NASA-CR-152107

UNCLASSIFIED

FINAL REPORT

NUMERICAL AERODYNAMIC SIMULATION FACILITY

PRELIMINARY STUDY EXTENSION

February 1978

(NASA-CR-152107) NUMERICAL AERODYNAMIC SIMULATION FACILITY. PRELIMINARY STUDY EXTENSION Final Report (Burroughs Corp.) 273 p HC A12/MF A01 CSCL 01A

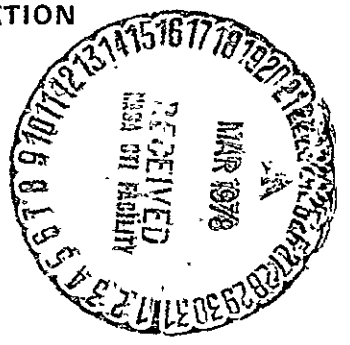
N78-19051

Unclas G3/02 08629

Prepared under Contract No. NAS2-9456 by
Burroughs Corporation
Paoli, Pa.

for

AMES RESEARCH CENTER
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION



UNCLASSIFIED

FINAL REPORT

NUMERICAL AERODYNAMIC SIMULATION FACILITY

PRELIMINARY STUDY EXTENSION

February 1978

Prepared under Contract No. NAS2-9456 by
Burroughs Corporation
Paoli, Pa.

for

AMES RESEARCH CENTER
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

CONTENTS

CONTENTS (Cont'd)

CHAPTER ONE

INTRODUCTION AND SUMMARY

1.1 INTRODUCTION

Burroughs Corporation is pleased to present this report which is the result of work carried on under an extension to contract No. NAS2-9456, a preliminary study for a Numerical Aerodynamic Simulation Facility. The primary objective of this extension is to produce an optimized functional design of key elements of the candidate facility defined in the Final Report[1] of the basic contract. This is accomplished by effort in the following tasks:

- To further develop, optimize and describe the function description of the custom hardware.
- To delineate trade-off areas between performance, reliability, availability, serviceability and programmability.
- To develop metrics and models for validation of the candidate systems performance.
- To conduct a functional simulation of the system design.
- To perform a reliability analysis of the system design.
- To develop the software specifications to include a user level high level programming language, a correspondence between the programming language and instruction set and outline the operation system requirements.

The results of this effort are presented in five separate chapters:

Chapter 2. Functional Description includes a summary of the system parameters, block diagrams, descriptions, of the major elements and the instruction set with detailed timing.

Chapter 3. Software Issues describes the extensions and restrictions on the FORTRAN language and compiler at the functional level a discussion of converting statements in extended FORTRAN into machine language and a statement regarding the operating system.

Chapter 4. Simulations presents the models, metrics and methodology for conducting the simulation along with preliminary results.

Chapter 5. Reliability includes two sections. The first presents the results of an availability analysis of the systems and the second present further discussion of the error detection, correction and control to be employed.

Chater 6. Trade-offs delineates and discusses a large number of design and operating factors for which reasonable alternatives exist.

While the information in this report is designed to stand alone, it is also considered to be a supplement to the Final Report (Ref. 2) of the basic NAS2-9456 contract where appropriate, reference is made to this report rather than to unnecessarily repeat previously reported information.

In addition, it should be pointed out that certain terminology used in the previous report have been revised. The new terms are:

- Flow Model Processor (FMP). This is the portion of the system previously called the Navier-Stokes Solver (NSS).

- Processor Data Memory (PDM) was previously called Processing Element Memory (PEM)
- Processor Program Memory (PPM) was previously called Processing Element Program Memory (PEPM)
- Execution Unit (EU), the logic portion of the array processor, formerly called Processor Element (PE).

The following sections summarize the chapters in additional detail.

1.2 FUNCTIONAL DESIGN

TheFMP is an array processor of 512 processors, a control unit, and 521 modules of extended memory, as described in Reference 1. The major additions found in Chapter 2, to the description of reference 1, are, first, the provision of SECDED, instead of parity-plus-retry, as the expected means of error control in the processors' memory, second, the addition of four on-line spare processors as definitely a part of the design (they are mentioned briefly as a possibility in reference 1); third, significant revisions and additions to the instruction set; fourth, the restriction of the extended memory instructions to fetching 512 words (one per processor) per instruction, (the earlier description had EM instructions fetching 512 X N words per instruction); and fifth, provision for special hardware for computing any floating-point variables that are not members of a vector.

Chapter 2 includes diagrams and figures of every element of the FMP.

1.3 SOFTWARE

The software chapter covers the FORTRAN language, to a depth necessary to cover simple test cases, discusses hand compiling, and is charged with the task of reporting on progress in defining the operating system during this contract extension. Three and only three extensions are visualized for the initial FORTRAN language. First, the DOALL construct declares to the compiler that the iterations of a particular loop can be done in any sequence, or all in parallel, without affecting the result; second, declarations of several types of use of variables are used to allocate those variables among the different types of memory; third, certain system library functions are required, because of the parallel nature of the machine, that would not be required in serial FORTRAN. None of these library functions are required for the initial benchmarks.

The operating system is extensively described in reference 1. The level of detail in that document is such that the effort of the contract extension was spent more fruitfully on language definition, compiler considerations, and hand compilation procedures. Thus, the operating system discussion in reference 1 still stands as the best description so far produced of the operating system of the FMP. No attempt has been made to update that description for this report.

1.4 SIMULATION

Chapter 4 discusses the separation of the simulation effort into two levels, instruction and FMP level, and the system level. Metrics for each level are discussed, and SUBROUTINE TURBDA has been selected as the metric for the simulation done in this extensionis also given. The BOSS simulator, in which our simulation is being done, is described briefly in chapter 4.

1.5 RELIABILITY

A detailed computer model for the reliability of the FMP was run. The results of this model bound the availability at 96 percent being the lower limit of availability using pessimistic assumptions, and better than 99 percent availability being achieved under the most optimistic assumptions. The use of spare processors with operating system automatic restart (assumed successful for some fraction of all attempts) produces a very significant improvement over the model that has no spare processors.

The reliability section also includes a discussion of the use of SECDED in all memory, of the process of "scrubbing" out the errors that spontaneously arise in CCD storage (DBM), and of other error control strategems that are used in the FMP.

1.6 TRADEOFFS

Chapter 6 discusses tradeoffs in many areas. These include ease of programming versus execution efficiency, where one wishes to have most of both, word and instruction formats, error control methods versus their cost in reduced throughput, several specific design issues, relative speeds of specific blocks of the system, alternate methods of supplying the floating-point scalar capability, and other topics, with a final section on the expansibility of both the specific FMP, once built, and the expansibility of the design from which it was built.

CHAPTER 2
FUNCTIONAL DESCRIPTION OF NSS HARDWARE

2.1 INTRODUCTION

This functional description is arranged in several successive sections. First, a brief system description of the SAM that is the baseline system for FMP is given. Second, a brief list of system parameters is provided. Third, the elements of the system block diagram are each described in turn. Fourth, the instruction set of the FMPis given, together with its timings.

