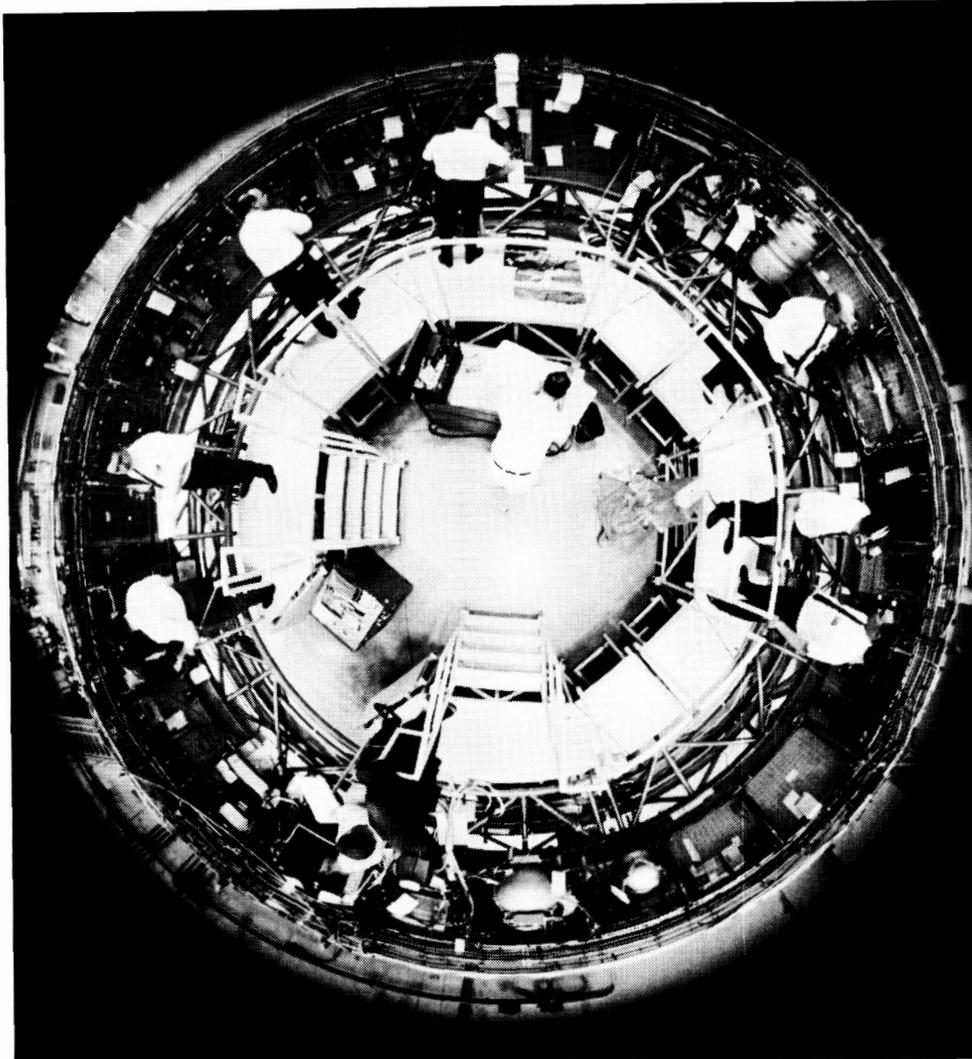


Figure 7-2. IBM engineers work inside the Saturn launch vehicle Instrument Unit. (IBM photo)

manually, a theme repeated in other parts of the Apollo program¹³. At that time, computers were seldom used for on-line work, and most engineers were still unfamiliar with them and wary of any more innovations in an already innovative program. However, Chrysler converted some factory tests to automatic, using a special language, "HYLA," to define them. Additionally, several Packard Bell computers connected to a common memory automatically checked out parts of the Saturn I. Use of a common memory as a computer interconnection device reappeared in several later systems and is critical to the success of the Shuttle's Launch Processing System. Engineers wrote the language "SOL," or Systems-Oriented Language, for the Packard Bell

machines. By late 1962, the Saturn V stage contractors accepted the concept of automatic checkout and settled on a common system, the Control Data Corporation CDC-924A computer, as the factory test machine, with 110As assigned to the S-I stage and for the assembled vehicle at the launch site¹⁴.

By this time, it was clear to Ludie Richard, a NASA engineer, and his team at Marshall that preparing a language to help test engineers write automated procedures was the key to continued acceptance of the principle. A custom-designed programming language would leave control over the definition of the tests in the hands of the engineers, avoiding communication problems that might arise with computer programmers inexperienced in checkout techniques¹⁵. IBM eventually wrote routines for the RCA computers in assembly language, but the majority of the automated tests were ATOLL (Acceptance, Test, or Launch Language) programs stored on tape. Richard acquired the over two dozen RCA 110As that were eventually used. His deputy, Charles Swearingen, was put in charge of managing the flight computer, ground computer, and checkout software¹⁶. James Lewis and Joseph Medlock were instrumental in developing the checkout systems and defining ATOLL¹⁷. IBM wrote both the flight programs and the Saturn Operating System that ran on the RCA computers and executed ATOLL procedures.

By mid-1963 the final configuration of the Saturn checkout computers was set by Richard's group. At Launch Complex 34, the Saturn IB launch site, one master RCA 110A was in the blockhouse and a slave underground at the pad. For Saturn Vs at Complex 39, one RCA 110A was located in each of the four firing rooms in the Saturn Launch Control Center, which was attached to the Vehicle Assembly Building in which the Saturns would be stacked. Each of four mobile launchers also contained a computer. In addition to the 110As, the firing rooms also had a DDP-224 minicomputer as a display driver for the CRTs showing output data to the engineers, as well as a controller for slides and other visuals. Computers in the mobile launchers could be used for checkout in the Assembly Building as well as at the pads, a foreshadowing of the later Launch Processing System. Due to reliability problems with the 110As, the launcher computers used a dual memory configuration. Checkout programs filled just half the memory, so the other half acted as a duplicate for redundancy, the same principle as applied to the LVDC memory.

Part of the credit for the perfect success record of the Saturn vehicles (*all* Saturn I, IB, and V boosters flew without a failure) must be due to the effectiveness of the checkout procedures. Without automatic testing the confidence in the rockets could not have been attained, since they were too complex for effective manual procedures. In addition to checkout methods specific to the launch vehicle, the launch directors in the firing rooms had access to automated test data

ORIGINAL PAGE IS
OF POOR QUALITY

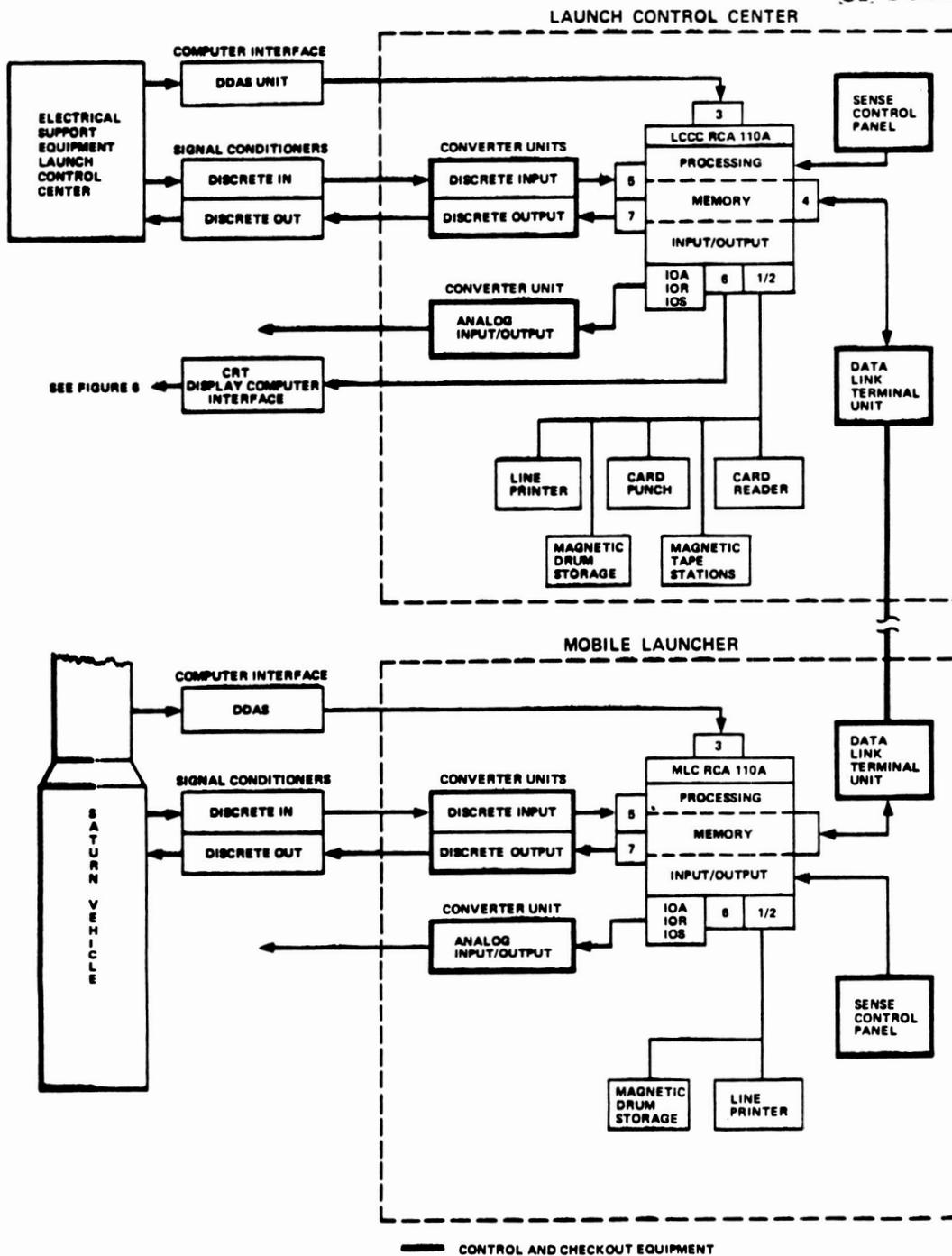


Figure 7-3. A block diagram of the automated preflight checkout hardware for the Saturn launch vehicle. (From IBM, *SLCC Programming System*)

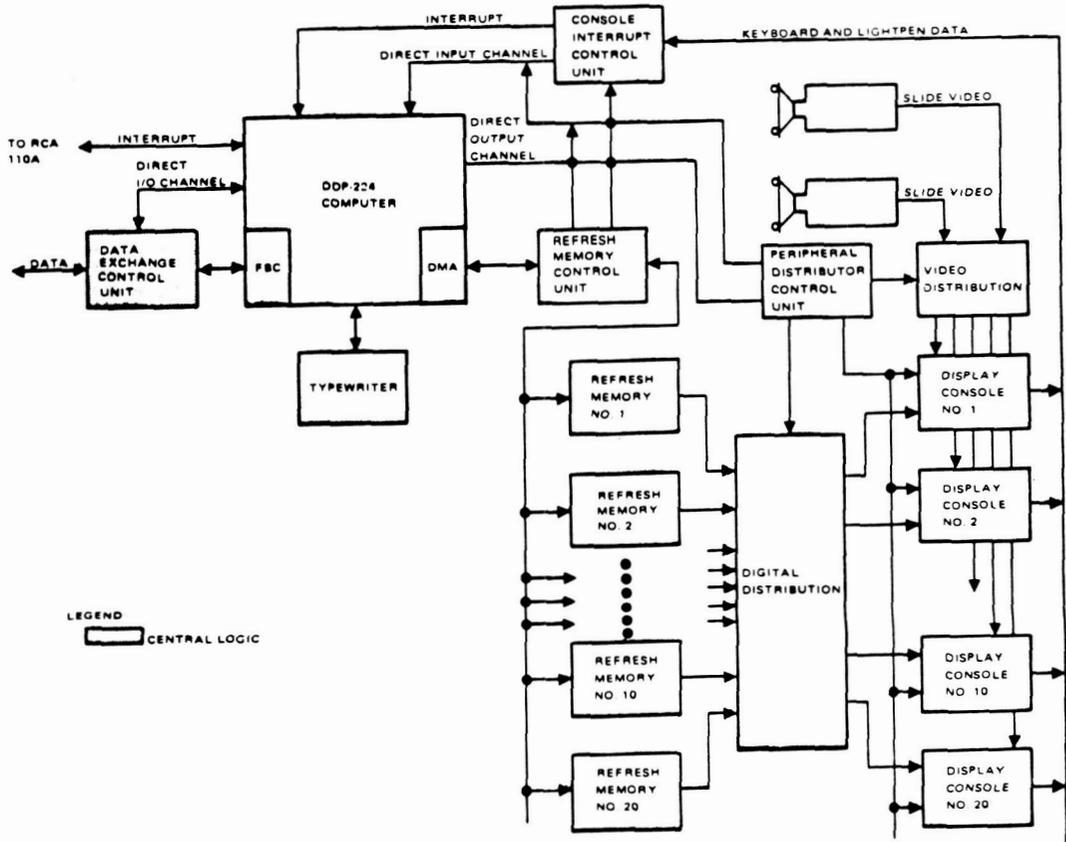


Figure 7-4. A block diagram of the Saturn Operational Display System. (From IBM, *SLCC Programming System*)

from the spacecraft preflight test equipment developed by both the Launch Operations Center and Manned Spacecraft Center.

Development of Apollo's Acceptance Checkout Equipment

From the first Apollo earth orbital flights through the lunar missions, Skylab, and the Apollo-Soyuz Test Project, ground testing and countdown support of the spacecraft and its associated systems were the responsibility of the ACE, or Acceptance Checkout Equipment**.

**The acronym ACE evolved from PACE, or Preflight Acceptance Checkout Equipment, which appears in some of the literature. It was discovered that the name conflicted with a commercial product, so the "Preflight" was dropped. Prior to PACE, there was a short period when the equipment was known as SPACE, but apparently not officially.

ACE stations were located in the Apollo Operations and Checkout Building in the Industrial Area of the Kennedy Space Center, at Launch Complexes 34 and 37, at the Johnson Space Flight Center, and at North American Aviation and Grumman Corporation assembly plants. Two of the North American stations were modified for use in assembling the Shuttle in Palmdale, California¹⁸.

ACE resulted when a spacecraft checkout engineer figured that there had to be an alternative to manual methods. Thomas Walton transferred from a job in the computer room at Langley Space Flight Center to the Cape Canaveral Launch Operations Center in early 1961. Assigned to the checkout of Mercury spacecraft, by September he had enough of manual testing and applied his background in computers to devising another way of doing things¹⁹. Walton convinced his boss, Harold G. Johnson, to let him build a digital ground station for telemetry from the capsule. Using the Mercury missions as prototypes, he proved that digital equipment could display the engineering data previously shown on dials and strip charts. His success led to a search for a computer system to handle the data in real time. With NASA's Gary J. Woods, he traveled to several companies in search of a machine. Walton did not believe that a computer like an IBM mainframe of that era could do the job. Since they were designed for large-scale batch processing, the difficulties of adapting such a computer to the real-time world of telemetry displays and automated checkout would be too great. Instead, he and Woods looked for simpler minicomputers such as the Digital Equipment Corporation PDP-1 and the Control Data Corporation CDC-168²⁰. Walton convinced the Gemini Project Office to buy a pair of CDC-168s to be used for checking out their spacecraft. Meanwhile, plans continued to create a system dedicated to the Apollo.

Marshall and Launch Operations personnel met in 1963 to determine whether the checkout equipment for both the Saturn vehicle and the Apollo spacecraft could be combined. Richard's and Walton's teams decided to continue separate paths²¹. The results were the Saturn checkout system and the first ACE unit using General Electric discrete equipment and CDC-168 computers going on line in late 1964.

Although the first ACE stations were under construction, a small political battle was raging over who would have ownership of the program. Joseph Shea, then at NASA Headquarters in Washington, wanted to control it from the Apollo Project Office there. Transferred to Houston in 1963 to take over management of Apollo, he moved the ACE development group of between 50 and 60 persons there instead²². This act reinforced the feelings that the Kennedy Space Center was to be strictly an operations center, staying clear of research and development activities.

ORIGINAL PAGE IS
OF POOR QUALITY,



Figure 7-6. An ACE station with twin Control Data computers. (NASA photo 107-KSC-67C-919)

Each ACE station used two digital computers with a common memory. One was the Digital Command Computer, which processed commands from the control or firing rooms to the spacecraft and was interconnected with the ground support equipment. A second machine was the Data Processing Computer, which drove the displays and controlled peripherals. Memory could be directly accessed by discrete circuits in the ground station, so data for both computers could be placed there²³. ACE stations could function in manual command mode, semiautomatic, or fully automatic with manual override. Stations in the Kennedy Industrial Area serviced the spacecraft both before and after mating with the Saturn V boosters. When Launch Complex 39 went into use, checkout wires carrying digital formatted data ran over 15 kilometers from the ACE stations to and from the pad and firing rooms in the Saturn Launch Control Center using a video wideband transmission system²⁴. Of course ACE had to cooperate with the RCA 110As at various points, so interfaces between the different computers consisted of dedicated I/O registers, sense lines, and priority interrupts²⁵. ACE also had to talk to the Apollo guidance computers in the command module (CM) and the lunar excursion module (LEM). On the average, the CM computer operated for 50 hours in support of the countdown. A CRT display controlled by ACE duplicated the data shown on the display and keyboard while the Apollo computer was in operation.²⁶

Walton judged that the development of ACE did a lot to stimulate the technology of on-line processing. Certainly it helped create techniques of interconnecting multiple different computer systems. Also, this was one of the first times that data transfers in the megabit range were accomplished over distance.

Digital Displays of Telemetry

Telemetry transmissions from the vehicle are one important source of data for rocket engineers. In the early days of rocket flight research, the causes of failures often could only be guessed. When the larger size of later rockets made it possible to carry radios for sending back data, sensors were added to supply engineering data from critical components throughout the flight of the vehicle. If a failure occurred, it was often possible to determine which components contributed most to it by examining the reams of data sent back and originally recorded in analog form on the ground. However, Tom Walton's pioneering digital ground station for Mercury displayed the data in processed digital form. In 1962, the Atlas-Centaur project automated postflight telemetry data reduction²⁷. By the mid-1960s, digital telemetry displays were standard at Kennedy Space Center, provided by a pair of mainframe computers in the Central Instrumentation Facility.

Kennedy acquired two General Electric 635 computers for telemetry monitoring and batch processing of institutional programs. GE 635s were 36-bit processors capable of double-precision arithmetic²⁸. Programmers prepared separate code for each of the Delta, Atlas-Centaur, and Saturn flight vehicles. Delta and Atlas launch pads, as well as Complexes 34, 37, and 39, could be switch connected to the computers at any one time. Forty different displays were possible and could be transmitted to the appropriate blockhouse or Launch Control Center firing room²⁹. NASA's Bruce Miller was in charge of systems programming for the GE computers, with Bradley Hughes as the chief scientific programmer.

These computers had the longest operational life of any installed at the Kennedy Space Center. GE delivered the first machine in late 1965. A second came on line in early 1966. Until May of 1983—18 years later!—one was still in use driving blockhouse displays for Delta and Atlas-Centaur. GE had long been out of the computer business by then, having sold its digital computer division to Honeywell in the early 1970s. Kennedy retained a permanent systems programmer from GE (who later moved to Honeywell) to keep the operating systems going and used a retired blockhouse 635 from Wallops Island as a source of parts³⁰. From the beginning the computers had a dual operating system. Batch institutional jobs could be run at the same in-

stant a real-time telemetry program was running, except when a Saturn was being supported, as its program was so big it pushed out the batch programs. When Kennedy Space Center officials searched for a second machine for the Central Instrumentation Facility, they considered other vendors. IBM's branch manager in Cape Canaveral, W. O. Robeson, sent a letter suggesting a System 360/50 as an administrative computer, pointing out that evidence from prior telemetry computers indicated that they rarely failed³¹. The dual operating systems could then be abandoned. However, Kennedy bought the second 635 to provide a redundant backup anyway, accepting the loss of batch processing during Saturn operations.

Telemetry data reduction computers thus provided yet another source of information to the launch directors in the Apollo/Saturn era. Still, some engineers were convinced that the computer data were never accurate, just as their colleagues in the checkout world had to be dragged into automation³². Regardless, telemetry displays became an integral part of the technology of launch processing.

Impacts of the Apollo/Saturn Era on the Shuttle Launch Processing System

Developing the major computer components of the launch processing system for Apollo/Saturn provided software contractors such as IBM and the Kennedy Space Center staff valuable experiences later transferred to the Shuttle Launch Processing System and on-board software for the Shuttle program. Additionally, some techniques known in theory, but never properly applied, found justification during the Apollo/Saturn programs. The areas of impact included the modularization of software, lessons learned by IBM as a key future contractor, and Walton's continued influence on ground computer concepts.

Software written for the LVDC and the GE 635 computers started as single monolithic programs and evolved to modularized programs at just about the same time. Flight software for the ASC-15 computer used on the Saturn I vehicles was necessarily monolithic because it had to be sequentially executed and strictly timed³³. Any changes impacted on the execution time, and therefore had to be carefully integrated. The computer could not handle waiting for an interrupt to instigate an action. Actions had to be initiated by the program relative to its starting time. When the ASC-15 gave way to the LVDC, a more powerful and flexible machine, programmers continued in the monolithic mode. Finally, IBM staff realized that by preparing the software in essentially free-standing chunks, the impact of changes would be limited to the modules and not spread side effects throughout the software. This discovery came late in the Saturn program but

early enough to affect the development of the Skylab on-board software. Also, IBM separated the modules into groups consisting of the control subsystem and applications subsystem, which is a prototype of the Shuttle on-board software organization³⁴. IBM helped transfer this concept to the Shuttle by moving people such as Kyle Rone and Lynn Killingbeck from working on the Saturn computer directly to the Houston office to support the Shuttle software development. NASA also independently moved toward modularization when, in 1973, it broke down the programs used on the GE 635s to support telemetry data reduction. Before then, it took an average of 3 months to implement a simple change in the monolithic version of the program, because of the massive debugging necessary to eliminate side effects³⁵. Thus, modularization came to be expected by NASA as part of software design. If modularization was not used on the Shuttle on-board software, preparing new flight loads would have been impossible within the projected time between flights of an individual orbiter.

Besides modularization, Apollo/Saturn significantly influenced IBM's later work on the Shuttle's on-board software, especially the company's design of the system used for Shuttle launch processing. IBM summarized its conclusions in a document released in late 1972, just at the time both Shuttle ground and on-board software contracts were being let³⁶. The study recommended that the vehicle's flight software be capable of reloading all programs on board³⁷. This was implemented on the Shuttle, as the mass memory units (MMUs) contain all preflight and flight software for the primary avionics computers, the display computers, and the engine control computers. Ground software recommendations required that all checkout functions use a higher order language and that checkout be conducted using one computer system³⁸.

During Saturn, both ATOLL and machine language programs controlled preflight tests, the machine language routines absorbing a considerable amount of development and maintenance time. This lesson helped spur the creation of an improved checkout language, GOAL. In regard to consolidating all functions in one computer, IBM thought that the difficulties of integrating the two RCA computers, the DDP-224 display computer, and the telemetry reduction computers were excessive. By taking that position, IBM found itself squared off against the distributed concepts envisioned by Tom Walton and his team for the Shuttle system. Walton refused to move to Houston when Shea transferred the ACE team. By staying at Kennedy, he was able to influence the structure of the Shuttle Launch Processing System and help make the Center fully responsible for all checkout and launch operations for the entire vehicle, a significant change from the Apollo/Saturn program.

THE SHUTTLE LAUNCH PROCESSING SYSTEM

When NASA began planning for the Space Transportation System (STS), it espoused ambitious requirements, such as an eventual launch rate of 75 per year. A projected fleet of three orbiters would be limited to a maximum 2-week turnaround between flights and a 2-hour countdown in order to achieve that many firings³⁹. Compared to the 5-month checkout of a Saturn V and its 3-day countdowns, this seemed outrageous, especially since the Shuttle would be no simpler than an Apollo/Saturn. NASA put considerable effort into examining commercial aircraft maintenance techniques to see what could be adopted for Shuttle use. One study indicated that only 53% of the tests done on a Saturn V would need be repeated if the spacecraft were reusable⁴⁰. Even with this reduction, nearly 46,000 measurements have to be made and monitored in real time in the process of preparing a Shuttle for launch⁴¹. Clearly, there was no way NASA could do the Shuttle checkout with Apollo concepts⁴². As Henry Paul, who headed the Launch Processing System development for NASA, said, "Automation...becomes a requirement for operations, not an elective"⁴³. Still, some engineers needed to be convinced that hard-wired testing could be successfully eliminated, even though the last 20 hours of a Saturn countdown was 85% automated⁴⁴. Building the present system, during which almost all preflight testing and preparation is done under control of software, and in which much of the countdown, sometimes even including the calling of "holds," is done by computing machinery, was a remarkable effort⁴⁵. One of the biggest changes from the Apollo/Saturn preflight checkout systems is that Kennedy Space Center became responsible for the development of the Launch Processing System. Given the organization of NASA at the time, this was one of the biggest surprises as well.

Kennedy Space Center Gets the Job

During the late 1960s NASA began studies of the configuration of the eventual STS. Most designs were predicated on a winged booster, which would return to the launch site immediately after separation from an orbiter with internal fuel tanks. Such a design could theoretically be launched from anywhere in the United States isolated enough to handle aborts safely. Project staff examined a number of sites and made projections of the cost of an "ideal launch

site" that would have all the facilities necessary for handling the Shuttle. Chief among these were a hypergolic and cryogenic fuels facility, a hangar for the orbiters and boosters, a mating building, a control center, the launch pads, and a runway with a safing bay for emptying residual fuels after landing. One study placed the cost of a new facility with these characteristics at \$1.9 billion. On the other hand, modifying existing Apollo/Saturn facilities at Kennedy and adding new equipment where needed would cost \$355 million, a significant savings⁴⁶. In March 1972, NASA selected the solid rocket booster/external tank configuration for the Shuttle. All inland launch sites were thus eliminated, and just Kennedy Space Center, Vandenberg Air Force Base, and a site in Texas remained under consideration⁴⁷. Since existing facilities could be modified at both Vandenberg and Kennedy, the cost-conscious administrators settled on those two launch sites. Vandenberg was expected to handle polar orbit launches and most military payloads. Kennedy would launch eastward, continuing the established situation and giving Kennedy the opportunity to try for the development of the checkout system.

Phase B Shuttle studies conducted by a number of contractors included concepts of the checkout system⁴⁸. Some hinted at the direction the eventual system would take. One pointed out the need for efficient and simple man-machine interfaces, and called for having ATOLL, FORTRAN IV, and COBOL compilers available to the engineers⁴⁹. Kennedy's own early study, based on Rockwell and McDonnell-Douglas Shuttle configurations, called for a central data processing facility connected to every part of the Shuttle handling equipment, including a mission simulator on site and by communications link to Mission Control in Houston⁵⁰. Meanwhile, remnants of the old ACE group at Johnson had started work on a Shuttle checkout system.

A "Checkout Systems Development Lab" at the Johnson Space Center did research on new concepts of preparing manned spacecraft for flight⁵¹. Individual BIC, for "built-in checkout," cells would be located at test points throughout a spacecraft, each cell with I/O registers. Automated tests would read and write to these cells. Johnson's development team wanted a single central computer to be connected to several sets of Universal Test Equipment consoles and thence to the Shuttle⁵². General Electric built a prototype of a "Universal Control and Display Console" for the Laboratory. Each control console would have two color display tubes and be capable of supporting tests on any specified parts of the spacecraft⁵³. The system was similar to earlier Apollo/Saturn concepts, with a big computer in the middle doing all the testing and displays, communicating with the spacecraft, and so on. One improvement was that the Universal Test Equipment meant that units could be mass produced and assigned to different checkout tasks without significant hardware changes. When

the time came for the Shuttle project office at Johnson to make a decision about preflight checkout, the hometown lab made a proposal that was "underdeveloped" and vague⁵⁴. Most likely, the engineers in the Development Lab thought the job was theirs because in all previous programs the center responsible for the spacecraft was responsible for checkout. When Kennedy had tried to do some ACE development, it was moved to the center responsible for the Apollo. Therefore, a full-blown proposal did not seem necessary. They were in for a surprise.

Impetus to make Kennedy the development center for the Launch Processing System came from many levels. The center's director, Kurt Debus, made his support clear to his engineers in 1972⁵⁵. Walton saw a chance to do another ACE, but this time as a fully integrated system for all parts of the spacecraft. The consensus was that by having Kennedy Space Center do the development, much money would be saved and civil servants would be more actively involved⁵⁶. Even though the work originated with Walton's Design Engineering Directorate, talent for developing the Launch Processing System came from across the Center⁵⁷. A study group of about half a dozen engineers, led by Theodore Sasseen and including Henry Paul, Frank Bryne, George Matthews, and others who had key roles in the later implementation of the system, met and began work on a prototype⁵⁸.

Making the prototype turned out to be one of the key factors in landing the Launch Processing System development job for Kennedy. The engineers made a small model of a liquid hydrogen loading facility, with real valves and tanks. Using a Digital Equipment Corporation PDP 11/45, they devised software that graphically displayed a skeletal view of the piping and valves, with actual pressures printed next to the appropriate valve. The prototype could transfer fuel to the model spacecraft under software control, with the user able to monitor flows and pressures at the console. Confidence in their ability to create automated procedures encouraged the engineers, and they also now had a physical version of their system to help in selling it. Johnson's Universal Test Equipment had no counterpart in terms of functionality.

The prototype represented a single, and complete, part of the total system, a system quite different in concept from previous ideas. Launch processing and mission control prior to the Kennedy developments were based on using a minimum number of mainframe computers. Frank Byrne had the technical vision to develop a distributed computing system, in which dozens of small computers would do the checkout functions. Walton provided the leadership and tenacity to hold to the concept and see it put into place⁵⁹. Several important advantages result from using distributed computers. First, the tasks more closely fit the power of the machine. Using a mainframe computer for relatively simple procedures such as solid rocket booster checkout would be overkill⁶⁰. Second, a distributed system would free software

developers from worrying about fitting their programs in with others in a big machine's memory. Each discipline, such as engines, cryogenics, and avionics, would have a separate console⁶¹. Third, parallel testing could be done⁶². A mainframe would have to be inordinately large to contain all the checkout programs. Therefore, they would have to be loaded and run serially, as in the RCA 110As, defeating the short countdown requirement. Finally, Paul was convinced that overall hardware costs would be reduced compared with mainframe configurations⁶³.

In 1972 Robert F. Thompson was the head of the Shuttle project office at Johnson and in charge of deciding where to place the checkout development. Faced with a choice between a homegrown system similar to tried and true predecessors and a new concept developed at Kennedy that even had opposition there, he ruled in favor of Kennedy's proposal against the opinions of his advisors. The winners are gracious toward Mr. Thompson, calling him an "honest manager" and a "nonterritorial individual"⁶⁴. Thompson judged Kennedy's to be the best proposal, but he also thought it more efficient for NASA to develop the Launch Processing System where it would eventually be used and by the people who would use it.

Getting Started: Contracting For the Launch Processing System

Due to the earlier site studies and the building of the prototype, Kennedy Space Center had a good idea of what it wanted in the Launch Processing System. Reflecting the detailed requirements developed for the Shuttle on-board computers, the Design Engineering Directorate's engineers started in March 1973 to prepare the "Launch Processing System Concept Description Document"⁶⁵. Released in October, the document specified the architecture and concepts of the system in detail, before any major contractor involvement⁶⁶. Kennedy's efforts on the Launch Processing System are reflected by the fact that nearly 100 civil servants were involved in the planning between 1973 and the March 1976 freeze of the design⁶⁷.

Plans for the System included extensive remodeling of Saturn facilities. The Processing System itself is largely contained in the Launch Control Center. Hardware is divided into the Checkout, Control, and Monitor Subsystem (CCMS), the Central Data Subsystem (CDS), and the Record and Playback Subsystem (RPS). Small, task-dedicated computers are in the four firing rooms of the Control Center and are the primary component of the CCMS. Large mainframe computers located on the floor below the firing rooms make up the biggest part of the CDS. NASA's Joseph Medlock, Thomas Purer, and Larry

Dickison envisioned test engineers developing their own procedures using an engineer-oriented language like ATOLL in concept but better and easier to use⁶⁸. These procedures would then be developed on the mainframes and tested against simulations stored on the mainframes. When verified, they would be included in the system and stored on disk. When a firing room became active to support a vehicle, the engineer would load his test procedure from the mainframe to the minicomputer attached to his console and execute it from the console.

Depending on which spacecraft subsystem is involved, the tendrils of the Launch Processing System may follow it wherever it goes on the Space Center site. The firing rooms are connected to the Vehicle Assembly Building, the launch pads, the Cargo Integration and Test Equipment, and the new orbiter Processing Facility, a two-bay horizontal hangar. At each location, hardware interface modules make it possible to test and monitor the orbiter and other parts of the spacecraft from the firing rooms. So the System is locally distributed computationally, but physically centralized—especially compared with the RCA 110As and GE 635s of the Saturn era.

One critical side effect of using a mix of mainframe data base machines and minicomputers for individual system checkout, as well as the need to talk to a pervasive on-board computer system, was that for the first time, several different network architectures had to be combined into one⁶⁹. The inherent difficulties involved led NASA to award the software contract before choosing hardware so that the software contractor could help in the computer selection⁷⁰. Further, the minicomputers were chosen apart from the contract for the consoles and other hardware associated with the CCMS. Four source selection boards eventually convened: one each for software, minicomputers, the CDS, and the CCMS⁷¹.

Since test engineers would write the applications software, the software contractor would be primarily responsible for the operating system under which the applications would run, the new test language, GOAL (for Ground Operations Aerospace Language), and any modifications to the microcode for the minicomputers and other equipment needed to successfully connect them. Interfacing largely became a software problem because the changes were to be implemented in microcode. Six contractors tried for the job, with IBM beating out General Electric, TRW, Computer Sciences Corporation, McDonnell-Douglas, and Harris Computer Corporation⁷². The initial \$11.5 million contract ran from May 1974 to March 1979⁷³. This contract was extended several times due to delays in launching the first Shuttles, but IBM's involvement ceased in the operations era. The company did its usual good job, and users of the eventual system believed it fulfilled the requirements⁷⁴. IBM used a top-down structured approach in designing the software, holding weekly formal reviews during the development stage so that NASA could closely monitor activities⁷⁵.

By winning the software contract first, IBM found itself in the unusual position of having to program other people's computers. One IBM employee said that his company was encouraged *not* to bid on the hardware contracts⁷⁶. According to Byrne, IBM was not kept out of the hardware bids so much as they lacked a suitable minicomputer to offer. The System 34 was under development at that time, as was the Series/1, but IBM chose not to make its new minicomputers public⁷⁷. Three companies made the final round of bids on the minicomputers: Prime, Varian Data Machines, and a small new company called Modular Computers, Inc⁷⁸. Design Engineering had built a prototype of a launch processing console set for the solid rocket boosters using Prime computers. (Later shipped to Marshall for awhile, it finished its career in the Vehicle Assembly Building nearly 10 years after construction.⁷⁹) Because of this, many thought Prime had the contract won, but it was edged out by Modular Computers, much to the surprise of Byrne and Walton⁸⁰. ModComp initially contracted for 60 machines at a cost of \$4.2 million, a number later extended considerably as console sets were placed in all four firing rooms, the cargo integration facility, the Shuttle Avionics Integration Lab (SAIL) at Johnson, and the hypergolic maintenance facility, as well as at Vandenberg. Two months after the computer contract was let in June of 1975, Martin-Marietta defeated Grumman Aerospace Corp., Aeronutronic Ford, and General Electric for the remaining CCMS hardware.

By November 1976, IBM received the first minicomputer for software development, and by February of 1977, the first station for GOAL applications development was delivered⁸¹. Honeywell won the CDS hardware contract in the fourth quarter of 1975, and John Conway of NASA managed the acquisition of equipment and personnel for that Subsystem during 1976–1977⁸². By 1977, the Launch Processing System began to take physical shape.

The Common Data Buffer: Heart of the System

Most diagrams of the physical components of the Launch Processing System show an inordinately large rectangle at the center of the drawing, with all other components either directly or indirectly connected to it. That rectangle represents the common data buffer, which Thomas Walton called the "cornerstone of the system"⁸³. The biggest problem with creating distributed computing systems is devising a method of intercomputer communication that is reliable, fast, and simple. In a system such as the Launch Processing System, which depends on a number of computers "knowing" the same data about the spacecraft, some method of protecting and centralizing the common

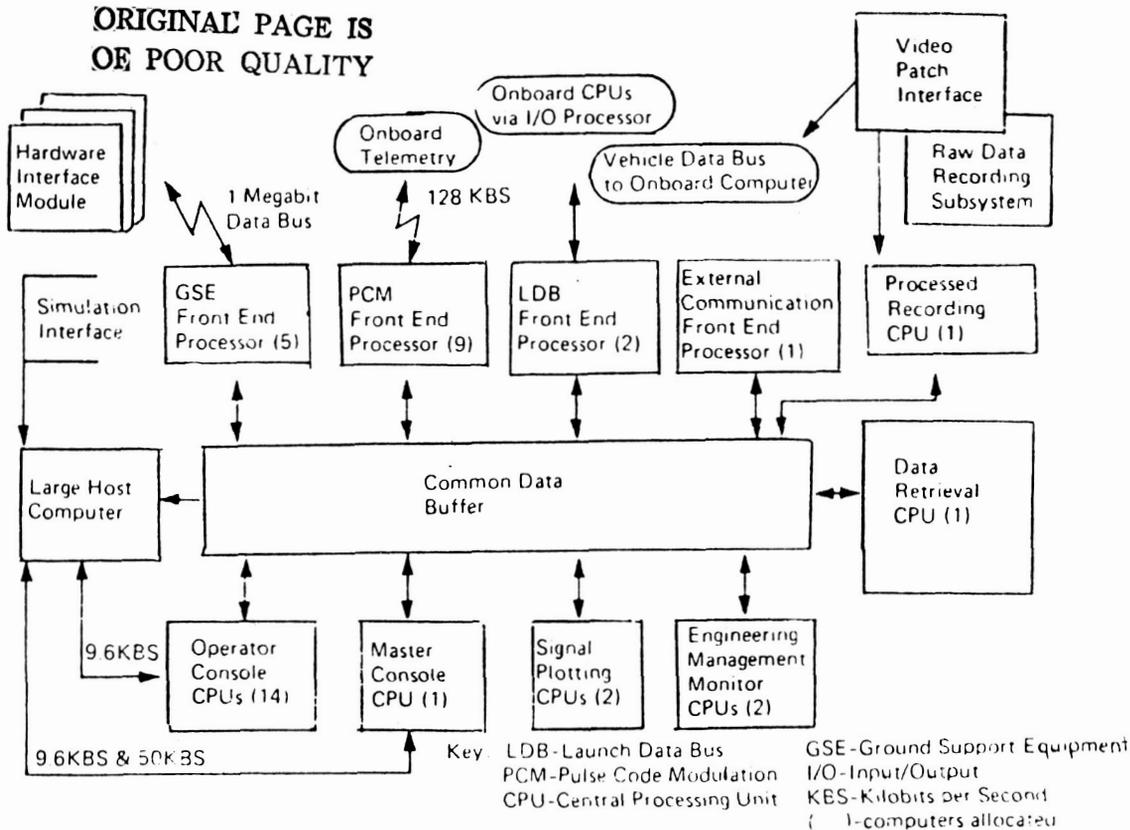


Figure 7-7. Shuttle Launch Processing System hardware structure. (Courtesy IBM)

data is needed. Frank Byrne, who was involved in the planning for the Processing System from the start, took on the job of designing a device to keep track of commonly needed data that also made it possible for the various computers to communicate with each other. Kennedy followed a plan to use commercially available equipment in as many parts of the Launch Processing System as possible. Minicomputers and mainframe computers are used largely unchanged. However, since no organization had tried to closely connect such large numbers of machines, some of which were quite different in architecture from the others, there was no commercially available solution to the common data problem. Byrne had to design one on his own: the common data buffer.

Byrne noted that as the number of computers in a distributed system increases, the complexity of intercomputer communication increases. He wanted to remove the complexity. By placing the common data at a central location he eliminated the need to update multiple copies of the data in separate machine memories. Possibly he got the idea for a common data area from his work on the GE 635s in the Central Instrumentation Facility. Those machines had a "data core"

which could be accessed by both⁸⁴. ACE stations also used common data areas. Basically the common data buffer provides each machine in the system with a set of "post office boxes." Specific parameters, such as valve pressures, voltages, and fuel levels, are assigned a location in the buffer memory. Each machine can read any location in the memory, but only the machines explicitly assigned to the task of maintaining a certain parameter can write to that parameter's location. That way a secondary machine cannot spuriously change a parameter's value. Programmers do not have to worry about where any particular parameter is kept. As long as it is referred to by its proper name in a GOAL program, the system build process will assign it its correct address as the program is compiled and integrated. In addition to acting as a common storage area for data, the buffer maintains the entire system interrupt stack and flags and status variables. Since it is centrally located, it is also used to temporarily store application programs as they are loaded from the repository in the CDS to the individual minicomputers. As such, it acts as a "way station."

Box 7-1: Inside the Common Data Buffer

Even though the common data buffer is a unique design, it uses standard commercial chips and boards. Nothing was custom built⁸⁵. Memory chips are made of negative metal-oxide semiconductors (N-MOS), with each section consisting of 64K of one bit and 32 sections making up 64K of 32-bit words, matching the word size of the Mod-Comp computers in the firing rooms and the error-correcting code used in transmissions⁸⁶. Memory can be read in 200 nanoseconds, very fast by any current standard. Motorola 6800 microprocessors are used in the buffer as controllers, each with 2K of read-only memory and 1K of read/write memory⁸⁷. The 64K main memory has the first 1K words set aside for interrupts, the next 1K as a common read/write area for flags and other variables, and the remainder as the protected memory area⁸⁸. Data can move through the temporary storage areas and out to the computers at a rate of 8 megabytes per second⁸⁹.

A common data buffer can have up to 64 devices (computers or other buffers) hooked to it at one time. Each device is connected to a buffer access card. The cards are scanned by the buffer in rotation, looking for incoming data or requests for data. If a device is needing the buffer and its request is noted on the access card, the device then has a slice of time to do its work, after which the scanner (which has been "looking ahead") goes to the next card indicating a usage request⁹⁰. In this way, when one machine is writing or even reading, all other machines are shut out, preventing both contention for resources or simultaneous attempts to update data⁹¹.

An early criticism of the common data buffer concept was that it would be a single point of failure⁹². Standard protections were built into the buffer, such as dual power supplies and the use of triple modular redundancy in some components⁹³. However, the biggest problem in a system of this type is protecting against communication errors. Millions of bits are speeding throughout the network each second, providing considerable opportunity for lost or garbled data. Byrne's answer was to include a powerful set of error-correcting codes, which he created with the help of Robert W. Hockenberger of IBM, who was brought in specifically to work on the problem⁹⁴.

The resulting codes enable the data buffer to successfully operate with any 2 bits of the 16-bit words in error! When a word is being transmitted between computers and the buffer or vice versa, it is sent as a 32-bit message. The first 8 bits are data, the second error-correcting code, then the next word's first 8 bits are code, and the last 8 data. Data are alternated in this way to protect against "big" signal losses⁹⁵. Individual bits are checked using the correcting codes at each end of a transmission. One hundred per cent of the 1-bit errors can be corrected, 99%+ of the 2-bit errors can be fixed, 70% of 3-bit errors, and even half of the four bit errors. Since the memory chips are arranged in 64K by 1-bit banks, the loss of an entire sector of memory means the loss of just 1 bit per word, which can then be corrected. The error-correcting codes themselves are generated by software on read-only memories⁹⁶. Even though such extensive protection is provided, in a decade of operation there has been no failure of a common data buffer, and internally never more than 1 bit has been garbled⁹⁷.

In terms of the architecture of distributed systems, the common data buffer was a pioneer. Currently, many distributed systems exist partly because of the proliferation of minicomputers and microcomputers. Micros, especially, can be connected to common data bases on shared hard disks. NCR Corporation briefly marketed a system called Modus from 1982 to 1984 that featured the ability to connect with different types of microcomputers, a shared data base, and microprocessor control of communications that effectively locked out other computers from corrupting data being updated by another one on the network. In general, though, no commercial system is as effective as the Launch Processing System in terms of speed, simplicity, and reliability. Most intercomputer communication is clouded by different protocols, nonadherence to declared international standards, and lack of speed. Frank Byrne's work stands as an original and brilliant solution to the key problem in implementing the Launch Processing System. Fittingly, Byrne received proper recognition for his achievement. NASA granted a \$10,000 bonus and an award⁹⁸. The buffer itself was patented, a rarity for the government side of the space program⁹⁹.

CCMS Hardware

The hardware of the CCMS consists of the common data buffer and everything else in the four firing rooms of the Launch Control Center. Even though the buffer appears on the charts as the largest item, in reality it is one of the smallest, filling two electronics racks in the back of a firing room. Most of the equipment in a room is blue-colored consoles and boxes: the ModComp computers and their consoles. The number and arrangement of consoles are dependent on the function of the particular room. Firing rooms one and three are for flight operations, with three capable of being made secure for DOD launches. Rooms two and four are for software development and testing, number four used for secure operations¹⁰⁰. Firing room two has three buffers to facilitate multiple parallel software development¹⁰¹. Operations firing rooms normally are configured for 12 consoles, plus a master, integration, and a backup console¹⁰².

During a countdown, consoles in the adjacent software development room are kept active as a further backup¹⁰³. Each ModComp has three display terminals, and those three make up one console. These are mounted in a half semicircle, so two computers and their attached consoles located side by side look like a "D" with the rounded part facing the front of the room. Each of the computers contains either a 5-megabyte hard disk or 80-megabyte hard disk for storing applications programs uploaded from the mainframes in the CDS¹⁰⁴. Early in the program each engineer had his own disk, and could carry his programs to different computers, but when configuration control began to be needed the removable disks were replaced¹⁰⁵. Loading an entire firing room through the buffer to the ModComps takes a full shift. Each computer can run up to six GOAL programs concurrently.