In all of this, it has not been felt necessary to repeat material that is found in the final report of contract NAS2-9456, except very briefly to refresh the reader's recollection. It is presumed that the reader has first read that report.

No design should be considered to be necessarily final if further investigation should show that the machine performs better with the feature modified. Chapter 6, "Tradeoffs", is a discussion of many of the features that will be studied in simulation during phase 2 (time permitting), and which are therefore likely to be modified in the direction of higher throughput if the baseline system is found wanting.

This functional description is intended to provide the base for the information input to a performance simulation of the SAM of the FMP. Some of the information, such as error correction capabilities, is included for completeness in spite of the fact that it has no apparent involvement in a performance simulation.

2.2 BASIC SYSTEM PARAMETERS

Most of the basic system parameters were covered in some detail in the final report Ref. 1. They are summarized here along with additional information of specific interest.

2.2.1 Logic Family - ECL is the preferred logic family. Final selection of circuits for implementation at this time would only lock us into choices that will become obsolete by 1979-1980 when the design is completed. We do not wish to preclude the use of up-to-date technology in the actual design. If the final design were being implemented at this time, Fairchild's 100K series would be chosen, together with compatible memory circuits. The chip count projected for 1979-1980 is the one assigned to the baseline system. Confidence in this package count is supported in most cases by the very similar chip count, of circuit types already available in 1977 (usually ECL 100K), which are also given.

2.2.2 Clock Rate - The clock has been assigned a 40 ns period. The instruction times, given below in terms of this clock period, are compatible with the instruction times derived from a preliminary processor design using ECL 100K.

2.2.3 Cabling Methods - The same flat belts used successfully in prior projects in Burroughs for transmitting high-speed signals with fast rise time and low crosstalk will be used for most of the interunit cabels. Reference 1 discusses this choice.

2.2.4 Power - While a number of comments on power were included in reference 1, certain detailed information was not. These details are provided in the following statements.

- Switching regulators will be used for the sake of efficiency. A net efficiency of 65% is expected from the total power supply.
- DBM is provided with whatever power is required to make it nonvolatile against glitches and short power outages. Since CCD is proposed for DBM, battery backup would be highly desirable.

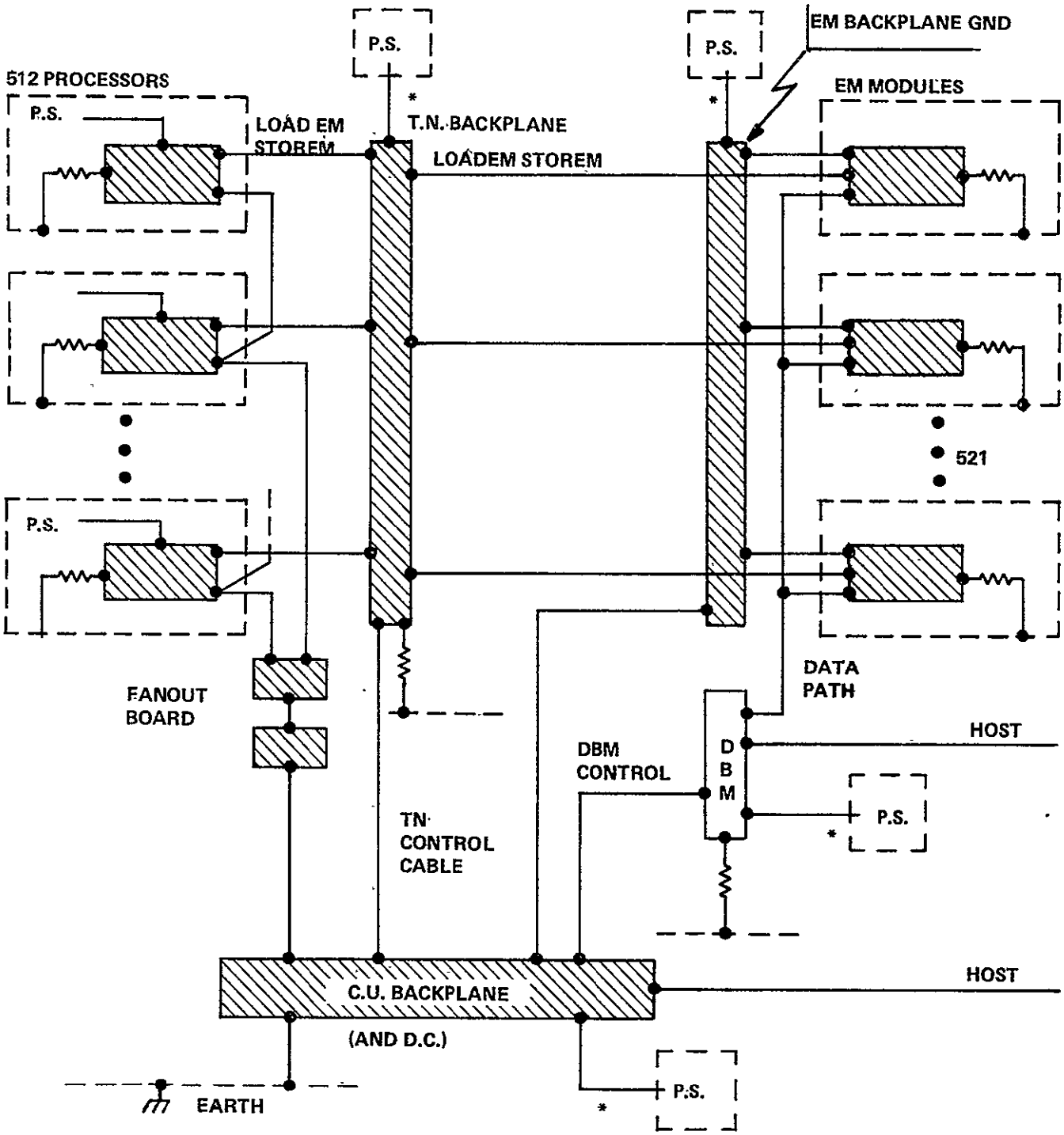


Figure 2-1. Grounding

- The ground return from backplane to power supply is never used as part of the path that connects one backplane ground to another backplane ground. Figure 2-1 shows the grounding arrangements expected

- Total power for the FMPis estimated (very approximately) at 250 kw, based on an average of 0.8w for each of the 200,000 circuit packages, and 65% efficiency in the power supply. These are for the 1980 projected circuit counts.

- Every module has its signal ground tied to chassis so that there will be no floating grounds when the modules are tested as stand-alone modules. In Figure 2-1 these ties are shown as resistors.

A requirement on power supplies employed at NASA AMES is that they must ride through the undervoltage transients produced by wind tunnel motor startup, and not pass voltage spikes. In addition, they should be reasonably respectful to the source. Several power supply configurations satisfy this requirement.