Individual consoles have marvelous capabilities. NASA commissioned Mitre Corporation to do a human factors study for the Launch Processing System¹⁰⁶. Some of the resulting concepts make the usability of the system outstanding, and it is superior to many workstations in existence today. Color displays, programmable function keys that make it possible to replace long strings of keystrokes with a single push, full cursor control, and other features make it possible for an engineer to create applications programs that can be run without using the keyboard¹⁰⁷. This concept antedates by 10 years the now ubiquitous "mouse" found on such machines as the Apple Macintosh. In addition, the consoles can be switched to become a terminal to the CDS for procedure development and to examine data recorded during operations. Special keys on the console enable program execution to be temporarily halted or single-stepped, aiding debugging of GOAL applications¹⁰⁸. As a further convenience, each console has hard-copy capability; "snapshots" of displays can be made, with all

the graphics intact, but in black and white. Graphics use is assisted by special keys that provide corners, standard symbols for valves, transducers, and other components that can be put at cursor positions on the screen. Thus, the engineers can build pictorial skeletons of the systems they are testing for greater clarity. In general, these consoles are among the best available in any computer installation and are ideally suited to the purpose of the Launch Processing System.

Besides the use of ModComps attached to consoles, other ModComps are used as front end processors to provide interfaces between the spacecraft and ground service systems and the buffer. ModComps used for applications programs have 64K words of memory, but the front end processors have 48K, 64K, or 304K, depending on their connections to other devices¹⁰⁹. Hardware interface modules at the actual points of entry to the ground support equipment plugged into the spacecraft send and receive data from the front end processors. Those, in turn, examine the data for parameters that are approaching their test limits. If a parameter nears a limit, the processor issues an interrupt and calls in a "control logic" program to handle the matter¹¹⁰. Control logic is a subset of GOAL used for making sure things are not done outside their proper order and within specific time constraints. For orbiter communication, a launch data bus front end processor communicates directly with the on-board general-purpose computers. Other "downlink" front end processors only receive pulse code modulated data to be processed for orbiter, main engine, and payload components.

Between the console computers and front end processors, a typical operations firing room contains over 30 minicomputers, each interconnected through the buffer. These computers can control tests and monitor the Shuttle anywhere hardware interface modules are available to connect it to the firing room, whether in the orbiter processing facility, the Vehicle Assembly Building, or the pad. Since each console can do the functions of any other console simply by changing its software load, the system has tremendous flexibility.

CDS Hardware

Supporting the CCMS is the CDS. Two sets of two Honeywell 66/80 mainframe computers are the heart of the CDS. NASA purchased the original pair of these 36-bit machines with half a million words of main memory each and an additional half a million words for sharing. As software for the Launch Processing System grew in size, the memories were upgraded to 1.5 million words each and 1 million words of shared memory¹¹¹. One hundred seventy-two disk drives are connected to the machines as mass storage, with a total capacity of almost 30 billion bytes. Originally, the computers used



Figure 7-8. Typical firing room layout for the Shuttle. Less than 50 engineers are needed for the countdown. (NASA photo 108-KSC-78PC-240)

Honeywell's 4JS1 operating system, which is no longer supported by the company. One NASA computer scientist said that "we have taken almost every piece of standard software and modified it" to meet the unique needs of the Launch Processing System¹¹². Most often the first pair of Honeywells support an operations firing room, whereas the second set is being used for software development. If one of the pair notices that it has failed self-tests for 10 machine cycles, it automatically switches control to its partner.

The third part of the Launch Processing System is the RPS. Initially implemented with Apollo-era equipment, it was later modernized with new recorders and computers. The RPS records most data telemetered from the spacecraft for later playback and produces records and printouts in real time for immediate analysis by system engineers conducting tests¹¹³. Firing room engineers can play back tests or other data directly to their firing room consoles for problem resolution or trend analysis¹¹⁴. At first, the RPS only had enough equipment to support one Shuttle at a time, so to switch from one to another required rearranging a number of connections¹¹⁵. This situation was corrected during the modernization so that the RPS can now handle multiple Shuttle data flows.

Launch Processing System Software

Software for the Launch Processing System is of three types: applications software, written in GOAL, performs the test and integration functions; simulations software enables engineers to verify their GOAL programs before using them on the real equipment; and systems software controls the execution of the other types. In the Launch Processing Division at the Kennedy Space Center, two branches support the hardware of the Launch Processing System, one for each major subsystem, whereas the Applications and Simulations Branch supports software by developing simulations and assisting test engineers in GOAL procedure writing. One of the reasons for the utility and success of the System is that civil service operations and maintenance personnel have been included in software planning and design from the beginning in order to make the System better meet their needs¹¹⁶. They essentially built their own tools¹¹⁷. That policy continues in the Applications and Simulations Branch, which helps the engineers refine their test requirements¹¹⁸.

GOAL applications programs are the largest part of the Launch Processing System software, totaling 14.2 million words by the early 1980s. As a comparison, the displays, control logic, and test control software added up to less than 700K words¹¹⁹. Despite the early resistance of engineers to the automation of testing, they found that they learned more about their assigned part of the spacecraft by writing ATOLL or GOAL programs¹²⁰. In instructing a computer, saying "pressurize the tank until the pressure is high enough" is too vague. Engineers writing programs are forced to think through the proper parameters and values and to account for anomalies ahead of time.

Kennedy engineers abandoned ATOLL because it had deficiencies in ease of use and in comprehensiveness. Too often assembly languages had to be used to do something ATOLL could not. Henry Paul assigned ATOLL veteran Joseph Medlock of Kennedy to head the GOAL development group¹²¹. Medlock and his team of civil servants received help from Martin–Marietta Corporation in defining the language, and then IBM implemented GOAL. The result was a highly readable, self-documenting procedural language. Just over four dozen statements are available, and training time is short, taking half days for 3 weeks (see Appendix III for an example of a GOAL program)¹²². IBM designed GOAL's compiler to disallow any undefined branches or procedures, making it more strict than FORTRAN compilers¹²³. GOAL is highly flexible and permits engineers to decide for themselves the degree of interaction required to do a test¹²⁴. GOAL programs are run within the computers in time slices of 10 milliseconds¹²⁵.

When an engineer is developing a GOAL procedure, he writes the procedure on his console using it as a terminal to the CDS. After the procedure is complete, it is tested against a simulation in the Honeywell computers, if a simulation is available. A Shuttle Ground Operations Simulator, consisting of GOAL-like statements, is available for developing models to test programs relating to ground equipment such as fueling systems and external power¹²⁶. However, due to the lack of an AP-101 processor and Shuttle on-board software, procedures for checking out the flight equipment are limited or nonexistent. There is no way at Kennedy to test those procedures except against an actual spacecraft, so they must be sent to SAIL at the Johnson Space Center¹²⁷. In addition to that restriction, simulations programs are limited to 256K words, the largest program a Honeywell 66/80 can run, since it is not a virtual memory machine. As a result, some models have to be run in parts¹²⁸.

A subset of GOAL is used to write control logic. Control logic prevents things from being done out of the proper order and within specific time constraints, avoiding disaster. It is necessary because of the parallel nature of testing. For example, before liquid oxygen can be moved through pipes and valves, they must be prechilled to near the temperature of the liquid, or the oxygen will flash evaporate. Control logic of the "prerequisite sequence" type checks to make sure the prechilling has been done. Or, if a valve pressure or voltage is nearing a dangerous level, "reactive sequence" control logic programs are automatically called by the front end processors to eliminate the anomaly¹²⁹. Control logic thus makes parallel operations safe.

GOAL and control logic procedures must be integrated with the rest of the System before use to resolve potential conflicts and assign real addresses in the buffer to logical addresses in the programs. Integration is done in a laboratory containing the "Serial 0" ModComp/console, the first set delivered¹³⁰. Integration requirements led NASA designers to abandon some of the flexibility envisioned in the early stages of the program¹³¹. Originally, they thought the engineers would have more responsibility for their programs and changes, but the complexity of the system required some measure of configuration control. Some 200 GOAL programs are needed just to load the liquid oxygen tank automatically¹³². With thousands of GOAL procedures to integrate, engineer autonomy had to be limited.

Cargo Integration and Test Equipment

One part of the Kennedy Space Center with an important role in the Shuttle program and also a user of Launch Processing System resources is the Cargo Integration and Test Equipment (CITE). With

hundreds of Shuttle flights planned to carry a variety of payloads from all over the world, the process of properly integrating cargo with the orbiter is a large task. Both electronic and physical interfaces must be checked in order to verify, for example, that a Spacelab module built in Germany will properly fit to a vacuum-proof seal in the cargo bay and be able to "talk" to the Shuttle computers as well.

Soon after beginning work as the prime contractor, Rockwell International and NASA did a study to find out how much and what kind of Interface Verification Equipment (IVE) would be needed for the operations era. By doing things the "traditional" way, in which the payload supplier did the interface verification, an estimated 20 sets of very expensive equipment were required. Robert Thompson favored sense over politics and decided in early 1976 to let Kennedy develop a centralized version of the IVE and do all the final interface testing for all the customers¹³³.

During June, July, and August of 1976 the formal requirements for the cargo facility were baselined. But when a source selection board met to begin choosing equipment, the members realized that they were about to violate the basic principles of the Launch Processing System by bringing in new equipment and doing things in a unique way instead of using existing contracts and computers¹³⁴. Accordingly, Kennedy stocked the cargo facility with the same physical and electronic interfaces present in the orbiter, permitting the same contractors and maintenance contracts to be used. In 1978, an AP-101 was added to provide a means to test software interfaces. Equipment in the cargo facility can also directly connect with the Launch Processing System so that payloads can be further integrated. CITE is another user of the simulations kept on the CDS¹³⁵.

Payloads delivered to Kennedy are checked out and further prepared in either the horizontal facility (Spacelab would be worked on there) or the vertical facility (communications satellites are integrated vertically). After completion of the integration tests, a special transporter with cargo space as large as the Shuttle's bay moves the payloads to the Vehicle Assembly Building for installation in the orbiter.

The Launch Processing System in the Operations Era

Originally, Henry Paul had a goal of reducing the number of technicians in a firing room from the 250 of the Apollo era to about 45. Although he succeeded, in the early 1980s, the Launch Processing System was still labor intensive, with 75 civil servants and 700 contractors involved¹³⁶. However, in late 1983, NASA awarded the shuttle maintenance contract to Lockheed, which is now responsible for physical equipment and software relating to Shuttle launch process-

ing. That award marks the end of the multicontractor development era and the beginning, for the first time in NASA's history, of an operations era for a manned spacecraft. Before the Shuttle, each flight and the preparations beforehand were idiosyncratic. Now some degree of standardization and routine is possible, largely because of the nature of the Launch Processing System. Carl Delaune, a NASA engineer in the Applications and Simulations Branch, is exploring ways of applying artificial intelligence to checkout procedures, such as creating a program that makes suggestions to test engineers if strange values occur¹³⁷. If his inquiry bears fruit, eventually the amount of human interaction during checkout will shrink even further.

As the development effort at Kennedy matured, the purposely staggered development of a Launch Processing System at Vandenberg Air Force Base began. Plans were to build the military facility after most of the developmental bugs were out of the NASA model. The Air Force saved money at its installation by modifying facilities built for the Gemini-technology Manned Orbiting Laboratory program in 1966¹³⁸. Originally designed as a Titan III launch site, the complex provides for mating orbiter, tank, and boosters at the pad, as no Vehicle Assembly Building exists there. Ground checkout facilities are split between locations at North Vandenberg and South Vandenberg, so the CDS, CCMS, and RPS are physically separated¹³⁹. With the commissioning of the western launch site in the early 1990s, the Shuttle program will have reached its full flowering.

Summary

Distributed computing, connecting different vendors' equipment successfully, good user interfaces, and automation are all topics of continued concern and research in the computer industry. The Launch Processing System solves all those problems in a specific arena. It is difficult to think of a system better suited to its task. A marvel of integration, efficiency, and suitability, it reflects the ingenuity and clear-sightedness of its originators. Lessons learned in the 1960s during the first attempts at automating checkout were applied in toto to the Launch Processing System. Rarely has a second system so completely eliminated the deficiencies of its predecessor.

8

Computers in Mission Control

Mission control begins when launch processing ends. At the point a missile is committed to flight—as when the Shuttle solid rockets are fired or a liquid-fueled booster rises an inch off the pad—responsibility for monitoring and control of the spacecraft shifts from the launch director and his crew to the flight director's team. Three major tasks occupy the flight controllers: sampling the telemetry stream to make certain everything is going well and to collect science data, doing navigation calculations, and sending commands. Manned and unmanned spacecraft require this support, with manned spacecraft having the advantage of carrying observers and decision makers to supplement what can be done from the ground. To successfully support both types of missions, digital computers must operate on massive amounts of data in real time. Mission control tasks are beyond the abilities of humans alone.

Mission control centers and their equipment are located far from the launch site. NASA's manned mission control began in 1961 with Project Mercury at the Cape Canaveral launch area, but its computers were at Goddard Space Flight Center near Washington, D.C. Since 1964, early in the Gemini program, both computers and controllers have been housed in Building 30 at the Johnson Space Center in Houston. NASA's unmanned near-earth missions are controlled mostly from Goddard, with most deep space missions handled through the Jet Propulsion Laboratory's (JPL) Spaceflight Operations Facility in Pasadena, California.

In addition to control centers, mission support requires numerous tracking stations to collect and format telemetry and radar data to help in monitoring and navigation and to transmit commands. These widely scattered stations and the control centers are linked together by the NASA Communications Network (NASCOM), headquartered at Goddard. The Space Tracking and Data Acquisition Network (STADAN), used to specialize in unmanned spacecraft but, having combined with the Manned Spaceflight Network (MSFN) in 1972, has become the general network. When all the specified Tracking and Data Relay Satellites are in place, they will take over much of the manned flight communications, yet tracking is still a STADAN responsibility. Lunar and planetary probes are the venue of the Deep Space Network, which operates three main stations at Goldstone, California, Madrid, Spain, and Canberra, Australia, each with a variety of antennas ranging up to 64 meters in diameter. The Deep Space Network helped with manned lunar missions when the Apollo spacecraft

passed a distance of 10,000 miles from earth*.

In contrast with on-board computers, computer systems used in control centers and tracking stations have primarily consisted of off-the-shelf equipment. NASA could take this approach to procurement because, so far, adequate processing power to achieve mission objectives has been available in commercial systems. When mission control began in the late 1950s and early 1960s, software technology had not reached the necessary level of sophistication. The prime contractor had to develop completely new operating system software for the Vanguard, Mercury, and Gemini programs, but was able to incorporate large chunks of existing operating systems into those used for Apollo and Shuttle, as well as some later deep space missions. This was possible in part because experience and techniques learned from designing the original operating systems were used in new commercial products.

MANNED MISSION CONTROL COMPUTERS

As with manned spacecraft on-board computers, computer systems used in manned mission control are more sophisticated and larger than those used for unmanned missions. Even though unmanned satellites and space probes pioneered the use of computers in mission control, the need for quick response and redundancy, the inherent complexity of manned spaceflight, and the rigors of the race to the moon forced rapid improvements and innovations in systems used in manned mission control so that they surpassed the older systems.

The story of computers in manned mission control is largely the story of a close and mutually beneficial partnership between NASA and IBM. There are many instances of IBM support of the space program, but in no other case have the results been as directly applicable to its commercial product line. When Project Vanguard and later NASA approached IBM with the requirements for computers to do telemetry monitoring, trajectory calculations, and commanding, IBM found a market for its largest computers and a vehicle for developing ways of creating software to control multiple programs ex-

*For the story of the tracking and communication networks, see William R. Corliss, *Histories of the Space Tracking and Data Acquisition Network (STADAN), the Manned Space Flight Network (MSFN), and the NASA Communications Network (NASCOM)*, NASA CR-140390, June, 1974, and N.A. Renzetti, ed., *A History of the Deep Space Network From Inception to January 1, 1969*, Jet Propulsion Laboratory TR 32-1533, September 1, 1971. Each has considerable detail about the technical developments involved, including the decision to use computers at stations.

ecuting at once, capable of accepting and handling asynchronous data, and of running reliably in real time. These things the company was able to do quite successfully, and the groups it assigned to the job impressed their NASA counterparts. When asked about IBM's performance in this field, one NASA manager said without hesitation, "IBM is the best"¹.

The company maintained its lock on mission control contracts through Gemini, Apollo, and the Shuttle. At each point, some experienced personnel were transferred to other parts of the company to share lessons learned. Several individuals contributed to OS/360, the first multiprogramming system made commercially available by IBM². One became head of the personal computer division³. NASA also used successful managers from mission control work to help other programs. Howard W. "Bill" Tindall started with Mercury and Gemini ground software and later made a significant contribution to the quality of the Apollo on-board software. No other software system developed under NASA contract in the 1960s was as well thought out and executed as manned mission control.

Beginnings: Vanguard and Mercury

America's most spectacular contribution to the International Geophysical Year (1957–1958) was the Vanguard earth satellite, which, in ignorance of Russian preparations, was thought to be the world's first orbiting spacecraft. In June of 1957, Project Vanguard established a Real-Time Computing Center (RTCC) on Pennsylvania Avenue in Washington, D.C, consisting of an IBM 704 computer⁴. The 40,000-instruction computer program developed for Vanguard did data reduction and orbit determination⁵. Orbit calculations needed to be done in real time so that ground stations could be warned of the approach of the satellite in time to listen for its signals and know where in space the data came from. Thus, IBM gained early practical training in the primary skills needed for mission control. In 1959, when NASA was ready to contract for a control center for Project Mercury, IBM had experience it could point to in its proposal, as well as an existing computer system about to be freed from Vanguard work.

NASA awarded Western Electric the overall contract for the tracking and ground systems to be used in Project Mercury on July 30, 1959⁶. By late 1959, IBM received the subcontract for computers and software⁷. Washington remained the site for the computer system because it could benefit from centralized communications already in existence⁸. NASA founded Goddard Space Flight Center the next year, and since it was less than half an hour from downtown Washington, the same advantages would accrue from locating the

computers there. Combined NASA and IBM teams used the old computer system downtown until about November 1960, when the first of Mercury's new 7090 mainframe computers was ready for use at Goddard. James Stokes of NASA remembers the first time he and Bill Tindall went to the new computer center, they had to cross a muddy parking lot to where a "building" with plywood walls, window air conditioners, and a canvas top confounded the IBM engineers who were trying to keep the system up and running under field conditions⁹. That structure evolved to become Building Three of the new Space Flight Center and housed the system through the Mercury era¹⁰.

IBM's 7090 mainframe computer was the heart of the Mercury control network. In 1959, the DOD issued a challenge to the computer industry in the form of specifications for a machine to handle data generated by the new Ballistic Missile Early Warning System (BMEWS). The 7090 was IBM's response. Essentially an improvement of the 700-series machines like the one being used as a development machine for Mercury, the 7090 adapted the new concept of I/O channels pioneered in the 709 and was so large that it needed up to three small 1410 computers just to control the input and output. The DOD's needs for BMEWS closely paralleled those of Mercury in terms of data handling and tracking. Thus, IBM was in a good position with its hardware.

To provide the reliability needed for manned flight, the primary Mercury configuration included 7090s operating in parallel, each receiving inputs, but with just one permitted to transmit output. Called the Mission Operational Computer and Dynamic Standby Computer, the names stuck through the Apollo program. This was NASA's first redundant computer system. Switching from the prime computer to the Dynamic Standby was by manual switch, so it was a human decision¹¹. During John Glenn's orbital mission, the prime computer failed for 3 minutes, proving the need for an active standby¹².

Three other computers completed the Mercury network. One was a 709 dedicated to continuously predicting the impact points of missiles launched from Cape Canaveral. It provided data needed by the range safety officer to decide whether to abort a mission during the powered flight phase and, if aborted, information about the landing site for the recovery forces. Another 709 was at the Bermuda tracking station with the same responsibilities as the pair at Goddard. In case of a communications failure or double mainframe failure it would become the prime mission computer. Lastly, a Burroughs-GE guidance computer radio-guided the Atlas missile during ascent to orbit¹³.

Locating the computers near Washington while placing the mission control personnel at Cape Canaveral led to a communications problem that resulted in a unique solution. In early digital computers, all input data went to memory by way of the CPU. Large amounts of data that needed to be accepted in a short time often backed up, wait-

ing for the central processor to handle the flow. A solution is direct memory access, which sends data directly from input devices into storage. Transfers of large blocks of data directly to memory are conducted through data channels, first used by IBM on its 709 and then on the 7090. By using channels, processing could continue while I/O occurred, increasing the overall throughput of the system. Mercury's 7090s were four-channel systems. Normally, the peripherals handling input and output would be connected to the channels physically close to the machine, but the peripherals (plotters and printers) driven by the Mercury computers would be about 1,000 miles away in Florida. The solution was to replace Channel F of the 7090 with an IBM 7281 I Data Communications Channel, a device originally created for Mercury that has had great impact on data processing¹⁴.

Four subchannels divided the data handled by the 7281 device. One was an input from the Burroughs-GE guidance computer to provide data used in calculating the trajectory during powered flight. The second input radar data for trajectory and orbit determination. Two output subchannels drove the displays in Cape Canaveral's Mercury Control Center and locally at Goddard¹⁵.

Connecting the two ends of the system was a land line allowing transmission at 1,000 bits per second¹⁶. Although this was a phenomenal rate for its time, now a simple microcomputer routinely transmits at 1,200 bits per second on nondedicated public telephone lines. The distance and newness of the equipment occasionally caused problems. Once in a while during a countdown, data such as the lift-off indicator, which was a single bit, would get garbled and give erroneous signals¹⁷. Most times such flags could be checked by other sources of information, such as radar data contradicting the lift-off message. Also, up to a 2-second time lag on the displays in the control center was common¹⁸. During powered flight, such delays could be significant; thus, the need for a separate impact prediction computer and another machine in Bermuda.

Software development for the Mercury program was another area in which IBM advanced the state of the art¹⁹. In the beginning of the computer era, operators ran programs on computers one at a time. Each program was assigned peripherals, loaded, run and, if errors occurred, stopped individually. As machines grew larger and the number of users increased, some way of making the process of loading and executing programs more efficient was needed. The result was the concept of "batch" processing, in which a set of several programs could be loaded as a unit and executed in sequence. A special control program called a "monitor" watched for errors and aborted programs trapped in loops or that spun off into corners. To handle the many jobs needed by manned spacecraft mission control, IBM set up a method for programs to be interrupted and suspended while other programs of greater priority ran, and then resumed when the high-priority jobs

ended. Thus, a number of programs could be loaded into the machine and run, giving the illusion of simultaneous execution, even though only one had the resources of the central processor at any one time. This was the only way the processing of radar data, telemetry, and spacecraft commands could be accomplished in the split seconds of time allotted.

IBM called the control program the Mercury Monitor, but that is a misnomer in that it superceded the capabilities of the known monitors of the time. It was event driven, which means that certain flight events (lift-off, sustainer engine cutoff, retrofire) formed the basis of the starting times of certain processes²⁰. The Mercury Programming System's primary functions included capsule position determination, retrofire time calculation, warning ground stations of the acquisition times, and impact prediction after retrofire. Three separate groups of processing programs, each stored on tape until needed, did these functions at different times: launch, orbit, and re-entry²¹. No matter which group of processors was loaded into the machine, the Monitor frequently checked a table listing processes waiting for input or output. Software placed entries in the table when the Data Communications Channel signaled that data were ready to be transferred²². The Monitor then handled the requests in priority order. Within a processor group, such as orbit, a set of different single-function processors would be defined. Thus, the entire mission control program was highly modular, allowing easier maintenance and change. In fact, some modules from the Vanguard programs could be adapted to Mercury use.

NASA wanted to take over the software as soon as possible, so 15 or so civil service employees were assigned to the IBM group while it was still in downtown Washington. However, the Space Task Group retained direct control over the software development, a somewhat frustrating situation for NASA engineers much closer to the actual project and in a better position to make suggestions²³. At the time, NASA saw its role as that of a knowledgeable user and recognized it lacked the expertise to handle some of the calculating tasks involved. James Stokes, a NASA engineer, admitted that "we didn't know enough to specify the requirements" for the software²⁴. IBM was not much better off and acquired its expertise by contracting for the services of Dr. Paul Herget, then director of the Cincinnati Observatory, who had privately published a book on orbit determination in 1948²⁵.

The Mercury network provided continuous height, velocity, flight path angle, retrofire time, and impact points. During powered flight, the main computer center, the Cape impact prediction computer, and the Bermuda tracking station computer all would give GO/NO GO recommendations to the flight director. After engine shutdown, the system needed to give GO/NO GO data within 10 seconds, so that a safe recovery could be effected if orbit had not been reached. During

the orbital cruise, the astronaut could be given updated retrofire times each time he came in contact with a ground station²⁶.

As the Mercury program wound down during 1962 and NASA began to accelerate preparations for Gemini and Apollo, the Agency decided to place both the computers and flight controllers for manned spaceflight mission control in a combined center in Houston. Goddard staff proceeded under the assumption that the new control center would not be ready in time for the first Gemini flights, which turned out to be correct. Gemini I, II, and III used Goddard as the prime computer center, with the new system in Houston acting in an active backup role for flight three. Beginning with flight four, the second manned mission, Houston took over as prime, with Goddard acting as the backup throughout the Gemini program²⁷.

For IBM and NASA, the development of the Mercury control center and the network was highly profitable. IBM's Mercury Monitor and Data Communications Channel were the first of their types²⁸. Future multitasking and priority interrupt operating systems and control programs owed their origins to the Monitor. Large central computers with widely scattered terminals, such as airline reservation systems, have their basis in the distant communications between Washington and a launch site in Florida. For both organizations, the experience gained by staff engineers and managers directly contributed to the success of Gemini and Apollo.

Second System: The Gemini–Apollo RTCC

Before the first Mercury orbital flight was off the ground, NASA engineers working on mission control tried to influence the design of the new center in Houston. Bill Tindall, who worked on ground control for NASA from the beginning, realized that locating the Space Task Group management at Langley Research Center, the computers and programmers at Goddard, and the flight controllers at Cape Canaveral created serious communication and efficiency problems. In January 1962, he began a memo campaign to consolidate all components at one site, obviously the new Manned Spacecraft Center²⁹. On February 28, just 8 days after John Glenn's flight, Tindall made his strongest case in a detailed essay in which he noted that IBM was the only company capable of creating real-time software. He wanted the Ground Systems Project Office, then in charge of oversight of the RTCC development, to allow representatives from the Flight Operations Division to assist in mission programming³⁰. As the eventual users of the system, it made sense to include them.



Figure 8-1. IBM 7094s in the Gemini Real Time Computer Complex. (IBM photo)

In April, the Western Development Laboratories of Ford's subsidiary Philco Corporation began a study of the requirements for the new mission control center. One aspect of the study was to take numeric data and give it pictorial content, making the jobs of the flight controllers less hectic but necessitating much more sophisticated computer equipment³¹. As Philco worked through the summer, NASA Administrator James Webb announced on July 20 that there would be an expanded replacement for Mercury Control. A "request for proposal" was prepared, including concepts developed by Philco and documented by them in their final facilities design released on September 7.

Philco's design was broad in scope, covering physical facilities, information flow, displays, reliability studies, computers, and even software standards. Philco specified that modularity in program development was a must, as it would ease maintenance and allow the use of "lower caliber" people to code subprograms, leaving the real stars to do the executive software³². This organizational rule became standard for large program projects. Another specification required that the probability of successful real-time computer support for a 336-hour mission be 0.9995. Also, due to rendezvous plans for Gemini and the dual-spacecraft Apollo lunar missions, the center had to control two spacecraft at one time. To meet the reliability and processing goals, Philco examined existing computer systems from

IBM, UNIVAC, and Control Data Corporation, as well as its own Philco 211 and 212 computers, to determine what type and how many would be needed. The calculations resulted in three possible configurations: five IBM 7094s (the immediate successor to the 7090, essentially a faster machine with a better operating system, IBSYS); nine UNIVAC 1107s, IBM 7090s, or Philco 211s; or four Philco 212s or CDC 3600s³³. No matter which group would be chosen, it was obvious that the complexity of the Gemini–Apollo Center would be much higher than its two-computer predecessor. To help keep the system as inexpensive and simple as possible, NASA specified to potential bidders that off-the-shelf hardware was essential.

IBM moved quickly to respond to NASA's call for proposals, delivering in September a 2-inch thick, three-ring binder full of hardware and software bids, including a detailed list of personnel they would commit to the project, complete with employment histories. Although the company knew it was the leading candidate (Tindall's endorsement could hardly have escaped notice), it carefully matched the specifications, such as clearly stating that modularization and unit testing would be the norm in software development. One area in which they differed from Philco's calculations was the number of machines needed. Perhaps to keep the total bid low, IBM proposed a group of three 7094 computers. By splitting the software into a Mission Computer Program and a Simulation Computer Program, one machine could run the Mission Program as prime, another run it as the dynamic backup, and the third run the simulation software to test the other two, thus fulfilling requirements for redundancy and preflight training and testing. This forced IBM to explain its way around the 0.9995 reliability requirement. Three machines yielded reliability of 0.9712, slightly over four being needed to achieve the specification (thus, Philco's suggested number of five). IBM made a case that the reliability figures were misleading and that during so-called "mission-critical" phases the reliability of three machines would exceed 0.9995³⁴.

Eighteen companies bid on the RTCC, including such powerful competitors as RCA, Lockheed, North American Aviation, Computer Sciences Corporation, Hughes, TRW, and ITT. NASA assigned Christopher Kraft, the eventual chief user, to chair the source board that studied the responses to the request for proposal. Tindall served also, with James Stroup, John P. Mayer, and Arthur Garrison, all of the Manned Spacecraft Center. They awarded the original contract NAS 9-996, covering the Gemini program, to IBM on October 15. Worth \$36 million, it was to run until the end of August 1965. Extended to December 1966, the total cost came to \$46 million³⁵.

With 6 weeks of preparation already done before the contract award, IBM's core of engineers were ready for business in Houston by October 28. J. E. Hamlin started as project manager and interim

head of systems engineering. He had 12 years of IBM experience, first as a hardware engineer, later as a group leader for SAGE software, and then manager for the Mercury system implementation. He had barely started work at JPL's Deep Space Instrumentation Facility when the RTCC contract came up. In his first report in January 1963, he was able to announce the arrival of the first 7094 to be used for software development. The computer and, later, two others were installed in an interim facility on the Gulf Freeway. Each started with 32K words of memory and 98K words of auxiliary core storage, with a 1401 as a front end for input and output³⁶. On the negative side, Hamlin's early projection of a peak staff of 161 had leaped to 228 by the time of the first report. Eventually, 608 IBM people worked simultaneously on the project, with 400 of them on software development. The magnitude of the task was greatly underestimated both by IBM, which made the bid, and NASA, which accepted it.

Hardware needs grew along with the staff. The original three machines moved from the interim center to Building 30 at the Manned Spacecraft Center. Two more were added, fulfilling Philco's prophecy. The size and rating of the machines was also increased to model 7094-II's with 65,000 words of main core storage and 524,000 words of additional core as a fast auxiliary memory³⁷. In the new configuration, one machine was the Mission Operational Computer, the second, the Dynamic Standby Computer, and the third, the Simulation Operations Computer as before, with the two new ones used as the Ground System Simulation Computer and a standby for future software development. The Ground System simulator acted like the tracking network and other ground-based parts of mission control to test software.

IBM's original proposal projected completion of the new system within 18 months. As time passed and problems occurred, the plan altered to begin with support of the Gemini VI mission. But slips in Gemini and steady progress on the software enabled the use of the Center for passive parallel computations during the Gemini II unmanned flight on December 9, 1964, just under 26 months after the contract award. On Gemini III, the Houston control center did its final test as an active backup. The results were so promising that from Gemini IV on, mission control shifted from the Cape to Houston.

Gemini Ground Software Development

NASA's requirements for the Gemini mission control software resulted in one of the largest computer programs in history. In addition to all the needs of the Mercury system, Gemini's proposed rendezvous and orbit change operations caused a near-exponential in-

crease in the complexity of the trajectory and orbit determination software. Placing a computer on board the spacecraft made it necessary to parallel its computations as a backup and also necessary to devise a way to use the ground computer system to update the Gemini flight computer. Also, by the the time the Gemini program matured, all data on the tracking network were in digital form, and thus compatible, so the amount of data that passed through the ground system increased further³⁸.

IBM reacted to the increased complexity in several ways. Besides adding more manpower, the company enforced a strict set of software development standards. These standards were so successful that IBM adopted them companywide at a time when the key commercial software systems that would carry the mainframe line of computers into the 1970s were under construction³⁹. IBM approached the more difficult areas by acquiring the services of specialist consultants and sponsored a group of 10 scientists pursuing solutions to problems in orbital mechanics. It included Paul Herget and some men from IBM's Cambridge, Massachusetts "think tank"⁴⁰.

Key to the flight system was the Mission Computer Program. It centered on a control program called the Executive, which took over the functions of the Mercury Monitor. Under the Executive, three main subprograms operated in sequence. NETCHECK performed automatic tests of equipment and data flow throughout the entire Manned Spaceflight Network, certifying it ready for the launch of the spacecraft. It succeeded the CADFISS (Computation and Data Flow Integrated Subsystem) program used in Mercury⁴¹. ANALYZER did postflight data reduction. However, the Mission Operations Program System remained the heart of the software, responsible for all mission operations, such as trajectory calculations, telemetry, spacecraft environment, backup of the on-board computer, and rendezvous calculations. It divided into a number of modules: Agena launch, Gemini launch, orbit, trajectory determination, mission planning, telemetry, digital commands, and re-entry, with several subprograms within each section⁴². Each subprogram was highly sophisticated and very powerful. The re-entry program, for example, could calculate retrofire times 22 orbits in advance⁴³.

IBM found it impossible to complete this complicated system with the tools used in the Mercury program. All of the Mercury control software was in assembly language. Aside from the assembler, software tools were minimal, reflecting the state of the art circa 1960. Partly inspired by the difficulties of developing a large system such as Mercury and SAGE and partly to help commercial customers creating new software to match the size and capabilities of the new line of mainframe computers, IBM provided a much better set of tools with its 7094 series machines than with earlier models. A fairly robust operating system, IBSYS, could be used with the 7094, and a

modification of it gave the Gemini software developers a decent editor and compilation tools for high-level languages. Called the Compiler Operating System, it included a combination FORTRAN/Mercury compiler called GAC (for Gemini-Apollo Compiler), making it possible to do some programming in FORTRAN. The Mercury compiler contained all the functions of SOS, the Share Operating System, which was IBM's standard system of the late 1950s and the predecessor to IBSYS⁴⁴.

Besides using better tools, the Gemini programmers tried to keep the architecture simple and changeable. Using process control tables was an important design decision, as they could be changed to fit different mission requirements with some ease and without disturbing software in place. Their use continued throughout the Apollo and Shuttle programs⁴⁵. The Executive was a further refinement of the real-time control program first approached in Mercury. A relatively spare 13,000 words in size, the Executive provided priority-based multiprogramming. It could transfer needed data to supervisory routines which, in turn, started processes⁴⁶. At the lowest level, contention between cyclic processes and demand processes characterized the RTCC⁴⁷. Its obvious success helped form NASA's ideas of what a good real-time operating system should be, which later influenced the nature of the operating system on board the Shuttle. NASA personnel were close to the Gemini-Apollo ground system development, sometimes defining test cases and duplicating programs to check whether requirements had been met⁴⁸.

Even with better tools and a more powerful computer, the processing needs of the mission control software quickly exceeded the capacity of the 7094. IBM recognized that the usual 32K memory of the machine would be insufficient when the company prepared its proposal. Therefore, it suggested the use of look-ahead buffering, which meant the next set of programs needed during a mission would be loaded over the ones going out of use⁴⁹. The commercial practice of using tape storage for waiting programs became impossible due to the size and speed demands of the Gemini software. Thus, IBM added large core storage (LCS) banks to the original machines. These banks, even though not directly addressable, provided a higher speed secondary memory. Tapes would be loaded to the large core and then transferred to primary storage as needed⁵⁰. An IBM engineer credited work in the use of LCS and paging memory as being influential in the development of IBM's version of virtual memory, the main software technological advance of its fourth generation 370 series machines of the early 1970s⁵¹. As the Gemini program continued, NASA grew more concerned about the ability of the 7094s to adequately support Apollo, considering the expected greater complexity of the navigation and systems problems. Kraft expressed concern that the "real time" in the RTCC needed enhancement⁵². As the large core filled, loading

from tape for certain programs became common practice. Once, when President Lyndon B. Johnson was visiting the control center, the NASA official leading the tour wanted to show the president a fancy display. Not fully conversant with the software, he chose one that ran off tape, so the entire party stood uncomfortably, minutes seeming like hours, while the machine dutifully found the program and put up the display⁵³. NASA wanted a change.

It was about this time that IBM announced its System 360 series, a compatible line of several computers of different sizes using a new multiprocessing operating system that owed some of its characteristics to the company's NASA experiences. NASA thought the upper level machines of the new product, specifically the 360/75, would have sufficient power to replace the 7094s for Apollo, although the LCS would have to be continued due to the sheer size of the software. IBM's announcement, as is usual with the company, preceded the shipping dates of the machines by some months. It did not take long for NASA to realize this and become impatient. Control Data Corporation (CDC) released its 6600 line of computers in 1965 and was actually shipping to customers as IBM failed to deliver. Robert Seamans of NASA Headquarters suggested that the Manned Spacecraft Center buy 6600s and let IBM retain the software contract⁵⁴. CDC's machine was actually faster and more powerful than the 360. Later, CDC sued IBM, claiming its premature 360 announcement sought to hold the market and that claims made for the 360 were not realized when the product actually came out. IBM settled out of court with major concessions totaling nearly \$100 million, rushing delivery of the first 360 to Houston in time to stave off the movement to other vendors. NASA announced the conversion to the 360 in a news release dated August 3, 1966.

Transition to Apollo

Although the four remaining 7094 computers continued to support flight operations through the first three Apollo (unmanned) missions, IBM used the first replacement 360 to begin software development for the Apollo lunar flights. As in Gemini, two spacecraft, the command module (CM) and the lunar excursion module (LEM), needed support, with five computers each contributing to the overall system. Again, LCS provided added memory. Unfortunately, all the software could not be moved directly from one machine to the other due to the change in operating systems. The new operating system for the series, OS/360, had the multitasking capability developed during Mercury days but operated primarily in batch mode. Many programs could be entered, either by cards or through remote entry from terminals, and run together, but not in real time. The priority-interrupt

provisions on the standard operating system were not sophisticated enough to handle the sorts of processing Apollo needed. Beginning in 1965, IBM modified the operating system into RTOS/360, the real-time version⁵⁵. Extensive use of modularization helped in the transition. Separately compiled subprograms in FORTRAN, moreover, could be moved to the 360 with relative ease, but the assembler-based code had to be modified. This work continued for nearly as long as it took to get the original system operating, even though the architecture remained essentially intact.

One problem would not go away: memory. Each 360 had 1 million bytes of main memory, about four times the size of 7094 main store. A further 4 million bytes of LCS was added to each machine⁵⁶. Even with some of the NETCHECK functions transferred to the new twin 360s in the Goddard Real-Time System (GRTS) and with seldom-used programs such as the radiation dosage calculator and ground telescope pointing program permanently located off-line, memory use rose to match the additional space. Simply meeting the requirements for ascent filled the main store⁵⁷. At this time, NASA's Lynwood Dunseith, who had worked on the ground software since Mercury, realized that the worry over memory was causing programmers to develop idiosyncratic, "tricky" code in an effort to save a few words⁵⁸. Dunseith knew the danger of that attitude, since it made the programs even more complex than their absolute complexity warranted. During the period he managed the software development, he tried to reduce the dependence on such expedients. It helped him that the 360s made it possible to develop significant parts of the software in FORTRAN⁵⁹. Although FORTRAN is not as easily readable as some other procedural languages, it far exceeds 360 assembler in understandability.

As the Apollo system moved into the operations phase, the use of the Dynamic Standby Computer waned. During the first manned flight, Apollo 7, the Mission Control Center used a single computer for just under 181 hours of a 284-hour support period, which included countdown and postflight operations⁶⁰. During Apollo 10, a dual spacecraft flight with LEM operations near the moon, the plan was to use the standby for 5 hours before a maneuver. Therefore, on only six occasions in an 8-day flight would there be two-computer support. To assist an off-line standby in coming to the rescue of a failed primary, operators made checkpoint tapes of current data every 1.5 hours. A failure of the Mission Operations Computer occurred at 12:58 Zulu on May 20, 1969. By 13:01, the standby had been brought up, using a checkpoint tape made at 12:00⁶¹. No significant problems resulted, which is actually a good summary of mission control operations throughout the Apollo era, Skylab, and the Apollo-Soyuz Test Project.

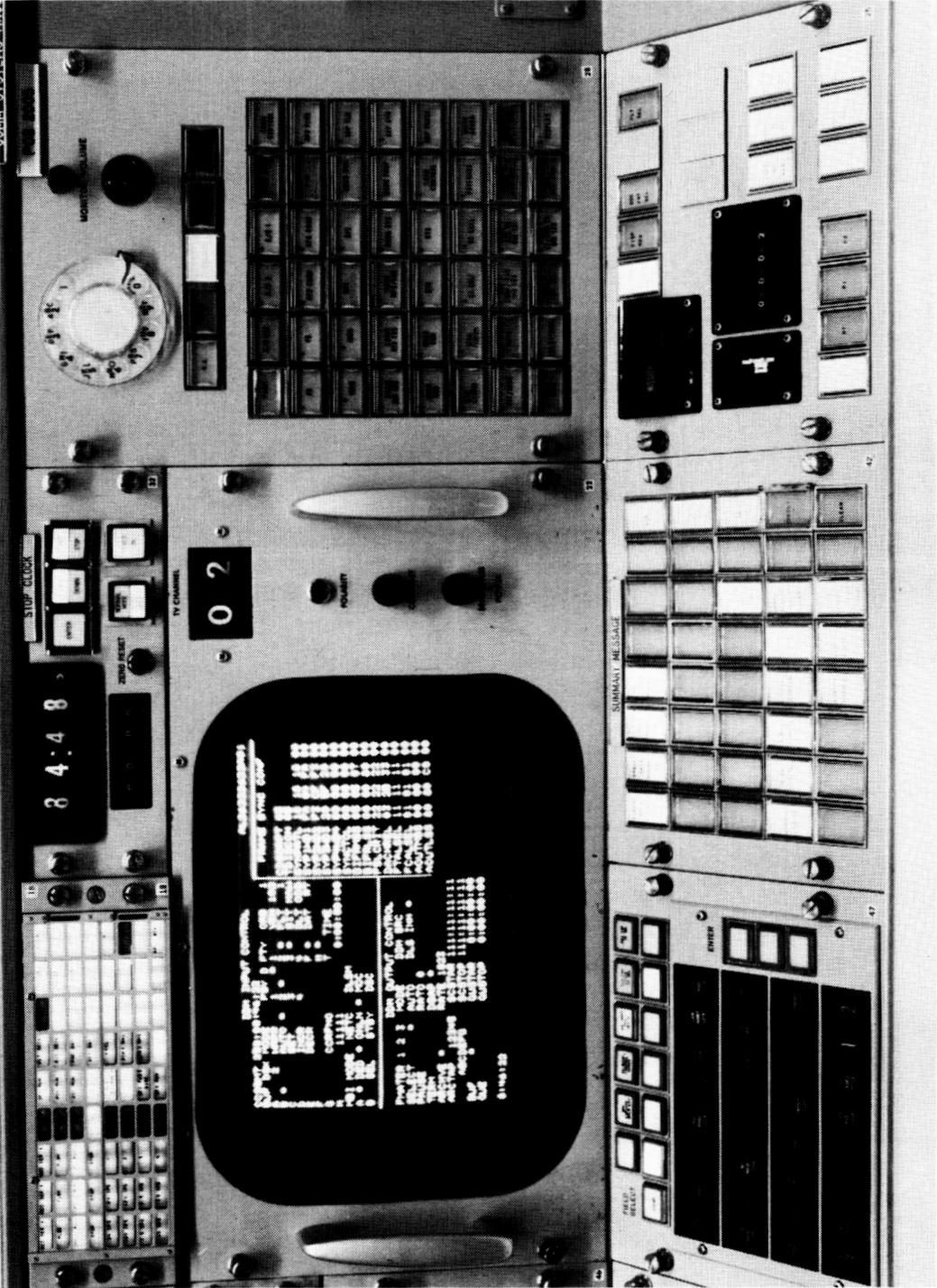


Figure 8-2. A display and control panel in Mission Control for the Shuttle program. (NASA photo S-80-26315)

ORIGINAL PAGE IS
OF POOR QUALITY

Reducing Mission Control: Conversion to the Shuttle

During planning for the Space Transportation System, with frequent launches and multiple missions aloft expected, NASA studied ways to make the spacecraft more autonomous and thus reduce the functions of mission control. IBM again won the ground support contract, this time over primary competitor Computer Sciences Corporation⁶². Beginning in June 1974 and continuing into the 1980s, IBM worked on a new software system and mission-specific changes⁶³. Five System 370/168 mainframe computers make up the Shuttle Data Processing Complex, the nominative successor to the RTCC. Each has 8 million bytes of primary storage, and, being virtual memory machines, do not need auxiliary storage of the LCS type. Disk is used instead. Three computers are involved during operations: One computer is the Mission machine, one, a Dynamic Standby Computer, and a third, the Payload Operations Control Computer. Now, in the late 1980s, these computers are being replaced by IBM 3083 series machines, marking Mission Control's fourth generation.

By this time, quite experienced and fairly knowledgeable about what would be needed, NASA and IBM approached the ideal of thorough design before coding began⁶⁴. Reflecting the structure of the on-board software, the requirements documents proceeded through different levels of complexity. For the first time in ground software development, a quality assurance group from outside the development organization watched over software production⁶⁵.

The efficiency of the software developers increased with the conversion from batch processing to interactive processing. During Mercury, Gemini, and Apollo, programmers tested new software in batch. With the main IBM Federal Systems Division office nearly a mile from the actual computers housed in Building 30, it was necessary for a courier to pick up card decks, deliver them to the Computing Center, and later return the results. In this manner, an average of only 1.2 runs per programmer per working day was possible. During 1974–1976, NASA commissioned a study of batch versus interactive programming, in which programmers using terminals could prepare jobs and run them from the IBM building. Using IBM's Time-Sharing Option (TSO) system, interactive processing clearly won out over batch in terms of effectiveness. NASA accordingly ordered all Shuttle ground software to be done under the time-sharing system⁶⁶.