- Motor-generator set. Inertia enables an M-G set to ride through large transients. The inefficiency of the M-G set is multiplied into the inefficiency of the system power supplies. The advantage of an M-G set is that it can be added to a system after the fact, without impacting any existing design.

- Transformerless rectifiers, like the old AC-DC radio, require a filter capacitor, which suppresses spikes, and if large enough, will ride through undervoltage transisents. The unregulated DC (about 280v) is distributed around the equipment and used as input to individual switching regulators. SCR rectifiers are to be avoided, since they inject noise back into the line.

. Battery back-up Uninterruptible Power Supply (UPS).

Of the three schemes, the transformerless rectifier is most efficient, and takes the least space. It also has the advantage that back-up batteries can be supplied to a selected subset of the equipment (DBM, in this case). It is also easy to make the rectification redundant. Three-phase full wave rectifiers are actually six-phase for ripple characteristics. They often need no chokes, and have wide conduction angles in the rectifier diodes.

2.2.5 Number of Processors - A key decision in the design of the FMP is the choice of the number of processors to be implemented. The design presented here is based on using the fastest processor that is consistent with the speed of memory built of 16k-bit static RAM chips. Projecting 100 ns speed for such chips, we arrive at a 360 ns floating point multiply as being approximately in balance. A faster processor would yield increased speed only if the memory were changed to the faster 4k-bit chips, implying a four-fold increase in the number of components in memory. Reliability, even more than cost, tells us to keep the parts count down, and therefore to design a system consistent with 16k-bit memory chips. It takes about 512 processors, at these speeds, to yield the desired billion floating point operands per second with sufficient margin for inefficiencies.

2.3 OVERVIEW OF FUNCTIONAL DESCRIPTION

2.3.1 Block Diagram

Figure 2-2 (a slightly expanded copy of Figure 1-2 of the Ref. 1) shows the array processor consisting mostly of 512 processors attached by a switch, the Transposition Network, to 521 Extended Memory modules which hold the main data base of the program. Used

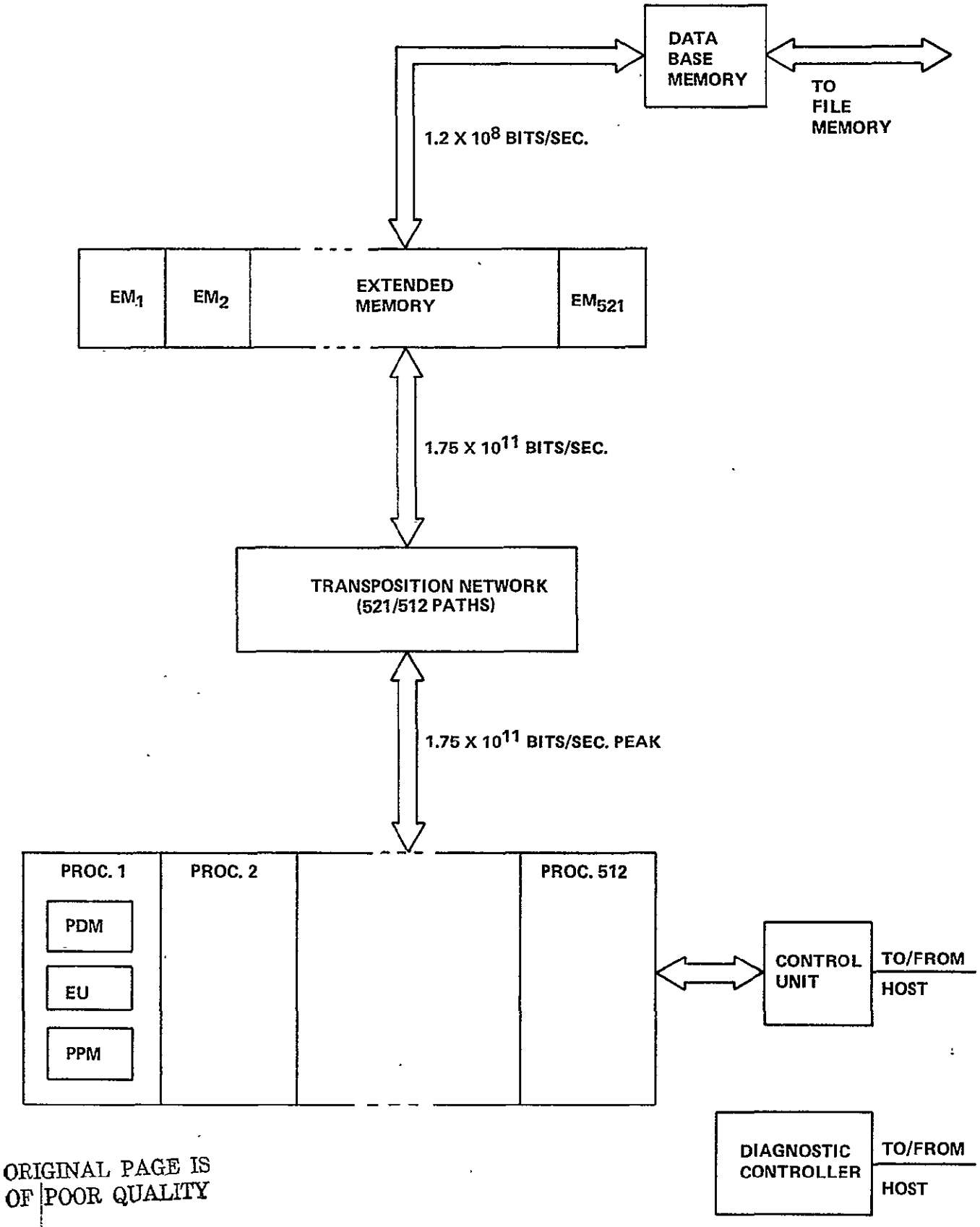


Figure 2-2. SAM Block Diagram

as a staging area for jobs not yet started, and as the output area for jobs in process or completed, is Data Base Memory. A Control Unit synchronizes the action and controls the transposition network and the transfers in and out on both faces of the extended memory. The controller for the Data Base Memory also accepts requests from the host processor to transfer to and from the host disk pack file system. The Data Base Memory controller resolves access conflicts to and from data base memory. The Control Unit resolves accesses to and from Extended Memory. There is also a Diagnostic Controller used for maintenance and cold starts.

Each processor is self-contained, with integer and floating-point arithmetic units, its own instruction decoder, its own program memory, and its own data memory. In addition to the 512 processors, four processors are included as on line spares to help achieve system availability requirements. The use of these on-line spare processors is discussed in Chapter Five.