Regardless of the intentions of the Shuttle managers to shrink the ground operations software, the ground support functions provided by the Data Processing Complex have not been reduced. Some parts of the original tasks are handled more completely on-board, but the continued addition of new equipment and concepts increased the size of the software. It supports over 40 digital displays and 5,500 event

lights. The total size of the system is 600,000 lines, roughly 26% larger than Gemini and rivaling Apollo⁶⁷. Shuttle missions are approaching the complexity that a single computer can no longer support⁶⁸. In addition, high between-flight change traffic delayed the transition to the operations era. As late as 1983, 8% of the total code changed each mission, keeping 185 programmers busy. New and more powerful computers can always be added, but the process of changing software must be automated or the expense of labor intensive maintenance will continue to the end of the Shuttle program.

UNMANNED MISSION CONTROL COMPUTERS

Mission control of unmanned spacecraft is significantly different from that of manned spacecraft. Most important of the differences is the long duration of many unmanned flights. Except for Skylab, no American manned flight has lasted more than 2 weeks. In contrast, when Voyager 2 encounters Neptune in 1989, it will have flown for 12 years. During that time, the Voyager Project staff must monitor the health of the spacecraft and gather and interpret the data it is collecting. Few of the original engineers will still be associated with the mission, so conceptually mission planning for a long-duration unmanned flight must concentrate on an extended view of operations and the development of detailed documentation⁶⁹. Another difference is that the manned mission control centers are used for one project at a time, whereas the unmanned centers may be controlling a wide variety of missions. So far, there has been no overlap in the manned *programs* in the sense that no Mercury flights continued after Gemini flew, and so on. In contrast, the Jet Propulsion Laboratory (JPL) commanded Surveyors, Lunar Orbiters, Pioneers, and Mariners all at once in the mid-1960s, and has continuously been responsible for multiple missions.

Control of Near-Earth Missions at the Goddard Space Flight Center

NASA formed Goddard Space Flight Center with the Naval Research Laboratory's Vanguard Project team as a nucleus. After Vanguard ended, use of the IBM 704 in downtown Washington ceased, and a model 709 was installed at Goddard on May 23, 1960, as a replacement machine for use in working with earth-orbiting satellites. Within 2 months, the first of six 7090 computers also arrived. Folklore has it that Goddard soon housed 1% of the total computing power in the entire United States. Although two of the 7090s and later

other computers supported Mercury flights, Goddard's most substantial customer base has been the plethora of scientific, navigational, communications, mapping, and weather satellites launched in the last quarter of a century.

Goddard pioneered the use of dedicated small computers for specific missions, thus eliminating the complexity of handling multiple missions on a single mainframe. This occurred in spite of the presence of large numbers of big computers. Some command and control and definitely navigation calculations are carried out on large machines, but each project has a small computer to handle data reduction and the day-to-day operation of the spacecraft. As examples, the Nimbus weather satellite program used Control Data 160A computers, the Orbiting Geophysical Observatory had Scientific Data Systems SDS 910 and 920 computers, and so did the Orbiting Solar Observatory⁷⁰. These machines could be sent on to another project when their current job ended, and in fact some of the SDS machines had rather long lifetimes of nearly two decades. In addition to using small computers at the control center, Goddard installed UNIVAC 1218 computers in the Manned Spaceflight Network ground stations, originally for control of Gemini and Agena and later for Apollo. Both the 160As and 910s were among the first products of their respective fledgling companies, and, with the 1218 and Digital Equipment Corporation's PDP series, the forerunners of the minicomputer boom of the 1970s.

Relatively little changed in the general techniques of mission control at Goddard for about two decades. As the 1980s continue, the trend is for the majority of unmanned satellites to be commercial rather than scientific in nature. Commercial satellites are controlled by their owners, although NASA provides orbit determination and some command services on a reimbursable basis. However, sufficient missions exist, such as the expected 17-year duration Hubble Space Telescope, to keep Goddard involved in ground control activities for some time, along with its continued commitment to NASCOM and STADAN.

To the Sky's Limit: Mission Control at JPL

As Goddard strove to standardize earth orbital operations and distribute its functions, JPL approached the similar problems in a different way, centralizing operations as much as possible. In many respects, Goddard and JPL are fraternal twins. Each has a set of ground-tracking stations, plus on-site control centers for a variety of missions. The difference is that JPL is responsible for deep space exploration. In fact, the lower limit of its responsibilities is set at 10,000 miles. For a short period, it did satellite work. JPL developed the

guidance system and propulsion for the Sargeant battlefield missile and studied adapting clusters of the motors as upper stages to the Redstone missile. The resulting Jupiter-C launch vehicle put America's first satellite into orbit on the night of January 31, 1958. Called Explorer I, the satellite carried JPL-developed instrumentation. A room near the office of laboratory director Dr. William Pickering became an active unmanned mission control center since it contained communications equipment connected to the tracking network that confirmed Explorer reached orbit. That same year NASA was formed and JPL became closely affiliated, changing its mission to deep space work.

In 1959, the early Pioneer flights aimed at the moon. JPL built a series of tracking stations, beginning at Goldstone in the high desert of California, to track the missions⁷¹. Unlike earth orbiters, whose closeness to the planet make it necessary to have a large number of stations to stay in contact, deep space probes needed only three stations spaced so that one would always face the spacecraft. Initially, the stations were located in Australia and South Africa as well as at Goldstone, but later one in Madrid replaced the African station and the Australian one moved from Woomera to Canberra⁷². The stations were collectively named the Deep Space Instrumentation Facility.

From the beginning, JPL considered using computers in the stations as data-gathering devices. One 1959 report suggested using IBM 650 machines, which were small computers⁷³. In 1962, Dr. Eberhardt Rechtin, head of the Instrumentation Facility, sent Paul Westmoreland and Carl Johnson to evaluate the computers of Scientific Data Systems, a new company⁷⁴. Westmoreland and Johnson thought that the SDS 910 could be used as the data gatherer, with the slightly more powerful 920 as a data processor. Accordingly, Rechtin directed that the machines be ordered and got the first 920 built and the second 910. The 910s and 920s still functioned in similar tasks as late as 1985!^{**}

Functionally, the SDS computers took data received from the spacecraft and formatted and recorded it on magnetic tape. A computer at JPL processed the data more completely. Initially, an IBM 704 similar to the one used for Vanguard did the work. JPL installed the computer in late 1958 to use with Pioneer 3 and 4⁷⁵. Early Ranger lunar impact flights later had all data reduction done off tape on that machine. Data in analog form on the tapes would be translated into numbers that spewed out on teletypes and punched paper tape. Aerojet-General Corporation also owned a 704 that JPL used as a backup⁷⁶.

^{**}After 1968, the SDS machines were known as XDS 910 and 920. Xerox bought out SDS and renamed the products "Xerox Data Systems."

Planning for the first Mariner missions revealed that more computing power would be needed at JPL to handle the increased data generated both by more instrumentation and longer mission lifetimes. Dual 7090 computers similar to those installed at Goddard were bought for data reduction. To provide flight controllers with more up-to-date information about spacecraft telemetry, a Digital Equipment Corporation PDP-1 computer served as a near-real-time data processor. Data could be displayed on teletypes from 4.5 to 7 minutes after it was received⁷⁷. By this time the Deep Space Instrumentation Facility could transmit data via NASCOM instead of having to wait for airmail to deliver the tapes. Operations with this equipment taught JPL at least one useful lesson: Power fluctuations in September and December 1962 caused both 7090s to go down at once, eliminating the redundant capability⁷⁸. As a result, JPL built an auxiliary power generation facility, perhaps leading the manned Mission Control Center, under construction at this time, to do the same.

Centralizing the Effort

During the 1960s, NASA found itself about to be involved in a large number of critical deep space projects. Ranger would be followed by the Surveyor series of lunar landing missions. Mariners would continue to fly to Venus and Mars, with several targeted for Martian orbit and imaging duty. Lunar Orbiters would look for Apollo landing sites and Pioneers were aimed at deeper space. JPL did not have primary responsibility for all of these programs. Lunar Orbiter came from Langley Space Flight Center, and Pioneer from the Ames Research Center. If each responsible organization had to set up a control center for its spacecraft, considerable overlap and duplication would occur. Accordingly, in 1963, NASA decided to have JPL track and command all deep space missions, with the help of project personnel from home centers stationed at JPL⁷⁹. On December 24, 1963, JPL's director William Pickering formally established the Deep Space Network⁸⁰. Managed by William H. Bayley, with Eberhardt Rechtin as technical head, it would serve all of NASA, just like NASCOM did from Goddard⁸¹.

JPL was already building a Space Flight Operations Facility to house new, more powerful computers and the various teams from its own projects. Anticipating NASA's decision, Eugene Giberson, then of the Surveyor project, directed some of his money to help develop the centralized computer center⁸². The combination of the Operations Facility and the Deep Space Instrumentation Facility was the Deep Space Network. After opening on May 15, 1964, the Operations Facility supported Mariner Mars 1964 as its first flight⁸³.

Even though expected to handle all deep space missions, some organizations fought to retain mission functions. Ames set up the Pioneer Off-Line Data Processing System (POLDPS) in 1965 to handle non-real-time data recorded by the SDS 910s at the stations⁸⁴. Both the Lunar Orbiter and Surveyor projects also wanted to record their telemetry data at the stations, so the Network bought dual SDS 920s for each site. Later, Pioneer 10 and 11 data were processed with these systems⁸⁵. Langley originally wanted to control the Viking Lander, but costs and common sense forced that job back to JPL.



Figure 8-3. The Space Flight Operations Facility central control room at the Jet Propulsion Laboratory. (JPL photo P23358BC)

Evolution of the Space Flight Operations Facility

JPL's Space Flight Operations Facility has had three generations of equipment. Beginning in 1964, two strings of solid-state computers formed the basis of the system. Each consisted of an IBM 7094 mainframe, an IBM 7040 medium-sized computer, and an IBM 1301 disk storage system placed between them. Later, a trio of System 360/75 computers replaced this configuration. More recently, the control center adopted a distributed computing strategy similar to Goddard's.

As in the manned programs, during critical mission phases both strings of the original generation of equipment would be running at the same time but with the data from the stations only routed to one of them. If a 7094 failed, its associated 7040 could be connected to the other 1301 (and, thus, the second 7094), leaving the second 7040 as another layer of backup⁸⁶. Later upgraded to a 7044, the smaller computer acted as a traffic cop on the incoming data. All inputs (teletype, telephone, microwave) went to the machine before they went anywhere else, and the software in the 7040 routed the data to active programs, inactive programs, or administration stations⁸⁷. George Gianopolis of JPL, one of those charged with the responsibility of getting the system to work, remembers that the 7040s were especially difficult to install⁸⁸. The 7040s deposited data on the 1301 disk storage system. A 54-megabyte hard disk, the 1301 served both the 7040 and the 7094 from the middle, so both could access data at identical addresses. This concept presages the network file servers in the modern office and the Common Data Buffer in the Launch Processing System. Airline reservation systems and other large data base operations utilized the same configuration beginning at about the same time. Using a smaller computer to handle resource-hungry input and output tasks and a common storage area is a standard network concept today. As for the 7094, the flight operations director could control its use by "percentage time sharing" in which higher priority jobs simply got more machine time⁸⁹. The primary functions of the 7094 were telemetry analysis, tracking, predictive work for the stations, and maneuver commanding. UNIVAC computers in the JPL institutional computer center did the navigation calculations as batch jobs, separate from the Operations Facility computers⁹⁰.

Although a powerful system, the 7040/7094 combination had to stretch to meet mission requirements. Upgrading it to 7044/7094 Model II status helped some, but the system could handle only a Mariner mission (two spacecraft) or a Surveyor but not both⁹¹. Surveyor project officials even had to add a PDP-7 as a front end computer to the front end computer, putting it between the stations and the 7044 and driving strip chart recorders⁹². More assistance came during the Mariner Project when engineers realized the UNIVAC 1218 computers used in preflight testing of the spacecraft could also do engineering telemetry analysis⁹³. This was not done until Mariner Mars 1971. Soon, though, the acquisition of more powerful 360 series machines ended the reign of the 7094s.

Monolithic Computer Systems

In October of 1969, JPL installed its first System 360/75, a gift from the Manned Spacecraft Center, where it was considered surplus. A second machine arrived in April 1970, this one left over from the demise of NASA's Electronics Research Center in Cambridge, Massachusetts. JPL bought a third machine, which survived until August 1983⁹⁴. Each 360 had 1 megabyte of primary core storage and 2 megabytes of LCS, half that of an Apollo-configured machine⁹⁵. Two of the 360s controlled missions as a redundant set, with the third used for development work. A special switch connected the 360s to the institutional UNIVAC 1108 mainframe computers so that tracking data could be directly transferred for use in navigational computations⁹⁶. But the gift from Houston was not entirely welcome at JPL, for along with it came the Real-Time Operating System (RTOS) developed by IBM for the Apollo program. As Gianopolis saw it, "what we picked up from Houston was good for Houston, but not necessarily for us"⁹⁷.

Unmanned spacecraft missions needed to create large data bases capable of handling the long series of telemetry signals that might go on for months or years. IBM's RTOS tried to keep all data in core memory, using disk storage as read-only devices. JPL needed to be able to write to the disks. Also, each Apollo computer concentrated on real-time functions and did not do development work. JPL wanted to run FORTRAN jobs on the machine, but RTOS could not handle it⁹⁸. A crisis of sorts arose with the Mariner Mars 1971 orbital mission. During the cruise period to the planet the ground software failed every 5 hours. By the time Mariner reached orbit around Mars, the failure rate fell to once every 20 hours⁹⁹. Still, something had to be done, so JPL contracted for an overhaul of the operating system, culminating in 1972 with the JPL/OS, which incorporated the needed changes.

Since the 360s lacked a small computer for a front end (original thinking being that the machine could handle the load by itself), JPL implemented the idea of using the preflight testing computers in mission support for Mariner Mars 1971¹⁰⁰. Incoming telemetry went to the UNIVAC 1230/1219 set first. Then the 360s did commanding, tracking data evaluation, predictions for the stations, and engineering work. Besides the UNIVAC test set, the UNIVAC 1108s provided navigational data and, by then, the Image Processing Laboratory at JPL had its own 360/44 for processing planetary imaging¹⁰¹.

Viking, a much more complex project than Mariner, and with essentially four spacecraft (two orbiters, two landers) to control, stretched the 360s and their helpmates to the limit. JPL assigned the small UNIVACs to handle the Viking Orbiter data, since the spacecraft were built and tested at JPL and the software was in place. System 360s controlled the landers¹⁰². At peak, 700 controllers

worked the Viking mission, more than on any other space program to date (The count was double that of Skylab, the largest manned support group)¹⁰³. Facing dual Voyager missions and Galileo, with the prospect of continuing Viking far past the original mission life estimates, JPL was again looking for a way to upgrade mission control.

Distributed Computing Becomes the Answer

As JPL discovered planning the computers for the Voyager on-board systems, functionally distributed small computers offered more reliability and cost savings than large single processor systems. The Laboratory implemented a distributed system to fill its Voyager ground control needs as well. Viking was the last mission to be supported from a large mainframe computer. By the time Voyager neared Jupiter, two strings of dedicated minicomputers performed the telemetry, tracking, and engineering monitoring functions. A single minicomputer shared by several projects did the commanding. Why did the change occur? First, the Deep Space Network was unhappy with the level of support it derived from a centralized system. Second, even though centralizing deep space mission control at JPL was a sound idea, putting too many missions on a single computer system was less so. No matter how much JPL tried to standardize things, each mission had its unique characteristics, calling for changes in the support software. With a distributed system, changes could be made without affecting other software. When missions neared critical phases, such as launch or encounter, software had to be frozen until the phase passed. With enough spacecraft aloft, the amount of time available to change software became quite short¹⁰⁴.

NASA provided an additional impetus to switch to a distributed system. Acknowledging the Deep Space Network's concern over using the 360s in the JPL control center and worried that the Network could not monitor its performance when supporting projects originating at other centers (such as Pioneer), the Agency directed the Network to develop monitoring capability in separate computers. Between 1972 and 1974, a set of ModComp 2 minicomputers was connected in a local area network at JPL to implement this directive¹⁰⁵.

In 1976, the control center itself converted from 360s to ModComp 2s and 4s in preparation for Voyager. Later the Laboratory added ModComp Classics and retained some of the UNIVAC 1218s and 1230s (renamed 1530s after upgrades)¹⁰⁶. These computers are arranged in redundant sets. Each project (Voyager, Galileo, etc.) has its own telemetry machine and shares a command machine. A routing computer in the basement of the Space Flight Operations Facility building is the entry point of all data from NASCOM, sending the data to the appropriate computer. The command computer reverses the process for outgoing data.

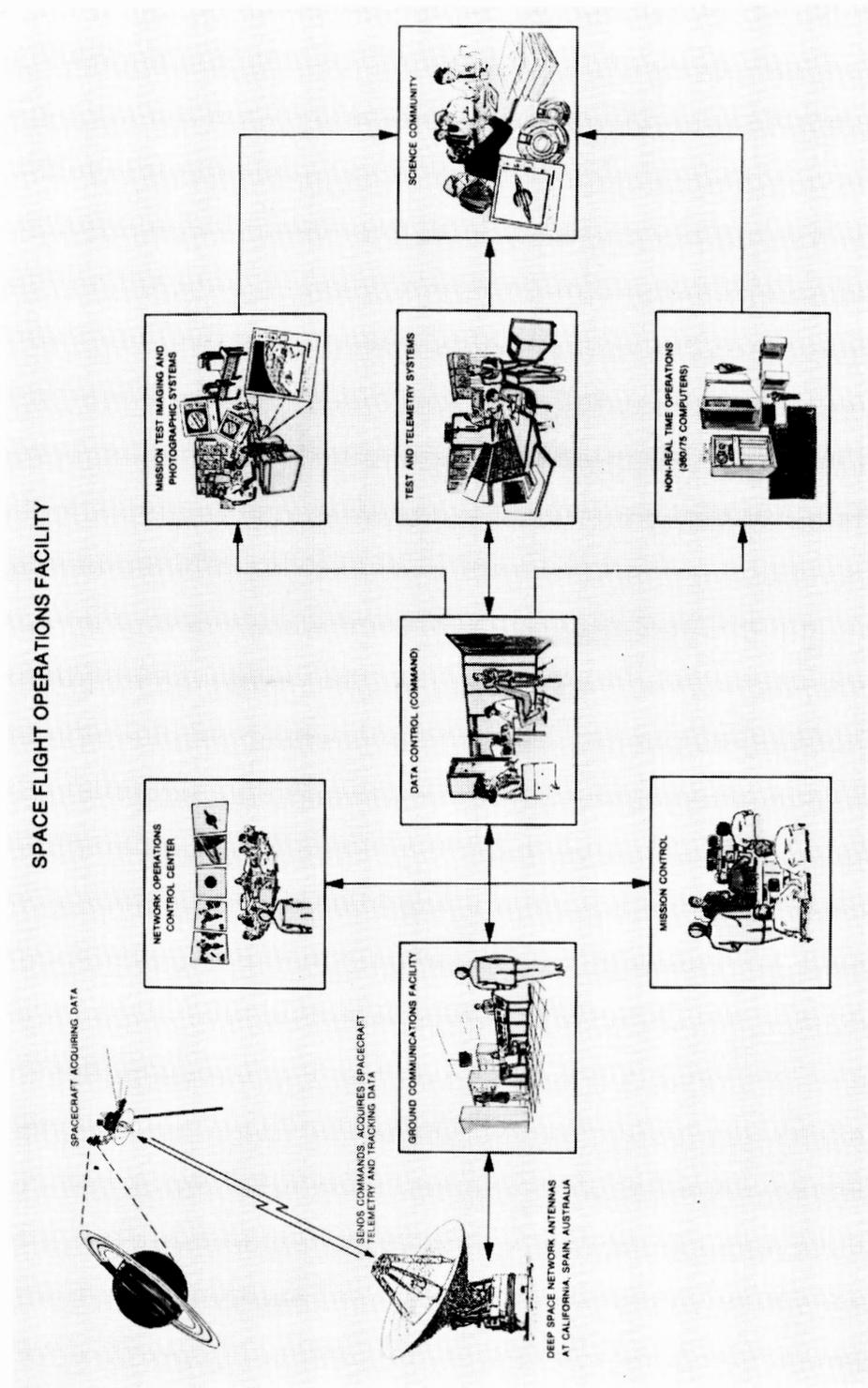


Figure 8-4. A schematic of the components of the Space Flight Operations Facility. (JPL 333-6620)

By the early 1980s, the Deep Space Network was heavily into distributed computing. It converted from the 920s to ModComp 2s at the stations and ordered three Digital Equipment Corporation 11/780 VAX superminicomputers for use at JPL. Nearly 100 minicomputers were connected on an Ethernet. The use of high-level languages became the rule rather than the exception¹⁰⁷. Key to the future success of the Deep Space Network is the inherent flexibility of distributed computing centers. They mirror the use of modules in software: interchangeable parts in a changing field.

Software Development in the Deep Space Network

Software development for the control center and the stations has always been a challenge, as programmers have struggled to use machines built primarily for commercial use in the arena of real-time control. In keeping with the centralization of the computers in 1963, the original software developers worked under Frederick Lesh in a "program analysis group"¹⁰⁸. JPL separated software development for mission operations from that of the network stations just before Mariner Mars 1969¹⁰⁹. Also, at that time emphasis began to be placed on making the software more parameter-based and, thus, more flexible and capable of use on multiple missions¹¹⁰. A new management concept led to the assignment of a program cognizant engineer to each software system engineer. The software engineer would define requirements, prepare test cases, and oversee the program engineer, who would produce the code. This turned out to be quite successful and avoided the difficulties encountered when an engineer thought (wrongly) that he could do both jobs alone¹¹¹. In microcosm, this is the "outside verification" concept used extensively in programming now.

Martin-Marietta Corporation, the Viking Lander contractors, had to do some dangerously unique software development when NASA decided to move control of the Lander from Langley to JPL. Since Orbiter software development and giving support to other missions tied up JPL's computers, Martin took the chance of developing the Lander software in a "minimal higher order language," specifically a hopefully transportable subset of FORTRAN. Martin's solution reflected its recent migration to IBM 370 series and Control Data 6500 series computers at its Denver plant. These were technologically more advanced than the JPL computers and could not be trusted to produce directly transportable software¹¹². The idea worked, but Martin admitted that the requirement for delivering mission support software 10 months before the flight provided strong motivation¹¹³.

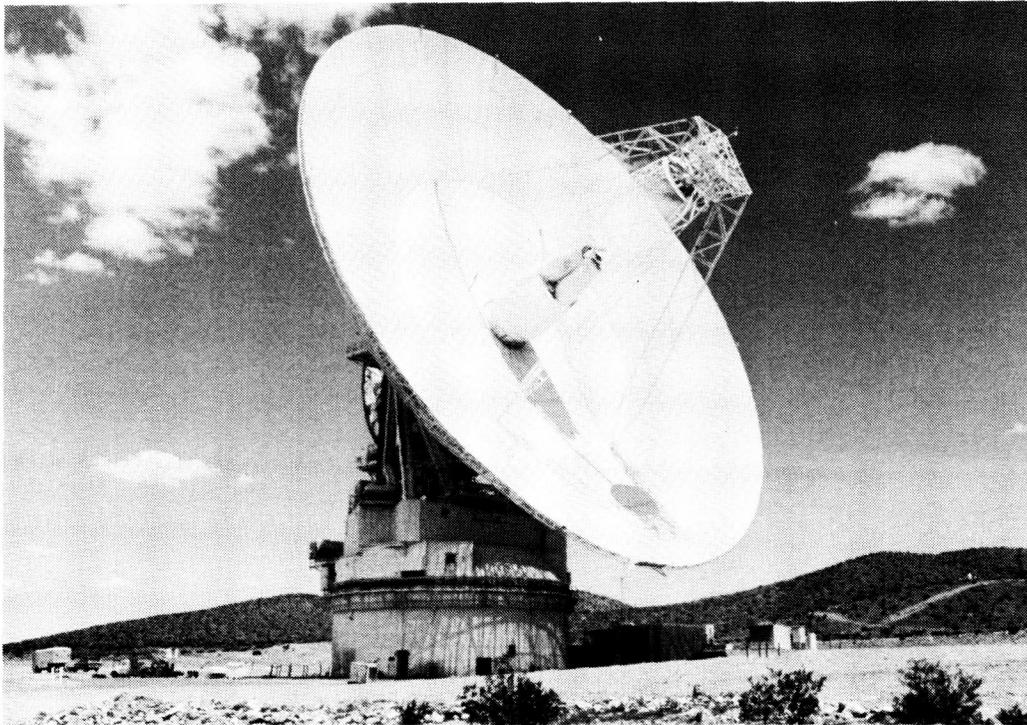


Figure 8-5. The 64-meter antenna of the Deep Space Network at Goldstone, California. (JPL photo 333-5967BC)

As JPL moved to a distributed system, a concerted attempt at establishing software standards has resulted in a state-of-the-art set of documents¹¹⁴. Based on structured programming and software engineering principles, these documents and the decision to use more high-level languages such as HAL, C, and Pascal make the Deep Space Network one of the most sophisticated software organizations within NASA. A further decision to no longer change commercial operating systems (possible now that computers are more general purpose), will help ensure continued cost reduction and consistency¹¹⁵.

Mission control is the most computer-intensive part of spaceflight operations. From the beginning of both the unmanned and manned programs, the computer industry has been constantly forced to stretch the capabilities of both hardware and software in order to meet NASA's needs. In this way, NASA was a driving force in the development of multiprocessing operating systems and large computer complexes.

9

Making New Reality: Computers in Simulations and Image Processing

The computers discussed so far actually flew in space or worked in direct support of launches and missions. Yet NASA found numerous uses for computers in areas somewhat removed from flight operations. Chief among these are simulations and image processing, which made the training of crews, development of launchers and spacecraft, and analysis of image data possible.

Simulations are used in hundreds of ways in the space program. Simulation programs and hardware test the workings of vehicles and spacecraft, determine the accuracy of flight paths, train controllers, check out designs, and actively contribute to the software development process. Simulations help NASA find out whether its programs and projects will work as planned, lessening the risks for crews and equipment. Especially important are simulations used in crew training and simulations used to test hardware. Both provide models by which to judge the extent and efficacy of NASA's dependence on simulations and to demonstrate the dependency of such simulations on computers.

Image processing was developed to make the analysis of digital images transmitted by unmanned deep space craft more consistent and fruitful. At first largely driven by the needs of the Jet Propulsion Laboratory's (JPL) scientific community, imaging spread quickly with applications such as Landsat and the Shuttle's imaging radar. From spectacular images of distant worlds to detailed pictures of the neighbor's farm, imaging technology has contributed to the quality of life on earth. Without the use of high-speed computers, the analysis and use of the billions of bits of imaging data would be impossible.

CREW-TRAINING SIMULATORS

NASA's requirements for flight simulators far exceeded the state of the art when the first astronaut crews reported for duty in 1959. Feeling obligated to prepare the astronauts for every possible contingency, NASA required hundreds of training hours in high fidelity simulators. Each crewman in the Mercury, Gemini, and Apollo programs spent one third or more of his total training time in simulators. Lunar landing crews used simulators more than half the time¹.

Simulators must provide the astronaut trainee with as close an approximation of spaceflight as is possible on earth, without losing sight of the need to extensively practice procedures to respond to failures as well as nominal events. Requirements for realism increase the complexity of the simulation. For example, when an astronaut fires thrusters, the simulator must activate readouts and lights showing the thrusters firing, fuel reducing, velocity changes, and also show movement in the scene outside the cabin window. In a moving base

simulator, such as a simulator in which a spacecraft cabin is suspended on hydraulically moved pylons to enable it to tilt, physical motion must take place. Causing all these things to happen and coordinating them to happen simultaneously is the difficult task of the simulator designer².

A manned spaceflight program always had more than one type of simulator. Usually there was a pair of full-function simulators, one fixed base and one moving base, used for procedures training and extended simulations. Often, part task trainers were needed for more difficult mission phases as well. NASA built a simulator for the last 200 feet of the landing of the lunar module. One part-task trainer exists to train astronauts in using the Shuttle on-board computer system and its software.

Mission simulators today are so dependent on computers that it has become necessary for proper design to think of it as large data processing complex that incidentally is driving displays and performing other functions in a crew trainer³. For the Shuttle Mission Simulator several dozen mainframe, mini, and on-board computers interconnect to create window scenes, change displays, move indicators, and light event lights. Reaching this level of computer involvement resulted from a steady evolution since the beginnings of manned spaceflight.

Project Mercury Mission Simulators

When the time came to design the Mercury flight simulators, experience with aircraft simulators and with those built for the X-15 rocket plane were all that were available. There is one critical difference between training needs for test pilots of aircraft and those of astronauts. Although flying experimental aircraft is always dangerous, they are rarely taken to their projected limits on the first flight⁴. Even the X-15 had a long series of buildup missions, first with a smaller engine, later incrementally increasing speed, then altitude, until a series of full out flights sent the plane to the edge of space. In rocket flight the spacecraft is pushed to the outer limits of stress and endurance from the instant of ignition. Its crews must be fully prepared for all contingencies before the first flight and continue to be prepared for every flight afterwards.

The primary simulator for the first manned spacecraft was the Mercury Procedures Simulator (MPS), of which two existed. One was at Langley Space Flight Center, and the other at the Mission Control Center at Cape Canaveral. Analog computers calculated the equations of motion for these simulators, providing signals for the cockpit displays⁵. In addition to this primary trainer, a centrifuge at the U.S. Naval Air Development Center in Johnsville, Pennsylvania, served as

a moving-base simulator. A Mercury capsule mock-up mounted at the end of the centrifuge arm provided ascent and entry training⁶. Additionally, Langley built a free-attitude trainer that simulated the attitude control capabilities of the spacecraft and two part-task trainers for retrofire and entry practice.

Analog computers commonly supported simulation in the 1950s and early 1960s. Having the advantage of great speed, the electronic analog computer fit well into the then analog world of the aircraft cockpit and its displays. By 1961, though, it became obvious that the simulation of a complete orbital mission would be impossible using only analog techniques⁷. The types and number of inputs and calculations stretched the capabilities of such machines so that when NASA defined requirements for Gemini simulators, digital computers dominated the design.

Computers in the Gemini Mission Simulators

Training crews for the more complicated Gemini spacecraft and its proportionately more complicated missions required the use of digital computers in the simulators. Aside from the tasks done during Mercury, such as ascent, attitude control, and entry, the Gemini project added rendezvous and controlled entries utilizing the spacecraft's greater maneuvering capabilities. At the Manned Spacecraft Center, NASA installed simulators to provide training for these maneuvers, including a moving-base simulator for formation flying and docking and a second moving-base simulator for launch, aborts, and entry. Besides these, two copies of the primary Gemini Mission Simulator, which had the same purpose as the Mercury Procedures Simulator, and the Johnsville centrifuge completed the list of Gemini trainers. One of the Mission Simulators was at Cape Canaveral; the other at Houston.

Gemini Mission Simulators used between 1963 and 1966 operated on a mix of analog and digital data and thus are a transition between the nearly all analog Mercury equipment and the nearly all digital Apollo and later equipment. Three DDP-224 digital computers dominated the data processing tasks in the Mission Simulator. Built by Computer Control Corporation, which was later absorbed by Honeywell Corporation, the three computers provided the simulator with display signals, a functional simulation of the activities of the on-board computer, and signals to control the scene generators⁸.

Functional simulation of various components was made easier by the use of digital computers. In a functional simulation, the actual component is not actually located in the simulator, its activities and outputs being created by software within the computer. Thus, in the Gemini Simulator, the on-board computer was not installed, but the

algorithms used in its programs were resident in the DDP computers, and when executed, activated computer displays such as the incremental velocity indicator just as on the real spacecraft.

Scene depiction in the Gemini era still depended on the use of television cameras and fake "spacescapes," as in aircraft simulators. Models or large photographs of the earth from space provided scenes that were picked up by a television camera on a moving mount. Signals from the computers moved the camera, thus changing the scene visible from the spacecraft "windows," actually CRTs. A planetarium type of projection was also used on one of the moving-base simulators at Johnson Space Center to project stars, horizon, and target vehicles.

Gemini simulations often included the Mission Control Center and worldwide tracking network. No commercially available computer could keep up with the data flowing to and from the network during these integrated simulations, so NASA asked the General Precision group of the Link Division of Singer Corporation to construct a special-purpose computer as an interface⁹. Singer held the contract for the simulators under the direction of prime contractor McDonnell-Douglas, which supplied cabin and instrumentation mock-ups. Fully functional simulators came on line at the Cape and Houston during 1964.

Moving-base simulation came into its own during the Gemini program. The docking simulator was in a large rectangular cube that permitted great freedom of motion in training crews for station keeping and docking. The dynamic crew procedures simulator that replicated launch, abort, rendezvous, tethered (with the Agena upper stage), and entry maneuvers and procedures suggested the feeling of acceleration at lift-off by tilting the spacecraft at a rate equal to the g buildup during launch from about a 45-degree angle to nearly horizontal to the floor. This resulted in a push on the astronaut's back, which increased from 0.707 g to 1 g . Engine cutoff and weightless flight could be suggested by returning the spacecraft to its original position, giving a feeling of maximum comfort to the crew¹⁰. Negative g s could be simulated by tilting the nose down, causing the astronauts to feel their weight on their shoulder harnesses.

Designing and using the Gemini simulators gave NASA a lot of experience in producing high fidelity simulations. Actual flight experiences from Mercury went into improving the Gemini simulators. Gemini rendezvous and maneuver experience helped make the Apollo simulations better. NASA adopted some of the Gemini equipment for Apollo. The use of Honeywell's DDP-224 computers continued, while moving-base simulators were adapted to Apollo use by changing the spacecraft mock-up and modifying existing techniques¹¹. Still, the Apollo program requirements demanded a further increase in the amount of computer power.

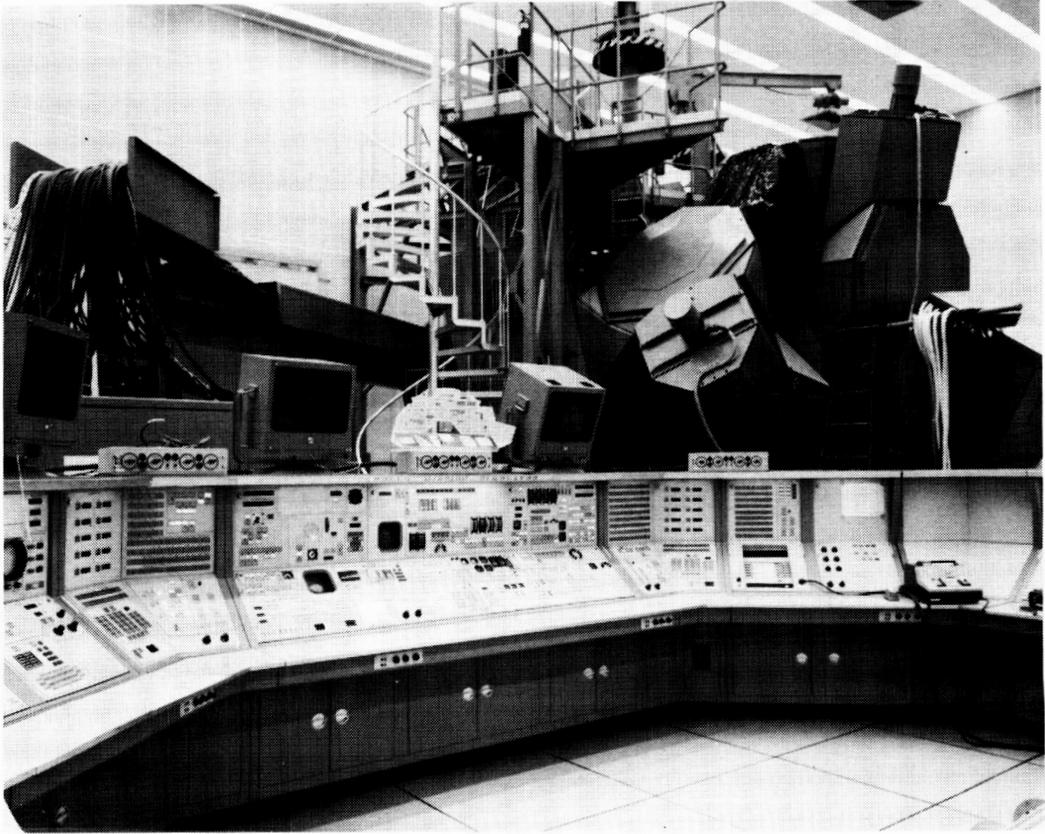


Figure 9-1A. The Apollo Command Module Mission Simulator. (NASA photo 108-KSC-67PC-178)

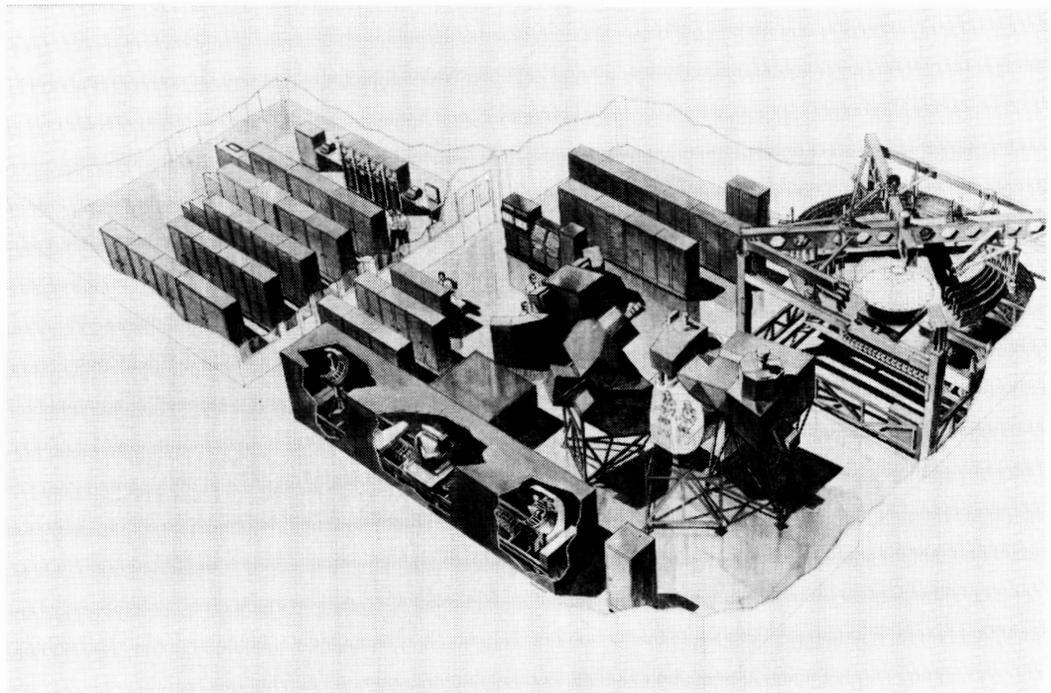


Figure 9-1B. An artist's conception of the Apollo Lunar Mission Simulator.

ORIGINAL PAGE IS
OF POOR QUALITY.

Computers in the Apollo Mission Simulators

No less than 15 simulators trained crews during the Apollo Program. Three were the primary Command Module Simulators, with one at Houston and a pair at the Cape. Two were the primary Lunar Module Simulators, one at each site. At Houston, a Command Module Procedures Simulator trained crews just to rendezvous with the command module, as there was a Lunar Module Procedures Simulator for lunar module rendezvous and landing training. Gemini's Dynamic Crew Procedures Simulator became the same for Apollo. Additional moving-base simulators at the Manned Spacecraft Center were for lunar module formation flying and docking, and a centrifuge (to avoid trips to Johnsonville). Langley Space Flight Center pioneered the research into the final 200 feet of lunar landing by suspending five sixths of a simulator's weight to give astronauts practice in controlling the lander in the gravity of the moon¹². Another lunar landing simulator used a jet engine to support five sixths of its weight and permit free-flight landing training. That simulator required a simulator of its own to keep the crews from crashing it. Finally, a pair of partial-gravity simulators gave the astronauts the chance to walk in space suits while having five-sixths of their weight supported. Later in the program, Marshall Space Flight Center built a simulator for the lunar rover vehicle.

Among the plethora of simulators, use of the Command Module Simulators and Lunar Module Simulators nonetheless occupied 80% of the Apollo training time of 29,967 hours¹³. These simulators and their associated computer systems were crucial to the success of the program. The Apollo 13 emergency in April 1970, when there was an explosion in the service module on the way to the moon, demonstrated the high fidelity and flexibility of the simulators as all lunar module engine burns, separations, and maneuvers could be tested and ad hoc procedures developed as the crippled mission progressed.

In contrast to the procedures simulators, all of which were driven by a single mainframe computer, the Mission Simulators used networks of several computers¹⁴. Honeywell won a \$4.2 million contract on July 21, 1966 to supply DDP-224 computers for the complexes¹⁵. Singer-Link was again the contractor for the simulators. Singer allocated three computers for the Command Module Mission Simulator and two for the Lunar Module Simulator. The sets of computers could communicate among themselves by using 8K words of common memory, where information needed throughout the simulation could be stored¹⁶. Later, a third and fourth computer were added, respectively, to the Lunar and Command Module Simulators. These com-

puters simulated the on-board computers. By the Apollo 10 flight a fifth computer, simulating the launch vehicle, completed the Command Module Simulator computer complex¹⁷. The two types of simulators and the Mission Control Center could do integrated simulations, thus requiring up to 10 digital computers to be working on one large problem simultaneously¹⁸.

Software became as important to the simulated world of Apollo as it was in the real world. Software development for the Apollo Mission Simulators required the efforts of 175 programmers at the peak, compared to 200 hardware persons¹⁹. Over 350,000 words of programs and data eventually ran in the two simulators. Using digital computers, trainers could return the crews to a certain point in a simulation and try again by simply recording the status of the computers and data on magnetic tape and reloading memory to match the state of the software at the time desired. This sort of flexibility made the training task much easier.

Early in the development of the Apollo simulators, a problem arose that would have had critical consequences if not solved. The importance of the on-board computer to the guidance and navigation of a moon-bound spacecraft was obvious. Crews interacted with the computer thousands of times in a typical mission; its keyboards contained the most used switches in the spacecraft. Initially, the Apollo Guidance Computer (AGC) for both the command module and the lunar module were simulated functionally, just like the rest of the spacecraft hardware²⁰. This meant that the major components of the Apollo modules existed as software in a DDP-224 rather than in their physical form in the simulator.

Even so, functionally simulating the on-board computer soon proved to be nearly impossible. Mathematical models and algorithms for specific Apollo missions had to be sent to the simulator programmers from the Instrumentation Laboratory at MIT. Although Singer contracted for over 20 experienced IBM programmers, the development of functional simulations lagged²¹. The programmers had to take logic and create software for the DDP-224s that executed just like the software on the AGC. Essentially they coded programs already being coded for the real computer but in a different machine language. Warren J. North of the Computational Analysis Division at the Manned Spacecraft Center studied the process of creating the new software and found it took about 4 months to write the functional simulation. Since crews needed the software for training at least 6 months before the mission, and some buffer had to be allowed for last-minute glitches and their solutions, software designs for the AGC, developed at MIT, had to be available a full year before a flight, a very difficult schedule to meet at the time²². As a result of this study and the continued concern of the Apollo Spacecraft Project Office, W. B. Goeckler of the Systems Engineering Division of the program

asked James L. Raney of Computational Analysis to do a feasibility study of using a DDP-224 to simulate the AGC²³. Goeckler thought it might be possible to make the Honeywell computer think it was the MIT computer and execute the MIT code, thus eliminating the need for rewriting the programs and solving the time problem.

When Raney joined Apollo in February of 1966, he faced a rather interesting question: Could a floating-point arithmetic, two's complement representation, 24-bit computer with accumulators and index registers run programs written for a fixed-point, one's complement representation, 16-bit machine that buried its registers in memory? Hardly likely, just about everybody thought—except Raney.

Instead of a functional simulation, a computer running another computer's code uses interpretive simulation techniques. It takes a single instruction from the other's program, executes it using as many instructions as necessary from its own repertoire, and then goes on to the next. Since the AGC had a unique interrupt structure and limited arithmetic capabilities (limited compared with the DDP-224), many Apollo instructions took multiple Honeywell instructions to get around the differences.

Raney suggested both hardware and software modifications to the DDP-224. He specified a switch to disable the machine's floating-point capability. Instructions were added to enable more efficient table searching and other operations that the AGC did well. To handle the different word sizes, Raney let the right-most 14 bits of the DDP word be the value of a corresponding AGC word. The left-most bit was always set to zero to indicate that it was an Apollo word, and the intervening bits matched the sign bit of the original word. Words that could not be translated (i.e., executed one for one), had to be executed by interpretive subroutines written for the purpose and stored in the lower part of the Honeywell memory. Raney figured that since the DDP had a 10-to-1 advantage in execution speed over the AGC, several instructions could be used to do one Apollo instruction without slowing down the program. He used the index registers in the Honeywell DDP to act as the Fixed Bank Register, which kept track of which core rope memory module the AGC was currently using, as well as the address of the next instruction. Finally, to store the AGC code, the flight program was put in the upper half of the 64K words of core, with the interpreter used in the AGC to execute its own instructions in an area in lower core. The contents of the AGC's 2K erasable memory and the 8K of common core addressable by all the simulator computers also was in lower core, along with Raney's interpretive subroutines²⁴.

Despite Raney's careful evaluation of the situation and proposed solution, many Apollo project personnel opposed it, simply feeling it was unworkable²⁵. In desperation, NASA approved the attempt at an interpretive simulator and bought the modified computers. In the end,

the simulation within a simulation was spectacularly successful. Even though Raney and his team took care to time the subroutines so that they matched execution of the actual Apollo code, the simulated computer was faster than the real article. Following the Apollo 9 earth-orbiting mission that tested the command module and lunar module rendezvous techniques, pilot Dave Scott complained that he had up to 12 seconds less time to react when the computer signaled for a maneuver to begin. This was adjusted for later flights.