2.3.2 Instruction Streams

As described in Ref. 1, the FMP is controlled by two instruction streams, which are created in parallel by the compiler from a single sequence of source statements. One instruction stream is being executed in the control unit; the other is being executed by all processors asynchronously of each other. Some statements in the source code result in instructions in both instruction streams. Examples are "CALL subroutine", or an arithmetic statement using an EM variable, and therefore requiring a fetch to all processors from the EM. Some of these joint instructions require that the control unit and the processors synchronize themselves. It has been observed that reference 1 does not seem to be clear in explaining synchronization, nor in explicating the means of accomplishing it. Therefore, the discussion digresses here to a detailed discussion of the synchronization mechanism.

2.3.3 Synchronization

The process of synchronization occurs within instructions. It involves two signal lines which go from the control unit to all processors, namely "CUready" and "go". "CUready" is a level, "Go" is a pulse that arrives at all processors simultaneously. From each processor there are two lines, "Enabled" is a copy of the "enabled" flipflop that exists in each processor; "I got here" is a signal, a level, which is raised during the execution of some instructions.

To explain the process, consider the example of a LOADEM instruction fetching N words from EM. In the control unit, the LOADEM causes the raising of the "CUready" line as soon as the TN controls have been set to the proper value. In each processor where "enabled" is true, "I got here" is raised as soon as the processor starts executing the LOADEM instruction.

When any processor executing LOADEM sees "CUready" true, the processor sends the address through the TN to the EM module that is connected to this processor. The strobe accompanying the address causes the loading of the address within the EM module.

An "all processors ready" signal, marking the time at which the last enabled processor arrives at the LOADEM instruction is created for the CU (The logic creating this signal is actually contained within the fanout tree). Using E_n as the "enable" bit of the nth processor, and H_n as the "I got here" line of the nth processor, the "all Processors ready" signal is given by the formula

$$\text{All-processors-ready} = (H_1 \text{ OR } \overline{E_1}) \text{ AND } (H_2 \text{ OR } \overline{E_2}) \text{ AND } \ldots \text{ AND } (H_{512} \text{ OR } \overline{E_{512}})$$

There is also "any processor enabled", the OR of all the "enable" bits.

When the CU sees "all processsors ready", the CU issues, after an appropriate delay to let addresses be loaded, a series of N "read" commands to the EM module and also issues, appropriately timed with respect to the last such command, a "go" pulse to the processors. In the processor, we load N words under control of the N strobes coming from EM module through the TN. The "go" signals the end of the instruction.

As a second example, consider the instruction WAIT. Here no processor action timed to the "CUready" is required, so the CU sends no "CUready". When the CU sees the "all processors ready" signal formed from the "I got here"s and the "enable"s, it issues a "go" to all processors, who have refrained from executing their next instruction until the "go" is received.

When the processor has raised its "I got here" line, but before it has received a "go" signal, it is said to be "waiting". The "I got here" line is dropped upon receipt of the "go" pulse.

In addition to the above synchronization, the CU also has the power to transmit commands. The commands are carried on a 4-bit-wide bus accompanied by a strobe line. Many of these commands are used in the diagnostic programs. Ref. 1, p 4-27, has a tentative list of operations called forth by these commands. Some of these commands will be conditional on the "enable" bit of the processor, some are unconditional independent of the enable bit. The only such command that is used in user-generated FORTRAN programs is the command that simultaneously loads the program counter and sets the enable bit.

The control unit's command power is exerted over all processors at once, not over individual processors. Processors that do not join in some array-wide operation avoid it by a) jumping around the operation, if it is local to each processor, b) executing certain

instructions (LOADEM, STOREM, SHIFTN) as noops conditional on the last bit of an integer register in the processor, or c) executing the STOP instruction, which turns off the "enable" bit until the CU reaches some point in its instruction stream that turns it back on.

There is also an interrupt line from processor to CU.

2.3.4 Starting a Run

During normal operation, all data and program for the next run will be loaded into data base memory prior to the beginning of the run. When the run starts, system software in the CU loads program from data base memory to the memory of the control unit (via extended memory). The initialization phase of the program then transfers necessary data to extended memory, and transmits the processors' program to them. These actions are automatically inserted by the compiler and the linker. With data in place in extended memory, and allocated space initialized to "invalid"N and with code files in place in control unit and processors, user execution starts.

2.3.5 FMP Hardware Summary

The Flow Model Processor therefore consists of

- One Control Unit (CU) with its own memory (CUM) with optional scalar processor capability.
- 512 Processors, (plus 4 spares) each with its own Processor Data Memory (PDM) and Processor Program Memory (PPM)
- One Transposition Network

- 521 Extended Memory modules
- One Data Base Memory and Controller
- One Diagnostic Controller

All of the above is shown in Figure 2-2 except for the optional scalar processor and the four spare processors. The scalar processor is an ingredient of the design which was not needed in order to successfully match the SAM to the aerodynamic flow models. Since the scalar processor was not discussed in reference 1, further discussion thereon is found in Chapter 6.

2.4 INDIVIDUAL BLOCKS

Following is a brief description of each of the elements of the FMPtogether with a formatted tabulation of pertinent features and a block diagram of each.

2.4.1 Description of Tables

For each element of the FMP, there is a table of characteristics given. A very short narrative description gives the intended function of the element in user programs. Source of control is identified, and the storage capabilities, both capacity and speed, are also given. Connectivity to other elements is broken down to a rather detailed level, with each group of signals that has an identifiably different function being so identified. In some cases, such as CU to processor, signals in the same belt are identified as a different group in order to more clearly identify their use.

The table also discusses the mode of error control built into the design. Some mechanisms of error control were included in the baseline system design in the final report. Some further mechanisms of error control are proposed in Chapter 5. This section represents a particular state of the design, not the final state.

Two chip counts are given. The 1979-1980 projected chip count is the one projected for the baseline system. The second chip count, using parts now existing in 1977, is given only for corroboration, to indicate the reasonableness of that projection. It also represents the chip count of the FMP if design were frozen now. There are also in some cases estimates of the power drain. All these are included only for interest. These are preliminary. They have no direct bearing on the performance evaluation simulation.

"TBD" means "to be determined".

2.4.2 Processor The array of 512 processors is charged with the task of executing the user computations in the program, namely the floating-point operations on the problem variables.

The processor executes code contained in its own program memory, and accepts commands from the control unit. Certain instructions (see Table 2-13) are executed in synchronism with the control unit (and hence, by implication, in synchronism with the entire array, since the control unit expects cooperation from all processors.)