Developing the interpretively simulated AGC had several impacts on the program. MIT could use the simulator as a field test of its code before flight. Since MIT used tape rather than core rope to send the programs to Houston and the Cape, errors discovered could be corrected and then the corrections tested in a "real" situation. Crews could react to the way the software worked with them. Also, the simulator cost just \$4.6 million, compared to an estimated \$18 million for functionally simulating the programs.

Actually, the Apollo Mission Simulators were the last of their type in that the analog environment of the spacecraft that dictated hybrid and functional simulations changed to a digital environment that lent itself to full digital simulations for the Shuttle program. Evolution to full digital simulation, including digital imaging of window scenes, meant even more dependence on digital computers. Making the Shuttle a more autonomous and thus more complex spacecraft contributed to a massive increase in the size of the computer systems needed to support simulations.

Full Digital Reality: Computers in the Shuttle Mission Simulators

The difficulty of producing a fully digital simulation of the Shuttle may be appreciated by considering the fact that when NASA issued the first request for proposals for the Mission Simulators, there was no response²⁶. Singer, which by then had converted Precision Link to the Simulator Products Division, eventually responded with a plan for a detailed analysis of the simulation problems of the Shuttle. NASA had already decided that the extreme cost of developing Shuttle simulators would be moderated by acquiring fewer of them²⁷. Shuttle program director Robert F. Thompson formed a committee in 1970 to monitor development of the simulators and involve the projected users, the Flight Crew Operations Division, and the Flight Operations Division, in its design²⁸. Singer considered the requirements and suggested a large complex of mainframe computers functioning through limited task minicomputers to drive the simulator.

All Shuttle simulators are located at the Johnson Space Center. The fixed-base simulator replicates the four crew stations on the flight

ORIGINAL PAGE IS
OF POOR QUALITY.

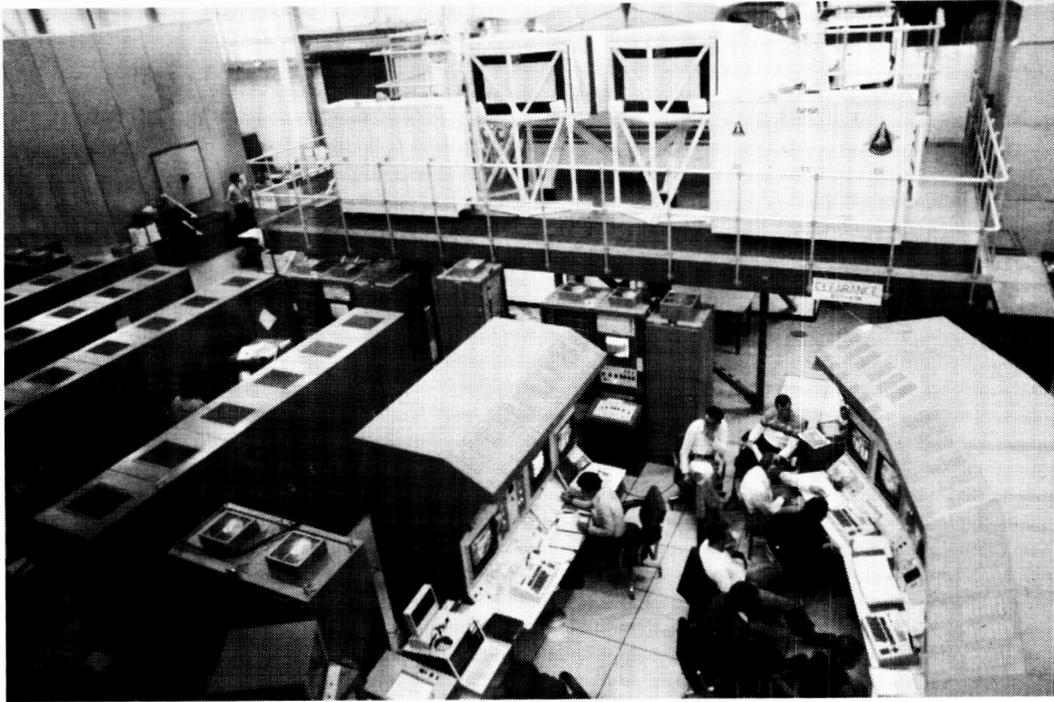


Figure 9-2. The fixed base Shuttle Mission Simulator (upper center) with some of its electronics. (NASA photo S-81-27526)

deck of the orbiter. It has window views through both aft windows and the overhead windows. Hosted by four Sperry Corporation UNIVAC 1100/40 mainframe computers, 15 Perkin-Elmer minicomputers (mostly 8/32s) provide digital images for the windows, interface with the on-board computers, and perform other functions, acting as fancy channel directors for the mainframes²⁹. A motion-base simulator recreates the two forward crew stations, all forward window views, and the heads-up display used in landing. Also hosted by four 1100s, it has 11 minicomputers due to the lesser digital image requirements. The fixed-base simulator not only has to display proper images of the earth and the cargo bay but it also must image the remote manipulator arm and any payloads, thus requiring the power of five of the 8/32s. Supplementing the two primary Mission Simulators is the Shuttle Procedures Simulator. Also called the "Spare Parts Simulator," it was often cannibalized to keep the more critical Mission Simulators running³⁰. In the early 1980s it was scrapped, and a Guidance and Navigation Simulator was built out of its remaining parts. It is used for some part-task training.

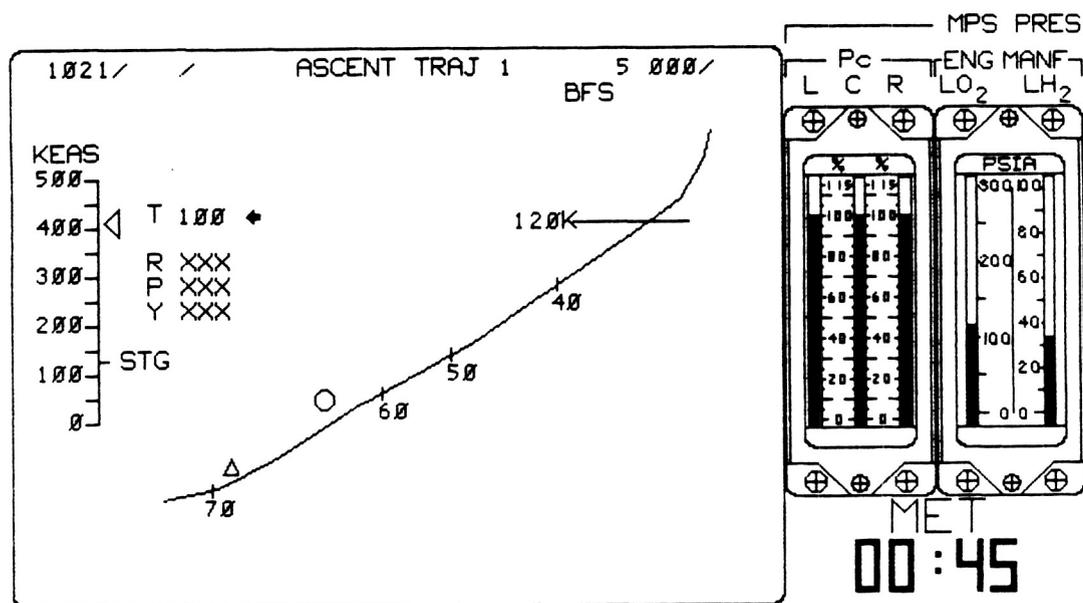
Singer quickly decided that the Shuttle's on-board computers could not be interpretively simulated, as the AGC was³¹. IBM's AP-101 machines used on the spacecraft were roughly as fast as the

UNIVAC computers, eliminating the time advantage the DDP machines had over the AGC³². Functional simulation of five computers working in concert was also out of the question. Therefore, each simulator had five computers, just as in the real spacecraft, and NASA bought two more as spares. During the course of the program, however, the computers began failing. With training schedules calling for simulation runs of 16 hours a day plus maintenance and reloading, several computers reached 30,000 hours of operation, far greater than the operational life of the flight version. Roughly 12 or 13 are actually available at any one time, with the two primary simulators always kept at a full complement of five, and the Spare Parts Simulator using the rest³³. The mass memory unit (MMU) of the on-board computer system, the magnetic tape drive that stores the software, is functionally simulated. It proved impossible to keep the actual mass memories running long enough to be cost effective. Designed for only a few minutes of use in each flight, they fell apart under the demands of the simulators. A disk drive controlled by an IBM Series/1 processor replaced the MMU, with delays built in to make it load as slowly as a tape would.

Software for the Shuttle Mission Simulators is based on a 20-millisecond cycle controlled by a special real-time clock that sends a signal to all participating computer systems³⁴. This is about the only way the large number of computers can be kept in step. The operating system for the UNIVAC machines is a commercial version that is no longer supported by Sperry, so NASA has had to specifically contract for maintenance on the system to avoid having to change the rest of the software to match a new one³⁵. Singer wrote the real-time operating system used on the Perkin-Elmer machines³⁶. Despite the large number of programmers on Singer's Shuttle Simulator payroll (200+ of 611 people), it subcontracts with Perkin-Elmer for some software, creating a situation where the developers are removed from NASA managers by another layer of management, which has resulted in unsatisfactory products³⁷. In 1980, NASA's Robert Ernull, with years of experience in the on-board software division, was named head of the simulator division to help clear up problems with the complex simulators. He tried to reduce the throughput required of the computers to 70% of the total capability to allow for changes. This did not help what he thought was a second major problem—lack of memory. Memories were so full any modifications caused a crisis³⁸.

Aside from the more traditional Mission Simulators, NASA is beginning to use microcomputers to replace the expensive part-task trainers of the past. A system called Regency provides a programmable 64 by 64 spot touch screen. Detailed graphics of switches and indicators can be displayed, and also component schematics, so that trainees can communicate with the teaching software by touching the screen in the appropriate place. The teaching software is based on

techniques developed for the PLATO system at the University of Illinois. Increased use of microcomputers and other small computers for more generalized training will come as the space program enters the Space Station era. Simulating large spacecraft will be financially impossible, but simulation of critical crew stations using software and graphics for flexibility will be possible. Given the present direction, it appears that some sort of generic trainer with its characteristics controlled by software will be the mainstay of the training program, replacing the large computer complexes of the past and present.



THRUST BUCKET

Thrust bucket is a term used to describe a temporary reduction in thrust for aerodynamic loading. The thrust level changes are controlled by engine valves responding to GPC guidance programs. Orbiter flight requirements will determine the time, duration, and thrust levels.

Percent of thrust (T) is displayed on the BFS TRAJ1 and on the 3 P_c (chamber pressure of each engine) meters on panel F7.

TUOCH SCREEN FOR ANIMATION AND CHECK THRUST LEVELS

Figure 9-3. One of the instructional screens of the Regency system used in training.

ORIGINAL PAGE IS
OF POOR QUALITY

ENGINEERING SIMULATORS

While flight simulators are the glory of the simulation business, engineering simulations help make spaceflight possible. Many times highly innovative systems proved themselves in extremely accurate simulations. One example is the control moment gyro system used in attitude control of Skylab. A large simulator constructed at the Marshall Space Flight Center gave engineers valuable data about the behavior and feasibility of the system, which was understood by few aside from its inventor. Also at Marshall was a simulation of the Shuttle's main engines. These first computer-controlled rocket motors run much hotter and closer to destruction than any predecessors. Software for the engine controllers can be tested and certified in the simulator. At Johnson Space Center and the Rockwell plant in Downey, California are full-scale engineering simulators of the entire Shuttle orbiter. Early in the program, engineers led by Kenneth Mansfield at Johnson used these simulators to work out preliminary concepts, flight techniques, and procedures development using functional simulations (no flight hardware). After the installation of the actual hardware, changes to the individual hardware and software components could be checked for integration with the remainder of the spacecraft in those simulators. Thus, engineering simulators provide engineers with help in requirements analysis, prototyping, verification of concepts, and integration testing.

Simulation of components involved in rocket flight began in the late 1930s with the German development group at Peenemunde, where attitude control systems were simulated. In 1939, a one-axis mechanical simulator of the A-4 rocket's motion about its center of gravity provided valuable control data without the expenditure of test vehicles³⁹. That device led conceptually to a more robust electronic analog simulation of the control system designed by Helmut Hoelzer and built under his direction by Otto Hirschler. Included in that simulator was an analog device to correct for the vehicle's lateral drift while in flight. Completed in 1941, the simulator was the most advanced analog computer built to that time⁴⁰.

Following World War II, the Peenemunde group brought the concepts of simulation to the United States. Hoelzer became head of the Computation Laboratory at the Army Ballistic Missile Agency research site in Huntsville, Alabama. When NASA absorbed the Agency's Huntsville facilities in 1959, Hoelzer continued his work and gathered a powerful set of digital and analog computation devices at the Marshall Space Flight Center. So much simulation work needed to be done that Hoelzer developed a simulation system consisting of a set of general-purpose digital, analog, and hybrid computers that several projects could use. Usually consisting of a large analog device and supporting digital minicomputers, the hybrids modeled booster

flight characteristics and tasks such as payload loading, space telescope pointing, attitude control problems, circuit design, and mission support⁴¹. One system, the SMK-23, modeled moving vehicles such as the lunar rover, providing television window views inside a closed control cockpit⁴². Besides this central facility, Marshall Space Flight Center also developed two large stand-alone simulators for special complex problems: the Skylab Attitude and Pointing Control Simulator and the Hardware Simulation Laboratory used to model the space Shuttle main engines.

Skylab Simulators

Prior to Skylab, the primary method of attitude control in a spacecraft was the use of reaction control system jets that burned liquid fuels. With a mission profile of up to a year's worth of occupancy and experimentation, Skylab could hardly carry enough fuel to maneuver its bulk for that length of time. An alternative solution was a control moment gyro (CMG) system that was very innovative and complex. A redundant digital computer system provided commands to the system in orbit. To study the operation of the complete system, including the computer and its attendant software, required the construction of a complete laboratory dedicated to the task.

Three rooms of the Astrionics Laboratory at Marshall were set aside for the simulator. In the Black Room sat the hardware that simulated the motion of the space station. The Green Room held the control devices and some of the computing equipment, with the remainder in the adjoining computer room. Primary computer for the simulator was a hybrid consisting of an XDS Sigma V digital computer and Comcor Ci5000 and Ci550 analog computers. These drove the simulation of the orbital workshop and interfaced with the ATMDC which flew on the actual spacecraft. A SEL 840 digital computer sent digital commands to the on-board computer⁴³.

Originally, the use of the simulator concentrated on mission planning and hardware and software verification tasks. Engineers expected to operate it less than half the working hours of a normal week. However, due to the severe hardware failures on the actual mission, the simulator reverted to 24 hours a day, 7 days a week operation. First the micrometeoroid shield and solar panels were damaged during ascent. This meant that the workshop had to be oriented in ways not set out in the requirements. For nearly 2 weeks, while Marshall prepared tools and techniques to effect repairs with the first crew aloft, the simulator tested attitude control maneuvers that would keep the workshop from excessive internal heating. Later failures, especially the loss of a CMG, were successfully modeled and solutions devised. As in the Apollo 13 flight, ground simulation of actual flight damage led to safe alternatives.

Space Shuttle Main Engine Simulator

The main engine of the space Shuttle is another complicated device that needs its own simulator. The Hardware Simulation Laboratory is the primary site for verifying the design of the main engines, testing the engine controller software, preparing for hardware changes such as new controllers, and modeling failures such as faulty valves and sensors that caused engine shutdowns on the pad and in flight during the Shuttle program. Begun in the early 1970s, by 1975 the engine simulator became operational. At the heart of the first version of the Laboratory were two Ci5000 analog computers and a SEL 840 MP digital computer. The engine, actuators, and sensors are simulated with the hybrid system. Actual engine control computers are mounted in the simulator⁴⁴.



Figure 9-4. A collage depicting the Hardware Simulation Laboratory at the Marshall Space Center used for testing the Shuttle Main Engine Controllers. (NASA photo 331594)

Since Marshall was responsible for all the booster components of the Shuttle, it developed other devices that modeled those components in the largest of the active engineering simulators, the Shuttle Avionics Integration Laboratory.

SAIL: Fully Operational Shuttle Skeleton

The Shuttle Avionics Integration Laboratory, or SAIL, one of the largest engineering simulators ever built, sits in a big bay at the Johnson Space Center. A fully functioning skeleton of the Shuttle orbiter, it contains all avionics components used on the real orbiter, totaling nearly 1,750 black boxes weighing 6,000 pounds⁴⁵. In fact, they are placed in exactly the same positions as in the actual spacecraft so that components can be certified and any changes made to the avionics can be tested. Also, software for a particular flight can be run to check for errors. Through the first six flights of the Shuttle program, the SAIL accounted for 241 errors found in the primary software and 196 errors in the backup software. For the first flight, SAIL operated for 644 shifts and since then has averaged 80 shifts in support of a mission. Just short of 350 contractors and NASA personnel manned the Lab in its operational phase.

Planning and construction of the SAIL began in 1968, when the Shuttle Engineering Simulator first began operations. This simulator, still functioning after many modifications 15 years later, replicates a cockpit. Scene generators for one forward window and both rear and overhead windows, as well as four SEL minicomputers and a Control Data Corporation Cyber 74 mainframe, drive the simulator. Preliminary work on this simulator gave experience that contributed to the SAIL, which started in 1972⁴⁶.

Until January of 1983, the SAIL itself consisted of a guidance and navigation test station; the Shuttle Test Station, which is the skeletal orbiter; the Marshall Mated Element System, which simulates the propulsion system; a ground standard interface unit, which sends commands and acquires data from the SAIL for display; and a subset of the Launch Processing System. Since the avionics system is the only real hardware in the orbiter mock-up, the orbital maneuvering engines, reaction control system, main propulsion, and other non-avionics boxes must be simulated by computer software or analog devices. To preserve the exact signal timing, these simulators must, in some cases, be located farther from the spacecraft skeleton than the real equipment. The forward reaction control jets simulation boxes, for example, are over 10 meters from the spacecraft nose. Since Marshall contributed 55 racks of electronics, and Kennedy Space Center sent the Launch Processing System subset, each center can use the SAIL to verify software written for equipment under their develop-



Figure 9-5. Astronaut John O. Creighton in the cockpit of the Shuttle Avionics Integration Laboratory. (NASA photo S-79-39162)

ment control, such as the engine controllers from Marshall and the interfaces and selected test software from Kennedy⁴⁷.

SAIL operators can monitor tests from display control modules connected to the interface unit. The consoles have color monitors and individual processors used for fault detection. Aside from validating engineering changes and software, the SAIL is used for validating tests to be carried out later on the spacecraft while it is being prepared for flight⁴⁸.

An extensive hybrid computing center drove the SAIL and its attendant simulators during its first decade. A pair of EAI 8800 analog computers simulated the landing gear, runway, and braking. A pair of 7800s represented the aerosurfaces and rate gyros. These analog computers were replaced with a pair of Gould SEL 32/8780 digital minicomputers in 1983. Other SELs provide a digital autopilot simulation, equations of motion, radar altimeter, and other nonavionics functions⁴⁹. Separate computers generate the window scenes. These are so much better than those done in the Shuttle Mission Simulator, especially in regard to the Remote Manipulator System, that crews prefer to use the Shuttle Engineering simulator in the SAIL for training when a mission requiring use of the arm is coming up⁵⁰.

Since the Shuttle was the first manned spacecraft to fly without unmanned development flights, the SAIL's importance cannot be minimized. By essentially replicating the entire spacecraft and its operations exactly as the spacecraft currently exists, the SAIL provides NASA and the astronaut crews confidence in the hardware and software for each mission. In its role, SAIL is the ultimate engineering simulator.

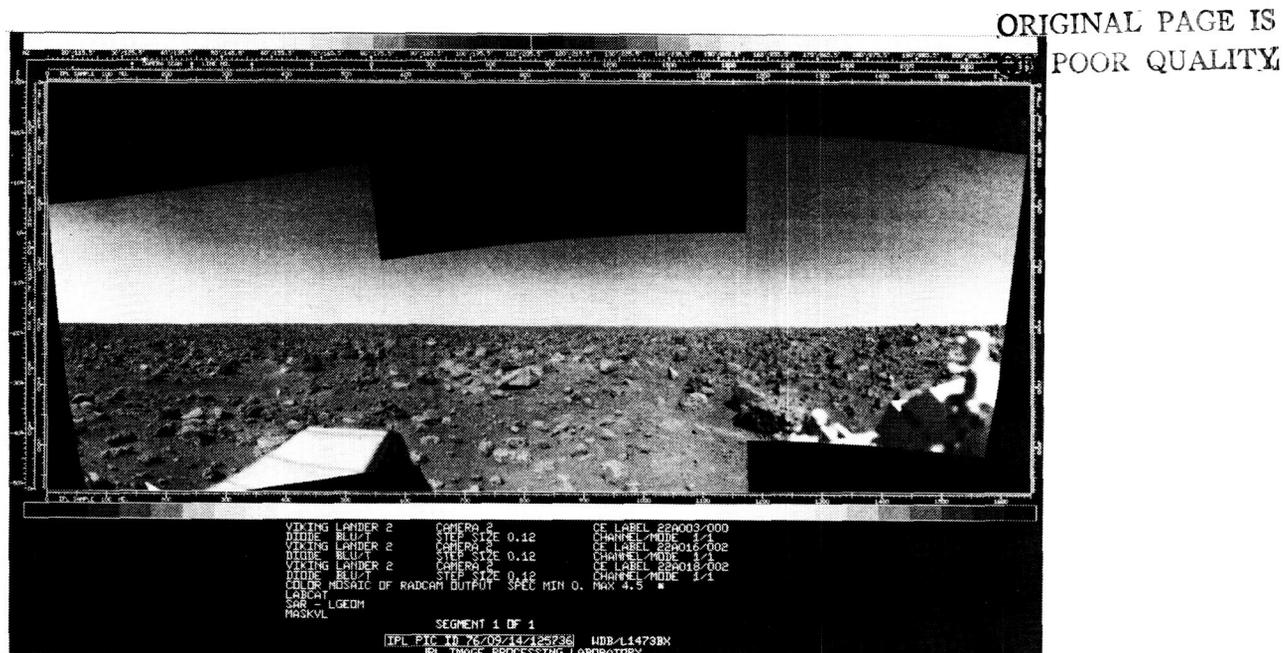


Figure 9-6. Image processing makes possible scenes from alien worlds such as this panorama of the Martian surface. (JPL P-17982)

IMAGE PROCESSING

Image processing is one area in which NASA, primarily through work done at JPL, clearly leads the field. Ironically, even though the production of high-quality images from space probes and Landsat earth orbiters has great scientific and public relations value, the concept of digital image processing was not incorporated in the original planning of a number of early missions. Instead, it had to gain acceptance as a "tack-on" to the Ranger and Surveyor programs⁵¹. Robert Nathan led the development of digital image processing in its early stages, and with the technical help of other JPL scientists, won for it a featured place on the planetary missions of the late 1960s and beyond. Of the early resistance, he later said that he "had to prove to [project management] each time what they needed" to get the most out of the first American pictures coming from space.

Nathan came to the California Institute of Technology as a graduate student in 1952. He earned a Ph.D. in crystallography in 1955 and soon found himself running CalTech's fledgling computer center, where he received a good grounding in the potential of digital computers. In 1959, he went to JPL to help develop imaging equipment to map the moon. When he saw the Russian pictures of the far side of the moon, he thought he could do better and began developing digital techniques for image enhancement using a small NCR 102D computer. Nathan reasoned that analog equipment, such as television cameras, could only be controlled by hardware changes, just like an analog computer can only have its internal program changed by rewiring or switching components. However, digital processing allows changes to be made with software, allowing a wider variety of enhancements⁵².

Before an image can be processed, it must be put into digital form. Frederick Billingsley and Roger Brandt of JPL devised a Video Film Converter (VFC) that could transform analog video signals, such as those sent back by Ranger spacecraft, into digital data. While they supervised the construction of the device, John Morecroft of JPL used the NCR computer to begin programming processing algorithms. These events took place in 1963, and by the next year Howard Frieden had programmed the Laboratory's institutional IBM 7094 computer to process Ranger data. Success with Ranger images led the Surveyor project to use Nathan's techniques, as well as Mariner Mars 1964. By the Mariner Mars 1969 missions, the concept of digital image processing was fully accepted.

Why is image processing needed? Due to the resolution and design of the video cameras used to make the images, they must be processed in order to return the most information possible. The surface of Mars is such a low-contrast object that without enhancements, features would be lost in the wash of monochrome⁵³. Also, because the human eye cannot adjust to differences in illumination across a field of view, illumination must be normalized⁵⁴. The cameras operate by taking an instantaneous view of the scene; the values of the light impressed on the vidicon tube are then made into digital data. Since images are taken one after the other, very close together in time, residual images from prior "snapshots" affect the current view⁵⁵. These residual images must be removed, a technique that took several missions to perfect. Finally, noise from transmitting a signal over planetary distances must be accounted for.

To see how such processing is done, the real-time display system used for the Mariner Mars 1971 orbital mapping mission provides a useful example. A UNIVAC MTC 1230 computer extracted 9-bit pixel data from the telemetry stream. A pixel is a single picture element, or dot. The spacecraft had a camera capable of recording frames of 700 lines by 832 pixels, or 580,000 individual dots. Such large

numbers of pixels were only practical as interplanetary communication advanced. Mariner Mars 1964's 200 by 200 pixel imaging equipment transmitted at the rate of $8 \frac{1}{3}$ bits per second. Thus, it took nearly one entire shift at a Deep Space Network station to record a single frame. At that data rate it would take over 1 week for a Mariner Mars 1971 frame! But by 1971, the data rate increased to 16,200 bits per second, giving a complete picture in 5 minutes and 40 seconds. Even these rates increased by over seven times in the next few years.

ORIGINAL PAGE IS
OF POOR QUALITY

ORIGINAL PAGE IS
OF POOR QUALITY

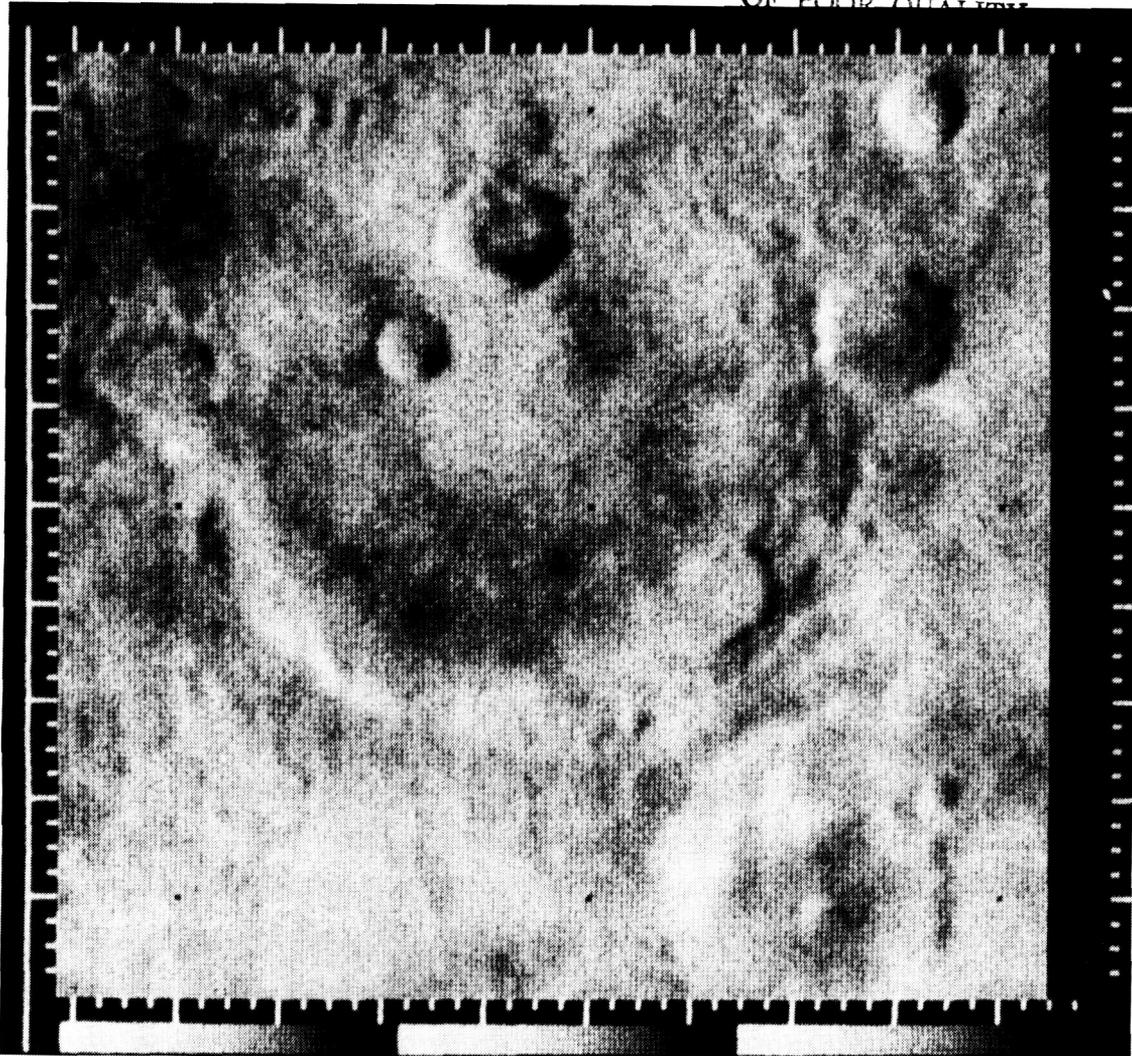


Figure 9-7A. Image processing's decade of progress: Mariner Mars 1964 returns the first closeups of Mars....

C-4

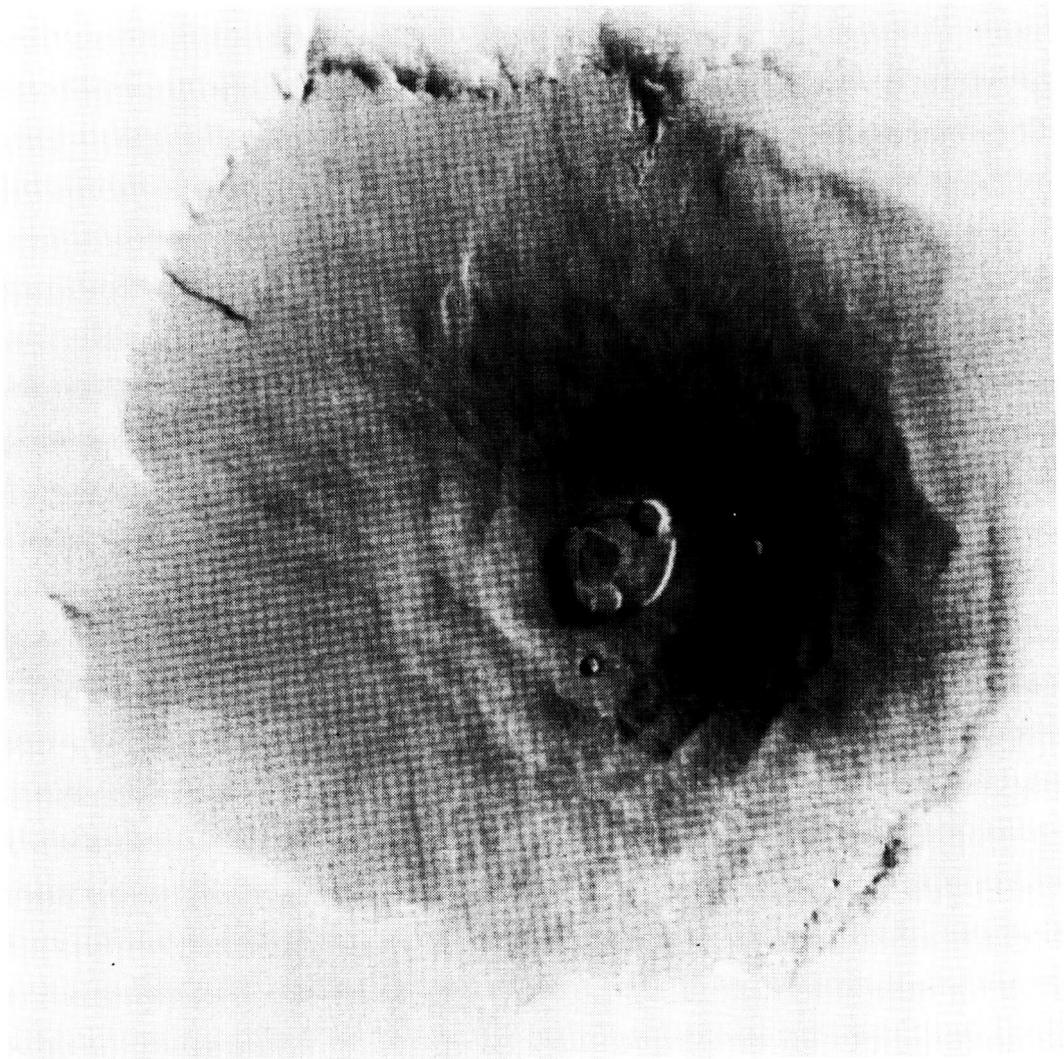


Figure 9-7B. ...As the planetwide dust storm clears, Mariner Mars 1971 scans Nix Olympica in January, 1972....

ORIGINAL PAGE IS
OF POOR QUALITY

ORIGINAL PAGE IS
OF POOR QUALITY

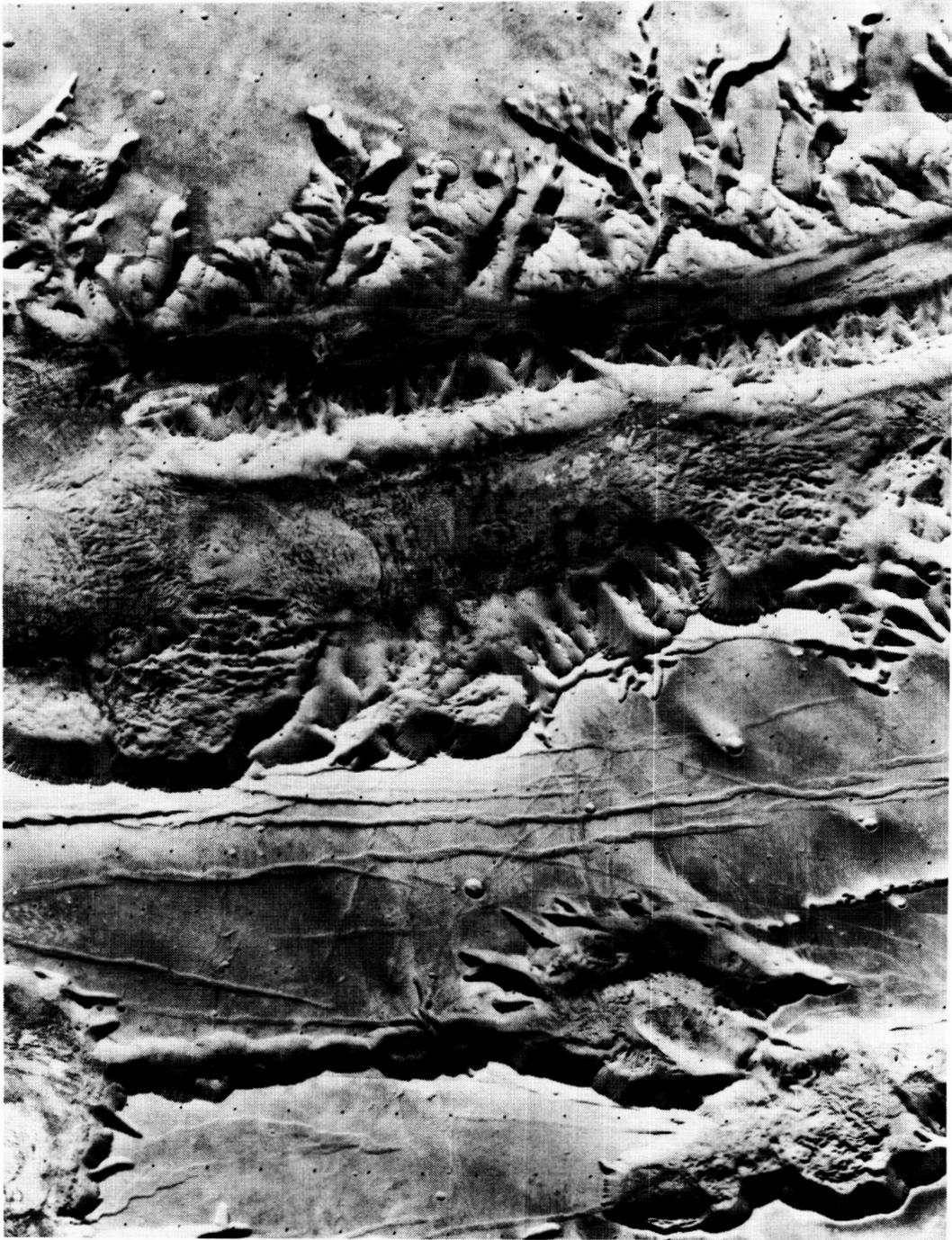
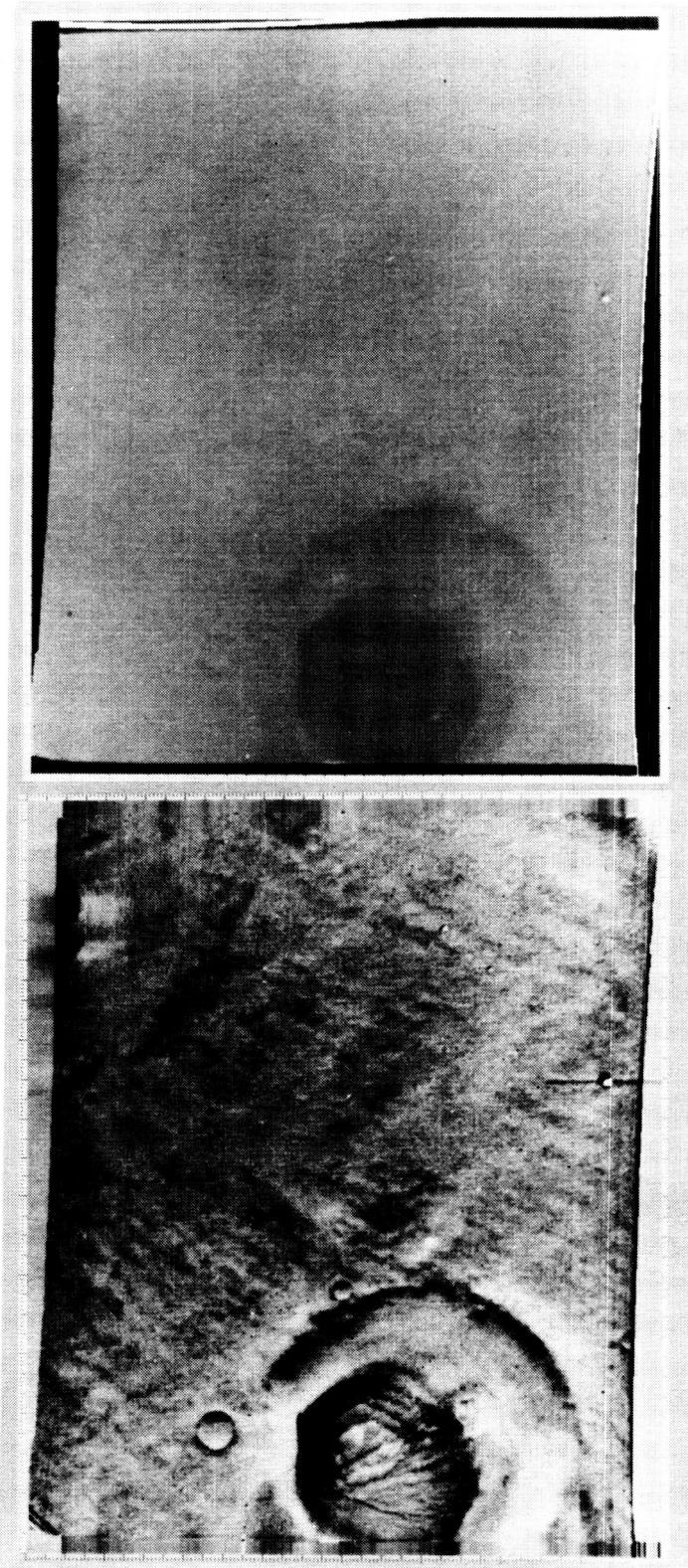


Figure 9-7C. ...Details from the Valles Marineris canyon taken by the Viking Orbiter in 1976. (JPL photos P-7875A; P-13074; P-17872)

Several techniques could be applied to the data by the computer. Contrast stretching helped increase the contrast of the single color Martian surface. Original values of the pixels ranged from 0 (black) to 255 (white). The computer truncated these to 6 bits, which yielded 64 levels. Since humans can only discern about 25 levels of grayness, this was more than enough. By increasing the brighter grays toward the white end of the scale and decreasing the darker grays toward the black end, the contrast was increased⁵⁶. Illumination could also be normalized using the computers. A "high pass filter" corrected the value of the pixels by averaging the immediately surrounding 125 pixels and then subtracting the running average from the value of the pixel⁵⁷. Another process compensated for geometric distortion. Simply because of the way the cameras were made, there was distortion in the image frames. Reference points marked on the image served to help distortion elimination algorithms properly square off the image. These techniques were also applicable to developing mosaic maps by taking images shot at oblique angles and flattening them out in any one of several projections⁵⁸. Noise elimination could be done by assuming that any pixel exceeding a difference of 32 levels of brightness from its neighbors was a spike and then changing the value of the spike to the average of its two immediate neighbors. From 20 to 10,000 spikes could be found on a single raw image, so without removal the image would be noticeably damaged⁵⁹.

Aside from the near-real-time imaging provided by the UNIVAC and other computers on later missions, long-term processing with a number of techniques is done in the Image Processing Laboratory at JPL. First established in 1965 with a new IBM 360/44 computer that lasted 10 years, the Processing Lab pioneered new imaging techniques and developed support software to implement them. Central to the success of image processing was the Video Information Communication and Retrieval language, or VICAR. Written in 1966 after a design by Stan Bressler and Howard Frieden, VICAR enabled users to define a pipeline of processes without having to use cumbersome job control language. For instance, VICAR could define an image file to be processed and then specify the type of processing to be performed on it in a sequential manner. Output from the stretching program could thus be directed to the input to the geometric transformation program. The existence of this language significantly increased the value of the imaging⁶⁰.

By 1975, when a 360/65 replaced the older computer, the Image Lab did roughly half of its work on planetary imaging and half on earth resources work using Landsat images⁶¹. Also, by that time numerous spinoffs from the program began to turn up in other fields, chief among them astronomy and medicine. Astronomers now use digital techniques to enhance their photographs of celestial objects in the same way spacecraft images are processed. Nathan left the



ORIGINAL PAGE IS
OF POOR QUALITY

Figure 9-8. Increasing contrast enhances a Mars image. (JPL 511-4353)

planetary imaging to his colleagues in 1968, when he turned his attention to a series of grants from the National Institutes of Health to study applications of digital image processing to microscopy and medical diagnosis. Robert Selzer of JPL had applied the techniques to x-ray enhancement. For Nathan, with a background in x-ray crystallography, this was a natural step. Unfortunately, by 1973 the government canceled all fundamental research grants in the field and Nathan found himself without support and nearly without a JPL position⁶².

Nathan managed to hold on for a few more years at JPL on other projects until, in the late 1970s, he thought of a way to increase the speed of the then computer-time-hungry image-processing programs. With Mariner Mars 1971 it became possible to send images faster than they could be processed. Since then, the ratio between transmission time and processing time has gone way up in favor of transmit time. In general, it does not really matter, since instant images are not now a requirement, but for users of image processing other than planetary scientists, additional speed is attractive. Also, as the number of images has skyrocketed from Mariner Mars 1964's 22 to literally tens of thousands in the Voyager and Galileo projects, time to process the images is of interest even to the most patient. The problem is that as the number of pixels has increased, the number of individual computations also increases. A 1,000 by 1,000 pixel image weighted 35 by 35 times requires 1.225 billion multiplications⁶³! If these are done in sequence, the amount of processing time would be formidable.

To solve this problem, Nathan suggested putting 35 sets of 35 multipliers in parallel on very large-scale integration (VLSI) chips. By doing that, the amount of calculations is reduced by 1,225 to 1. Recently, he has begun design of a set of VLSI chips that will speed up the geometry or reprojection operations⁶⁴. Basically, the weighting algorithm is encapsulated in a single chip as a unit of hardware, rather than as software. Logic in hardware executes faster than logic in software because all 1,225 multipliers are operating simultaneously in parallel rather than one at a time serially as in a central processor. Nathan's chips have been plugged into Digital Equipment Corporation VAX 11/780 computers. When the computer is executing an image-processing program and reaches the point where it wants to do the algorithm on the chip, the computer "calls" the chip just as though it were calling a software subroutine.

ORIGINAL PAGE IS
OF POOR QUALITY

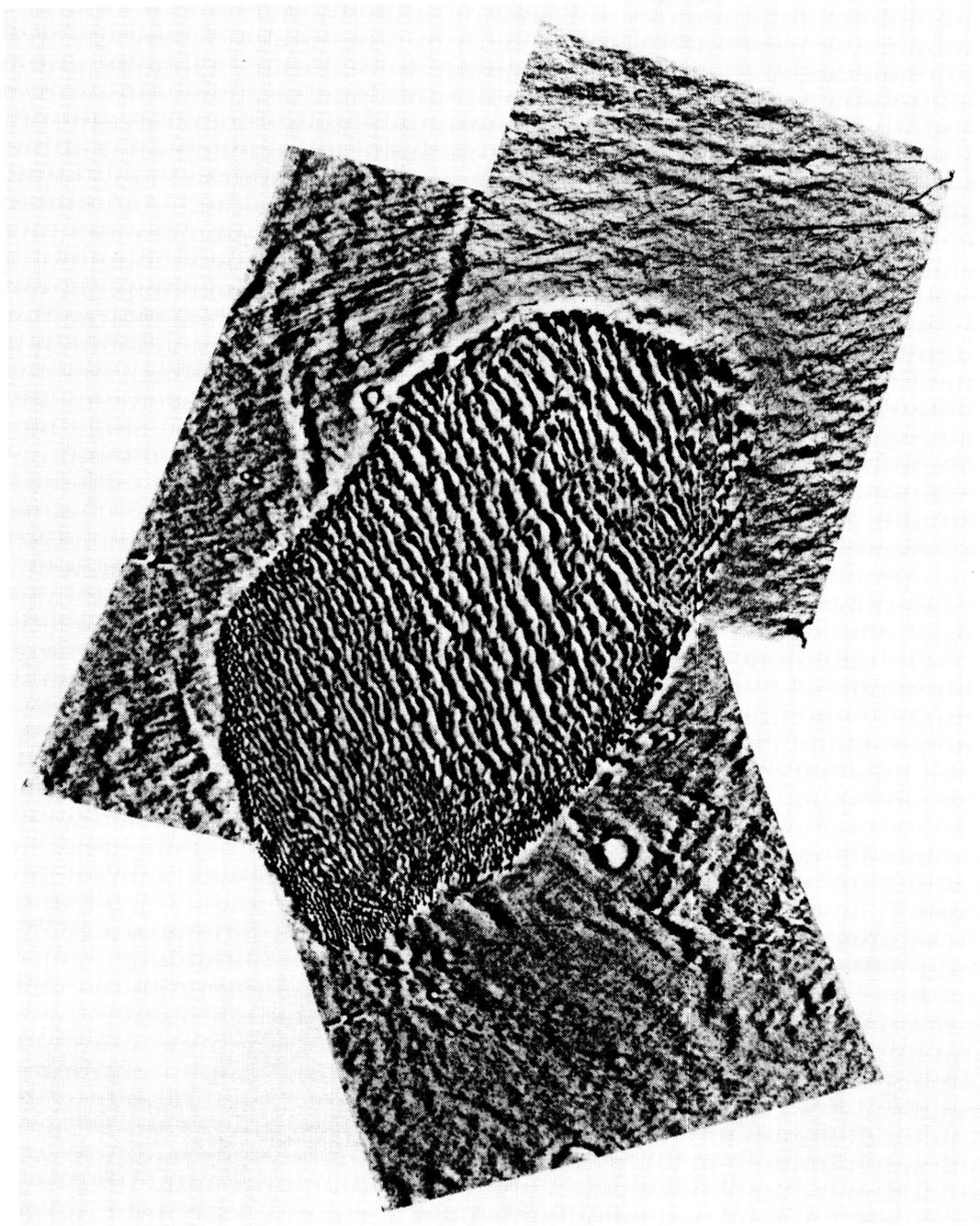


Figure 9-9A. Mosaics combine detailed images into detailed maps: a Martian desert....

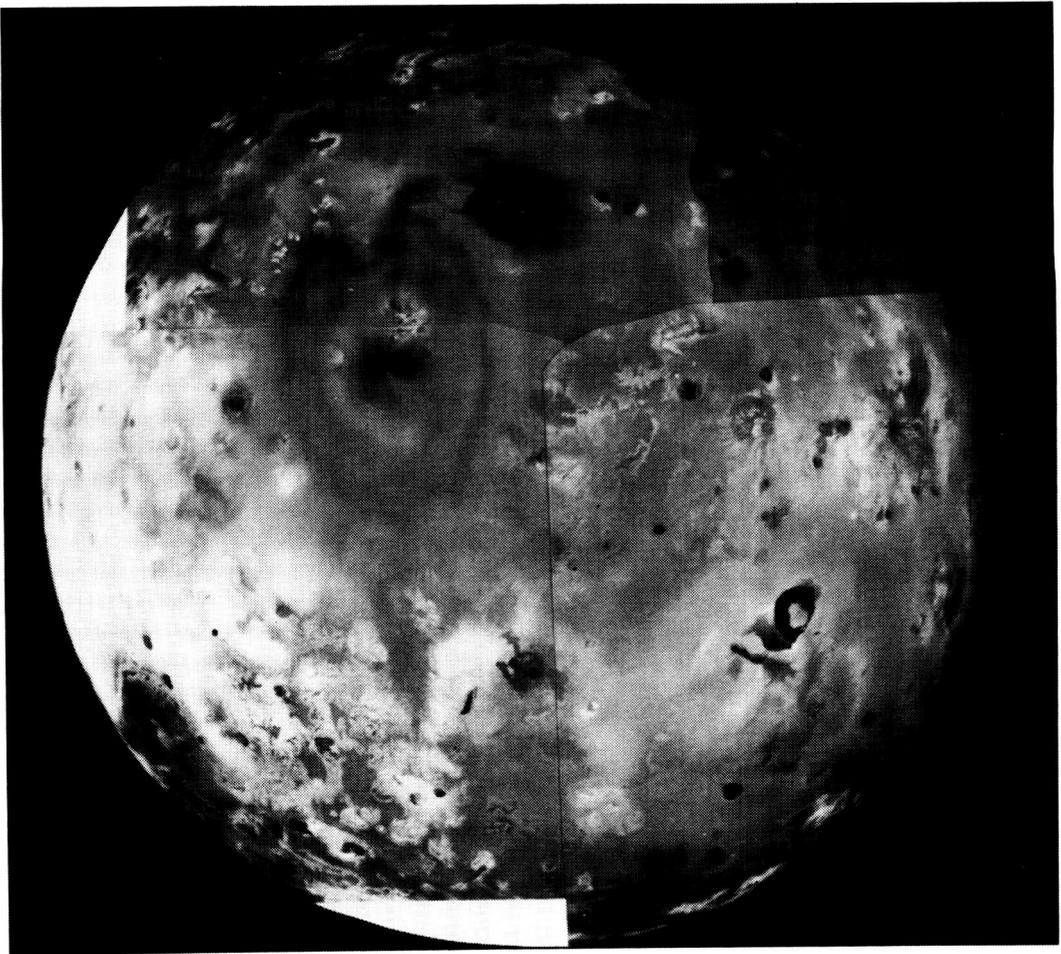


Figure 9-9B.Volcanic Io....

ORIGINAL PAGE IS
OF POOR QUALITY

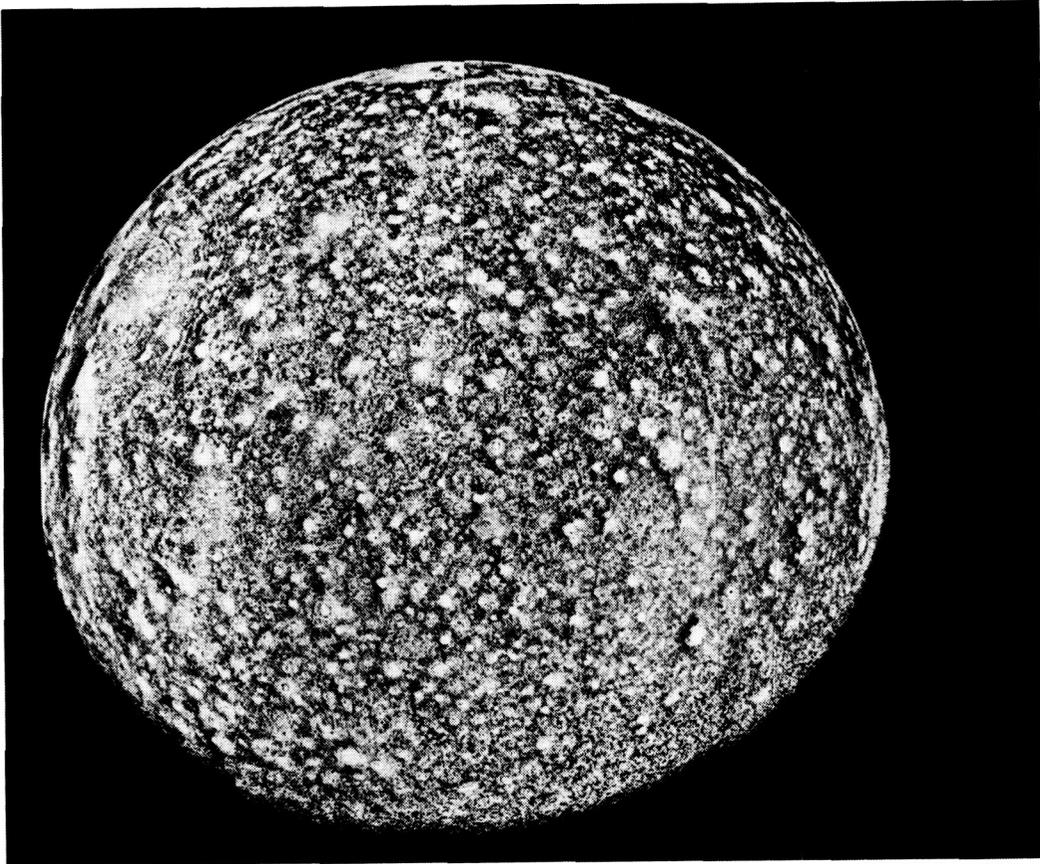


Figure 9-9C.Heavily cratered Callisto. (JPL photos 211-4704; P-21278; P-21746)

Nathan sees his invention not only as the solution to a problem in image processing but also as the beginning of a new future in computing. Using this technique, special-purpose computers with a lot of logic embodied in hardware could easily outstrip the existing systems in speed and accuracy. In some ways, it would be like electronic analog computers, but better in that the rearrangement of components would be simpler.

It is fitting to end on this note, as Nathan's application of computers to fulfill a need in space exploration mirrors the entire story of NASA's use of computers. He approached his tasks in the late 1950s and early 1960s as a pragmatist. He had some computing background, as well as grounding in other fields, so he could see the possibilities of applications. He used equipment usually behind the state of the art but got beyond the state-of-the-art results with it. And, finally, he repays computing by finding one way to improve it on the path to solving yet another problem. Nathan himself said that "NASA is not to be given credit for initiating advances in image-processing technology, but NASA has supported the grass roots initiatives." In general, that is true. NASA never asked for anything that could not be done with the current technology. But in response, the computer industry sometimes

pushed itself just a little in a number of areas. Just a little better software development practices made on-board software safe, just a little better networking made the Launch Processing System more efficient, just a little better operating system made mission control easier, and just a little better chip makes image processing faster. NASA did not push the state of the art, but nudged it enough times to make a difference.

Epilogue: Themes in NASA's Computing Experience

Running throughout the individual histories of American space flight computer systems are five themes that encapsulate NASA's intentions and experiences. Developing and evolving over the last quarter century, they promise to dominate NASA's use of computers for space flight well into the future. The themes are: the need for real time systems, the use of redundancy to maintain reliability and safety, the choice of off-the-shelf equipment wherever possible, the adoption of distributed processing, and adherence to the principles of software engineering in system development.

Real-Time Systems

NASA had no choice but to become a leader in the development of real-time systems, beginning with the decision to use computers to support manned and unmanned flights. Including a computer on-board spacecraft further sealed NASA's fate as a developer and user of embedded computer systems—computers within larger systems replacing or enhancing existing hardware. Therefore, it is in this field of computing that NASA has had its greatest impact.

Contractors working on NASA's real-time systems have been able to benefit from what they learned in the process of completing their contract obligations. For example, an immediate application of techniques used in the Mercury Monitor was IBM's System 360 operating system. Later, experience with fly-by-wire systems quickly spread to civilian and military applications. Within 10 years of the first digital fly-by-wire aircraft flight, airliners using the technology were in prototype. As computers continue to shrink in size and increase in power, the applications of real-time computing will grow enormously.

Reliability and Safety Through Redundancy

NASA has achieved increasing levels of reliability through a concurrent increase in the levels of redundancy. Ground systems always had an active backup. On-board systems acquired them as size and performance improvements made it possible. The use of computers running in parallel, working on the same calculations, made necessary the development of redundancy management techniques. Thus, again, NASA pioneered an area which was as yet poorly developed.

Proven Equipment

Even though NASA led the way in the development and use of some aspects of modern computing, one area in which innovation was purposely avoided was hardware. Acting in the belief that existing equipment is inherently more reliable and less risky than new, custom-designed computers, NASA sought to acquire proven processors wherever possible. As a result, flight systems are often years behind the current state-of-the-art. Nevertheless, they can complete the missions for which they were purchased. In long-term programs, such as the Shuttle, processors are being replaced by newer (but not the *newest*) equipment where possible.

Distributed Processing

Partly as a result of safety considerations, partly for convenience, and partly because different organizations often contribute subsystems to the same spacecraft, there is a continuing trend toward the use of distributed computing both in flight and on the ground. Most new NASA computer systems are functionally distributed. On an unmanned spacecraft, for instance, separate computers handle command interpretation, data acquisition and attitude control. Other examples include the Shuttle Launch Processing System and the Shuttle itself, which has computers on the main engines as well as other components. Again, improved processors will make it cheaper and easier to continue this trend in the future.

Software Engineering

Software engineering has always been a big part of NASA's business, even in the era before 1968 when the term did not yet exist. In recent years, it has become a central focus of activity. NASA has developed an Agency-wide software development standard and made it available to the various Centers. Short courses on software engineering topics are being taught routinely. The Jet Propulsion Laboratory has established a software resource center. Goddard Space Flight Center regularly sponsors a software engineering conference. Conferences have been held to get an early start on the use of the the Ada programming language in the Space Station project. Obviously, NASA is committed to improvement and high quality in this field, as more and more functions on space flights are taken over by computers.

In all, NASA has very effectively adapted its operations to the Computer Age. Computers, frankly, make useful spaceflight possible. Even though a spacecraft could theoretically be placed in orbit using a World War II tilt-table missile guidance system and mechanical clocks, landing safely on the moon, flying within kilometers of the outer planets, and landing on runways after descending from space would all be unlikely happenings with the old technology. As Man begins the era of permanent presence in space, his partner will be millions of bits flashing in a sea of transistors, a helpmate in the discovery of the universe.

Source Notes

Chapter One

1. J. M. Grimwood and B. C. Hacker, *On the Shoulders of Titans*, NASA SP-4203 Washington, D.C., 1977, p.xvi.
2. McDonnell Corporation, *NASA Project Gemini Familiarization Manual*, 1965, vol. 2, pp. 8.7, 8.45.
3. Lenz, in Conrad D. Babb, Charles E. Dunn, John J. Lenz, and John L. Sweeney interview, IBM at Owego, NY, by Ivan Ertel, April 25, 1968, transcript in Johnson Space Center History Office; McDonnell Corporation, *Gemini Familiarization Manual*, p. 8.45.
4. McDonnell Corporation, *Gemini Familiarization Manual*, p. 8.8.
5. Grimwood and Hacker, *On The Shoulders of Titans*, pp. 356, 358.
6. Grimwood and Hacker, *On The Shoulders of Titans*, p. 370.
7. McDonnell Corporation, *Gemini Familiarization Manual*, pp. 8.10-8.11, 8.48-8.49.
8. J. M. Grimwood, B. C. Hacker, and P. J. Vorzimmer, *Project Gemini: Technology and Operations*, G.P.O., Washington, D.C., 1969, p. 40.
9. Grimwood, Hacker, and Vorzimmer, *Technology and Operations*, p. 110; Lenz in Babb, Dunn, Lenz, and Sweeney interview.
10. Lenz, in Babb, Dunn, Lenz, and Sweeney interview.
11. McDonnell Corporation, *Gemini Familiarization Manual*, p. 8.72.
12. Sweeney, in Babb, Dunn, Lenz, and Sweeney interview.
13. IBM, *IBM Gemini Guidance Computer*, Fact Sheet at Johnson Space Center History Office, February 17, 1966.
14. 'Astronauts Control Flights, Aided by a Versatile Digital Computer,' *Electronics* 71-76, (May 3, 1965).
15. P. W. Malik and G. A. Souris, *Project Gemini: A Technical Summary*, NASA CR-1106 Washington, D.C., 1968, p. 117.

16. McDonnell Corporation, *Gemini Familiarization Manual*, pp. 8.75, 8.80; Lenz, in Babb, Dunn, Lenz, and Sweeney interview.
17. Lenz, in Babb, Dunn, Lenz, and Sweeney interview.
18. Homer W. Hutchison interview, Owego, NY, by Ivan Ertel, April 25, 1968; transcript in Johnson Space Center History Office.
19. McDonnell Corporation, *Gemini Familiarization Manual*, p. 8.143.
20. 'Magnetic Tape Memory Stores More Data in Space,' *Prod. Eng.*, 47 (April 25, 1966).
21. Hutchison interview.
22. Babb, in Babb, Dunn, Lenz, and Sweeney interview; 'Magnetic Tape Memory,' *Prod. Eng.*, 47.
23. Hutchison interview.
24. Malik and Souris, *Gemini Technical Summary*, p. 120.
25. Harold E. Dodge interview, Owego, NY, by John J. Lenz, April 25, 1968; transcript at Johnson Space Center History Office.
26. Grimwood and Hacker, *On the Shoulders of Titans*, p. 252.
27. Grimwood and Hacker, *On the Shoulders of Titans*, p. 254.
28. McDonnell Corporation, *Gemini Familiarization Manual*, pp. 8.82-8.89.
29. See the October 1983 issue of the *Annals of the History of Computing* (Vol. 5, [4]) for several articles on SAGE.
30. See Frederick P. Brooks, *The Mythical Man-Month* Addison-Wesley, Reading, MA, 1975.
31. Lenz, in Babb, Dunn, Lenz, and Sweeney interview.
32. James Joachim interview, Owego, NY, by Ivan Ertel, April 25, 1968; transcript in Johnson Space Center History Office; Malik and Souris, *Gemini Technical Summary*, p. 251.
33. Grimwood, Hacker, and Vorzimmer, *Technology and Operations*, p. 102; Malik and Souris, *Gemini Technical Summary*, p. 251.

34. Malik and Souris, *Gemini Technical Summary*, p. 254.
35. McDonnell Corporation, *Gemini Familiarization Manual*, p. 8.104.
36. Dale F. Bachman interview, IBM at Owego, NY, by John J. Lenz, April 25, 1968, transcript at Johnson Space Center History Office; Lee Jackson interview, IBM at Owego, NY, by Ivan Ertel, April 25, 1968, transcript at Johnson Space Center History Office; Leroy S. Jimerson, Jr., interview, IBM at Owego, NY, by Ivan Ertel, April 26, 1968, transcript at Johnson Space Center History Office; Malik and Souris, *Gemini Technical Summary*, pp. 254–255; Sweeney, in Babb, Dunn, Lenz, and Sweeney interview.
37. Joachim interview.
38. Malik and Souris, *Gemini Technical Summary*, p. 118.
39. Dodge interview; Malik and Souris, *Gemini Technical Summary*, p. 118; McDonnell Corporation, *Gemini Familiarization Manual*, pp. 8.44, 8.51–8.52, 8.121, 8.123.
40. Malik and Souris, *Gemini Technical Summary*, p. 137; McDonnell Corporation, *Gemini Familiarization Manual*, pp. 8.46, 8.48.
41. Gene Cerman interview, telephone from Houston, November 7, 1983.
42. John Young interview, telephone from Houston, March 6, 1984.
43. IBM, p. 3.
44. Cerman interview.
45. Young interview.
46. McDonnell Corporation, *Gemini Familiarization Manual*, p. 8.47.
47. Bachman interview.
48. 'Astronauts Control Flights', *Electronics*; Sweeney, in Babb, Dunn, Lenz, and Sweeney interview.
49. Sweeney, in Babb, Dunn, Lenz, and Sweeney interview.
50. Sweeney, in Babb, Dunn, Lenz, and Sweeney interview.
51. Westkaemper, in Lee Jackson and Robert M. Westkaemper interview, IBM at Owego, NY, by Ivan Ertel, Apr. 25, 1968; transcript in Johnson Space Center History Office.

52. Hutchison interview.
53. Joachim interview.
54. Babb, in Babb, Dunn, Lenz, and Sweeney interview.
55. Lenz, in Babb, Dunn, Lenz, and Sweeney interview.

Chapter Two

1. D. G. Hoag, *Apollo Navigation Guidance Computer Systems*, Report E-2411 MIT, Cambridge, MA, April 1969, p. 2.
2. A. L. Hopkins, 'Electronic Navigator Charts Man's Path to the Moon,' *Electronics*, 109 (January 9, 1967); Ralph Ragan interview, MIT, Cambridge, MA, by Ivan Ertel, April 28, 1966, Johnson Space Center transcript.
3. Hoag, *Apollo Navigation Systems*, p. 3.
4. Hoag, *Apollo Navigation Systems*, pp. 3–4.
5. P. Hersch, 'Engineers Reassessing Electronic Hardware in the Light of Some Near Failures on *Apollo 12*,' *IEEE Spec.*, 23 (January 1970).
6. B. K. Thomas, Jr., '*Apollo 8* Proves Value of Onboard Control,' *Aviation Week*, 43 (January 20, 1969).
7. Steve Bales interview, Johnson Space Center, Houston, TX, May 31, 1983; Howard W. Tindall, notes in review of draft chapter, August 1985.
8. John R. Garman interview, Johnson Space Center, Houston, TX, May 25, 1983 and June 1, 1983.
9. C. G. Brooks, L. S. Grimwood, and L. S. Swenson, Jr., *Chariots for Apollo: A History of Manned Lunar Spacecraft*, NASA SP-4205 Washington, D.C., 1979, p. 355.
10. Hopkins, 'Electronic Navigator,' p. 116.
11. Brooks, Grimwood, and Swenson, *Chariots for Apollo*, pp. 38–39.
12. Ragan interview.
13. Hoag, *Apollo Navigation Systems*, p. 5.

14. Ragan interview.
15. Eldon Hall lecture, "The Apollo Guidance Computer—A Designer's View," Digital Computer Museum, Marlboro, MA, June 10, 1982, transcript at museum's Boston location, p. 4.
16. P. G. Felleman and D. C. Fraser, "Digital Fly-by-Wire: Computers Lead the Way," *Astronaut. and Aeronaut.*, 30 (July–August 1974).
17. H. J. Goett to G. Low, "Recommendations for Apollo On-board Guidance Computer," June 6, 1961, JSC History Office.
18. L. A. Wood to R. Chilton, NASA Space Task Group, "Information of RCA Computers Adaptable to Guidance," June 6, 1961, JSC History Office.
19. R. G. Chilton to multiple addresses, NASA Space Task Group, "Discussions at Goddard of a Possible Joint Development Program for Airborne Computers for OAO and Apollo," memo, June 6, 1961, JSC History Office.
20. R. Alonso to multiple addresses, MIT, "Review of Saturn Computer Discussions, memo, June 27, 1961, JSC History Office.
21. D. W. Gilbert to multiple addresses, JSC, "Guidance Computer for Apollo," memo, July 9, 1963, JSC History Office.
22. Maj. Gen. Samuel C. Phillips, deputy director of the Apollo Program, reported on a meeting to discuss the use of the triple modular redundant Saturn launch vehicle computer in Apollo. S. C. Phillips to multiple addresses, "Saturn V/Apollo Spacecraft Guidance Computer Conference," memo, May 14, 1964, JSC History Office.
23. Hopkins, "Electronic Navigator", p. 110.
24. Ragan interview.
25. E. C. Hall, *Reliability History of the Apollo Guidance Computer*, NASA CR-140340 MIT, Cambridge, MA, 1972, p. 25.
26. C. D. Brady to multiple addresses, "Integrated Circuit Packages for the AGC," memo, June 19, 1964, JSC History Office.
27. D. G. Hoag interview, MIT, Cambridge, MA, by Ivan Ertel, April 29, 1966, Johnson Space Center transcript.
28. Brooks, Grimwood, and Swenson, *Chariots for Apollo*, p. 187.
29. R. Alonso, H. Blair-Smith, and A. Hopkins, "Logical Description for the Apollo Guidance Computer," MIT, Cambridge, MA, March 1963, p. 1.1.

308 COMPUTERS IN SPACEFLIGHT

30. Hopkins, "Electronic Navigator," p. 110.
31. Hall, *Reliability History*, p. 9.
32. Alonso, Blair-Smith, and Hopkins, "Logical Description," p. 1.1.
33. Alonso, Blair-Smith, and Hopkins, "Logical Description," p. vii.
34. Ragan interview.
35. Ragan interview.
36. Hall, *Reliability History*, p. 5.
37. Hopkins, "Guidance Computer Design," *Spacecraft Navigation, Guidance, and Control* MIT, Cambridge, MA, 1965, pp. 13–15.
38. Hopkins, "Electronic Navigator," p. 112.
39. R. H. Battin and F. H. Martin, "Computer Controlled Steering of the Apollo Spacecraft," *J. Spacecr. Rockets*, 5, 402 (1968); A. Drake and B. I. Savage, *AGC4 Basic Training Manual* MIT, Cambridge, MA, 1967, pp. 1.1–1.2.
40. F. Bedford, S. P. Cockrell, and R. T. Savely, *Apollo Experience Report: Onboard Navigational and Alignment Software*, MSC-04238, Houston, TX, 1971, p. 2; Hopkins, "Electronic Navigator," p. 113.
41. Drake and Savage, *AGC4*, p. 1.3.
42. Hopkins, "Electronic Navigator," p. 118.
43. Hall, *Reliability History*, p. 10; Raytheon Corporation, *Apollo Guidance Computer Program Block 1 (100) and Block 2 Final Report* (July 25, 1969–December 31, 1969), p. 2.17.
44. Hall lecture, p. 5.
45. R. Alonso and A. Hopkins, *The Apollo Guidance Computer*, NASA CR-118183, MIT, Cambridge, MA, p. 8.
46. E. M. Copps, *Recovery from Transient Failures of the Apollo Guidance Computer*, NASA CR-92255, MIT, Cambridge, MA, 1968, p. 2.
47. Drake and Savage, *AGC4*, p. 1.13.

48. H. Kreide and D. W. Lambert, "Computation: Aerospace Computers in Aircraft, Missiles, and Spacecraft," *Space/Aeronaut.*, 42 77 (1964).
49. Alonso and Hopkins, *Apollo Computer*, p. 12.
50. L. J. Carey and W. A. Sturm, "Space Software at the Crossroads," *Space/Aeronaut.* 63 (December 1968); Kreide and Lambert, "Computation," pp. 97-98.
51. A. Hopkins, "Design Concepts of the Apollo Guidance Computer," Mimeograph, MIT Instrumentation Lab, Cambridge, MA, June 1963, p. 2.1.
52. Alonso, Blair-Smith, and Hopkins, "Logical Description," p. 1.3; Stan Mann interview, Johnson Space Center, Houston, TX, June 6, 1983.
53. E. C. Hall, *MIT's Role in Project Apollo: Computer Subsystem*, Charles Stark Draper Laboratory, Cambridge, MA, 1972, vol. 3, p. 3.
54. Alonso, Blair-Smith, and Hopkins, "Logical Description," p. 4.2; Hopkins, "Design Concepts," June 1963, p. 2.5; P. Kuttner, "The Rope Memory—A Permanent Storage Device," *Proc. AFIPS*, 49 (November 1963).
55. Hopkins, "Electronic Navigator," p. 114.
56. Raytheon Corporation, *Final Report*, p. 2.29.
57. Drake and Savage, *AGC4*, p. 1.6.
58. John R. Garman interview, Johnson Space Center, Houston, TX, May 25, 1983 and June 1, 1983.
59. Raytheon Corporation, *Final Report*, p. 4.5.
60. A. Laats and J. E. Miller, *Apollo Guidance and Control System Flight Experience*, NASA CR-101823, MIT, Cambridge, MA, 1969, p. 1.
61. Raytheon Corporaton, *Final Report*, p. 2.56.
62. Ragan interview.
63. R. C. Seamans, Jr., to multiple addresses, "Raytheon Negotiations on Apollo Guidance Computer," memo, November 9, 1962, JSC History Office.
64. J. F. Shea to multiple addresses, Johnson Space Center, "Integrated Circuit Packages for the Block II Apollo Guidance Computer (AGC)," memo, September 9, 1964, JSC History Office.

310 COMPUTERS IN SPACEFLIGHT

65. Hall, *Reliability History*, p. 43.
66. C. W. Frasier to multiple addresses, Johnson Space Center, "Block II Computer Design Deficiency," memo, August 10, 1965, JSC History Office.
67. I. V. Ertel, *The Apollo Spacecraft*, GPO, Washington, D.C., 1969, pp. 31–32.
68. Hall, *Reliability History*, pp. 15–16, 19.
69. Raytheon Corporation, *Final Report*, p. 3.12.
70. Felleman and Fraser, "Digital Fly-By-Wire," p. 30.
71. W.M. Keese, et al, *Management Procedures in Computer Programming for Apollo—Interim Report*, Bellcomm. Inc., Washington, D.C., November 30, 1964.
72. Garman interview.
73. Hopkins, "Design Concepts," June 1963, p. 3.11.
74. Frank Hughes interview, Johnson Space Center, Houston, TX, June 2, 1983.
75. Ertel, *Apollo Spacecraft*, p. 288.
76. Mann interview.
77. Guidance Software Validation Commission, *Apollo Guidance Software Development and Verification Plan*, Manned Spacecraft Center, Houston, TX, October 4, 1957, p. 2.1.
78. Ed Lineberry interview, Johnson Space Center, Houston, TX, June 2, 1983.
79. Dick Parten interview, Johnson Space Center, Houston, TX, June 3, 1983 and June 16, 1983.
80. M. D. Richter to multiple addresses, MIT, "Summary of AGC Program Processing Procedures," memo, August 13, 1965, JSC History Office.
81. Garman interview.
82. RASPO to MIT memo, JSC History Office Archives, October 22, 1963.
83. Hopkins, "Guidance Computer Design," p. 48.
84. Mann interview.

85. C. A. Muntz, *Users Guide to the Block 2 AGC/LGC Interpreter*, NASA-CR-126815, MIT, Cambridge, MA, April 1965.
86. Hopkins, "Electronic Navigator," p. 117.
87. Battin and Martin, "Computer Controlled Steering," p. 403.
88. Hopkins, "Electronic Navigator," p. 117.
89. Copps, *Recovery From Transient Failures*, p. 3.
90. D. J. Bowler, *Apollo Guidance Computer Improvement Study—Apollo Guidance, Navigation, and Control*, NASA CR-114898, MIT, Cambridge, MA, 1970, p. 2.
91. George W. Cherry to multiple addresses, "Exegesis of the 1201 and 1202 Alarms Which Occurred During the [Mission G] Lunar Landing," memo no. 370-69, MIT Instrumentation Lab, Cambridge, MA, August 4, 1969, JSC History Office.
92. Hopkins, "Electronic Navigator," p. 117; T. Lawton and C.A. Muntz, *Organization of Computation and Control of the Apollo Guidance Computer*, MIT Instrumentation Lab., Cambridge, MA, E-1758, 1965, p. 15.
93. Drake and Savage, *AGC4*, p. 1.21.
94. Garman interview.
95. Hopkins, "Design Concepts," June 1963, pp. 2.8–2.9; Lawton and Muntz, *Computation and Control*, p. 15.
96. Garman interview.
97. Raytheon Corporation, *Final Report*, p. 2.32.
98. Howard "Bill" Tindall to multiple addresses, "Spacecraft Computer Program Names," memo, May 23, 1967, JSC History Office.
99. Ertel, *Apollo Spacecraft*, p. 238.
100. Copps, *Recovery From Transient Failures*, p. 4; Hall, *Reliability History*, pp. 7–8.
101. Copps, *Recovery From Transient Failures*, pp. 1, 5; Cherry, "Exegesis of the 1201 and 1202 Alarms."

312 COMPUTERS IN SPACEFLIGHT

102. Copps, *Recovery From Transient Failures*, p. 3.
103. Hopkins, "Design Concepts," p. 2.11.
104. Hopkins, "Design Concepts," p. 2.11.
105. Hall, *Reliability History*, p. 31.
106. Cherry, "Exegesis of the 1201 and 1202 Alarms."
107. Garman interview.
108. Cherry, "Exegesis of the 1201 and 1202 Alarms."
109. T. Lawton and C. A. Muntz, *Verification Plan for AGC/LGC*, MIT, Cambridge, MA, E-1786, 1964, p. 5.
110. Lawton and Muntz, *Verification Plan*, p. 5; MIT, *Guidance System Operations Plan for Manned CM Earth Orbital and Lunar Missions Using Program COLLOSSUS*, sec. 5, NASA CR-97515, MIT, Cambridge, MA, n.d., p. 6.2.
111. Mann interview.
112. Carey and Sturm, "Space Software," p. 63.
113. Mann interview.
114. Lawton and Muntz, *Verification Plan*, p. 14.
115. Guidance Software Control Panel File, JSC History Office Archives, May 13, 1966.
116. Bowler, *Improvement Study*, p. 2.
117. Hoag interview.
118. Tindall to multiple addresses, "Apollo Spacecraft Computer Program Development Newsletter," memo, May 31, 1966, JSC History Office.
119. Tindall, "Program Development Newsletter."
120. See Madeline S. Johnson, *MIT's Role in Project Apollo: The Software Effort*, Charles Stark Draper Laboratory, Cambridge, MA, 1971, vol. 2, pp. 21–22 for a chart showing the "manloading" at MIT during the Apollo effort.

121. Mann interview.
122. Tindall to multiple addresses, "Apollo Spacecraft Computer Programs—Or, A Bucket of Worms," memo, June 13, 1966, JSC History Office.
123. Tindall to multiple addresses, "Another Apollo Spacecraft Computer Status Report," memo, July 1, 1966, JSC History Office.
124. Tindall to multiple addresses, "Spacecraft Computer Programming Development Improvements to be Utilized by MIT," memo, 67-FM-1-T:85, October 18, 1967.
125. Tindall, "Program Development Newsletter."
126. Tindall to multiple addresses, untitled memos, May 13, 1966 and May 14, 1966, JSC History Office.
127. Tindall, "Apollo Spacecraft Computer Programs—Or, A Bucket of Worms."
128. Tindall, "Apollo Spacecraft Computer Programs—Or, A Bucket of Worms."
129. Tindall to multiple addresses, "AS-204 Computer Program Status," memo, July 21, 1966, JSC History Office.
130. Tindall to multiple addresses, "Spacecraft Computer Program Status for AS-501," memo, September 20, 1966, JSC History Office.
131. Tindall, "Apollo Spacecraft Computer Programs—Or, A Bucket of Worms."
132. Ray Morth, "De-orbit Burn Program," (Flight 204), memo no. 35, January 23, 1967, MIT, Cambridge, MA, JSC History Office.
133. Hughes interview; Tindall concurs. He wrote in a memo for distribution on April 28, 1967 that "it is almost certain that deficiencies will exist in the program we will ultimately fly." He then proceeded to improve the software during the delay in the Apollo program caused by the fire.
134. Tindall to multiple addresses, "In Which is Described the Apollo Spacecraft Computer Programs Currently Being Developed," memo, March 24, 1967, JSC History Office.
135. Tindall to multiple addresses, "Spacecraft Computer Program Status, memo, October 17, 1967, JSC History Office.
136. Tindall to multiple addresses, "A New Spacecraft Computer Program Development Working Philosophy is Taking Shape," memo, 67-FM-1-39, May 17, 1967.

314 COMPUTERS IN SPACEFLIGHT

137. Ertel, *Apollo Spacecraft*, p. 250.
138. Ertel, *Apollo Spacecraft*, pp. 203–204.
139. John P. Mayer to multiple addressees, "Notes on Meeting of the Apollo Guidance Software Task Force in Washington on February 9, 1968," memo no. 68-RM-10, February 13, 1968.
140. Apollo Guidance Software Task Force, "Final Report," memo, September 23, 1968, JSC History Office, p. 4.
141. Apollo Guidance Software Task Force, "Final Report," p. 7; Ertel, *Apollo Spacecraft*, p. 250.
142. Ertel, *Apollo Spacecraft*, p. 288.
143. Mann interview.
144. W. J. North and C. H. Woodling, "Apollo Crew Procedures, Simulation, and Flight Planning," *Astronaut. and Aeronaut.* 58 (March 1970).
145. Howard W. Tindall, in his review of the draft of this chapter, noted that the crew insisted on flexibility and control from the start of the program, so in a way they set the level of computer-related activity for themselves.
146. Raytheon Corporation, *Final Report*, p. 2.9.
147. J. M. Dahlen, *Apollo Guidance Navigation and Control: Guidance and Navigation System Operations Plan, Apollo Mission 202*, NASA CR-65770, MIT, Cambridge, MA, 1966, p. 3.19; Laats and Miller, *Apollo Guidance Flight Experience*, p. 6.
148. Hopkins, "Design Concepts," June 1963, p. 2.3.
149. Hall lecture, p. 8.
150. Copps, *Recovery From Transient Failures*, p. 2.
151. Hopkins, "Electronic Navigator," p. 116.
152. Hopkins, "Electronic Navigator," p. 117.
153. Copps, *Recovery From Transient Failures*, p. 2.
154. Garman interview.

155. Gene Cernan, telephone interview from Houston, November 7, 1983.
156. Vance Brand interview, Johnson Space Center, Houston, TX, June 2, 1983; John Young interview, telephone from Houston, TX, March 2, 1984.
157. Ragan interview.
158. Cernan interview.
159. Mann interview.
160. P. M. Kurten, *Apollo Experience Report: Guidance and Control Systems—Lunar Module Abort Guidance System*, NASA TN -D-7990, Johnson Space Center, Houston, TX, 1975, p. 4.
161. TRW, news release, July 1, 1969, JSC History Office Archives.
162. TRW, news release, July 1, 1969.
163. Kurten, *Abort Guidance System*, p. 1; B. Miller, "Abort Backup for LEM Near Production," *TRW Executive Clips*, 1 (January 15, 1966).
164. Kurten, *Abort Guidance System*, p. 5.
165. Kurten, *Abort Guidance System*, p. 6.
166. B. Miller, "Abort Backup," p. 4.
167. J.J. Seidman, "LEM/AGS Marco 4418", mimeograph, September 1966, JSC History Office, p. 1.
168. Kurten, *Abort Guidance System*, p. 7; B. Miller, "Abort Backup," p. 4; TRW, news release, July 1, 1969.
169. B. Miller, "Abort Backup," p. 3.
170. Jonas Beraru, "The TRW Systems MARCO 4418—A Man Rated Computer," (TRW, 1979), p. 26.
171. TRW, news release, July 1, 1969.
172. Kurten, *Abort Guidance System*, pp. 8–9.
173. Kurten, *Abort Guidance System*, p. 42.

316 COMPUTERS IN SPACEFLIGHT

174. Tindall, notes in review of the draft of this chapter, August, 1985.
175. Kurten, *Abort Guidance System*, p. 38.
176. Kurten, *Abort Guidance System*, pp. 22–23.
177. TRW, news release, July 1, 1969.
178. Young interview.
179. Young interview.
180. Apollo Guidance Software Task Force, *Final Report*, pp. 5–6; R. R. Regelbrugge, *Apollo Experience Report: Apollo Spacecraft and Ground Software Development for Rendezvous*, MSC-02676, 1970, p. 5; S. A. Sjoberg, "Objectives for Software Controlled Aerospace Systems in the Next Decade," memo for distribution, Manned Spacecraft Center, July 14, 1970, p. 2.
181. Bowler, *Improvement Study*.

Chapter Three

1. Bill Chubb interview, Marshall Space Center, Huntsville, AL, June 22, 1983.
2. Steve Bales interview, Johnson Space Center, Houston, TX, May 31, 1983.
3. IBM, *Skylab Reactivation Mission*, IBM Federal Systems Division, Huntsville, AL, September 12, 1979, p. 1.2.
4. See Charles D. Benson and W. David Compton, *Living and Working in Space: A History of Skylab*, NASA SP-4208, Washington, D.C., 1983.
5. IBM, *Design and Operation Assessment of Skylab ATMDC/WCIU Flight Hardware and Software*, IBM Federal Systems Division Electronics System Center, Huntsville, AL, May 9, 1974, p. 1.1.3.
6. IBM, *Skylab Reactivation Mission*, p. 1.1.
7. IBM, *Design and Operation Assessment*, p. 1.1.1.
8. James McMillion interview, Marshall Space Center, Huntsville, AL, June 22, 1983.
9. Charles Swearingen interview, Huntsville, AL, June 21, 1983.

10. IBM, *Design and Operation Assessment*, p. 1.1.14.
11. Martin Marietta Corporation, *Skylab Data Handbook*, Marshall Space Center, Huntsville, AL, n.d., p. 4.13.
12. IBM, *Design and Operation Assessment*, p. 1.1.3.
13. John Copeland interview, IBM at Marshall Space Center, June 23, 1983.
14. IBM, *Design and Operation Assessment*, p. 1.1.22.
15. Ibid., p. 1.1.1.
16. Copeland interview.
17. IBM, *Design and Operation Assessment*, p. 1.1.12.
18. Ibid., p. 1.1.1.
19. Ibid., p. 1.1.28.
20. Ibid., p. 1.1.11.
21. Ibid., p. 1.1.5.
22. Ibid., pp. 1.1.7, 1.1.15, 1.1.18.
23. Ibid., p. 1.1.12.
24. Ibid., p. 1.1.11.
25. For a complete discussion of these problems, see Ibid., pp. 1.1.18–1.1.22.
26. IBM, *Design and Operation Assessment*, p. 1.1.24.
27. Ibid., p. 1.1.3.
28. For the complete story, see Frederick P. Brooks, *The Mythical Man–Month*, Addison-Wesley, Reading, MA, 1975, Harlan D. Mills, *Software Productivity*, Little, Brown, and Co., Boston, MA, 1983.
29. Madeline S. Johnson, *MIT's Role in Project Apollo: The Software Effort*, Charles Stark Draper Laboratory, Cambridge, MA, 1971, vol. 5, p. 18.

318 COMPUTERS IN SPACEFLIGHT

30. Copeland interview.

31. IBM, *Design and Operation Assessment*, p. 1.2.8.

32. Ibid., p. 1.2.9.

33. Ibid., pp. 1.2.33–1.2.34.

34. Ibid., p. 1.2.28.

35. Ibid.

36. Ibid., pp. 1.2.28–1.2.29.

37. Ibid., p. 1.2.29.

38. Ibid., p. 1.2.54.

39. Ibid., p. 1.2.29.

40. Ibid., p. 1.2.30.

41. Ibid., pp. 1.2.36–1.2.37.

42. Ibid., p. 1.2.36.

43. Ibid., p. 1.2.47.

44. Ibid., p. 1.2.36–1.2.37.

45. IBM, *NASA Skylab A: Apollo Telescope Mount Digital Computer Program Definition Document*, Electronic Systems Center, 1972, p. I.5.1.

46. IBM, *Design and Operation Assessment*, p. 1.2.48.

47. Ibid., p. 1.2.31.

48. Ibid., p. 1.2.64.

49. Ibid., p. 1.2.16.

50. IBM, *NASA Skylab A*, p. I.7.8.

51. Ibid., p. I.1.11.
52. Ibid., p. I.1.10.
53. Copeland interview.
54. IBM, *Design and Operation Assessment*, p. 1.1.6.
55. IBM, *NASA Skylab A*, p. I.7.3.
56. IBM, *Skylab Reactivation Mission*, p. 3.16.
57. Ibid., pp. 3.17–3.18.
58. IBM, *Design and Operation Assessment*, pp. 1.2.38–1.2.39, 1.2.51–1.2.52.
59. Ibid., pp. 1.2.1–1.2.3, 1.2.6.
60. Ibid., p. 1.2.67; IBM, *Skylab Reactivation Mission*, p. 3.25.
61. Copeland interview.
62. Jack Lousma, telephone interview from Houston, July 5, 1983.
63. Copeland interview.
64. IBM, *NASA Skylab A*, p. I.5.3.
65. Ibid., p. I.7.2.
66. IBM, *Skylab Reactivation Mission*, pp. 1.3, 2.1–2.2, 2.4.
67. Ibid., pp. 1.2, 5.3.
68. Ibid., pp. 3.19–3.20.
69. Ibid., pp. 3.3–3.4, 3.9.
70. Ibid., pp. 3.10–11, 3.20, 3.22.
71. Ibid., pp. 3.30, 3.33.

Chapter Four

1. Interview with Arnold Aldrich, Johnson Space Center, June 13, 1983.
2. Interview with Lynn Killingbeck, IBM, Johnson Space Center, June 7, 1983.
3. For a full discussion of the evolution of Shuttle design concepts, see the first two chapters of the forthcoming Space Shuttle Chronology.
4. NASA, *Space Vehicle Design Criteria: Spaceborne Digital Computer Systems*, SP-8070, March 1971.
5. Killingbeck interview; In 1967, the Manned Spacecraft Center contracted with IBM for a conceptual study of spaceborne computers; see M. Ball, and F.H. Hardie, "Computer Partitioning Improves Long-term Reliability in Space," *Space/Aeronaut.*, 114–118 (May 1967).
6. F.J. Hudson and J.C. McCall, "Integrated Electronics System for Space Shuttle," *AIAA Advanced Space Transportation Meeting*, Cocoa Beach, FL, February 4–6, 1970.
7. M. Hamilton and S. Zeldin "Higher Order Software Techniques Applied to a Space Shuttle Prototype Program," *Programming Symposium*, ed. B. Robimet, Springer-Verlag, New York, 1974, pp. 17–32; *Higher-Order Software—A Methodology for Defining Software*, AIAA Paper 75-593; *Higher Order Software Requirements*, MIT Draper Laboratory, Cambridge, MA, August 1973.
8. G.A. Vacca et al. "Mission Influences on Advanced Computers," *Astronaut. Aeronaut.*, 36–37 (April 1967).
9. NASA, *Space Vehicle Design Criteria*, p. 16.
10. R.L. Alonso; and G.C. Randa "Flight-Computer Hardware Trends," *Astronautics and Aeronautics*, April 1967, pp. 31.
11. B.W. Boehm, "Some Information Processing Implications of Air Force Space Missions: 1970–1980," Memorandum, Rand Corp., January 1970, V.
12. Boehm, "Information Processing Implications," p. 21.
13. H. Kreide and D.W. Lambert, "Computation: Aerospace Computers in Aircraft, Missiles and Spacecraft," *Space/Aeronaut.*, 42, 78 (1964); see also N.H. Herman and U.S. Lingon, "Mariner 4 Timing and Sequencing," *Astronaut. Aeronaut.*, 43 (October 1965).
14. A.E. Cooper, "The Shuttle Computer Complex," *Space Transportation System: The IBM Role*, IBM Corporation, 1981, p. 3.