The actions of the processor are delineated by the instruction set in the next section. Figure 2-3 shows pictorially the division of the processor into and execution unit, a data memory, and a

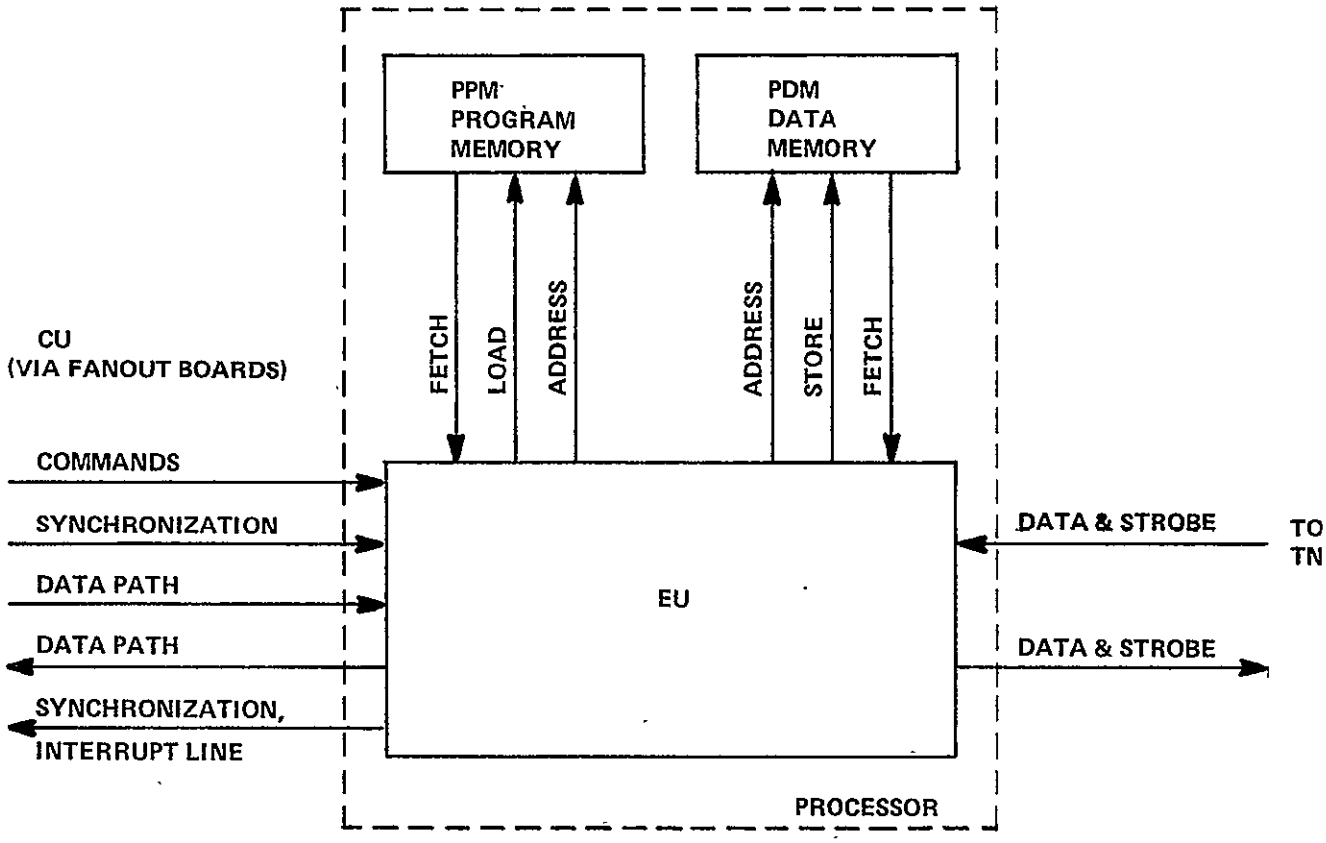


Figure 2-3. Processor Block Diagram

program memory. Figure 2-4 is a block diagram of the logic part of the processor, showing the independent integer and floating point units, with separate register files for each. Figure 2-5 is a diagram of the instruction fetching and overlap machinery, which is explained at length below in connection with the timing of instruction execution. The logic portion of the processor has been named the "execution unit." Table 2-1 provides data on the EU.

Connections to the processor come from the control unit and the transposition network. A byte-wide (8-bit) data path is found both from (BDCST) and to (HVST) the control unit. The synchronization signals discussed previously also come from the control unit. The 4-bit wide command path, and its strobe, also come from the control unit. The data paths to (STOREM) and from (LOADEM) the transposition network are each accompanied by a strobe. In addition, each processor is connected to backplane wiring that expresses its own number. Of the 129 processors in a cabinet, any one may be the spare processor. Suppose processor no. N is the spare processor. Then the backplane number for processors 0 through N-1 is correct, but the backplane number for processors Nl through 128 must be shifted own by one, to N through 127, in order that the processors being used by the program be consecutively numbered. Therefore, there is a one-bit signal coming from the switching machinery which tells the processor whether or not to subtract 1 from its hard-wired processor number to correct for the location of the spare.

Error control within the processor consists of bounds checks, reasonableness checks, and consistency checks, as listed in Ref. 1. See Sections 6.7 and 6.8 for further checks that may be implemented but at some cost in throughput.

For justification of the 1977 component count, see appendix E of volume II of reference 1.