15. NASA, *Space Vehicle Design Criteria*, p. 44.
16. Z. Strickland, "NASA Seeks Ways to Handle Data Flood," *Aviation Week*, June 22, 1970, p. 135.
17. E.S. Chevers, "Proposed Avionics Baseline for the Shuttle," Memorandum, Manned Spacecraft Center, Houston, TX, August 1971, JSC History Office Archives.
18. Interview with Stan Mann, Johnson Space Center, June 8, 1983.
19. Draper Laboratory, "Computer Hardware," charts, March 1971, Johnson Space Center Archives.
20. Draper Laboratory, "Computer Hardware."
21. P.H. Stakem, "One Step Forward - Three steps Backup: Computing in the U.S. Space Program," *BYTE*, 114 (September 1981).
22. Draper Laboratory, "Computer Hardware."
23. J. Kernan, "Desirable Computer Features," Memorandum, Cambridge: MIT, March 5, 1971, JSC History Office; see also Draper Laboratory, "Computer Hardware."
24. Boehm, "Information Processing Implications," p. 19.
25. R.E. Poupard et al., "Design Considerations for Shuttle Information Management," *Astronaut. Aeronaut.*, 53 (May 1973).
26. "Guidance Software Programming Advances," *Aviation Week*, 73 (November 8, 1976).
27. Interview with Dick Parten, Johnson Space Center, June 16, 1983.
28. Killingbeck interview.
29. "New Family of Computers: Military and Aerospace," *Electronics*, 42 (October 31, 1966).
30. IBM, *Space Shuttle Advanced System/4 Pi Model AP-101 Central Processor Unit*, File no. 75-A97-001, 1975, 4.64; see also Cooper, "Shuttle Computer Complex," p. 7.
31. A.J. Macina "Space Shuttle Program," Part I, Memorandum, IBM Federal Systems Division, Houston, p. 16; see also Cooper, "Shuttle Computer Complex," p. 5.

322 COMPUTERS IN SPACEFLIGHT

32. IBM, *Model AP-101 Central Processor Unit*, 1.2, 2.8.
33. Macina, "Space Shuttle," Part I, p. 16.
34. Cooper, "Shuttle Computer Complex", p. 4.
35. IBM, *Model AP-101 Central Processor Unit*, 2.14.
36. IBM, *Space Shuttle Model AP-101 C/M Principles of Operation*, File no. 6246156B, 1974, 2.2.
37. Cooper, "Shuttle Computer Complex", p. 6.
38. IBM, *Model AP-101 Principles of Operation*, 2.15.
39. IBM, *Model AP-101 Principles of Operation*, 2.17.
40. NASA, *Shuttle Flight Operations Manual: Volume 5—Data Processing System*, Flight Operations Directorate, Crew Training and Procedures Division, Johnson Space Center, Houston, TX, November 1978, p. 50.
41. Cooper, "Shuttle Computer Complex," p. 8.
42. IBM, *Model AP-101 Central Processor Unit*, 5.1.
43. Killingbeck interview.
44. Interview with John R. Garman, Johnson Space Center, June 1, 1983.
45. Killingbeck interview.
46. Interview with Dick Parten, June 3, 1983.
47. Garman interview, June 1, 1983.
48. Parten interview, June 16, 1983.
49. Cooper, "Shuttle Computer Complex," p. 9.
50. IBM, *Model AP-101 Central Processor Unit*, 4.36.
51. R.E. Poupard, "Redundant Computer Operations," in *Space Transportation System*, p. 21.

52. IBM, *Space Shuttle Advanced System/4 Pi Input/output Processor*, File no. 6246556A, 1976, part III, p. 1.
53. IBM, *Input/output Processor*, part II, p. 1.
54. IBM, *Space Shuttle Advanced System/4 Pi Prototype Input/Output Processor*, File no. 74-A31-016, 1974, p. 1.3.
55. IBM, *Prototype Input/Output*, p. 1.1.
56. NASA, *Data Processing System Overview Workbook*, DPS-OV-2102, 1979, p. 16.
57. A.E. Cooper and E.S. Flanders, "Shuttle Multifunction CRT Display and Mass Memory Subsystem," in *Space Transportation System*, p. 11; Macina, "Space Shuttle," Part I, p. 24; NASA, *Overview Workbook*, p. 54.
58. NASA, *Shuttle Operations Manual*, p. 4.58.
59. "Velocity, Altitude Regimes to Push Computer Limits," *Aviation Week*, 49 (April 6, 1981).
60. D.C. Fraser and P.G. Felleman, "Digital Fly-by-Wire: Computer Lead the Way," *Astronaut. Aeronaut.*, 12, 24-32 (July-August 1974).
61. Killingbeck interview.
62. For a more complete description see James E. Tomayko, "Digital Fly-By-Wire: A Case of Bidirectional Technology Transfer," in *Aerospace Historian*, 33 (1), 10.18 (March 1986).
63. B.R.A. Burns, "Control Configured Combat Aircraft," *Active Controls in Aircraft Design*, P.R. Kurzhals, ed., AGARDograph #234, NATO, London, 1978, p. 3.15.
64. Poupard, "Redundant Computer," p. 20.
65. Mann interview, June 8, 1983.
66. A.D. Aldrich, "A Sixth GPC On-Orbit," Memorandum, Johnson Space Center, Houston, TX, October 13, 1978, JSC History Office.
67. Garman interview, June 1, 1983.
68. Mann interview, June 8, 1983.

324 COMPUTERS IN SPACEFLIGHT

69. Killingbeck interview; also Parten interview, June 16, 1983.
70. Killingbeck interview.
71. Parten interview, June 16, 1983.
72. NASA, *Shuttle Operations Manual*, pp. 4.24–4.25.
73. NASA, *Shuttle Operations Manual*, p. 4.28.
74. Poupard, "Redundant Computer," p. 23.
75. NASA, *Shuttle Operations Manual*, pp. 4.28–4.29.
76. NASA, *Overview Workbook*, p. 6; NASA, *Shuttle Operations Manual*, p. 4.22.
77. Garman interview, June 1, 1983; Killingbeck interview.
78. Mann interview, June 8, 1983.
79. Interview with Kyle Rone, IBM, Johnson Space Center, June 3, 1983.
80. Killingbeck interview.
81. Interview with John Aaron, Johnson Space Center, June 17, 1983.
82. Interview with William Sullivan, Johnson Space Center, June 14, 1983.
83. Parten interview, June 3, 1983.
84. Sullivan interview.
85. Parten interview, June 3, 1983.
86. Mann interview, June 8, 1983.
87. Parten interview, June 16, 1983.
88. Parten interview, June 16, 1983.
89. Interview with Anthony Macina, IBM, Houston, TX, June 7, 1983.

90. There are widely disparate estimates of how many people actually contributed to the shuttle software. Macina of IBM says 275, but I think he means coders. John Aaron of NASA, head of Spacecraft Software in 1983, estimates 900 contractors and 90 civil servants. Parten said 2,000 but that may include everyone in all contracting organizations working on hardware and software. The figure of 1,000 seems reasonable for software developers, as it is consistent with similar projects.
91. Frederick Brooks, *The Mythical Man-Month*, Addison-Wesley, Reading, MA, 1975).
92. Parten interview, June 16, 1983.
93. Aaron interview; Sullivan interview.
94. NASA, *Development Specification*, Volume Five, Book 1, pp. 3.72-3.75.
95. Sullivan interview.
96. Parten interview, June 3, 1985.
97. M. Hamilton and S. Zeldin, *Higher Order Software Requirements*, MIT Draper Laboratory, Cambridge, MA, August 1973.
98. An example would be Draper Lab's, *Space Shuttle On-Orbit Flight Control Software Requirements*, December 1975.
99. Sullivan interview.
100. Rone interview.
101. Parten interview, June 3, 1983.
102. Killingbeck interview.
103. Sullivan interview.
104. W.A. Madden and K.Y. Rone, *Design, Development, Integration: Space Shuttle Primary Flight Software System*, IBM Federal Systems Division, Houston, TX 1980, p. 36; also reprinted in the *Commun. ACM*, 27, (9), 914-925 (September 1984).
105. Aaron interview.
106. NASA, *Overview Workbook*, p. 52.

326 COMPUTERS IN SPACEFLIGHT

107. Killingbeck interview.

108. Mann interview, June 8, 1983.

109. NASA, *Space Shuttle Operations*, p. 4.32.

110. Macina interview.

111. IBM, *Space Shuttle Model AP-101*, 2.22.

112. F.H. Martin, "HAL/S The Avionics Programming System for the Shuttle," AIAA, 315 (1977).

113. IBM, *Space Shuttle Orbiter Avionics Software Interface Control Document, HAL/S/FCOS*, version 5.

114. Macina interview.

115. Killingbeck interview.

116. Garman interview, June 1, 1983.

117. R.F. Thompson, "Minutes of Shuttle System Software Review," Memorandum, Johnson Space Center, Houston, TX, May 29, 1974, JSC History Office.

118. Killingbeck interview.

119. Macina, "Space Shuttle," Part One, p. 30.

120. David Gifford and Alfred Spector, ed., "Case Study: The Space Shuttle Primary Computer System," *Commun. ACM*, 27, 881 (September 1984).

121. Boehm, *Information Processing Implications*, p. 35.

122. Gifford and Spector, "Case Study," p. 885.

123. J.R. Garman, *Managing the Software Development for the Space Shuttle Orbiter*, Johnson Space Center, Houston, TX, December 9, 1981, JSC History Office, p. 10.

124. Garman interview, June 1, 1983.

125. Sullivan interview.

126. Macina interview.
127. IBM, *SDL Requirements Document. Pt. 2—Hardware Configuration*, May 31, 1974, JSC History Office, p. 2.3.
128. C.C. Kraft jr. "Automatic Data Processing Equipment (ADPE) Acquisition Plan for the Software Production Facility (SPF)," Memorandum, Johnson Space Center, Houston, TX, April 18, 1980, JSC History Office.
129. Garman, *Managing the Software Development*, p. 6.
130. Aaron interview.
131. Sullivan interview.
132. J.R. Garman, "Software Production Facility: Management Summary—Concepts and Schedule Status," NASA Data Systems and Analysis Directorate, Spacecraft Software Division, February 10, 1981, p. 12.
133. Sullivan interview.
134. Macina interview.
135. A.J. Macina, *Independent Verification and Validation Testing of the Space Shuttle Primary Flight Software System*, IBM, Houston, TX, December 1980, p. 4.
136. Macina, *Independent Verification and Validation Testing*, p. 8.
137. See IBM, *Flight Software Integrated Test Plan, Volume II*, #77-SS-3622 for examples of test cases.
138. Macina, "Space Shuttle," Part 2, p. 14.
139. Gifford and Spector, "Case Study," p. 884; Hamilton and Zeldin, *Software Requirements*, p. 39.
140. Macina, *Independent Verification and Validation Testing*, p. 8.
141. Macina, *Independent Verification and Validation Testing*, p. 8.
142. Macina interview.
143. Macina interview; Mann interview, June 8, 1983.

328 COMPUTERS IN SPACEFLIGHT

144. Garman, *Managing the Software Development*, p. 3.
145. Macina interview.
146. Parten interview, June 16, 1983.
147. Mann interview, June 8, 1983.
148. Mann interview, June 8, 1983.
149. Mann interview, June 8, 1983.
150. Parten interview, June 16, 1983.
151. See NASA, *OFTIS-19 Program Notes and Waivers*, Houston, November 1983.
152. Macina interview.
153. Mann interview, June 8, 1983.
154. Gifford and Spector, "Case Study," p. 893.
155. Garman, "Managing the Software Development," p. 5.
156. Mann interview, June 8, 1983.
157. Telephone interview with John Young, Johnson Spaceflight Center, March 6, 1984.
158. Interview with Frank Hughes, Johnson Space Center, June 2, 1983.
159. Aaron interview.
160. Hughes interview.
161. Interview with Vance Brand, Johnson Space Center, June 2, 1983.
162. Interview with Henry Hartsfield, Johnson Space Center, June 2, 1983.
163. Young interview; Brand interview; Interview with Terry Hart, Johnson Space Center, June 10, 1983.
164. Hart interview.

165. "Main Engine Controller Location," undated charts, no author, obtained from Russ Mattox, Marshall Space Flight Center.

166. Interview with Walt Mitchell, Marshall Space Flight Center, June 23, 1983.

167. R.M. Mattox and J.B. White, *Space Shuttle Main Engine Controller*, NASA Technical Paper 1932, Marshall Space Flight Center, Huntsville, AL, November 1981, p. 1.

168. "Main Engine Controller".

169. Interview with Russ Mattox, Marshall Space Center, June 23, 1983.

170. Mattox and White, *Space Shuttle Main Engine*, p. 6.

171. Mattox and White, *Space Shuttle Main Engine*, pp. 9, 11.

172. Mattox and White, *Space Shuttle Main Engine*, p. 5.

173. Mattox interview.

174. Mattox and White, *Space Shuttle Main Engine*, p. 23.

175. Mitchell interview.

176. Mattox and White, *Space Shuttle Main Engine*, p. 31.

177. Mattox and White, *Space Shuttle Main Engine*, p. 12.

178. Mitchell interview.

179. Mattox interview.

180. Mitchell interview.

181. Mattox and White, *Space Shuttle Main Engine*, p. 29.

182. Mattox interview.

183. Telephone interview with Russ Mattox, Marshall Space Center, November 16, 1984.

184. Telephone interview with Gary K. Raines, Johnson Spaceflight Center, November 1, 1985.

Chapter Five

1. J.R. Casani, A.G. Conrad, and R.A. Neilson, "Mariner 4 - A Point of Departure," *Astronaut. Aeronaut.*, 16-24, (August 1965).
2. Interview with Richard Malm, Jet Propulsion Laboratory, May 31, 1984.
3. Malm interview.
4. Casani et al, "Mariner 4—A Point of Departure," p. 17.
5. Jet Propulsion Laboratory, *The Mariner R Project Progress Report, September 1, 1962 to January 3, 1963*, Tech. Rep. no. 32-422, Vol. 1, Pasadena, CA, 1963.
6. H.K. Bouvier, R.G. Farney, and S.Z. Szirmay, "Mariner 4 Maneuver and Attitude Control," *Astronaut. Aeronaut.*, 38 (October 1965).
7. Casani et al, "Mariner 4—A Point of Departure," p. 17.
8. Jet Propulsion Laboratory, *Mariner-Venus 1962: Final Project Report*, NASA SP-59, Washington, D.C., 1965.
9. Jet Propulsion Laboratory, *Mariner Mars 1964 Project Report: Spacecraft Performance and Analysis*, Tech. Rep. no. 32-882, Pasadena, CA, February 15, 1967.
10. N.H. Herman and U.S. Lingon, "Mariner 4 Timing and Sequencing." *Astronaut. Aeronaut.*, 40 (October 1965).
11. Interview with Edward Greenberg, Jet Propulsion Laboratory, May 30, 1984.
12. Greenberg interview.
13. J.R. Scull, "Mariner Mars 1969 Navigation, Guidance, and Control," Jet Propulsion Lab, Pasadena, CA, 1970; A.J. Aukstikalnis, "Spacecraft Computers: A Survey," *Astronaut. Aeronaut.*, 33 (July-August 1974); C.R. Koppes, *The Jet Propulsion Laboratory and the American Space Program*, Yale University Press, New Haven, CT, 1982.
14. Malm interview.
15. Jet Propulsion Laboratory, *Mariner Mars 1969 Final Project Report: Development, Design, and Test*, Volume 1, Tech. Rep. no. 32-1460, Pasadena, CA, November 1, 1970.

16. Jet Propulsion Laboratory, *Development, Design, and Test*, p. 325.
17. Greenberg interview.
18. J.R. Scull, "Mariner Mars 1969 Navigation, Guidance, and Control," Jet Propulsion Lab, Pasadena, CA, 1970.
19. Interview with Don Johnson, Jet Propulsion Laboratory, May 16, 1984.
20. Greenberg interview.
21. Jet Propulsion Laboratory, *Development, Design, and Test*, p. 328.
22. A.J. Hooke, "In Flight Utilization of the Mariner 10 Spacecraft Computer," *J. Br. Interplanet. Soc.*, 29, 277 (April 1976).
23. Hooke, "Mariner 10 Spacecraft Computer," p. 277; Jet Propulsion Laboratory, *Mariner Mars 1969 Final Project Report: Development, Design, and Test*, p. 328.
24. Greenberg interview.
25. After the example given by A.J. Hooke, "The 1973 Mariner Mission to Venus and Mercury," *Spaceflight*, 30 (January, 1974).
26. Greenberg interview.
27. J.A. Gleason, *Mariner Mars 1971 Space Flight Operations Plan: Mission Operation Specifications and Constraints*, Jet Propulsion Laboratory, Pasadena, CA, April 6, 1971, Vol. 4, pp. 2, 112-113.
28. Hooke, "Mariner 10 Spacecraft Computer," p. 275.
29. C.E. Kohlhasse, H.W. Norris, H.M. Shurmeier, and J.A. Stallkamp, "The 1969 Mariner Mission to Mars," *Astronaut. Aeronaut.*, 84 (July, 1969).
30. Jet Propulsion Laboratory, *Development and Testing of the Central Computer and Sequencer for the Mariner Mars 1971 Spacecraft*, Tech. Rep. no. 33-501, October 15, 1971, pp. 2-3.
31. Hooke, "Mariner 10 Spacecraft Computer," p. 273.
32. Hooke, "Mariner 10 Spacecraft Computer," pp. 279-81.
33. Hooke, "Mariner 10 Spacecraft Computer," p. 285.

332 COMPUTERS IN SPACEFLIGHT

34. Jet Propulsion Laboratory, *Development and Testing of the Central Computer and Sequencer for the Mariner Mars 1971 Spacecraft*, p. 12.
35. A.J. Hooke, "Mariner Mission to Venus and Mercury," p. 29.
36. Jet Propulsion Laboratory, *Development and Testing of the Central Computer and Sequencer for the Mariner Mars 1971 Spacecraft*, p. 4.
37. Hooke, "Mariner 10 Spacecraft Computer," p. 285.
38. A. Avizienis, F.P. Mathur, D. A. Rennels, and J.A. Rohr, "Automatic Maintenance of Aerospace Computers and Spacecraft Information and Control Systems," Tech. Rep. no. 32-1449, Jet Propulsion Laboratory, Pasadena, CA, 1969, p. 1.
39. Avizienis et al., "Automatic Maintenance of Aerospace Computers," pp. 10–11.
40. Avizienis et al., "Automatic Maintenance of Aerospace Computers," p. 10.
41. A. Avizienis, "Design Methods for Fault-Tolerant Navigation Computers," Tech. Rep. no. 32-1409, Jet Propulsion Laboratory, Pasadena, CA, October 15, 1969, p. 4.
42. Avizienis et al., "Automatic Maintenance of Aerospace Computers," p. 9.
43. A. Avizienis, *An Experimental Self-Repairing Computer*, NASA-TR-32-1356, Jet Propulsion Laboratory, Pasadena, CA, 1968, p. E30.
44. Avizienis et al., "Automatic Maintenance of Aerospace Computers," p. 10.
45. Avizienis, *An Experimental Self-Repairing Computer*, p. E31.
46. Avizienis, *An Experimental Self-Repairing Computer*, p. E32.
47. A. Avizienis, "The STAR Computer: A Self-Testing-and-Repairing Computer for Spacecraft Guidance, Control, and Automatic Maintenance," in *The Application of Digital Computers to Guidance and Control*, AGARD, London, June 2, 1970, p. 17.1.
48. Avizienis, *An Experimental Self-Repairing Computer*, p. E31.
49. Avizienis et al., "Automatic Maintenance of Aerospace Computers," p. 3; Avizienis, "Design Methods for Fault-Tolerant Navigation Computers," p. 7.
50. Avizienis et al., "The STAR (Self-Testing and Repairing) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design," *IEEE Trans. Comput.*, C(1971), p.1316.

51. Avizienis et al., "Automatic Maintenance of Aerospace Computers," p. 2.
52. Avizienis, *An Experimental Self-Repairing Computer*, p. E33.
53. Avizienis et al., "Automatic Maintenance of Aerospace Computers," p. 6.
54. Avizienis et al., "The STAR (Self-Testing and Repairing) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design," p. 1314.
55. Avizienis et al., "Automatic Maintenance of Aerospace Computers," p. 7.
56. Avizienis, "The STAR Computer: A Self-Testing-and-Repairing Computer for Spacecraft Guidance, Control, and Automatic Maintenance," p. 17.8.
57. Avizienis, "Design Methods for Fault-Tolerant Navigation Computers," p. 5.
58. Avizienis, "The STAR Computer: A Self-Testing-and-Repairing Computer for Spacecraft Guidance, Control, and Automatic Maintenance," p. 17.8.
59. Avizienis, "The STAR Computer: A Self-Testing-and-Repairing Computer for Spacecraft Guidance, Control, and Automatic Maintenance," p. 17.8.
60. Avizienis et al., "The STAR (Self-Testing and Repairing) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design," p. 1320.
61. "A Star is Born," *Time* (December 7, 1970).
62. Carole McKelvey, "JPL's STAR Computer is Out of This World," *Glendale News-Press*, 3-594 (May 29, 1971).
63. H. Hecht, "Fault-Tolerant Computers for Spacecraft," *J. Spacecr. Rockets*, 14, 579-586 (1977).
64. Avizienis, "The STAR Computer: A Self-Testing-and-Repairing Computer for Spacecraft Guidance, Control, and Automatic Maintenance," p. 17.8.
65. Interview with William Stewart, Goddard Spaceflight Center, July 10, 1984.
66. J. Rhea, "Space Electronics: Electronics Research Center is Focal Point of Future Efforts," *Aerospace Tech.*, 54 (November 20, 1967).
67. Greenberg interview.
68. Samuel K. Deese, letter to the author, June 3, 1985.

334 COMPUTERS IN SPACEFLIGHT

69. The complete story of the Viking project can be found in Edward and Linda Ezell's history, *On Mars: Exploration of the Red Planet: 1958–1978*, NASA SP-4212, Washington, D.C., 1984.

70. Greenberg interview.

71. Greenberg interview.

72. W.H. Kohl, *Functional Requirement Viking Orbiter 1975 Flight Equipment Computer Command Subsystem Hardware*, No. V075-4-2005-2A, Pasadena, CA Jet Propulsion Lab, Jan 17, 1975, p. 20.

73. Greenberg interview.

74. Kohl, *Functional Requirement Viking Orbiter 1975 Flight Equipment Computer Command Subsystem Hardware*, p. 20; Kohl, *Functional Requirement Viking Orbiter 1975 Flight Equipment Computer Command Subsystem Software*, No. 075-4-2005-2A, Pasadena, CA Jet Propulsion Lab, January 17, 1975, p. 7.

75. Kohl, *Functional Requirement Viking Orbiter 1975 Flight Equipment Computer Command Subsystem Hardware*, p. 13.

76. R.A. Proud et al., *Viking 75 Orbiter Computer Command Subsystem Flight Software Design Description*, 612–28 (DRL Line Item No. N4-SE24), Vol. 1, Pasadena, CA Jet Propulsion Lab, November 20, 1974, pp. A.1.21–22; Kohl, *Functional Requirement Viking Orbiter 1975 Flight Equipment Computer Command Subsystem Software*, p. 30.

77. Kohl, *Functional Requirement Viking Orbiter 1975 Flight Equipment Computer Command Subsystem Hardware*, p. 13.

78. Kohl, *Functional Requirement Viking Orbiter 1975 Flight Equipment Computer Command Subsystem Hardware*, p. 5.

79. Greenberg interview.

80. Proud et al., *Viking 75 Orbiter Computer Command Subsystem Flight Software Design Description*, p. A.1.21.

81. Proud et al., *Viking 75 Orbiter Computer Command Subsystem Flight Software Design Description*, p. 3.15.

82. Kohl, *Functional Requirement Viking Orbiter 1975 Flight Equipment Computer Command Subsystem Software*, p. 16.

83. Proud et al., *Viking 75 Orbiter Computer Command Subsystem Flight Software Design Description*, p. 3.125.

84. Proud et al., *Viking 75 Orbiter Computer Command Subsystem Flight Software Design Description*, p. 3.165.
85. Proud et al., *Viking 75 Orbiter Computer Command Subsystem Flight Software Design Description*, pp. 228–234.
86. Kohl, *Functional Requirement Viking Orbiter 1975 Flight Equipment Computer Command Subsystem Software*, pp. 31–33 contains a complete list of the instructions.
87. Proud et al., *Viking 75 Orbiter Computer Command Subsystem Flight Software Design Description*, p. 3.6.
88. Kohl, *Functional Requirement Viking Orbiter 1975 Flight Equipment Computer Command Subsystem Hardware*; Kohl, *Functional Requirement Viking Orbiter 1975 Flight Equipment Computer Command Subsystem Software*.
89. Proud et al., *Viking 75 Orbiter Computer Command Subsystem Flight Software Design Description*, p. 4.2.
90. Proud et al., *Viking 75 Orbiter Computer Command Subsystem Flight Software Design Description*, p. 4.1.
91. Proud et al., *Viking 75 Orbiter Computer Command Subsystem Flight Software Design Description*, p.4.4
92. Proud et al., *Viking 75 Orbiter Computer Command Subsystem Flight Software Design Description*, p. 5.7.
93. Martin-Marietta Corp., *Viking Software Data*, Rome Air Development Center, TR-77-168, p. 107.
94. B.A. Claussen II and R.E. Wachs, "Software First: Our Viking Experience and Continuing Research," *Proceedings of the 1977 Summer Computer Simulation Conference*, New York: AIAA, 1977, p. 108.
- 95) Martin-Marietta Corp., p. 107.
96. Martin-Marietta Corp., p. 111.
97. Claussen and Wachs, p. 108.
98. An excellent introduction to microcode is contained in D.A. Patterson, "Microprogramming", *Scientific American*, March, 1983.
99. Martin-Marietta Corp., p. 118.

336 COMPUTERS IN SPACEFLIGHT

100. Martin-Marietta Corp., p. 107.
101. Claussen and Wachs, p. 108.
102. Martin Marietta Corp., p. 106.
103. Claussen and Wachs, p. 108.
104. Martin–Marietta Corp., pp. 131–132.
105. Martin-Marietta Corp., p. 110.
106. Martin-Marietta Corp., p. 111.
107. Martin-Marietta Corp., p. 114.
108. Martin-Marietta Corp., pp. 150, 154.
109. Martin-Marietta Corp., p. 104.
110. Martin-Marietta Corp., p. 149.
111. RADC-TR-77-168, May, 1977.

Chapter Six

1. See General Electric Missile and Space Division, "Final Report, Voyager Spacecraft, "Volume 1, Volume IV, Book 3, Volume VII, Book 3; October 16, 1967.
2. E. Greenberg to T. Gottlieb, "A Proposal for the Development of a New Spacecraft System Concept," December 8, 1972. From the files of E. Greenberg.
3. Interview with Edward H. Kopf, Jet Propulsion Laboratory, May 18, 1984.
4. E.C. Litty and R.D. Rasmussen, "A Voyager Attitude Control Perspective on Fault Tolerant Systems," AIAA Paper 81-1812, 1981.
5. F.T. Surber to H.M. Schurmeier, "Clarification of Spacecraft Memory Redundancy Policies," July 28, 1975, Voyager files, Jet Propulsion Laboratory Record Center.
6. K. Frewing to R. Draper, "MJS77 Software System Engineer," February 20, 1974, Voyager file, JPL Record Center.

7. K. Frewing to multiple addressees, "MJS77 On-board Software Design Team," May 10, 1974, Voyager file, JPL Record Center.
8. Raymond L. Heacock, memo to the author, May, 1985; Mr. De Jesus provided a tour of the facility when I visited JPL in 1984.
9. J.N. Bryden, memo to distribution, November 28, 1973, Jet Propulsion Laboratory Records Center.
10. S. Lingon, "Functional Requirement Mariner Jupiter/Saturn 1977 Flight Equipment-Computer Command Subsystem Hardware," No. MJS77-4-2005-1B, Pasadena, CA Jet Propulsion Lab, August 16, 1977.
11. Interview with Roy Otamura, Jet Propulsion Laboratory, 29 May 1984.
12. Interview with Dick Rice, Jet Propulsion Laboratory, 29 May 1984.
13. Lingon, "Equipment-Computer Command Subsystem Hardware," p. 35; U. S. Lingon et al., "Voyager Computer Command Subsystem Flight Software Design Description," Rev. G, Vol. I, No. 618-235, Pasadena, CA Jet Propulsion Laboratory, July 1983, p. 3.141.
14. U. S. Lingon et al., "Voyager Computer Command Subsystem Flight Software Design Description," pp. 3.30-38.
15. D. Linick to Laeser, R.P., "Preliminary List of MOS Concerns About Spacecraft Software Design." August 21, 1975, From the files of R.J. Rice, Jet Propulsion Laboratory.
16. Otamura interview.
17. Otamura interview.
18. Otamura interview.
19. Interview with Ted Kopf, Jet Propulsion Laboratory, May 18, 1984.
20. Interview with William Charlen, Jet Propulsion Laboratory, May 18, 1984.
21. Kopf interview, May 18, 1984.
22. E.H. Kopf and L.S. Smith, "The Development and Demonstration of Hybrid Programmable Attitude Control Electronics with Adaptable Analog/Digital Design Approach," *AGARD Real-Time Computer-Based Systems*.; Kopf interview, May 18, 1984.
23. Kopf interview, May 18, 1984.

338 COMPUTERS IN SPACEFLIGHT

24. Kopf interview, May 18, 1984.
25. Charlan interview.
26. Interview with Edward Greenberg, Jet Propulsion Laboratory, May 30, 1984.
27. Telephone interview with Wayne Kohl, Jet Propulsion Laboratory, January 31, 1985.
28. Kopf interview.
29. Telephone interview with Sam Deese, Jet Propulsion Laboratory, January 31, 1985.
30. Raymond L. Heacock, comments on the draft of this chapter, July 23, 1985.
31. Kopf interview, May 18, 1984.
32. Kopf interview, May 18, 1984.
33. Jet Propulsion Laboratory, *Voyager Guidance and Control Functional Description and Block Diagrams*, 618-623, August 26, 1980.
34. K. Frewing to multiple addresses, "AACS Fault Protection Proposal," June 16, 1975, Voyager file, Jet Propulsion Laboratory Record Center.
35. Kopf interview, May 18, 1984.
36. Litty and Rasmussen, p. 246.
37. Kopf interview, May 18, 1984.
38. Kopf interview, January 31, 1985.
39. Kopf interview, May 18, 1984.
40. Interview with John Wooddell, Jet Propulsion Laboratory, May 21 1984.
41. B.D. Martin, "Data Systems for 12-Year Missions," *Astronaut. Aeronaut.*, September 1970, p. 58.
42. Interview with Don Johnson, Jet Propulsion laboratory, May 16 1984; Undated at the time, Wooddell thinks he wrote the "Design of a CMOS Processor for Use in the Flight Data Subsystem of a Deep Space Probe" in 1974, which makes sense as the development process lasted from 1972 to 1974.

43. J. Wooddell, "Design of a CMOS Processor," pp. 2-3, files of J. Wooddell.
- 44) J. Morecroft to K. Frewing, "FDS Programming," January 14, 1975, files of R.J. Rice, Jet Propulsion Lab.
45. Rice interview.
46. Wooddell, "Design of a CMOS Processor," pp. 8-9.
47. Wooddell, "Design of a CMOS Processor," pp. 14-15.
48. Wooddell interview.
49. D. Johnson interview.
50. Rice interview.
51. Wooddell, "Design of a CMOS Processor," p. 9-11.
52. Interview with John Morecroft, Jet Propulsion Laboratory, 29 May, 1984.
53. D. Johnson interview.
54. J. Wooddell to multiple addressees, "FDS Processor Changes Allowing Better Utilization of the Added Memory," June 6, 1975, files of R.J. Rice, Jet Propulsion Lab.
55. Wooddell, "Design of a CMOS Processor," p. 16.
56. J. Wooddell to multiple addressees, "MJS FDS Processor Architecture and Instruction Set," October 7, 1974, files of R.J. Rice, Jet Propulsion Lab.
57. R. DeSantis, "Functional Requirement Mariner Jupiter/Saturn 1977 Flight Equipment Flight Data Subsystem Hardware," Number MJS77-4-2006-1A, Pasadena, CA Jet Propulsion Laboratory, March 20, 1978, p. 13.
58. DeSantis, "Flight Data Subsystem Hardware," p. 44.
59. Wooddell interview.
60. Wooddell, "Design of a CMOS Processor," p. 5.
61. Wooddell, "MJS FDS Processor Architecture and Instruction Set"; D. Johnson interview.
62. Wooddell interview.

340 COMPUTERS IN SPACEFLIGHT

63. Rice interview.
64. D. Johnson interview.
65. Rice interview.
66. Heacock, notes on the draft.
67. Morecroft interview.
68. H.M. Schurmeier to multiple addresses, "MJS77 Spacecraft Memory Usage, MJS77 FDS Program Development Priorities," June 23, 1975, From files of R.J. Rice, Jet Propulsion Laboratory.
69. K. Frewing to T. Sorenson, "Response to FDS Flight Software Design Review Action Items," March 17, 1976, Voyager file, JPL Record Center.
70. Wooddell, "MJS FDS Processor Architecture and Instruction Set."
71. DeSantis, "Flight Data Subsystem Hardware," pp. 4, 6.
72. Morecroft interview.
73. D. Johnson interview.
74. Otamura interview.
75. Heacock, notes on the draft.
76. Interview with Gentry Lee, Jet Propulsion Laboratory, June 1, 1984.
77. Kopf interview, May 18, 1984.
78. Charlan interview; Kopf interview, May 18, 1984.
79. C.P. Jones and M.R. Landano, "The Galileo Spacecraft System Design," *Proceedings of the AIAA 21st Aerospace Science Meeting*, Reno, Nevada, January 10-13, 1983, p. 19.
80. Jones and Landano, "Galileo Spacecraft System Design," p. 17.
81. Kohl interview.
82. D. Johnson interview.

83. Kopf interview, May 18, 1984.
84. B. Larman, *Functional Requirement Galileo Orbiter Data System Intercommunication Requirements*, GLL-3-270, Rev. A, Pasadena, CA Jet Propulsion Lab, March 27, 1981, p. 35.
85. Larman, *Functional Requirement Galileo Orbiter Data System Intercommunication Requirements*, p. 37.
86. Larman, *Functional Requirement Galileo Orbiter Data System Intercommunication Requirements*, p.34.
87. Telephone Interview with Bill Tindal, Washington, D.C., August 10, 1984.
88. D.A. Rennels, B. Riis-Vestergaard, and V.C. Tyree, "The Unified Data System: A Distributed Processing Network for Control and Data Handling on a Spacecraft," *NAECON '76 Record*, New York: IEEE, 1976, p. 283.
89. Greenberg interview; Rennels et al., "Unified Data System," p. 283.
90. D.A. Rennels, "Reconfigurable Modular Computer Networks for Spacecraft On-board Processing," *Computer*, July 1978, p.57.
91. Rennels et al., "Unified Data System," p.284.
92. Rennels, "Reconfigurable Modular Computer Networks," p. 54.
93. Rennels, "Reconfigurable Modular Computer Networks, p. 49.
94. P. Lecoq and F. Lesh, "Software Techniques for a Distributed Real-Time Processing System," *NAECON '76 Record*, New York: IEEE, 1976, p. 291.
95. Rennels et al., "Unified Data System" p. 285.
96. Kopf interview.
97. P.H. Stakem, "One Step Forward - Three Steps Backup: Computing in the U.S. Space Program," *Byte*, September 1981, p. 118.
98. Kohl, *Galileo Orbiter Flight Equipment*, p. 144.
99. Stakem, "One Step Forward - Three Steps Backup," p. 132.
100. Interview with John Zipse, Jet Propulsion Laboratory, May 22, 1984.

342 COMPUTERS IN SPACEFLIGHT

101. D. Johnson interview.

102. D. Johnson interview.

103. Jones and Landano, "Galileo Spacecraft System Design," p.11.

104. Jones and Landano, "Galileo Spacecraft System Design," p. 12.

105. Zipse interview.

106. Zipse Interview.

107. Jones and Landano, "Galileo Spacecraft System Design," p. 12.

108. W.H. Kohl, *Functional Requirement Galileo Orbiter Flight Equipment Command and Data Subsystem*, No. GGL-4-2006, Pasadena, CA., Jet Propulsion Laboratory, March 5, 1981, pp. 101–102,

109. Zipse interview; O.W. Adams, *Project Galileo: Software Requirements Document, Command and Data Subsystem*, 625-340-006000, Rev. A, Jet Propulsion Laboratory, Pasadena, CA, January 16, 1984, p. 13.

110. Zipse interview.

111. Kohl, *Galileo Orbiter Flight Equipment*, pp. 89–90.

112. Zipse interview.

113. Kohl, *Galileo Orbiter Flight Equipment*, p. 48.

114. D. Johnson interview.

115. O.W. Adams, *Project Galileo: Software Requirements Document Command and Data Subsystem*, 625-340-006000, Rev. A, Jet Propulsion Laboratory, Pasadena, CA, Jan 16, 1984, pp. 144–145.

116. Adams, *Project Galileo: Software Requirements*, p.215.

117. C. Chadwick to W.J. O'Neil, "Some Thoughts on the HAL/S System," June 19, 1978, Galileo file, Jet Propulsion Laboratory Record Center.

118. Adams, *Project Galileo: Software Requirements*, p. 12.

119. Henry Kleine, "Software Design and Development Language," Jet Propulsion Laboratory Publication 77-24, Rev. 1, August 1, 1979.

120. C. A. Ericson, *Apollo Logic Diagram Analysis Guideline*, Boeing Co., Seattle, WA, 1967, pp. 66.

121. J.T. Buchman, *Galileo: General Design Document, Attitude and Articulation Control Subsystem*, 625-350-007000, Jet Propulsion Laboratory, Pasadena, CA, October 1, 1983.

122. H.K. Bouvier and G.D. Pace, "Management of the Galileo Attitude and Articulation Control Flight Software Development," *3rd Computers in Aerospace Conference*, AIAA, San Diego, CA, October 26-28, 1981, No. 81-2127, pp. 112-118.

123. Charlan interview.

124. Bouvier and Pace, "Galileo Attitude and Articulation Control Flight," pp. 112-118.

125. Kopf interview.

126. Kopf interview.

127. ITEK, "ATAC—16M Principles of Operation," June 1979, p. 2.60.

128. Kopf interview.

129. ITEK, "ATAC—16M Principles of Operation," p. 1.8.

130. Bouvier and Pace, "Galileo Attitude and Articulation Control Flight," pp. 112-118.

131. E.H.Kopf, *Galileo Real-Time AACS Operating System, "Gracos" Reference Manual*, Jet Propulsion Laboratory, Pasadena, CA, September 1, 1983, p. 29.

132. Buchman, *Galileo: General Design Document*, p. 1.6.

133. Bouvier and Pace, "Galileo Attitude and Articulation Control Flight," pp. 112-118.

134. Kopf interview.

135. Kopf interview.

136. Kopf interview.

137. Buchman, *Galileo: General Design Document*, pp. 1.7, 4.1.

344 COMPUTERS IN SPACEFLIGHT

138. Kopf, "Gracos," p. 9.
139. Kopf, "Gracos," p. 25.
140. Charlan interview; Kopf interview.
141. Buchman, *Galileo: General Design Document*, p. 1.11.
142. Buchman, *Galileo: General Design Document*, p. 3.3.
143. Lee interview.
144. Washburn, *Distant Encounters*, pp. 87, 107.
145. Kopf interview.
146. Lee interview.
147. D. Johnson interview.

Chapter Seven

1. Interview with Dr. Helmut Hoelzer, Huntsville, AL, June 24, 1983.
2. Interview with Charles Swearingen, Huntsville, AL, June 21, 1983.
3. Interview with Jim Lewis, Marshall Space Center, June 20, 1983; IBM Corporation, "Apollo Study Report," October 1, 1963, p. 6.2.
4. Interview with Kyle Rone, IBM, Johnson Space Center June 3, 1983.
5. See IBM, "Apollo Study Report," Volume II, for a complete technical description.
6. IBM, "Apollo Study Report," p. 3.4.
7. IBM, "Apollo Study Report," Vol. II, p. 3.5; F.B. Moore and J.B. White, *Applications of Redundancy in the Saturn V Guidance and Control System*, NASA-TM-X-73352, Huntsville, AL, Marshall Space Flight Center, November, 1976, p. 10.
8. IBM, "Apollo Study Report," Vol. II, pp. 3.2, 3.3.

9. Swearingen interview; R. Alonso and G.C. Randa, "Flight-Computer Hardware Trends," *Astronaut. Aeronaut.*, 33-34 (April 1967).

10. Lewis interview.

11. IBM Corporation, *Saturn Launch Computer Complex Software System Description*, IBM Federal System Division, Huntsville, AL, March 14, 1972, p. 12.

12. Interview with Jim Willbanks, IBM, Kennedy Space Center, June 29, 1983.

13. J.W. Dahnke, "Computer-Directed Checkout for NASA's Biggest Booster," *Control Eng.*, 84 (August 1962); Lewis interview.

14. C.O. Brooks to multiple addressees, October 5, 1962, Marshall Space Flight Center.

15. R. Dutton and W. Jafferis, *Utilization of Saturn/Apollo Control and Checkout System for Prelaunch Checkout and Launch Operations*, NASA-TM-X-65271, Kennedy Space Center, Cocoa Beach, FL, July 22, 1968, p. 5.2.

16. Swearingen interview.

17. Willbanks interview.

18. Interview with Thomas S. Walton, Kennedy Space Center, July 6, 1983.

19. Walton interview.

20. Walton interview.

21. Walton interview.

22. Walton interview.

23. W.E. Parsons et al., "PACE: Preflight Acceptance Checkout Equipment," *Astronaut. Aerospace Eng.*, 52 (July 1963).

24. Walton interview.

25. Dutton and Jafferis, p. 3.40.

26. A. Laats and J.E. Miller, *Apollo Guidance and Control System Flight Experience*, NASA-CR-101823, MIT, Cambridge, MA, June, 1969, pp. 1,3.

346 COMPUTERS IN SPACEFLIGHT

27. Interview with Bruce Miller, Kennedy Space Center, July 5, 1983.
28. Interview with James E. Deming, Kennedy Space Center, July 6, 1983.
29. Miller interview.
30. Miller interview.
31. W.O. Robeson to Mr. Siefert, Deputy Director of NASA Kennedy Space Center, "Optimum Computer Support to NASA," IBM Corp., Cape Canaveral, FL, July 11, 1965, Kennedy Archives.
32. Deming interview.
33. IBM Corporation, *Saturn Software Systems Development Study*, December 8, 1972, p. 5.1.
34. IBM Corporation, *Saturn Software Systems*, p.5.2.
35. Deming interview.
36. IBM Corporation, *Saturn Software Systems*, December 8, 1972.
37. IBM Corporation, *Saturn Software Systems*, p. 4.16.
38. IBM Corporation, *Saturn Software Systems*, pp. 4.18, 2.3.
39. James J. Hart, "Analysis of Apollo Launch Operating Experience" in Space Shuttle Technology Conference, NASA, Kennedy Space Center, TR-1113, May 3, 1971, p. 15.
40. Hart, "Analysis of Apollo Launch Operating Experience," p. 26.
41. Interview with Pam Biegert, Kennedy Space Center, June 30, 1983.
42. Deming interview.
43. H.C. Paul, "A Standard Language for Test and Ground Operations," *Proceedings of the Space Shuttle Integrated Electronics Conference*, 2:355-358, Kennedy Space Center, Cocoa Beach, FL, p. 358.
44. H.C. Paul, "Launch Processing System Transition from Development to Operation," *Proceedings of the 14th Space Congress, Cocoa Beach, FL, 27-29 April 1977*, pp. 7.1-7.3, Canaveral Council of Technical Societies, Cocoa Beach, FL, 1977, p. 7.3.

45. Interview with Al Parrish, Kennedy Space Center, June 28, 1983.
46. *Space Shuttle Launch Operations Center Study*, NASA-TM-X-67292, Kennedy Space Center, Cocoa Beach, FL, November 4, 1970, pp. 6.5, 6.11.
47. A.B. Sloan, "Vandenberg Planning for the Space Transportation System," *Astronaut. Aeronaut.*, 47 (November 1981).
48. Interview with R.C. Bulkley, IBM, Kennedy Space Center, June 27 and 29, 1983.
49. W.E. Parsons, "Kennedy Launch Processing System," *Technology Today and Tomorrow: Canaveral Council of Technical Societies, Proceedings of the Eighth Space Congress, Cocoa Beach, FL, 19-23 April 1971*, 1:11.31-11.40, Canaveral Council of Technical Societies, Cocoa Beach, FL, 1971, p. 11.34.
50. *Space Shuttle Launch Operations Center Study*, pp. 2.1, 4.2.
51. E.A. Dalke, "Unified Test Equipment: A Concept for the Shuttle Ground Test System," *NASA Manned Space Center Proceedings of the Space Shuttle Integrated Electronics Conference*, vol. 2, pp. 329-354, Johnson Space Center, Houston, TX, p. 330.
52. Dalke, "Unified Test Equipment," p. 348.
53. General Electric Co., "Summary Report for the Universal Control and Display Console," NASA CR-115350, July 1971.
54. Interview with Henry Paul, Kennedy Space Center, July 7, 1983.
55. Interview with Bill Bailey, Kennedy Space Center, June 30, 1983.
56. C. Covault, "Cape Shuttle Capabilities Broadened," *Aviation Week and Space Technol.*, 40 (October 13, 1975).
57. Bailey interview.
58. Interview with Frank Byrne, Kennedy Space Center, June 29, 1983; Walton interview.
59. Henry C. Paul, letter to the author, July 3, 1985.
60. Bailey interview.
61. Byrne interview, June 29, 1983.

348 COMPUTERS IN SPACEFLIGHT

62. G.W. Cunningham and D.M. Welsh, "Launch Processing System: A Groundbased Component of Space Shuttle," reprinted from *Tech. Directions*, 3 (1) (Spring 1977), IBM Federal System Division, Bethesda, MD, p. 8.
63. H.C. Paul, "Launch Processing System Transition from Development to Operation," p. 7.2.
64. Walton interview; Paul interview.
65. KSC-DD-LPS-007, October 12, 1973.
66. Interview with Fred Heddens, IBM, Kennedy Space Center, June 27 and 29, 1983.
67. H.C. Paul, "LPS—A System to Support the Space Shuttle," *Technology Today and Tomorrow: Proceedings of the Twelfth Space Congress, Cocoa Beach, FL, 9–11 April 1975*, pp. 8.3–8.7, Canaveral Council of Technical Societies, Cocoa Beach, FL, 1975, p. 8.3; Bailey interview.
68. Paul letter.
69. Cunningham and Welsh, "Launch Processing System," p. 10.
70. Bulkley interview; Bailey interview.
71. Walton interview.
72. Covault, "Shuttle Capabilities Broadened," p. 40.
73. F. Byrne, G.V. Doolittle, and R.W. Hockenberger, "Launch Processing System," *IBM J. Res. Devel.*, 75 (January 1976).
74. Interview with Al Parrish, Kennedy Space Center, June 28, 1983.
75. Cunningham and Welsh, "Launch Processing System," p. 13.
76. Heddens interview.
77. Byrne interview, June 29, 1983.
78. Covault, "Shuttle Capabilities Broadened," p. 41.
79. Byrne interview, June 29, 1983.
80. Byrne interview, June 29, 1983; Walton interview.

81. W.W. Bailey, *Launch Processing System: Concept to Reality*, Kennedy Space Center, Digital Electronics Division, p. 5.
82. Parrish interview.
83. Walton interview.
84. Paul interview.
85. Byrne interview, June 29, 1983.
86. Cunningham and Welsh, "Launch Processing System," p. 10.
87. *CCMS Hardware Project Requirements*, Pamphlet, no date, p. 189.
88. Byrne, et al., "Launch Processing System," p. 81.
89. H.C. Paul, "Launch Processing System Transition from Development to Operation," p. 7.1.
90. Paul interview; Byrne, et al., "Launch Processing System," p. 82.
91. Byrne interview, June 29, 1983.
92. Paul interview.
93. Byrne interview, June 29, 1983
94. Byrne interview, June 29, 1983
95. Byrne interview, July 8, 1983.
96. *CCMS Hardware Project Requirements*, p. 197.
97. Byrne interview, July 8, 1983.
98. Parrish interview; Byrne interview, June 29, 1983.
99. Biegert interview.
100. Parrish interview.
101. Byrne interview, June 29, 1983.

350 COMPUTERS IN SPACEFLIGHT

102. Biegert interview.

103. Parrish interview.

104. T.E. Utsman, "KSC Ground Support Operation and Equipment for the Space Transportation System," *Shuttle Propulsion Systems Proceedings of the Winter Annual Meeting, Phoenix, AZ, 14–19 November 1982*, pp. 73–77, American Society of Mechanical Engineers, New York, 1982, p. 76.

105. Bailey interview.

106. Bailey interview.

107. D.A. Springer, "The Launch Processing System for Space Shuttle," *AIAA, ASME and SAE Joint Space Mission Planning and Execution Meeting, Denver, CO, July 10–12, 1973*, p. 4.

108. Biegert interview.

109. Bailey interview.

110. Biegert interview.

111. Interview with Jane Stearns, Kennedy Space Center, June 30, 1983.

112. Stearns interview.

113. Interview with Bobby Bruckner, Kennedy Space Center June 30, 1983.

114. Biegert interview; T.E. Utsman, "KSC Ground Support Operation and Equipment for the Space Transportation System," *Shuttle Propulsion Systems Proceedings of the Winter Annual Meeting, Phoenix, AZ, 14–19 November 1982*, pp. 73–77, American Society of Mechanical Engineers, New York, 1982, p. 76.

115. Bruckner interview.

116. H.C. Paul, "Launch Processing System Transition from Development to Operation," p. 7.2.

117. Walton interview.

118. Interview with Brad Hughes, Kennedy Space Center, July 5, 1983.

119. Bailey, *Launch Processing System: Concept to Reality*, p. 7.

120. Paul interview.
121. Paul interview.
122. *GOAL On-Board Interface Language Users Packages*, KSC-LP-OP-033-4, April 27, 1983, Kennedy Library; Paul, p. 7.2.
123. H.C. Paul, "LPS—A System to Support the Space Shuttle," p. 8.4.
124. Springer, "The Launch Processing System for Space Shuttle," p. 2.
125. Interview with Jack Bogan, IBM, Kennedy Space Center, June 29, 1983.
126. *Shuttle Ground Operations Simulator Users' Reference Manual*, Revision 1, KSC-LPS-UM-073, February 1983.
127. Interview with Dr. Carl Delaune, Kennedy Space Center, July 5, 1983.
128. Delaune interview.
129. Biegert interview; IBM Corporation, *Launch Processing System Checkout, Control and Monitor Subsystem Detailed Software Design Specifications, Book 1, part 2: Control Logic Language Design*, KSC-LPS-IB-070-1, part 2, S33 release, Kennedy Documents Library, Cape Canaveral, FL, May 5, 1983, p. 1.3.
130. Bogan interview; Remer Prince, a NASA employee, wrote most of the first GOAL compiler interpreter on this machine in such a short time it paid great dividends in gaining recognition of the Kennedy effort to build the Launch Processing System (Paul letter).
131. Byrne interview, June 29, 1983.
132. Utsman, "KSC Ground Support Operation and Equipment," p. 76.
133. Interview with Bob Yarborough, Kennedy Space Center, July 6, 1983.
134. Yarborough interview.
135. Paul interview.
136. Parrish interview.
137. Delaune interview.
138. Sloan, "Vandenberg Planning," p. 44.

139. D.J. Berrier, "Vandenberg Ground Support Equipment for the Space Shuttle," *AIAA 19th Aerospace Sciences Meeting*, St. Louis, MO, January 12-15, 1981, p. 5.

Chapter Eight

1. Interview with John Morton, Goddard Spaceflight Center, June 27, 1984.
2. Frederick P. Brooks, Jr., *The Mythical Man-Month*, Addison-Wesley, Reading, MA, 1975, p. 56.
3. Morton interview, June 27, 1984.
4. S.E. James, "Evolution of Real-Time Computer Systems for Manned Spaceflight," *IBM J. Res. Devel.*, 25, 418 (1981).
5. W.R. Corliss, *Histories of the Space Tracking and Data Acquisition Network (STADAN), The Manned Space Flight Network (MSFN), and the NASA Communications Network (NASCOS)*, NASA CR-140390, September 1974. p. 29.
6. Western Electric, Inc., *Final Project Report to NASA: Progress Mercury*, NAS1-430, June 1961, p. 1.
7. Interview with James Stokes, Johnson Space Center, June 14, 1983.
8. H.R. Karp, "NASA Plans Global Range to Track Man in Space," *Control Eng.*, 22 (July 1959).
9. Stokes interview, June 14, 1983.
10. Interview with J. Perry Chambers, Goddard Spaceflight Center, June 28, 1984.
11. Chambers interview, June 28, 1984.
12. Corliss, *Space Tracking and Data Acquisition Network*, September 1974, p. 138.
13. Western Electric, Inc., *Final Project Report to NASA: Progress Mercury*, June 1961, pp. 42, 45.
14. IBM, Goddard Monitor Programs, *Project Mercury*, prepared for NASA, NAS1-430, 1961, p. 1.4, files of John Morton, Goddard Spaceflight Center; James, "Evolution of Real-Time Computer Systems," p. 424.

15. IBM, Goddard Monitor Programs, *Project Mercury*, p. 1.8.
16. IBM, Goddard Monitor Programs, *Project Mercury*, p. 3.2.
17. Chambers interview, June 28, 1984.
18. Interview with Lynwood Dunseith, Johnson Space Center, June 2, 1983.
19. Chambers interview, June 28, 1984.
20. Chambers interview, June 28, 1984.
21. IBM, Goddard Monitor Programs, *Project Mercury*, pp. 1.13, 1.18.
22. IBM, Goddard Monitor Programs, *Project Mercury*, pp. 1.15, 1.16, 3.1.
23. Chambers interview, June 28, 1984.
24. Stokes interview, June 14, 1983.
25. Chambers interview, June 28, 1984; P. Herget, *The Computation of Orbits*, Cincinnati, by the author, 1948.
26. IBM, Goddard Monitor Programs, *Project Mercury*, p. 1.2; Stokes interview, June 14, 1983; IBM Corporation, *IBM Launch Monitor Subsystem*, MS 124, May 1, 1962, pp. 1.2, 1.3; James, "Evolution of Real-Time Computer Systems for Manned Spaceflight," 25 418 (1981).
27. Chambers interview, June 28, 1984.
28. Stokes interview, June 14, 1983.
29. J. Grimwood, B.C. Hacker, and P.J. Vorzimmer, *Project Gemini: Technology and Operations*, GPO, Washington, D.C., 1969, p. 23.
30. H. W. Tindall, jr., *Consolidation of Gemini Computer Programming and Operation at Houston, Texas*, memorandum, February 28, 1962, Johnson Space Center History Office.
31. T.L. Kraft, *WDL Tech. Rep. E-167: Mission Display Study*, Philco Corp., Palo Alto, CA, June 15, 1964, p. 2.1.
32. Philco Corporation, "IMCC Systems and Performance Requirements Specification," p. 4.1.2.

354 COMPUTERS IN SPACEFLIGHT

33. Philco Corporation, "IMCC Systems and Performance Requirements," p. 4.3.5.1.
34. IBM Corporation, *Real-Time Computer Complex for NASA Manned Spacecraft Center, Houston, Texas*, IBM Federal Systems Division, Rockville, MD, September 1, 1962, pp. 1.1, 1.2, 2.5, 2.53.
35. IBM Corporation, *Project Gemini Final Report-Summary, NASA CR-72180*, no date, p. 2.
36. IBM Corporation, *Project Gemini Final Report-Summary*, no date, p. 2; IBM Corporation, *Real-Time Computer Complex for NASA Manned Spacecraft Center*, pp. 2.51, 3.5.
37. IBM Corporation, *Project Gemini Final Report-Summary*, no date, p. 39.
38. Corliss, *Space Tracking and Data Acquisition Network*, p. 145.
39. Stokes interview, June 1, 1983.
40. IBM Corporation, *Real-Time Computer Complex for NASA Manned Spacecraft Center*, p. 2.1.
41. IBM Corporation, *Real-Time Computer Complex for NASA Manned Spacecraft Center*, p. 2.44.
42. IBM Corporation, *Project Gemini Final Report-Summary*, p. 7.
43. IBM Corporation, *Project Gemini Final Report-Summary*, p. 18.
44. IBM Corporation, *Real-Time Computer Complex for NASA Manned Spacecraft Center*, p. 2.3; IBM Corporation, *Project Gemini Final Report—Summary*, p. 38.
45. James, "Evolution of Real-Time Computer Systems," p. 421.
46. IBM Corporation, *Project Gemini Final Report-Summary*, p. 36.
47. James, "Evolution of Real-Time Computer Systems," p. 422.
48. Interview with William Sullivan, Johnson Space Center, June 14, 1983.
49. IBM Corporation, *Real-Time Computer Complex for NASA Manned Spacecraft Center*, p. 2.14.
50. Interview with Dub Pollen, IBM, Johnson Space Center, June 13, 1983.

51. James, "Evolution of Real-Time Computer Systems," p. 424.
52. B.C. Hacker and J.M. Grimwood, *On the Shoulders of Titans*, NASA SP-4203, 1977, p. 386.
53. Stokes interview, June 14, 1983.
54. R.C. Seamans, *RTCC Computer Requirement for Project Apollo*, memorandum, Washington, D.C., October 7, 1965, Johnson Space Center History Office.
55. Pollen interview, June 13, 1983.
56. Stokes, "Managing the Development of Large Software Systems—Apollo Real-Time Control Center," *IEEE Proc.*, A/31–A/315, p. 1 (1970).
57. Stokes interview, June 14, 1983.
58. Dunseith interview, June 9, 1983.
59. Stokes interview, June 14, 1983.
60. Stokes, *RTCC Computer Supervisor's Report for the AS-205/CSM-101 Mission*, memorandum, November 1, 1968, Johnson Space Center History Office.
61. Stokes, *RTCC Computer Supervisor's Report for the Apollo 10 Mission*, memorandum, June 27, 1969, Johnson Space Center History Office.
62. Stokes interview, June 14, 1983.
63. James, "Evolution of Real-Time Computer Systems," p. 419.
64. Stokes interview, June 14, 1983.
65. James, "Evolution of Real-Time Computer Systems," p. 424.
66. M. Forthoffer, "A Comparison of Time-Shared vs. Batch Development of Space Software," *Proceedings of 12th International Symposium on Space Technology and Science, Tokyo, Japan, 16–20 May 1977*, National Aerospace Laboratory, Chofu, Tokyo, 1977, pp. 1056, 1057, 1059.
67. James, "Evolution of Real-Time Computer Systems," p. 419.
68. Pollen interview, June 13, 1983; Interview with Gene Campbell, IBM, Houston, TX, June 13, 1983; Interview with Fred Riddle, IBM, Johnson Space Center, June 13, 1983.

356 COMPUTERS IN SPACEFLIGHT

69. Frank Hughes (JSC) and Kristan Lattu (JPL), *A Comparative Study of the Evolution of Command and Control Activities for Manned and Unmanned Spaceflight Operations*, AIAA-83-294, given by Lattu at International Aerospace Foundation Meeting, Budapest, October 10–15, 1983, pp. 2–3.
70. Interview with Ann Merwarth, Goddard Spaceflight Center, July 3, 1984.
71. Interview with Carl Johnson, Jet Propulsion Laboratory, May 23, 1984.
72. See Corliss, *Histories of the Space Tracking and Data Acquisition Network*; N.A. Renzetti, ed., *A History of the Deep Space Network From Inception to January 1, 1969*, Jet Propulsion Laboratory TR 32-1533, September 1, 1971.
73. C.R. Gates and M.S. Johnson, "A Study of On-Site Computing and Data Processing for a World Tracking Network," *Jet Propulsion Laboratory Publication no. 154*, Jet Propulsion Laboratory, Pasadena, CA, February 9, 1959, p. 11.
74. C. Johnson interview, May 23, 1984.
75. W.D. Merrick; E. Rechtin; R. Stevens; and W.K. Victor, "Deep Space Communications," *IRE Trans. Military Electron.*, 160 (April/July 1960).
76. Interview with George Gianopolis, Jet Propulsion Laboratory, June 4, 1984.
77. NASA, *Mariner-Venus 1962: Final Project Report*, NASA SP-59, Washington, 1965, pp. 278–279, 294.
78. NASA, *Mariner-Venus 1962*, 1965, p. 294.
79. Eberhardt Rechtin, interviewed by Cargill Hall, 1970?, Hall's files in the JPL Library Vault.
80. W.H. Pickering to multiple addresses, "Establishment of the Deep Space Network," December 24, 1963, memo. no. 218.
81. B. Sparks to multiple addresses, "Deep Space network Appointments," January 23, 1964, memo. no. 128; Rechtin interview.
82. Rechtin interview.
83. NASA, *Mariner Mars 1964: Final Project Report*, NASA SP-139, Washington, D.C., p. 216.
84. Corliss, *The Interplanetary Pioneers, Volume I: Summary*, NASA SP-278, Washington, D.C., 1972, p. 85.

85. Interview with Don Royer, Jet Propulsion Laboratory, June 7, 1984.
86. NASA, *Mariner Mars 1964: Final Project Report*, pp. 224–225.
87. Jet Propulsion Laboratory, "Spaceflight Operations Facility Data Processing System," Spec. no. SFOF 372-III-310, Jet Propulsion Laboratory, Pasadena, CA, October 1, 1963, p. 3.
88. Gianopolis interview, June 4, 1984.
89. NASA, *Mariner Mars 1964: Final Project Report*, p. 220.
90. Interview with Frank Jordan, Jet Propulsion Laboratory, May 31, 1984.
91. NASA, *Mariner-Venus 1967: Final Project Report*, NASA SP-190, Washington, D.C., 1971, pp. 166, 182.
92. Jet Propulsion Laboratory, *Surveyor Final Project Report*, Volume I, Tech. Rep. no. 32-1265, Jet Propulsion Laboratory, Pasadena, CA, July 1, 1969, p. 142.
93. Jet Propulsion Laboratory, *Mariner Mars 1969 Final Project Report: Development, Design, and Test*, Vol. 1, Tech. Rep. no. 32-1460, Jet Propulsion Laboratory, Pasadena, CA, November 1, 1970, p. 556.
94. Royer interview, June 7, 1984; Jet Propulsion Laboratory, *Mariner Mars 1971 Project Final Report*, Vol. III, Tech. Rep. no. 32-1550, Jet Propulsion Laboratory, Pasadena, CA, July 1, 1973, p. 27; Interview with Richard Moulder, Jet Propulsion Laboratory, May 21, 1984.
95. Royer interview, June 7, 1984.
96. Jet Propulsion Laboratory, *The Deep Space Network*, Space Programs Summary 37–66, Vol. II, Jet Propulsion Laboratory, Pasadena, CA, November 30, 1970, p. 91.
97. Gianopolis interview, June 4, 1984.
98. Royer interview, June 7, 1984.
99. JPL, *Mariner Mars 1971 Final Report*, p. 126.
100. R. Scott to W.J. Koselka, "History of the MCIF," memorandum no. RS-75-103, Jet Propulsion Laboratory, August 27, 1975, files of R. Scott.
101. JPL, *Mariner Mars 1971 Final Report*, p. 7.

358 COMPUTERS IN SPACEFLIGHT

102. Martin Marietta Corp., *Viking Software Data*, Rome Air Development Center, TR-77-168, May 1977, pp. 5, 6.
103. Hughes and Lattu, *Evolution of Command and Control Activities*, October 10–15, 1983, pp. 5, 6.
104. Royer interview, June 7, 1984; Interview with Frank Singleton, Jet Propulsion Laboratory, May 17, 1984.
105. Carl W. Johnson, letter to the author, September 12, 1985.
106. Moulder interview, May 21, 1984; Interview with Lloyd Jennings, Jet Propulsion Laboratory, May 15, 1984.
107. G.A. Madrid and P.T. Westmoreland, "Adaptation of a Software Development Methodology to the Implementation of a Large-Scale Data Acquisition and Control System," AIAA paper 83-2412, 1983, p. 360.
108. Gianopolis interview, June 4, 1984.
109. Jet Propulsion laboratory, *Mariner Mars 1969 Final Project Report*, November 1, 1970, vol. 1, p. 554.
110. C. Johnson interview, May 23, 1984.
111. JPL, *Mariner Mars 1971 Final Report*, p. 27.
112. Martin Marietta Corp., *Viking Software Data*, p. 10.
113. Martin Marietta Corp., *Viking Software Data*, p. 23.
114. Madrid and Westmoreland, *Implementation of a Large Scale Data Acquisition and Control System*, 1983, p. 361.
115. Royer interview, June 7, 1984.

Chapter Nine

1. C.H. Woodling et al., *Apollo Experience Report: Simulation of Manned Space Flight for Crew Training*, NASA MSC-07036, 1972, p. 2.
2. Interview with John Erickson, Johnson Space Center, June 14, 1983.
3. Interview with Robert Ernulf, Johnson Space Center, June 16, 1983.

4. Woodling et al., *Apollo Experience Report*, p. 6.
5. Woodling et al., *Apollo Experience Report*, p. 47.
6. H.I. Johnson, *Facilities for Manned Spacecraft Development Simulation and Training*, NASA Fact Sheet 132, Houston, TX, Manned Spacecraft Center, February 1963, Johnson Space Center History Office, p. 4.
7. N.R. Cooper, "X-15 Flight Simulation Program," *Aerospace Engineering*, 77 (November 1961).
8. D.K. Slayton, *Apollo Computer Software Report*, memorandum, Manned Spacecraft Center, Houston, TX, June 20, 1967, Johnson Space Center History Office; Woodling et al., *Apollo Experience Report*, p. 48; Interview with Ken Mansfield, Johnson Space Center, June 1, 1983.
9. Interview with Ray Palikowsky, Singer, Johnson Space Center, June 10, 1983.
10. Mansfield interview, June 1, 1983.
11. Erickson interview, June 14, 1983.
12. A.W. Vogeley, *Piloted Spaceflight Simulation at Langley Research Center*, NASA-TM-X-59598, Langley Research Center, Hampton, VA, 1966, p. 11.
13. Woodling et al., *Apollo Experience Report*, p. 3.
14. Woodling et al., *Apollo Experience Report*, p. 50.
15. Ivan Ertel, News release, September 21, 1966, Johnson Space Center.
16. J.L. Raney, *Feasibility Study—Use of a Simulated AGC in the Apollo Trainers*, February 24, 1966, p. 4, Johnson Space Center History Office.
17. Woodling et al., *Apollo Experience Report*, p. 49.
18. Interview with James Raney, Johnson Space Center, May 31, 1983.
19. Woodling et al., *Apollo Experience Report*, p. 10.
20. Mansfield interview, June 1, 1983.
21. Raney interview, May 31, 1983.

360 COMPUTERS IN SPACEFLIGHT

22. Warren J. North, memorandum to chief, Computational Analysis Division, "Simulation of Spacecraft Guidance Computer in the CM and LEM Mission Simulators," April 5, 1966, files of James Raney.
23. Raney, *Use of a Simulated AGC In the Apollo Trainers*, February 24, 1966.
24. Raney interview, May 31, 1983; Raney, *ISCMC*, Johnson Space Center, Houston, TX, April 1, 1969, Johnson Space Center History Office, pp. 5, 11, 13, 14; Raney, *Use of a Simulated AGC In the Apollo Trainers*, February 24, 1966.
25. Interview with Stan Mann, Johnson Space Center, June 8, 1983.
26. Palikowsky interview, June 10, 1983.
27. A.C. Bond, *Training and Simulation Requirements for Future Programs*, memorandum, October 23, 1969, Johnson Space Center History Office.
28. R.F. Thompson, *Shuttle Engineering/Training Manned Simulation Program Definition*, memorandum, Manned Spacecraft Center, Houston, TX, September 9, 1970, Johnson Space Center History Office.
29. Raney interview, May 31, 1983.
30. Erickson interview, June 14, 1983.
31. Singer Corp., *Shuttle Mission Simulator Requirements Report*, vol. II, Rev. C, December 21, 1973.
32. Raney interview, May 31, 1983.
33. Raney interview, May 31, 1983.
34. Erickson interview, June 14, 1983.
35. Raney interview, May 31, 1983.
36. Erickson interview, June 14, 1983.
37. Ernull interview, June 16, 1983; Raney interview, May 31, 1983.
38. Ernull interview, June 16, 1983.
39. J. Boehm and H.H. Hosenthien, "Flight Simulation of Rockets and Spacecraft," in *From Peenemunde to Outer Space*, eds. E. Stuhlinger, et al., Marshall Space Flight Center, Huntsville, AL, March 23, 1962, p. 437.

40. See James E. Tomayko, "Helmut Hoelzer's Fully Electronic Analog Computer," *Ann. Hist. Comput.*, 7 (3) 227-240 (July, 1985).
41. W.K. Polstroff and F.L. Vinz, "General-Purpose Simulation At Marshall Spaceflight Center," Presented to American Institute of Aeronaut. and Astronaut. Working Group on Flight Simulation Facilities at Marietta, Georgia, April 22, 1974, pp. 1-4; Interview with Jack Lucas, Marshall Space Center, June 21, 1983.
42. Lucas interview, June 21, 1983; Polstroff and Vinz, "General-Purpose Simulation At Marshall Spaceflight Center," April 22, 1974, p. 9.
43. Interview with Frank Vinz, Marshall Space Center, June 21, 1983.
44. P.W. Hampton and H.G. Vick, *Space Shuttle Main Engine Hardware Simulation*, Marshall Spaceflight Center, Huntsville, AL, no date, pp. 1, 2, 7, 8.
45. T.V. Chambers, "Shuttle Avionics Integration Laboratory," AIAA Paper, 1977, p. 212.
46. Mansfield interview, June 1, 1983.
47. Chambers, "Shuttle Avionics Integration Laboratory," 1977, pp. 219-220.
48. J.T.B. Mayer, "The Space Shuttle Vehicle Checkout Involving Flight Avionics Software," AIAA Paper no. 1981-2141, pp. 174-175.
49. Chambers, "Shuttle Avionics Integration Laboratory," 1977, p. 216.
50. Mansfield interview, June 1, 1983.
51. Kenneth R. Castleman, *Digital Image Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1979, p. 389. Castleman's book contains an excellent short history of the development of image processing at the Jet Propulsion Laboratory in his Appendix I, pp. 383-400, including an effective selected bibliography.
52. Interview with Robert Nathan, Jet Propulsion Laboratory, May 30, 1984.
53. J.A. Dunne et al., "Digital Processing of the Mariner 6 and 7 Pictures," *J. Geophys. Res.*, 395 (1976).
54. Interview with Albert Zobrist, Jet Propulsion Laboratory, May 23, 1984.
55. W.B. Green and D.A. O'Handley, "Recent Developments in Digital Image Processing at the Image Processing Laboratory at the Jet Propulsion Laboratory," *Proc. IEEE*, 60 (7) 822 (July 1972).

362 COMPUTERS IN SPACEFLIGHT

56. R.F. Stott, "Mariner Mars 1971: Real-Time Video Data Processing," April 4, 1972, from the files of R.F. Stott, p. 3A.
57. Stott, "Mariner Mars 1971: Real-Time Video Data Processing," April 4, 1972, p. 6.
58. Green and O'Handley, "Recent Developments in Digital Image Processing," July 1972, pp. 821–823.
59. Stott, "Mariner Mars 1971: Real-Time Video Data Processing," April 4, 1972, p. 4.
60. For a description of VICAR and its subroutines, see Chambers, *Digital Image Processing*, pp. 52–67, 401–411.
61. Zobrist interview, May 23, 1984.
62. Nathan interview, May 30, 1984.
63. R. Nathan, "Large Array VLSI Filter," Reprinted from 1983 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management—CAPAIDM, Silver Spring, MD, p. 15.
64. Nathan interview, May 30, 1984.

Bibliographic Note

The detailed references to sources given with each numbered note in the text serve as the primary record of the evidence used in writing this volume. This essay provides a general summary of the sources used and describes the method of locating and evaluating them. As the volume is a history of technology rather than an institutional history, the burden of the written evidence lies in technical reports, software documentation, training manuals, and feasibility studies rather than memoranda and executive orders. However, the latter sources often provide the time sense and structure that so quickly fades from an engineer's mind as he goes on to his next project. Because NASA's involvement in computer operations during the 1960s and 1970s mirrored the stumbling discovery of software engineering principles by other organizations, interviews not only with managers but programmers and contract liasons in both NASA and contractor offices are a major contribution to my understanding of the flow of events and their impact on later decisions and developments. Thus, the sources include the basic mix found in other NASA histories: written institutional records and oral interviews, with the addition of an extensive list of technical material.

Identification of source materials was conducted in several cycles. First, a comprehensive search through standard references, such as *The Applied Science and Technology Index*, was made to identify secondary sources that dealt with NASA's use of computers. The period surveyed was 1945 to 1981. Articles were found in journals such as *Electronics*, *Journal of Spacecraft and Rockets*, *Journal of Guidance and Control*, and various IBM, American Institute of Aeronautics and Astronautics, and American Federation of Information Processing Societies publications. This search revealed that even though NASA is critically dependent on computers for spaceflight operations, and that even though massive amounts of material have been written on the space program in general, relatively little has been published in public journals or in books on spaceflight specifically treating the use of computers. Most of what has been published is short and far from comprehensive. In books about space projects heavily dependent on computers, such as NASA's own *Chariots for Apollo*, generally nothing is said about the configuration, programming, or operation of those computers. Thus, the general public, even the technically sophisticated public, is largely in the dark about the specifics of NASA's computer use. That, of course, is one reason why this volume is needed.

The identification of primary source materials came next. Thanks to a Faculty Research Grant from Wichita State University, I was able

to make a preliminary visit to the Johnson Space Center while preparing my contract proposal. This visit provided access to the RECON bibliographic retrieval system that NASA uses. RECON is especially valuable in this subject area because key words are rather liberally applied to each item stored: anything remotely to do with computers had a "computer" key word. Therefore, search keys could be developed such as "Apollo*Computers," and items with both those key words could be separated from the mass of material on Apollo. The RECON search netted over 1,000 items, of which about 25% were rejected based on their abstracts or because the other associated key words indicated that the item was primarily concerned with another subject, with only passing reference to computing. The remainder were physically examined in order to eliminate those that actually did not have pertinent materials. This process of reading the remaining sources and doing the interviews turned up a number of new primary sources. The bulk of these sources are software and hardware specifications, operations reports, flight training manuals, and spacecraft systems familiarization manuals, which are not indexed either in standard bibliographies or RECON.

Most items in the NASA archives at the various centers are not listed in RECON, so memoranda and other such unpublished items were discovered the "old-fashioned" way: by physically going through the files. My contract provided for visits to Johnson Space Center, Kennedy Space Center, Marshall Space Center, the Jet Propulsion Laboratory, Dryden Flight Research Center, Goddard Space Center, and NASA Headquarters. Archives do not exist at either Marshall or Goddard, so individuals provided whatever new sources were gained at those places. In each of the other centers, a serious perusal of the materials relating to computer usage was done. At Johnson, the archives transferred to the Woodson Library of Rice University were also consulted.

The tour of the various NASA facilities demonstrated that those with full-time historians or archivists had the most useful archives. That is, of course, obvious, but it is interesting to contrast the situation at, say, Kennedy versus that at Marshall. Marshall has not had an archivist or historian since the early 1970s. There is no central repository. The only way information could be located was by finding division chiefs in the areas to be researched and then depending on them to help identify the people who had experience with the actual program. Those people could then be interviewed and some had kept copies of appropriate documentation. Others had not. For example, the entire story of the Saturn launch vehicle preflight checkout system is in danger of being lost. The people who built it are nearing federal retirement, they have thrown out almost all of the documentation, and their memories are clouded by the other projects in which they have been involved in the last 20 years. At Kennedy, even though a lot of interviews were conducted, the main source of information was the

well-kept library and archives, which included a technical documents section. There the bulk of the story of the Shuttle Launch Processing System, the successor to the Saturn preflight checkout system, can be reconstructed from specifications, development reports, and user manuals. The point is that most of NASA will soon be in the state that Marshall is in. Johnson has lost its full-time historian, and the archives are maintained part-time by administrative personnel, with some items transferred to the Woodson Library. The latter facility has no provision for extensive xeroxing and is closed-shelf, two crippling defects for the historian with limited time on-site.

With no one at the various centers to choose what should be saved, documents are being lost at a prodigious rate. It is true that much paper generated by NASA is not needed for later historical research, but there is no apparent system of sifting out the material that has potential for later use. At the Jet Propulsion Laboratory, where files are regularly collected for archiving, there is no active control of what is sent to the records center. Some boxes contained organized, indexed files. Others looked as though someone had simply emptied their desk drawers into them; they contained such items as old copies of the employees' newsletter amidst notes and memos in no particular order. Additionally, project managers can choose to delete materials and thus prevent historians from gaining a balanced perspective.

Personnel assigned as history liaisons at each NASA center, even if they had no historical experience, were unfailingly helpful and cooperative. They are mentioned and thanked individually in the Acknowledgments. By contacting them ahead of time, I was able to obtain the names of initial contacts, which led me to the large number of very candid interviewees whose collective memory adds so much to this book. They are listed at the end of this note. I was also able, through the individual efforts of interviewees, to obtain entry to areas normally restricted to the public. In that way I was able to see firsthand what I was writing about. There is no substitute for seeing the computers installed and operating and for looking at and using the crew interfaces. In that way, the true scale of things is established in the mind.

The remainder of this bibliographic note is a topic-by-topic summary of the main sources.

THE GEMINI DIGITAL COMPUTER

The most useful written source for the hardware section of the chapter on Gemini is the *NASA Project Gemini Familiarization Manual*, Volume 2, published by McDonnell Corporation in 1965. This manual contains a detailed hardware description of the Gemini

digital computer, its location in the spacecraft, and drawings of the user interfaces. For the software development cycle and contents of the Gemini software loads, *Project Gemini: A Technical Summary*, (NASA CR-1106, 1968) by P. W. Malik and G. A. Souris, is the most comprehensive. Ivan Ertel, then of the Manned Spacecraft Center History Office, conducted extensive interviews with IBM personnel who worked on the Gemini computer during a visit to the Owego, New York plant in April of 1968. These interviews are transcribed and available at Johnson Space Center. They were very useful in identifying development problems and procedures. Lastly, interviews with Gene Cernan and John Young provided information about the system from the user standpoint.

THE APOLLO COMPUTER SYSTEMS

Sources for this chapter were primarily technical reports issued by the MIT Instrumentation Lab, memos on file at the Johnson Space Center, and some very illuminating interviews. The best hardware description of the Block II computer is in R. Alonso and A. L. Hopkins, *The Apollo Guidance Computer* (NASA-CR-118183, August 1963). An introduction to developing software for the computer is B. I. Savage and A. Drake, *AGC4 Basic Training Manual* (MIT, January 1967). Copies of these are available at Johnson Space Center. For NASA's view of the hardware and software difficulties in developing the onboard computer, the files of Howard W. Tindall are the most helpful. These are also at Johnson. The best interview sources for this chapter are John R. Garman of JSC and Stan Mann, formerly of JSC. Both were involved in the Apollo software development effort and later in the Shuttle program. Both were extremely candid and very informative. Transcribed interviews of David Hoag and Ralph Ragan of MIT were also helpful. Astronaut users Vance Brand, Gene Cernan, and John Young gave good insights in their interviews. For the Abort Guidance Section, the best source is P. M. Kurten, *Apollo Experience Report: Guidance and Control Systems—Lunar Module Abort Guidance System* (NASA-TN-D-7990, Johnson Space Center, Houston, TX, July 1975).

THE SKYLAB COMPUTER SYSTEM

The Skylab chapter is overwhelmingly based on two excellent sources, both produced by IBM Corporation. They are the *Design and Operational Assessment of Skylab ATMDC/WCIU Flight Hardware and Software* (IBM No. 74W-00103, May 9, 1974) and the *Skylab*

Reactivation Mission (IBM No. 79W-0005, September 12, 1979). The Skylab hardware and software development was an operation largely local to Huntsville, Alabama, where IBM had a continuing corporate presence since the early 1960s when work on the computer systems for the Saturn launch vehicles began. These two sources are detailed histories of the development and use of the computer system in both the primary Skylab mission and the reactivation mission. They are quite frank, documenting both the first-time successes and needed restarts, although obviously proud of the highly reliable record of the system. By the time I reached Huntsville, the IBM office had closed, but some ATMDC programmers were still on-site working on Spacelab. By now, those few are scattered elsewhere. One, John Copeland, was kind enough to be interviewed and lent the reactivation documentation. Bill Chubb and Jim McMillion of Marshall Spaceflight Center were also very good sources on the computer system. Steve Bales of Johnson Space Center was able to give a perspective on the system from the flight controller's angle and was especially helpful regarding the first 2 weeks of the primary mission before the crew arrived.

THE SHUTTLE DATA PROCESSING SYSTEM

At the time this volume was being written, the Shuttle was an ongoing project. Therefore, abundant primary source materials in the form of actual requirements and design documents, program code, and managers involved in the day-to-day production of the hardware and software were available. Additionally, the astronauts have fresh memories, and the artifacts described in the chapter can be actually seen and touched. I decided to try to base this chapter on these sources as much as possible, plus my personal experiences in using the equipment and software in simulators. Thus, there are a great number of references to interviews (of which roughly 35 hours were done) and to current documentation. Despite this plethora of sources, some things could not be settled. An example would be the question of who thought up the eventual scheme used in redundancy management. No one could name a specific person or a time. Everyone asked about the subject said "it just evolved," or "no one person thought it up," both of which are true, but frustrating!

NASA STANDARD SPACECRAFT COMPUTER-1

Since the origination and development of this computer took place at one place, Goddard Spaceflight Center, it was relatively

simple to find materials and persons. Ann Merwarth, Bill Stewart, and John Azzolini were the key informants in describing the design and capabilities of this device. Stewart led me to a documents distribution point where I was able to get copies of Merwarth's guide to the executive. A fine source of information is an article in the September 1984 issue of *Communications of the ACM*, authored by Merwarth, Stewart, and others, which tried to show the evolution of the system from its beginnings in the mid-1960s. A chapter on this computer, which was written for an early draft of the volume was later deleted because it was too much a restatement of previously published materials.

COMPUTERS ON DEEP SPACE PROBES

The Jet Propulsion Laboratory has three methods of archiving documentation. One is the "Vellum File," located in the basement of the Library and containing on microfilm all technical documentation used in projects. It is possible to obtain hard copy of critical documents, which I did when told of their existence by my informants. The Library itself contains indexes of publications written by JPL personnel, wherever published and holds copies of most of those in its collection. A third source, and one very critical for historians, is a central depository that contains memos and other unpublished documentation from the project offices and permanent section offices. Materials in this archive are arranged by JPL section number and stored in boxes. This collection is very erratic in quality. Almost all the materials cited in Chapters 5 and 6 were found in one of these three locations.

If the section on Galileo contains omissions, it might be because the project director refused to let me examine his and his chief deputy's office files. No reason was given. In the face of the existence of the actual documents, I thought it was foolish to speculate on any matters possibly contained within them, as a later historian can examine the materials after they are retired—assuming, that is, that they are not destroyed beforehand.

Personal contacts at the Lab were among the most satisfying I had in all my travels. Engineers at JPL are more introspective and more history conscious than others I have met. Their help is reflected in the actual notes to Chapters 5 and 6.

EVOLUTION OF LAUNCH PROCESSING

Documentation for this chapter was hard to come by, both because the information was scattered among Johnson, Marshall, and Kennedy Space Centers, and because pre-Shuttle primary sources at

both Marshall and Johnson had been destroyed. However, the current Launch Processing System is heavily documented as to function because it is still operating. Also, Kennedy preliminary studies such as the *Space Shuttle Launch Operations Study* are in archives, and published summaries by IBM tended to be historical in nature. Therefore, the present System is easy to describe. For specifics of origin, though, I am again indebted to my informants, particularly Thomas S. Walton, who lived through the entire era at Kennedy, Jim Lewis of Marshall, Frank Byrne, the genius behind the common data buffer, and Henry Paul, who headed the development effort. A short manuscript history by Bill Bailey of Kennedy and an interview with him were also very helpful.

MISSION CONTROL

Fortunately, a fair amount of original source material is available on the subject of mission control. Documentation for the Mercury Control Center software system is contained in detailed IBM handbooks such as the "Goddard Monitor System," supplied by John Morton. He, J. Perry Chambers, and Ray Mazur were excellent sources of information on Goddard's Spaceflight Center's role both in manned and unmanned mission control. Philco's "IMCC Systems and Performance Requirements" study and IBM's proposal for the Gemini and Apollo mission control centers are the best sources for what was installed at the then Manned Spacecraft Center. Interviews with Lyn Dunseith and James Stokes helped considerably with that era. Shuttle mission control information is primarily based on interviews with Dub Pollen, Fred Riddle, and Gene Campbell of IBM and a publication by S. E. James of that company. Researchers interested in unmanned mission control of lunar and planetary probes should consult the final reports of the various Mariner, Viking, Ranger, Surveyor, and Voyager projects, usually issued as Jet Propulsion Laboratory technical reports. Each contains a detailed description of control considerations. George Gianopolis, Richard Moulder, Lloyd Jennings, Frank Singleton, Carl Johnson, and Don Royer, all of JPL, each contributed informative interviews for this section.

SIMULATIONS AND IMAGE PROCESSING

Again, interviews are the backbone of my understanding any materials analyzed for this chapter. Jim Raney, Bob Ernull, and Ken Mansfield of Johnson Space Center provided both knowledge and materials related to mission and engineering simulators. Jack Lucas

and his staff at Marshall helped with the engineering simulators located there. Finally, Bob Nathan of JPL, founder of image processing, and his colleague Al Zobrist clarified the complex world of digital imaging. Almost all written materials used as sources for this chapter were either given to me by these informants, or they directed me to them. The monograph *Digital Processing of Remotely Sensed Images*, by Johannes G. Moik (NASA SP-431), is a good reference for NASA's work in this field.

Interview List

Note: Unless identified otherwise, all persons on this list were NASA employees at the time they were interviewed. Locations are also indicated at the time of the interview.

AARON, JOHN, Johnson Space Center, June 17, 1983.

ALDRICH, ARNOLD, Johnson Space Center, June 13, 1983.

AZZOLINI, JOHN, Goddard Spaceflight Center, July 2, 1984.

BAILEY, WILLIAM, Kennedy Space Center, June 30, 1983.

BALES, STEVEN, Johnson Space Center, May 31, 1983.

BIEGERT, PAMELA, Kennedy Space Center, June 30, 1983.

BLIZZARD, EDGAR, Jet Propulsion Laboratory, May 29, 1984.

BOGAN, JACK, IBM, Kennedy Space Center, June 29, 1983.

BORNCAMP, FRANZ, Jet Propulsion Laboratory.

BRADFORD, CLIFFFORD, Marshall Space Center, June 20, 1983.

BRAND, VANCE, Johnson Space Center, June 2, 1983.

BULKLEY, R.C., IBM, Kennedy Space Center, June 27 and 29, 1983.

BRUCKNER, BOBBY, Kennedy Space Center, June 30, 1983.

BYRNE, FRANK, Kennedy Space Center, June 29, and July 8, 1983.

CAMPBELL, GENE, IBM, Houston, June 13, 1983.

CERNAN, GENE, telephone interview from Houston, November 7, 1983.

CHAMBERS, J. PERRY, Goddard Spaceflight Center, June 28, 1984.

CHARLEN, WILLIAM, Jet Propulsion Laboratory, May 18, 1984.

- CHUBB, WILLIAM, Marshall Space Center, June 22, 1983.
- CLAYTON, ELDON, Johnson Space Center, June 1, 1983.
- COPELAND, JOHN, IBM, Marshall Space Center, June 23, 1983.
- COX, KENNETH, Johnson Space Center, June 14, 1983.
- DEESE, SAMUEL, Jet Propulsion Laboratory, telephone interview, January 31, 1985.
- DEETS, DWAIN, Dryden Flight Research Center, May 25, 1984.
- DELAUNE, CARL, Kennedy Space Center, July 5, 1983.
- DEMING, JAMES E., Kennedy Space Center, July 6, 1983.
- DUNSEITH, LYNWOOD, Johnson Space Center, June 2 and 9, 1983.
- EISENMAN, DAVID, Jet Propulsion Laboratory, May 21, 1984.
- ERICKSON, JOHN, Johnson Space Center, June 14, 1983.
- ERNULL, ROBERT, Johnson Space Center, June 16, 1983.
- FOY, LYNNE, Johnson Space Center, June 16 and 17, 1983.
- GARMAN, JOHN R., Johnson Space Center, May 25, and June 1, 1983.
- GIANOPOLIS, GEORGE, Jet Propulsion Laboratory, June 4, 1984.
- GREENBERG, EDWARD, Jet Propulsion Laboratory, May 30, 1984.
- HART, TERRY, Johnson Space Center, June 10, 1983.
- HARTSFIELD, HENRY, Johnson Space Center, June 2, 1983.
- HEDDINS, FREDERICK, IBM, Kennedy Space Center, June 27 and 29, 1983.
- HINSON SHIRLEY, Johnson Space Center, June 16, 1983.
- HOELZER, HELMUT, Huntsville, Ala., June 24, 1983.
- HUGHES, BRAD, Kennedy Space Center, July 5, 1983.

- HUGHES, FRANK, Johnson Space Center, June 2, 1983.
- JENNINGS, LLOYD, Jet Propulsion Laboratory, May 15, 1984.
- JOHNSON, CARL, Jet Propulsion Laboratory, May 23, 1984.
- JOHNSON, DONALD, Jet Propulsion Laboratory, May 16, 1984.
- JORDAN, FRANK, Jet Propulsion Laboratory, May 31, 1984.
- KILLINGBECK, LYNN, IBM, Houston, June 7, 1983.
- KOHL, WAYNE, Jet Propulsion Laboratory, telephone interview, January 31, 1985.
- KOPF, EDWARD H., Jet Propulsion Laboratory, May 18, 1984., telephone interview, January 31, 1985.
- LANIER, RONALD, Johnson Space Center, June 16, 1983.
- LEE, B. GENTRY, Jet Propulsion Laboratory, June 1, 1984.
- LEMON, RICHARD, Jet Propulsion Laboaratory, May 29, 1984.
- LEWIS, JAMES, Marshall Space Center, June 20, 1983.
- LINEBERRY, EDWARD, Johnson Space Center, June 2, 1983.
- LOCK, WILTON, Dryden Flight Research Center, May 24, 1984.
- LOUSMA, JACK, telephone interview from Houston, July 5, 1983.
- LUCAS, JACK, Marshall Space Center, June 21, 1983.
- MACINA, ANTHONY, IBM, Houston, June 7, 1983.
- MALM, RICHARD, Jet Propulsion Laboratory, May 31, 1984.
- MANN, STANLEY, Johnson Space Center, June 6 and 8, 1983.
- MANSFIELD, KENNETH, Johnson Space Center, June 1, 1983.
- MATTOX, RUSSELL, Marshall Space Center, June 23, 1983, telephone interview, November 16, 1984.
- MAZUR, RAYMOND, Goddard Spaceflight Center, June 28, 1984.

- McMILLION, JAMES, Marshall Space Center, June 22, 1983.
- MERWARTH, ANN, Goddard Spaceflight Center, July 3, 1984.
- MILLER, BRUCE, Kennedy Space Center, July 5, 1983.
- MITCHELL, WALTER, Marshall Space Center, June 23, 1983.
- MORECROFT, JOHN, Jet Propulsion Laboratory, May 29, 1984.
- MORTON, JOHN, Goddard Spaceflight Center, June 27, 1984.
- MOULDER, RICHARD, Jet Propulsion Laboratory, May 21, 1984.
- NATHAN, ROBERT, Jet Propulsion Laboratory, May 30, 1984.
- OTAMURA, ROY, Jet Propulsion Laboratory, May 29, 1984.
- PALIKOWSKY RAYMOND, Singer, Houston, June 10, 1983.
- PANCIERA, ROBERT, Marshall Space Center, June 20, 1983.
- PARRISH, ALBERT, Kennedy Space Center, June 28, 1983.
- PARTEN, RICHARD, Johnson Space Center, June 3 and 16, 1983.
- PAUL, HENRY, Kennedy Space Center July 7, 1983.
- PENDLETON, THOMAS, Johnson Space Center, June 9, 1983.
- PENOVICH, FRANK, Kennedy Space Center, July 1, 1983.
- PETYNIA, WILLIAM, League City, TX, June 8, 1983.
- POLLEN, DUB, IBM, Houston, June 13, 1983.
- RAINES, GARY K., telephone interview from Houston, November 1, 1985.
- RANDALL, JOSEPH, Marshall Space Center, June 20, 1983.
- RANEY, JAMES, Johnson Space Center, May 31, 1983.
- RICE, RICHARD, Jet Propulsion Laboratory, May 29, 1984.
- RIDDLE, FREDERICK, IBM, Houston, June 13, 1983.