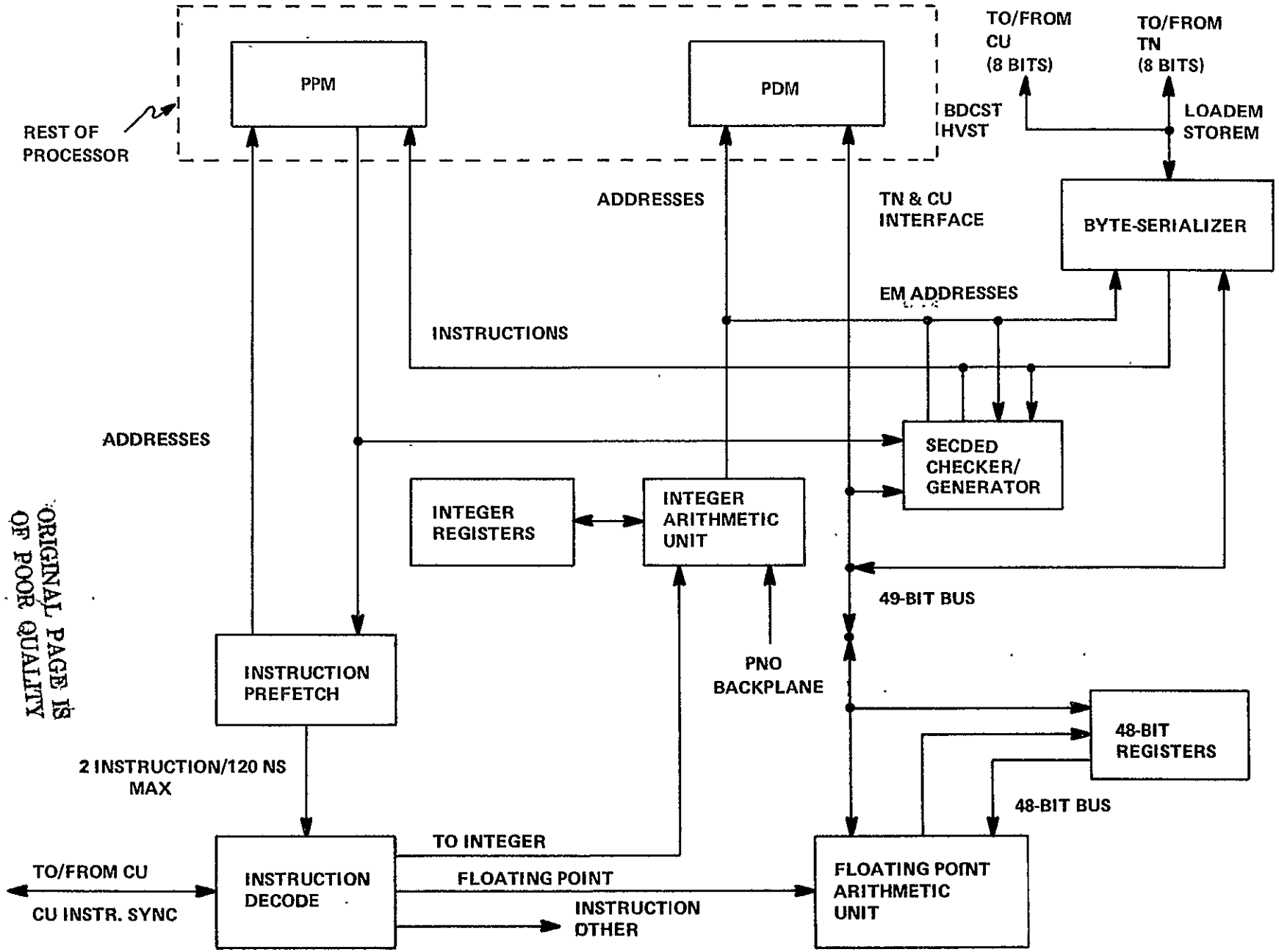


Figure 2-4. Internal Block Diagram of EU

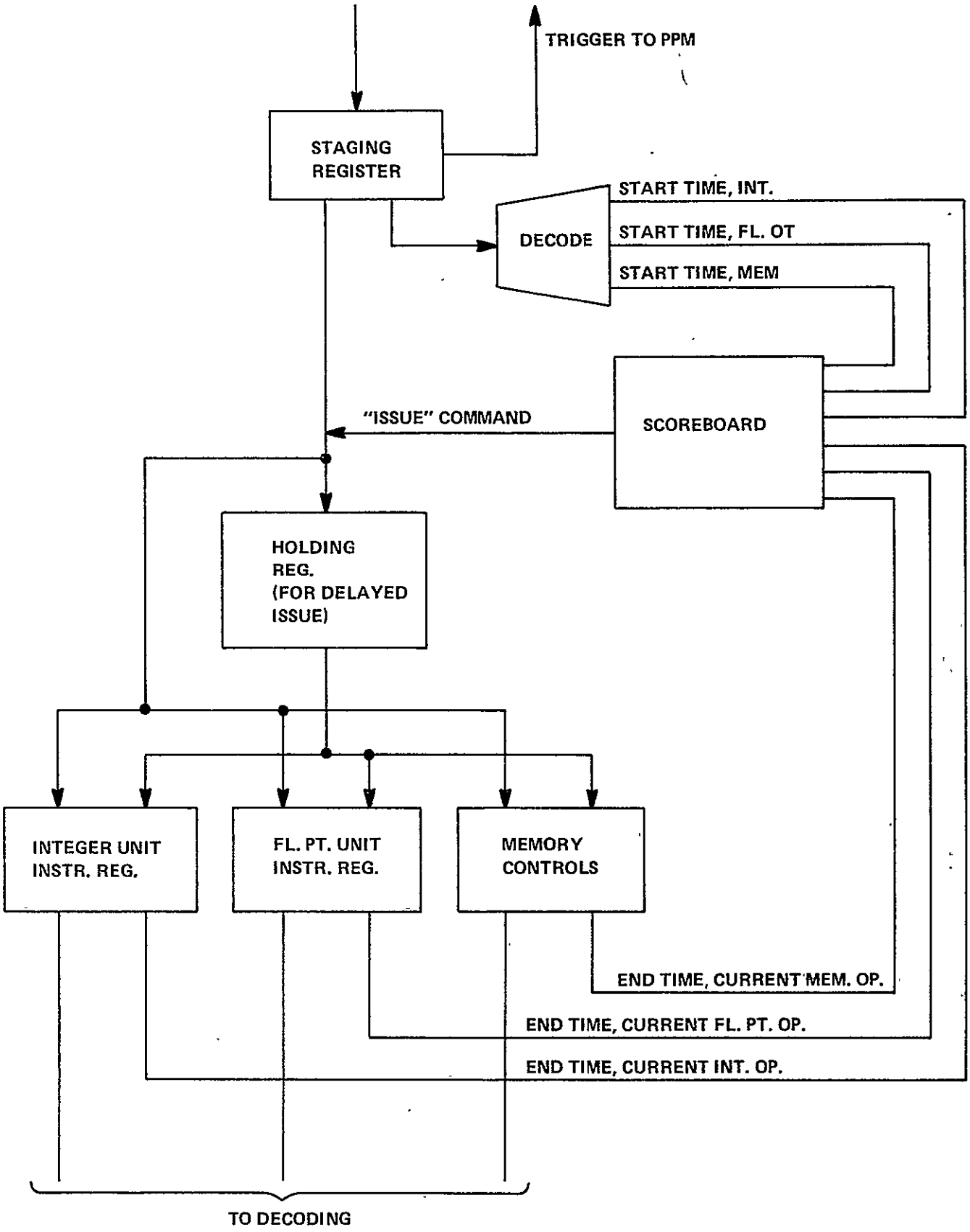


Figure 2-5. Instruction Fetching and Overlap

TABLE 2-1
EXECUTION UNIT CHARACTERISTICS

UNIT: Execution Unit (EU) No. In System: 512 + 4 on-line spares

FUNCTIONAL CHARACTERISTICS

Function: This is the logic portion of the processor, all the processor except memory. It executes code that has been written by the FMP FORTRAN compiler, including EM address computations, index calculations and floating point operations.

Source of Control; During User Program: Program stored in PPM, sync's from the CU.
During System Startup and Diagnostics: Same plus CU commands

Storages; Capacity: 16 16-bit integer registers
16 48-bit floating point registers
Other registers (see text)
Speed: Multiple accesses each 40 ns clock

Connectivity to Other Elements:

#	Path	To or From	No. Sig	Timing	Primary Use
1	BDCST	From CU	8	byte/20ns	Receive global variables from CU
2	HVST	To CU	8	byte/20ns	Transmit result to CU (global)
3	LOADEM	From TN	9	byte/20ns	Receive data from EM
4	STOREM	To TN	9	byte/20ns	Transmit data to EM
5	CUinstr	From CU	4	TBD	Primarily for diagnostics
6	sync	To CU	4	edge	Synchronization
7	sync	From CU	4	edge	Synchronization
8	PEno	Wired to backplane	9	D.C. level	Processor's own number

RELIABILITY/REPAIRABILITY/TRUSTWORTHINESS

Error Control Methods: TBD. Modulo 3 check on arithmetic is being evaluated. Error cases are detected (see text).
Repair Methods: Replace and restart from restart point. On-line replacement (with manual pull-and-replace at a later convenience of the repairman) is very feasible.
MTBF of Unit: See Chapter 5.
Degraded Modes Available: Programs can be compiled to use less than all the processors available, thereby bypassing any failed processors. On-line switching of spare processors.