- RONE, KYLE, IBM, Houston, June 3, 1983.
- ROYER, DONALD, Jet Propulsion Laboratory, June 7, 1984.
- SINGLETON, FRANK, Jet Propulsion Laboratory, May 17, 1984.
- SMITH, GEORGE, IBM, Kennedy Space Center, June 27 and 29, 1983.
- STEARNS, JANE, Kennedy Space Center, June 30, 1983.
- STEWART, WILLIAM, Goddard Spaceflight Center, July 10, 1984.
- STOKES, JAMES, Johnson Space Center, June 14, 1983.
- STORY, SCOTT, Ford Aerospace, Johnson Space Center, June 16, 1983.
- STOTT, RUSSELL, Jet Propulsion Laboratory, May 16, 1984.
- SULLIVAN, WILLIAM, Johnson Space Center, June 14, 1983.
- SWEARINGEN, CHARLES, Huntsville, AL, June 21, 1983.
- TINDALL HOWARD W., telephone interview from Washington, D. C., August 10, 1984.
- VICK, H.G., Marshall Space Center, June 21, 1983.
- VINZ, FRANK, Marshall Space Center, June 21, 1983.
- WALTON, THOMAS S., Kennedy Space Center, July 6, 1983.
- WILLBANKS, JAMES, IBM, Kennedy Space Center, June 29, 1983.
- WOODDELL, JOHN, Jet Propulsion Laboratory, May 21, 1984.
- YARBOROUGH, ROBERT, Kennedy Space Center, July 6, 1983.
- YOUNG, JOHN, telephone interview from Johnson Spaceflight Center, March 6, 1984.
- ZIPSE, JOHN, Jet Propulsion Laboratory, 22 May 1984.
- ZOBRIST, ALBERT, Jet Propulsion Laboratory, May 23, 1984.

Interviews Conducted by Other Persons

BACHMAN, DALE F., interviewed by John J. Lenz, at IBM, Owego, NY, April 25, 1968; transcript at Johnson Space Center.

BARTON, JOHN, interviewed by James Grimwood, at Motorola Aerospace Center, Scottsdale, AZ, July 11, 1966, transcript at Johnson Space Center.

DODGE, HAROLD E., interviewed by John J. Lenz, at IBM, Owego, NY, April 25, 1968; transcript at Johnson Space Center.

DRAPER, C. STARK, interviewed by Ivan Ertel at Cambridge, MA, April 23, 1968.

HOAG, DAVID G., interviewed by Ivan Ertel, at MIT, Cambridge, MA, May 15, 1967; transcript at Johnson Space Center.

HUTCHISON, HOMER W., interviewed by Ivan Ertel, at IBM, Owego, N. Y., April 25, 1968; transcript at Johnson Space Center.

JACKSON, LEE, interviewed by Ivan Ertel, at IBM, Owego, NY, April 25, 1968; transcript at Johnson Space Center.

JIMERSON, LEROY S. jr., interviewed by Ivan Ertel, at IBM, Owego, NY, April 26, 1968; transcript at Johnson Space Center.

JOACHIM, JAMES, interviewed by Ivan Ertel, at IBM, Owego, NY, April 25, 1968; transcript at Johnson Space Center.

LENZ, JOHN J., with JOHN L. SWEENEY, CHARLES E. DUNN, and CONRAD D. BABB, interviewed by Ivan Ertel, at IBM, Owego, NY, April 25, 1968; transcript at Johnson Space Center.

MILLER, JOHN E., interviewed by Ivan Ertel, at MIT, Cambridge, MA, April 28, 1966; transcript at Johnson Space Center.

RAGAN, RALPH, interviewed by Ivan Ertel, at MIT, Cambridge, MA, April 27, 1966; transcript at Johnson Space Center.

RECHTIN, EBERHARDT, interviewed by Cargill Hall, Jet Propulsion Laboratory, c. 1970?; transcript in the Jet Propulsion Laboratory Library Vault.

WESTKAEMPER, ROBERT M., with LEE JACKSON, interviewed by Ivan Ertel, at IBM, Owego, NY, April 25, 1968; transcript at Johnson Space Center.

Appendix I: Glossary Of Computer Terms

Accumulator---The register in the central processing unit of a computer used to store the current results of calculations.

Algorithm---A step-by-step solution to a problem that is the basis for writing the code that will enable the computer to solve it.

Analog Circuit---An electrical circuit that models the behavior of a real object or force, providing a nondigital means of calculation.

Analog Computer---A machine that computes by modeling objects and forces using either mechanical or electrical means.

Assembly Language---A low-level programming language for computers that express algorithms in statements consisting of mnemonics representing actions and numbers representing addresses. For example, the statement "ADD A,7FFF" tells a computer to add the contents of location 7FFF (a hexadecimal number) to the contents of the accumulator and leave the result in the accumulator. The specific mnemonics for assembly languages may be different for different processors but are closely related. Programs written in assembly languages, although conservative of machine resources and quite fast, are much more difficult to write and maintain than those written in high-level languages such as FORTRAN, Pascal, and HAL/S.

Asynchronous---Occurrences happening at no set time. Asynchronous interrupts mean that signals to a computer to start a specified process may come at any time.

Backup Program---A computer program shorter and with less functionality than the primary program that performs only critical functions in case the primary program or the hardware in which it resides fails.

Bandwidth---The amount of information that can be transferred in a discrete amount of time. The higher the bandwidth, the more information.

Batch Processing---A method of executing programs on a computer

that reserves resources for the use of a particular program and releases them upon completion. Older batch systems could process only one program at a time.

Binary Object Code---The result of an assembler processing an assembly language program or a compiler processing a high level language program. It consists of binary numbers in the machine language of the specific computer on which the program is to run.

Bit---A binary digit, representing either a one or a zero.

Breadboard---A prototype of a computer or other electronic device built by the design group to test the device before it is packaged for production.

Bubble Memory---A type of nonvolatile computer memory using materials that can retain a specific magnetic polarity when electrical power is cut off. The polarity determines whether a one or zero is being stored.

Buffer---A cache of memory used to store information temporarily during transfer operations. It is usually used to adjust for differences in operating speed between devices.

Buffer Register---A register used to temporarily store information in transit to another device.

Bug---Common term for an error in a computer program or hardware.

Bus---An interconnection device that can be used to speed up information transfer (as when a bus made up of multiple wires carries the bits of an entire computer word in parallel) and to act as a connector for multiple devices (as when several devices that do not need to transmit simultaneously time-share the use of the bus for intercommunication). Also used to refer to heavy-duty electrical power cables or bars supplying power to many devices.

Byte---A collection of bits, commonly eight.

Central Processor---The portion of a computer that contains the control circuits and does the actual calculations.

CMOS---Complimentary metal-oxide silicon circuits, characterized by

low-power requirements, tolerance of wide variation in voltages, and susceptibility to damage from discharges of static electricity.

CMOS Processor---A microcomputer built of CMOS circuits.

Coding---The act of writing a program for a computer.

Compiler---A computer program that accepts statements of a high-level language as input and generates machine code that will execute those statements as output.

Condition Code---A message several bits in length used to communicate the physical status of one device to another device.

Contiguous Memory Locations---Addresses in a machine memory located adjacent to one another.

Core Memory---A type of computer memory constructed of a series of two-dimensional planes containing networks of wires with ferrite rings called "cores" at their intersections. The magnetic polarity of the cores can be changed by electrical pulses. Each core stores 1 bit of information. Core memory is nonvolatile; when power is cut off, it does not lose information. Destructive-readout core memory loses the information stored in a core when the core is read, so a temporary register must be used to intermediately store the information before writing it back to its original location as it is simultaneously sent to other parts of the computer. Nondestructive-readout core memory can be read without the information being changed.

Core Rope---A type of core memory that stores entire computer words rather than individual bits. Each core in a core rope is permanently charged to represent a "one." A number of wires equal to the number of bits in a word is weaved through the cores. When a bit within a word is to represent a one, its wire is connected to a core. Bits representing zeroes are not connected. Thus, by selecting the correct core and sensing which wires represent ones and which zeroes, the word can be reconstructed. More than one word can be attached to a core by adding more wires to the rope. Core rope, once constructed, can only be read.

Core Storage---Another name for core memory.

Core Transistor Logic---Circuits made up of discrete transistors used to form the control unit in the central processor of a computer.

Cycle Time---The length of time it takes for a computer to do a fundamental operation, such as reading a word from memory into the central processor. Some instructions, such as multiply, take several cycles.

Data Flow Diagram---A software design tool that uses circles to represent operations and arrows to represent data movement. It is used to determine the ordering of processes and input and output requirements.

Data Formatters---Hardware or software that takes raw data from devices and puts it into a uniform format for transmission, usually adding some special error detection bits.

Data Word---A computer word containing only data, not instructions.

Demultiplexer---A device that receives data transmitted on a bus and routes it to the correct device.

Digital Circuit---A circuit constructed to handle discrete units of information that can represent ones and zeroes.

Digital Computer---A calculating device using digital circuits, usually consisting of a central processing unit, memory, and input and output devices.

Diode Transistor Logic Integrated Circuit---A type of miniaturized digital circuit used to construct logic units in the central processor of a computer.

Direct Addressing---Using the absolute address of a memory location to access data within it. For example, in a hypothetical machine with a word size of 4 bits, up to 16 memory locations can be directly addressed, simply by matching them one for one with the 16 numbers 4 bits can represent. Thus, memory location 1011 is the twelfth location in the memory. Since it is often necessary to have more memory than the number of locations a single word can represent, indirect addressing schemes must be devised. The 4 bit computer can indirectly represent a memory location by using two words; one word can indicate which bank of 16 words to access, the second word can indicate which of the 16 addresses in that bank to read.

Direct Memory Access---Reading or writing to a memory location in a computer without passing the information through the central processing unit for disposition.

Discrete Component---A component containing a single entity, such as a transistor, as opposed to containing many entities, such as an integrated circuit with thousands of transistors.

Disk Drive---A type of mass storage device in which bits are represented by magnetized areas on a plane, or disk, covered with a suitable material of the same type used for magnetic tape. A disk drive may have one or many disks.

Double Precision---Using two computer words to represent a number instead of one.

Drum Memory---A type of mass storage device in which the material (similar to material used to make magnetic tape) that contains the information is placed on a rotating drum.

Emulator---A device that can be programmed to replicate the logic and functions of another device and operate at the same speed.

Erasable Memory---Memory in which information can be overwritten by new information.

Event Word---A word of information containing code to activate devices or functions.

File---A collection of related information, such as a computer program or imaging data, which can be thought of as a unit.

Firmware---Software stored in read-only memory devices used to control logic flow in a computer. Changing the firmware changes the nature of the computer.

Fixed Memory---Memory that can only be read.

Fixed Point---A method of representing numbers in a computer in which the decimal point is permanently fixed. Therefore, numbers used in calculations must be properly scaled relative to the location of their decimal point or the results will be meaningless. Such scaling is usually left to the programmer.

Flat Packs---Collections of integrated circuits packaged in modules for use in a computer.

Flip-Flop---A logic device that can change from containing a one to a zero and vice-versa depending on inputs. Flip-flops are often used in the central processing unit of a computer.

Floating Point---A method of storing numbers in a computer in which the location of the decimal point is stored with the values of the individual places.

Flowchart---A method of program design in which algorithms are represented by specific two-dimensional shapes and connecting arrows. Each shape represents a specific logical act. For example, a diamond indicates a true/false decision.

Full Word---All the bits of a computer's word size.

Gate---A logic device. For example, an "AND Gate" returns the result of a Boolean AND operation on its inputs.

General-Purpose Register---A register in the central processing unit of a computer not assigned to a specific task but that can be dynamically required to act as an accumulator, program counter, or index register.

Half Word---One half of the bits of a computer's word size.

Hard-Wired Logic Circuits---Logic implemented in hardware, as opposed to implementation in software.

Hard-Wiring---Permanently representing logic in hardware.

Hard Logic---Logic permanently represented in hardware.

Hardware---Physical components of a computer system or other device, such as memories, registers, and control logic circuits.

Hexadecimal---Base 16. One-digit numbers include 0 through 9 and A through F.

High-Level Language---A language in which algorithms can be represented in a series of structured, formal statements using selected easily recognizable words from a natural language. For example, "IF VALVE_POSITION = 2 THEN SET FUEL_FLAG TO TRUE" is a high-level language statement.

Image Processing---Using computers to operate on the digital information that represents images to enhance its value for specific purposes. Most images are represented by collections of 8-bit "gray scale" values, which contain a number ranging from 0 to 255 indicating the level of darkness in one picture element, or dot, in an image. Image processing works on these 8-bit values to increase contrast, translate oblique images to vertical images, and emphasize certain colors.

Imaging---The process of acquiring images using vidicon tubes and digital circuits.

Index Register---A register in the central processing unit of a computer that contains the value of the memory bank currently being accessed.

Instruction Set---The list of instructions that a computer can execute. It varies from a few to several hundred depending on the computer.

Instruction Word---A word in a computer containing the bits representing an instruction and an address on which the instruction is to operate.

Integrated Circuit---An electronic circuit containing hundreds, thousands, or millions of components, such as transistors, and used for a specific purpose, such as logic or memory.

Interactive Processing---Executing computer programs so that the user can actively send information to the program and receive information from it while the program is running.

Interface---The connection between two devices for the exchange of data.

Interface Table---A collection of information containing instructions for connecting devices so that data can be exchanged.

Interpreter---A computer program that executes statements written in a high-level language one at a time.

Interrupt Stack---Storage of interrupts so that they can be handled in a last-in first-out fashion.

Interrupt-Driven System---A computer that is programmed to execute processes on demand, the demand taking the form of signals sent from other devices or itself that cause processes of lower priority to halt execution and be replaced by processes of higher priority. If the interrupt is of a lower priority than the current process, it is saved for later execution.

Kilobyte---One thousand twenty-four (1,024) 8-bit bytes. Abbreviated "Kb" or, more commonly, "K."

Listing---The content of a computer program, often used to refer to the printed result of sending a program through a compiler.

Logic Channels---Hardware that represents logic and through which data flows for processing.

Logic Circuit Board---A board containing electrical connections into which circuits are plugged representing the logic of a device.

Logic Gate---See *Gate*.

Machine Code---The representation of instructions as a series of bits, which cause the computer to execute the specified actions. Machine code is idiosyncratic to a particular type of computer.

Machine Cycle---See *Cycle Time*.

Machine Time---The amount of time a computer takes to execute a program or function.

Macro---A subroutine in assembly language that can be invoked by name.

Magnetic Tape---A mass storage device in which bits are represented in areas on a magnetic surface.

Main Memory---The memory of a computer used for both reading and writing operations, and of a faster type than secondary storage devices, such as magnetic tape or disk. Main memory is often made from core or semiconductor devices.

Mainframe Computer---A large, fast computer system capable of supporting hundreds of individual users, usually with a long word size, millions of words of main memory, and many peripherals.

Medium-Scale Integration---Integration of several thousand transistors or other devices on a single chip. Abbreviated "MSI."

Megabyte---One million 8-bit bytes. Abbreviated "Mb" or, more commonly, "M."

Microprocessor---A small computer built of integrated circuits, often on a single chip. Usually a microprocessor will support a single user or function.

Microcode---The programs used to create firmware.

Microsecond---One millionth of a second.

Millisecond---One thousandth of a second.

Minicomputer---A computer sized between a microprocessor and a mainframe computer, capable of supporting from one to several dozen users or tasks.

Mnemonics---Short groups of letters representing instructions in an assembly language. The mnemonic for "decrement the number in the specified register by one and branch to another address if the number is zero" is "DBZ."

Modularization---A technique for creating large computer programs based on the principle of "divide and conquer." Each module of a large program performs one task, can be entered at only one point, and exited at only one point. For example, the "BOOST THROTTLING TASK" module of the Shuttle on-board software handles the throttling of the main engines during the ascent phase of a mission. It is scheduled to execute many times each second. By isolating the function to this one module, it can be tested more easily and also reused in software loads for many Shuttle flights.

Multiplexer---A device that controls the time-sharing of a bus so that many devices can send information over the same interconnection.

Multitasking---A method of using computer resources so that more than one program can be in the process of execution at one time. The operating system of the computer will do calculations for one program while another is using the printer, for instance.

Nanosecond---One billionth of a second.

Networking---The process of interconnecting several computers together so that they can share data and programs.

Noise---Stray electromagnetic signals that may or may not interfere with data transmission and calculations. Noise may be generated locally, as when devices that leak electromagnetic radiation are placed next to one another, or from radiation fields in space.

NOR Gate---A type of logic gate that executes a Boolean OR operation on its inputs and then complements the result (reverses the value) before outputting it.

Object Code---See *Binary Object Code*.

Octal---Representation of numbers in base eight. Octal digits range from 0 through 7.

One's Complement---A method of storing binary numbers in which each bit in a word is complemented (reversed in value). The one's complement of 101 is 010.

Operation Code---That part of an instruction word that contains the bits that represent the specific mnemonic to be executed.

Parallel Data---Data transmitted in several bits at once.

Parameter---Data made available as input or output to a module or procedure. In general, the current value of specific information, such as fuel remaining, angle of flight, or re-entry time.

Parity---A method of ensuring accurate data transfer. The number of ones or zeroes in a specific computer word is kept either even or odd

by the addition of a changeable "parity bit." If the device is using even parity based on the number of ones, or if the number of ones in the word is odd, then the parity bit is set to one. When the transfer to another device is complete, that device examines all incoming words for even parity. If it detects odd parity, it requests a retransmission of the data that failed the parity test.

Parity Bit---See *Parity*.

Peripheral Device---Hardware associated with a computer used for input, output, or memory functions, such as disk drives, printers, terminals, and card readers.

Pixel---Short for *picture element*. One dot of a digital image.

Plated-Wire Memory---A type of nonvolatile computer memory using areas of wires plated with material that can be magnetically polarized to store bits. Its function and advantages are similar to core memory.

Primary Memory---See *Main Memory*.

Primary Storage---See *Main Memory*.

Procedural Language---A computer language that can represent algorithms, such as FORTRAN, Pascal, or Ada.

Processor---Alternative term for computer.

Propagation Time---The amount of time it takes for a signal to get from one part of a device to another, or to another device.

Pseudocode---A program design tool using structured English to represent algorithms. It has the advantage of being easily understandable and independent of the syntax of a particular computer language.

Radiation-Hardened Chips---Integrated circuits that have been protected from the effects of radiation, either by shielding, decreasing the density of components, or both.

Random Access Memory---A computer memory in which data can be written to or read from any location directly.

Read---The process of moving information from a storage device to some other place.

Read Only Memory---A type of computer memory that can only be read from, not written to, such as core rope.

Real-Time Processing---A type of processing in which the computer accepts or initiates continuous asynchronous inputs and outputs.

Redundant Circuits---Sections of hardware replicated to increase reliability.

Register---A storage location containing a set of bits. Registers in memory contain data words or instruction words, registers in the central processing unit of a computer contain data, instructions, the bank of memory currently being accessed, the location of the next program step, and intermediate results of calculations.

Secondary Memory---Mass storage such as tapes or disks, usually slower than main memory.

Secondary Storage---See *Secondary Memory*.

Semiconductor---Name for the electronic devices made of semiconducting metals such as silicon. A transistor is a semiconductor.

Sense lines---The wires in a core or core rope memory that sense the change in polarity during a read operation on the core and transmit the value of the core to a register.

Sequencer---A hardware device that commands other devices according to a fixed time or event-initiated sequence.

Serial Access---Transmission of information one bit at a time.

Serial Data---See *Serial Access*.

Sign Bit---A bit of a computer word reserved for indicating the sign of a number.

Signed Two's Complement---Storage of the value of a number in two's complement form with an associated sign bit.

Simulator---A device or software that replicates the functions of another device, though not necessarily at the same speed or in exactly the same way.

Single Precision---Representing the value of numbers using one computer word.

Soft Logic---The representation of logic in software.

Software---Part of a computer or other device that is the representation of algorithms of functions and problem solutions and that is easily changeable.

Software Engineering---The creation of software according to engineering principles; emphasizing proper specification, design, development, accuracy, and reliability.

Solid-State Computer---A computer built wholly out of solid-state devices such as transistors and integrated circuits as opposed to vacuum tubes.

Spike---A portion of the storage medium on a magnetic storage device such as a tape or disk ruined by an excessive electrical discharge or other event, preventing that area from being used for reading or writing.

State Vector---The current position of a spacecraft in three dimensions plus time.

Status Variable---A parameter indicating the current state of data processed by a module, either reliable or corrupted. For example, if a calculation within a module cannot be done because of insufficient or damaged data, a status variable can be set to a specified value and sent to the main program indicating that a failure has occurred and, often, what type of failure.

Stored Command Processor---One of the virtual machines of the Galileo spacecraft command and data computers that executes stored commands.

Structured Macros---A pseudo-high-level language made by naming routines written in assembly language after statements in a typical high-level language. These routines can then be invoked by name. Thus, an IF-THEN statement can be represented by an assembly language macro which accepts as its parameters the information to be tested and what to do after the result of the operation is known.

Subroutine---A software module that is part of a larger program, often repeated many times during the execution of the program.

Subroutine Linkages---Code that collects and connects subroutines together for execution.

Sumword---The result of adding the current values of specified commands together. Used to check against sumwords of other computers operating redundantly on the same processes.

Superminicomputer---A minicomputer with the speed and accuracy characteristics of a mainframe.

Telemetry---Signals sent from a spacecraft to the earth containing data gathered or generated by experiments and flight hardware.

Time-Sharing System---A method of allocating computer resources so that a number of processes can be executing simultaneously.

Time-Slice Method---A time-sharing or multiprocessing method in which each currently executing process is given a discrete length of time to use machine resources. At the end of that time, its execution is temporarily suspended and the next process in line is activated for a discrete length of time. Eventually, all current processes are serviced and the original interrupted program can pick up where it left off, beginning the cycle again.

Transfer Register---A register used for temporary storage of data, such as data read from a core memory, before it is sent to the device that requested it.

Transistor-Transistor Logic---A type of integrated circuit for representing logic in hardware.

Two's Complement---A method of storing binary numbers that first complements (reverses) each bit, and then adds one to the result. For

example, the two's complement of 101 is 011. This is useful for "subtracting by adding." To subtract a two from a five, the five (101) is added to the one's complement of two (110), with the carry past the left-most bit discarded. The result is 011, or three.

Uplink---Sending signals from the ground to a spacecraft.

Uplink Telemetry---See *Uplink*.

User Friendliness---A term relating to the degree of ease in the use of a computer system.

Very Large-Scale Integration---Combining millions of components on a single chip. Also "VLSI."

Virtual Machine---A system of managing machine resources so that many users have the impression that each has the full attention of the computer when in actuality it is rapidly servicing the processing needs of all of them. See *Virtual Memory*.

Virtual Memory---A system of memory management in which small segments of a program or data are brought from secondary mass storage into main memory on demand. For example, if a programmer is examining a very large program on an interactive terminal, the code for one page of the program is in real memory, with the rest on a disk drive. If the programmer moves to a different portion of the code, the computer automatically retrieves the correct segment of the code from secondary storage and places it in main memory. In this way, many users can be serviced with a main memory much smaller than secondary storage, each having the impression that large amounts of main memory are available to them. The speed of moving information from main memory to secondary storage is such that it is not usually noticeable.

Voter Circuit---A circuit in a device that has multiple identical logic paths, which compares the results of calculations and outputs the value that the majority of the input circuits carried. In triple modular redundant circuits, three logic paths are examined by voter circuits at frequent intervals in the machine.

Word---A single unit of information in a computer, made up of a number of bits. Small microprocessors often have 8-bit words; large mainframe computers, up to 64-bit words.

*Write---*The act of placing data in a storage device.

Appendix II: HAL/S, A Real-Time Language for Spaceflight

HAL/S is a high-level programming language commissioned by NASA in the late 1960s to meet the real-time programming needs of the Agency. At the time, programs used on board spacecraft were either written in assembly languages or in interpreted languages. The former make programs difficult to write and maintain, and the latter are insufficiently robust and slow. Also, future systems were expected to be much larger and more complex, and cost would be moderated by the use of a high-level language.

Since NASA directed the development of the language from the start, it influenced the final form it took and specifically how it handled the special needs of real-time processing. Statements common to other high-level languages such as FORTRAN and PL/1 were put in HAL. These included decision statements such as IF and looping statements such as FOR, DO, and WHILE. NASA added to the list of statements several specifically designed to create real-time processes, such as WAIT, SCHEDULE, PRIORITY, and TERMINATE. The objective was to make HAL quickly understandable to any programmer who had worked in other languages and to give a variety of tools for developing the new real-time programs. To make the language more readable by engineers, HAL lists source in such a way as to retain traditional notation, with subscripts and superscripts in their correct position, as contrasted with other languages, which force such notation onto a single level (see Fig. II-1.)

In addition to new statements, HAL provided for new types of program blocks. Two of these specific to real-time processing are COMPOOL and TASK. "Compools" are declarations of data to be kept in a common data area, thus making the data accessible to more than one process at a time. It was expected that several processes would be active at once and that many data items would need to be dynamically shared. Task blocks are programs nested within larger programs that execute as real-time processes dependent on one of the most powerful HAL statements, SCHEDULE.

Scheduling the execution of specific tasks was simplified by the syntax of HAL. Fig. II-2 shows the final page of the procedure STARTUP, written for use on the Galileo spacecraft attitude control computers, containing the master scheduling for the entire program. Note that the components of the SCHEDULE statement are the task name, start time, priority, and frequency. The statement "SCHEDULE ERROR0 ON RUPT0 PRIORITY(22);" tells the operating system to execute the task ERROR0 when an interrupt named RUPT0 occurs with a relative priority of 22. A different form of the SCHEDULE

statement is "SCHEDULE RG1 PRIORITY(12), REPEAT EVERY 6./90," which initiates the task handling the highest frequency rate group and repeats it 15 times per second. The statement TERMINATE cancels a specified task upon a designated interrupt or time.

HAL did not have the widespread use NASA had hoped for when the language was designed. Although the Shuttle on-board programs are exclusively in HAL, the Galileo attitude control system is the only other flight project to make significant use of the language. Other projects, though instructed to use HAL, found reasons to avoid it, although the Deep Space Network applied it to some ground software. In late 1985, NASA announced that the language of choice for the upcoming Space Station project would be Ada. Commissioned by the Department of Defense in the late 1970s to serve as a standard for all contractor software development, Ada includes real-time constructs pioneered by HAL such as task blocks, scheduling, and common data. The announcement made NASA the first nonmilitary agency to use Ada. Ada was adopted because commercial compilers were available and because the DoD's insistence on its use meant that it would be around for a long time. It appears that HAL will be phased out, destined to join the hundreds of other dead computer languages.

More information on the HAL/S language is contained in the following sources:

Intermetrics, Inc., *HAL/S-360 Compiler System Specification*, Version IR-60-7, February 23, 1981.

Intermetrics, Inc., *HAL/S Language Specification*, Version IR-542, September 1980.

Intermetrics, Inc., *HAL/S Programmer's Guide*, Version IR-63-5, December 1981.

Ryer, Michael J., *Programming in HAL/S* Intermetrics, Inc., Cambridge, MA, 1978.

ORIGINAL PAGE IS
OF POOR QUALITY

Figure II-1

```

M: general:
M: COMPOOL;
M: DECLARE num_vehicles CONSTANT(10);
M: STRUCTURE veh_state:
M:   1 time SCALAR,
M:   1 pos VECTOR,
M:   1 v VECTOR,
M:   1 accel VECTOR;
M: STRUCTURE vehicle:
M:   1 status,
M:   2 nav_state veh_state-STRUCTURE,
M:   2 mass SCALAR,
M:   2 electrical_systems BIT(12),
M:   2 computer_systems BIT(5)
M:   1 com_info,
M:   2 pilot_name CHARACTER(30),
M:   2 call_letters CHARACTER(10),
M:   2 receive_frequency INTEGER;
M: DECLARE ship vehicle-STRUCTURE(num_vehicles) LOCK(1),
M:   coord_trans MATRIX,
M:   possible_collision EVENT LATCHED INITIAL(OFF),
M:   nav_cycle EVENT,
M:   guid_cycle EVENT;
M: CLOSE general;
-----
M: read_accel:
M: PROCEDURE ASSIGN(loc) REENTRANT;
M:   DECLARE loc veh_state-STRUCTURE(num_vehicles);
M:   DECLARE a BIT(30) AUTOMATIC;
M:   DO FOR TEMPORARY veh = 1 TO num_vehicles;
E:
M:   CALL get_accel(veh) ASSIGN(a);
M:   loc.time = RUNTIME;
S:     veh
E:
M:   loc.accel = coord_trans VECTOR(SCALAR(a, a, a));
S:     veh;           10 AT 1  10 AT 1  10 AT 21
M: END;
M: CLOSE read_accel;
-----
M: gnd_startup:
M: PROGRAM;
M: STRUCTURE state:
M:   1 time SCALAR,
M:   1 pos VECTOR,
M:   1 v VECTOR,
M:   1 accel VECTOR;
M: DECLARE old_time ARRAY(num_vehicles) SCALAR,
M:   state state-STRUCTURE(num_vehicles),
M:   old_accel ARRAY(num_vehicles) VECTOR,
M:   t ARRAY(num_vehicles) SCALAR;
M: DECLARE collision_check FUNCTION BOOLEAN;

```

Figure II-1 (Continued)

```

M: CALL Kalman;
M: CALL new_state;
E:
M: IF collision_check THEN
M: DO;
C:     inform all interested processes of collision threat.
M:     SET possible_collision;
M:     SCHEDULE fast_nav IN 2 PRIORITY(35),
M:     REPEAT EVERY 2 UNTIL 10 FLOOR(RUNTIME / 10) + 9;
M:     END;
M: ELSE
M:     RESET possible_collision;

M: fast_nav:
M: TASK;           /*perform a fast intermediate update of the state vectors*/
M: [t] = {time} - [old_time];
E:               +
M: CALL read_accel ASSIGN({state});
C:     Update the entire array of position vectors and velocity vectors.
E:     -         -         -         -         -         2
M: {pos} = {pos} + {v} [t] + .25 ({accel} + [old_accel]) [t] ;
E:     -         -         -         -         -
M: {v} = {v} + .5 ({accel} + [old_accel]) [t];
M: CALL new_state;
M: CLOSE fast_nav;

M: new_state:
M: PROCEDURE;           /*internal procure to update the state vectors*/
M: [old_time] = {time};
E:     -         -
M: [old_accel] = {accel};

M: UPDATE;           /*use update block to access shared data incontrolled manner*/
E:     +         +
M: {ship.status.nav_state} = {state};
M: CLOSE;

M: collision_check:
M: FUNCTION BOOLEAN;           /*check if any pair of vehicles is too close together*/
M: DELCARE too_close SCALAR INITIAL(5000);
M: DO FOR TEMPORARY veh = 1 TO num-vehicles;
M:     DO FOR TEMPORARY other = veh = 1 TO num_vehicles;
E:     -         -
M:     IF ABVAL(pos     - pos     ) < too_close THEN
S:         veh;     other;
M:     RETURN TRUE;
M:     END;
M: END;
M: RETURN FALSE;
M: CLOSE collision check;

M: Kalman:
M: PROCEDURE;           /*perform a sophisticated but slow navigation algorithm*/
C:     .
C:     .
C:     .
M: CLOSE Kalman;

M: CLOSE nav;

```

ORIGINAL PAGE IS
OF POOR QUALITY

Figure II-2

```
C
C      SCHEDULE THE ERROR INTERRUPT SERVICE ROUTINE, 'RTI_UPT' .
C
C      112 CORRECTION #12
C      113 PROB 76
C
C      SCHEDULE ERRORF ON FAILURE PRIORITY(23);
C      SCHEDULE ERROR0 ON RUPT0 PRIORITY(22);
C      SCHEDULE ERROR1 ON RUPT1 PRIORITY(21);
C
C      SCHEDULE THE RTI INTERRUPT SERVICE ROUTINE, 'RTI_UPT' .
C
C      SCHEDULE RTI_RUPT ON RUPT4 PRIORITY(20);
C
C      SCHEDULE THE STAR INTERRUPT SERVICE ROUTINE, 'STAR_RUPT' .
C
C      SCHEDULE STAR_RUPT ON RUPT5 PRIORITY(16);
C
C      SCHEDULE THE 33.33 MS. RATE-GROUP, 'RGO'
C
C      ITB PROB #128: TEMPORARY CHANGE FOR RGO
C      FSW #210: 33 MSEC RATE GROUP AND SAFERATE VALUE
C      SCHEDULE RGO PRIORITY(13), REPEAT EVERY 3/90
C
C      SCHEDULE THE 66.66 MS. RATE-GROUP, 'RGI'
C
C      SCHEDULE RGI PRIORITY(12), REPEAT EVERY 6./90
C
C      SCHEDULE THE 133.3 MS. RATE-GROUP, 'RG2'
C
C      SCHEDULE RG2 PRIORITY(11), REPEAT EVERY 12./90.;
C
C      SCHEDULE THE 666.67 MS. RATE-GROUP, 'RG3'
C
C      SCHEDULE RG3 PRIORITY(8), REPEAT EVERY 2./3.;
C
C      SCHEDULE THE RAM CHECKSUM FUCTION TO RUN CONTINUOUSLY
C      (ONLY FOR AIAC NOT FOR FUNSIM)
C
A SCHEDULE CHKSUM PRIORITY(1);
CLOSE STARTUP;
```

02880000
02890000
02900000
02910000
02911000
02912000
02920000
02930000
02940000
02950000
02960000
02970000
02980000
02990000
03000000
03010000
03020000
03040000
03050000
03051000
03052003
03060003
03070000
03090000
03100000
03110000
03120000
03130000
03140000
03150000
03160000
03170000
03180000
03190000
03200000
03210000
03220000
03230000
03240000

Appendix III: GOAL, A Language for Launch Processing

GOAL is a high-level language that uses the terminology of test engineers to write tests and procedures to certify that a Shuttle vehicle is ready for launch. When the first automated preflight checkout programs were written in the mid-1960s, Marshall Space Flight Center originated ATOLL, a special high-level language for use in preparing test procedures. GOAL superseded that language in the early 1970s.

Fig. III-1 is a segment of a GOAL program used to safe various spacecraft systems if a NOGO condition causes the final countdown to be suspended. Note that names of data items held in common in the Launch Processing System appear within brackets, <>, and data local to the program is named between parentheses, (). Statements familiar to high-level programming language users, such as READ, IF-THEN-ELSE, and LET, have similar functions in GOAL. Additional statements, such as VERIFY, make it possible for the engineers to test whether valves or switches are set properly or whether a value is within a specified range. SET permits switches to be activated.

Although seemingly highly structured, GOAL allows engineers to frequently repeat the most common error of their peers using FORTRAN: excessive unconditional jumps such as the one on line 2030, making it difficult for someone to read and modify the program. Whereas in older versions of FORTRAN it was necessary to create structures such as those found between lines 2026 and 2039 to handle multiple statements in the THEN and ELSE blocks of a selection structure, later versions of the language and GOAL itself (see lines 1980 through 1988) permit multiple lines of code to be included within the blocks. Therefore, the GOTO statements are often used less to create structure than to provide a "quick fix" when the logic of the program needs expanding.

GOAL is used both at the Kennedy Space Center and Vandenberg Air Force Base in launch processing systems and is expected to last for the duration of the Shuttle program.

Further information about GOAL is contained in the following documents:

IBM Corporation, *Launch Processing System Checkout, Control and Monitor Subsystem Detailed Software Design Specifications, Book 2, Part 1: GOAL Language Processor*, KSC-LPS-IB-070-2, pt. 1, release S33, Cape Canaveral, FL, June 3, 1983.

IBM Corporation, *Launch Processing System Checkout, Control, and Monitor Subsystem: GOAL On-Board Interface Language*, KSC-LPS-OP-033-4, release S33, Cape Canaveral, FL, April 27, 1983.

400 COMPUTERS IN SPACEFLIGHT: THE NASA EXPERIENCE

Figure III-1

GOAL LANGUAGE PROCESSOR SOURCE INPUT LISTING

```
RECORD      SOURCE RECORD
1951
1952          SEND INTEGER <NO12INTGR> TO CONSOLE <GOXARM>;
1953
1954          $ SEND NOTIFICATION OF "OK TO START GOX ARM EXTEND"
1955          VIA REMOTE COMM INTERRUPTS TO ECS CONSOLE.      $
1956
1957          RECORD TEXT (BFS      PASS      LDB      ) TO <PAGE-B>
1958          LINE 5 COLUMN 46 INVERT WHITE;
1959
1960          READ <NGPCLMCNFG>,
1961          <V98U2408C1>,
1962          <V90Q8001C1> AND SAVE AS (LDB), (BFS), (PASS);
1963
1964          VERIFY <SGPCAREA1> IS ON AND <SGPCFIDA1> = 21,
1965          BEGIN SEQUENCE;
1966
1967          IF (PASS) = 102,
1968          RECORD (PASS) TO <PAGE-B> LINE 5 COLUMN 60 INVERT RED;
1969          ELSE
1970          RECORD (PASS) TO <PAGE-B> LINE 5 COLUMN 60 INVERT GREEN;
1971
1972          END SEQUENCE;
1973
1974          ELSE
1975          RECORD (PASS) TO <PAGE-B> LINE 5 COLUMN 60 INVERT WHITE;
1976
1977          VERIFY <SGPCAREA2> IS ON AND <AGPCFIDA2 IS BETWEEN 12 AND 13,
1978          BEGIN SEQUENCE;
1979
1980          IF (BFS) =102,
1981          BEGIN SEQUENCE;
1982
1983          ASSIGN (BFS SAFING) = ON;
1984          RECORD (BFS) TO <PAGE-B> LINE 5 COLUMN 50 INVERT RED;
1985
1986          END SEQUENCE;
1987          ELSE
1988          RECORD (BFS) TO <PAGE-B> LINE 5 COLUMN 50 INVERT GREEN;
1989
1990          END SEQUENCE;
1991
1992          ELSE RECORD (BFS) TO <PAGE-B> LINE 5 COLUMN 50 INVERT WHITE;
1993
1994          RECORD (LDB) TO <PAGE-B> LINE 5 COLUMN 69 INVERT GREEN;
1995
1996          INHIBIT PROGRAM LEVEL INTERRUPT CHECK FOR <PFPK2>
1997          <PFPK3>
1998          <PFPK5>;
1999
2000          LET (APUNOGO) = 0;
```

ORIGINAL PAGE IS
OF POOR QUALITY

Figure III-1 (Continued)

GOAL PROCESSOR SOURCE INPUT LISTING

RECORD	SOURCE RECORD
2001	
2002	IF (GLS EVENT COMPLETED) IS GREATER THAN
2003	(ET SRB RSS IGN S A TO ARM) \$ -04:58 \$
2004	THEN GO TO STEP 5150; \$ PRIMARY SAFING \$
2005	\$ IMMEDIATE SAFING OF
2006	SRB IGN S/A DEVICE REQUIRED ? \$
2007	
2008	IF (GLS EVENT COMPLETED) IS GREATER THAN
2009	(FWD CMD DCDR PWR OFF) \$ -10 SEC \$
2010	THEN GO TO STEP 5103;
2011	
2012	LET (C3ERR) = 0;
2013	
2014	VERIFY <N03IS091E> IS ON, GO TO STEP 5103;
2015	
2016	RECORD TEXT (CMD DECODERS PWR ON)
2017	TO <PAGE-B>
2018	LINE 5 COLUMN 0 YELLOW;
2019	
2020	SET (PWR UP AFT CMD DECODER) FUNCTIONS TO ON;
2021	
2022	DELAY 0.5 SEC;
2023	
2024	SET (PWR UP FWD CMD DECODER) FUNCTIONS TO ON;
2025	
2026	IF (C3ERR) IS NOT EQUAL TO 0 THEN GO TO STEP 5102;
2027	
2028	MODIFY <PAGE-B> LINE 5 COLUMN 0 TO COLUMN 22 GREEN;
2029	
2030	GO TO STEP 5103;
2031	
2032	STEP5102 RECORD (CMDERR GSE)
2033	TO <PAGE-B>
2034	LINE 5 COLUMN 23 RED;
2035	
2036	STEP5103 IF (GLS EVENT COMPLETED) IS GREATER THAN
2037	(FWD MDM LOCKOUT) \$ -35 SEC \$
2038	THEN GO TO STEP 5107;
2039	
2040	VERIFY <N03IS100E> IS ON, GO TO STEP 5107;
2041	
2042	LET (3CERR) = 0;
2043	
2044	RECORD TEXT (UNLOCK SRB FWD MDMS)
2045	TO <PAGE-B>
2046	LINE 6 COLUMN 0 YELLOW;
2047	
2048	UNLOCK SRB MDM FOR <B75K3065XL> CRITICAL;
2049	
2050	UNLOCK SRB MDM FOR <B75K3066XL> CRITICAL;

Appendix IV

Mariner Mars 1969 Flight Program

Figure IV-1

This segment of a Mariner programmable sequencer flight program is given as an example of the sort of flexibility gained by adding a memory to the system. The first segment, the Executive, is only seven lines, yet it essentially controlled the software. The remaining code demonstrates a typical subroutine. The entire length of this program was 128 lines.

```

LOC  OPC  A/TIME B/EVENT      SYMB  OPC  A/TIME B/EVENT

*COM      EXECUTIVE ROUTINE                                ( 7 WORDS)
*COM
0  CLJ    35      102      EXC0  CLJ    36      REQM      ABORT+EXTRA FE+NAA TESTS, SLEWS
1  DSJ    126     104      EXC1  DSJ    OV       ROJ1      OV,M/C,OPT FE TEST,TV PIC CTR
2  DHJ    125     31       EXC2  DHJ    LCH1   CR01     CRUISE AND POST ENCOUNTER EVENTS
3  DHJ    11      8        EXC3  DHJ    CY04   CY01     Y1 CYCLIC GENERATOR
4  CLJ    256     12       EXC4  CLJ    256    RT01     READOUT TEXT,NF NON SLEW EVENTS
5  CLJ    2       18       EXC5  CLJ    2      RDIN     TEST FOR EVENT ADDRESS READIN
6  HLT    368     489     EXC6  HLT    368    489     END OF SCAN

*COM
*COM      END OF CRUISE CHAIN
*COM
7  ROJ    2       3       ENDC  ROJ    EXC2   EXC3     END CRUISE SEQUENCE

*COM
*COM      CYCLIC SUBROUTINE FOR Y1 EVENTS
*COM
8  TAB    10      11      CY01  TAB    CY03   CY04     ENTRY----RELOAD CYCLIC TIME
9  UNJ    0       4       CY02  UNJ    0      FXC4     RETURN TO EXECUTIVE PROGRAM
10 DATA  ****    288     CY03  DATA  FILL   0024     CYCLIC TIME STORAGE
11 DATA  ****    288     CY04  DATA  FILL   0024     COUNTING LOCATION FOR Q1 ENENLK

*COM
*COM      ENABLE FAR ENCOUNTER*ENTRY PT. IS CYEO        ( 11 WORDS)
*COM
20 ADD    48      112     OFE0  ADD    OFE4   CZ11     ADD 5HRS TO TIME OF N1(2)
21 ADD    27      109     OFE1  ADD    OFE3   CZ14     ADD F3 EVENT TO CZ14 EVENT TIME
22 TAB    29      79      OFE5  TAB    OFE5   CGC6     MOD CYCLE GEN FOR OPT FE(CTR)
23 TAB    30      84      OFE6  TAB    OFE6   CGD4     MOD CYCLE GEN FOR OPT FE(EVENT)
24 CLJ    16      20      CYE0  CLJ    16     OFE0     TEST FOR OPTIONAL FE(DC-32)
25 DAJ    71      26      CYE2  DAJ    CTR4   CYE3     UPDATE CYCLE GENERATOR (EVENT)
26 DAJ    69      68      CYE3  DAJ    CTR2   CTR1     UPDATE CYCLE GENERATOR (COUNTER)
27 DATA  0       8       OFE3  DATA  0      0200     STORAGE FOR F3 EVENT (OPTION)
28 DATA  5       0       OFE4  DATA  5      0000     TIME STORAGE FOR OPTIONAL FE SHFT
29 CTJ    ****    43      OFE5  CTJ    FILL   ROJ0     COUNT FOR OPTIONAL FE PICTURES
30 DATA  ****    451     OFE6  DATA  FILL   3007     DATA FOR OPTIONAL FE PICTURES

*COM

```

ORIGINAL PAGE IS
OF POOR QUALITY

1. Report No. NASA CR-182505		2. Government Accession No.		3. Recipient's Catalog No.	
4. Title and Subtitle Computers in Spaceflight - The NASA Experience				5. Report Date March 1988	
				6. Performing Organization Code	
7. Author(s) James E. Tomayko				8. Performing Organization Report No.	
				10. Work Unit No.	
9. Performing Organization Name and Address Department of Computer Sciences Wichita State University Wichita, KS 67208				11. Contract or Grant No. NASW-3714	
				13. Type of Report and Period Covered Contractor Report	
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546				14. Sponsoring Agency Code	
15. Supplementary Notes					
16. Abstract This book examines the computer systems used in actual spaceflight or in close support of it. Computer systems used in administration and in aeronautical and other research not directly related to spaceflight are ignored. Each chapter deals with either a specific program, such as Gemini or Apollo onboard computers, or a closely related set of systems, such as launch processing or mission control. A glossary of computer terms is included.					
17. Key Words (Suggested by Author(s)) assembly language modularization core memory virtual memory core rope time-slice method image processing bubble memory			18. Distribution Statement Unclassified  Subject Category 60		
19. Security Classif. (of this report) Unclassified		20. Security Classif. (of this page) Unclassified		21. No. of pages 405	22. Price