PHYSICAL

Chip Count; 1980 Projection: 100 If use 1977 parts: 160 (100K ECL etc.)
(based on preliminary logic design using 100K)
Pysical Size: 1980: One large pc. sized module. 1977: Single removable module

Power Drain: 1980: 150 w 1977: 300 w

2.4.3 Processor Data Memory - The processor data memory (PDM) contains work space for each processor. It is also used to hold local copies of global information, to facilitate their being fetched by the processor's program. It can be used to window data from EM. Control is from the memory address register in the processor. There are 16384 words of 55 bits, consisting of 48 bits data and 7 bits of single-error correcting, double-error-detecting code. Data address, and control connections are solely to the processor. 16k-bit static RAM chips are used. Figure 2-6 shows some of the logic in the processor associated with the port into PDM. Table 2-2 describes major characteristics of the PDM. See sections 6.6, 6.12, 6.13 for discussion of tradeoffs in PDM design.

2.4.4 Processor Program Memory. Processor Program Memory (PPM) contains the code file from which the processor executes. It is addressed directly by the program counter. Overlay comes from the CU via the "broadcast" (BDCST) path. Except for the size of 8192 words, design is identical with that of PDM.

2.4.5 Control Unit (CU)

2.4.5.1 Basic Control Unit

The control unit, during user programs, is in charge of synchronizing the array for those instructions that require a synchronized array; it issues the "go" signal. It also handles those portions of the address computation that must be issued from a central point. The control unit executes the FMP-resident portion of the system software. It has a single shared memory (CUM) for both program and data.

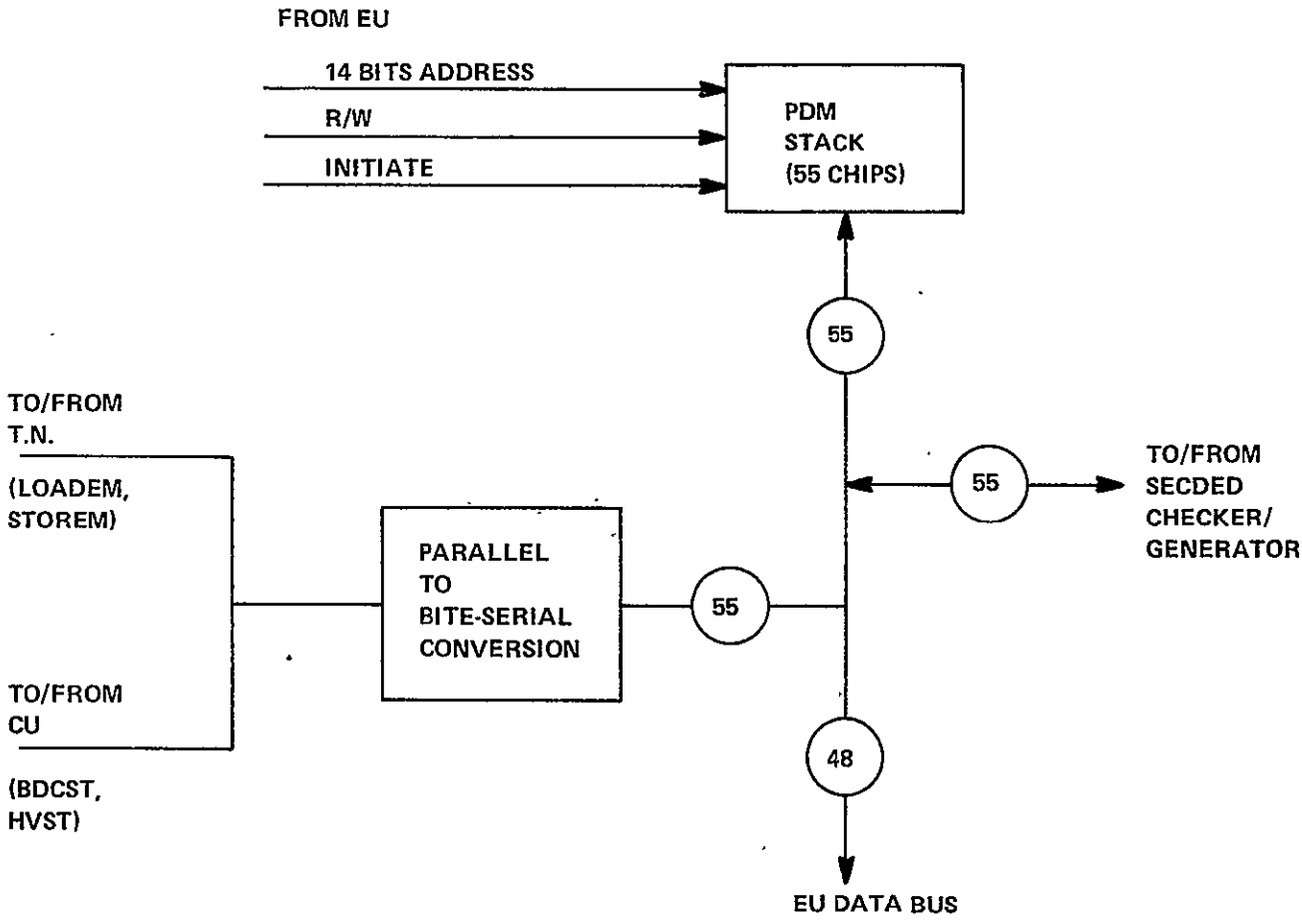


Figure 2-6. PDM Logic

TABLE 2-2
CHARACTERISTICS OF PROCESSOR DATA MEMORY

UNIT: Processor Data Memory (PDM) No. In System: 512 + 4 spares with spare processor (formerly processing element memory PEM)

FUNCTIONAL CHARACTERISTICS

Function: Stores temporary variables generated by the processor during computation. Work space. Subroutine return information. Windows EM data.

Source of Control; During User Program: EU command lines
During System Startup and Diagnostics: Same

Storages; Capacity: 16,384 words.
Speed: 120 ns cycle

Connectivity to Other Elements:

#	Path	To or From	No. Sig.	Timing	Primary Use
1	data	To/from EU	55	static	Fetch and store data
2	address	From EU	16	static	Address
3	control	From EU	2	edge or static	Command

RELIABILITY/REPAIRABILITY/TRUSTWORTHINESS

Error Control Methods: SECDED
Repair Method: Removed with entire processor. Not a separate entity.
MTBF of Unit: Dominated by control chips because of SECDED.
Degraded Modes Available: Programs compiled to less than 512 processors bypass failed PDM's. Error correction allows program to continue, but with reduced reliability, in single-bit failure cases. On-line switching of failed processors.

PHYSICAL

Chip Count; 1980 Projection: 70 (55 16k-bit mem + 15 control)
If use 1977 Parts: 250 (100K ECL, etc.) (220 4k-bit mem. + 30 control)

Physical Size; 1980: Part of processor assy. 1977: Part of processor assy.
Power Drain; 1980: 1977:

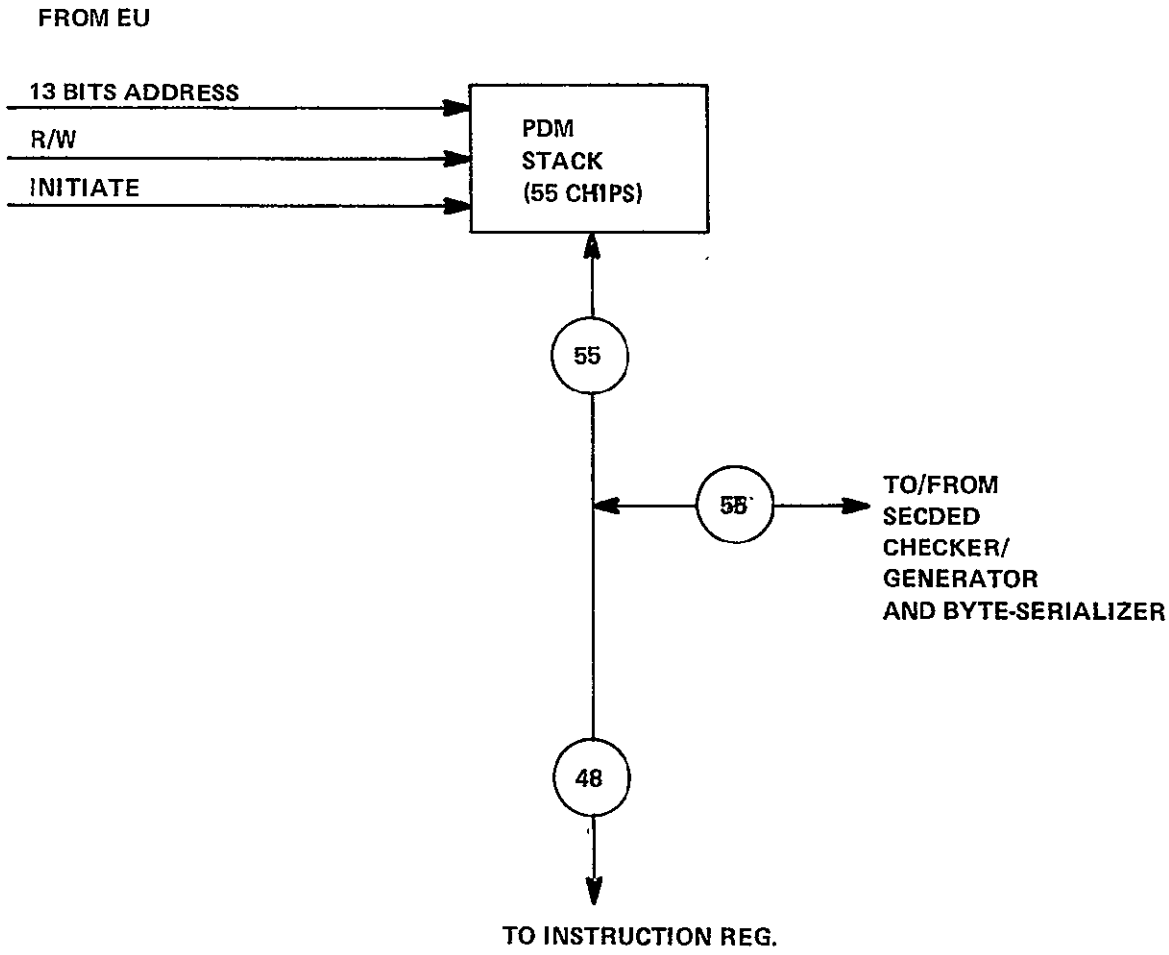


Figure 2-7. PPM Logic

TABLE 2-3
PROCESSOR PROGRAM MEMORY CHARACTERISTICS

UNIT: Processor Program Memory (PPM) No. In System: 512 + 4 spares with spare processor

FUNCTIONAL CHARACTERISTICS

Function: Contains program for the processor. Is loaded using the BDCST path from the CU.

Source of Control; During User Program: Processor's program counter.
During System Startup and Diagnostics: Same

Storages; Capacity: 8,192 words
Speed: 120 ns

Connectivity to Other Elements:

#	Path	To or From	No. Sig.	Timing	Primary Use
1	program	To/From EU	55	static	Fetch and load program
2	address	From EU	16	static	Address
3	control	From EU	2	edge or static	Command

RELIABILITY/REPAIRABILITY/TRUSTWORTHINESS

Error Control Methods: SECDED
Repair Method: Remove with entire processor. Not a separate entity.
MTBF of Unit: See Chapter 5
Degraded Modes Available: Program compiled to less than 512 processors bypass failed PPM's. Error correction allows program to continue at reduced reliability, in single bit failure cases. On-line switching of failed processors.

PHYSICAL

Chip Count; 1980 Projection (28 mem + 15 control)	43	If use 1977 parts: 140 (100K ECL, etc.) (110 mem + 30 control)
Physical Size; 1980:	Part of processor assy.	1977: Part of processor assy.
Power Drain; 1980:		1977:

The control unit can also be controlled by commands from the host computer issued via the Diagnostic Controller (DC). This mode of operation is supplied for the purpose of performing diangostics.

The control unit is at once the most complex, in terms of variety of functions performed, and the most pedestrian, in terms of the demands it makes on the logic designer, of all the units in the FMP. Such hand analysis as has been done indicates that for the aerodynamic flow problems, the control unit will most of the time be waiting on the processors. One of the aims of the simulation is to find out if this statement is really true, or whether an investment in a faster control unit will pay off.

The frequency with which the CU executes system software upon interrupt, in the middle of user executions, will affect the required speed of the CU. The present plan is to so allocate the tasks in the system that during normal executions no interrupts either from host or resulting from FMP code are expected.

The host initiates file-system-to-DBM transfers using its copy of the DBM allocation map and issuing I/O commands directly to the DBM controller. No FMP-resident routine is involved in the initiation or completion of these transfers. The DBM controller resolves any potential conflict between these host transfers and a CU-initiated DBM-EM transfer.

Figure 2-7 is the block diagram of a control unit built around a single bus for transferring all data to and from memory, and using this same bus for one of the register file outputs. Such a structure defeats overlap but simplifies design. If simulation were to show that a faster CU is needed, a faster CU would be built.

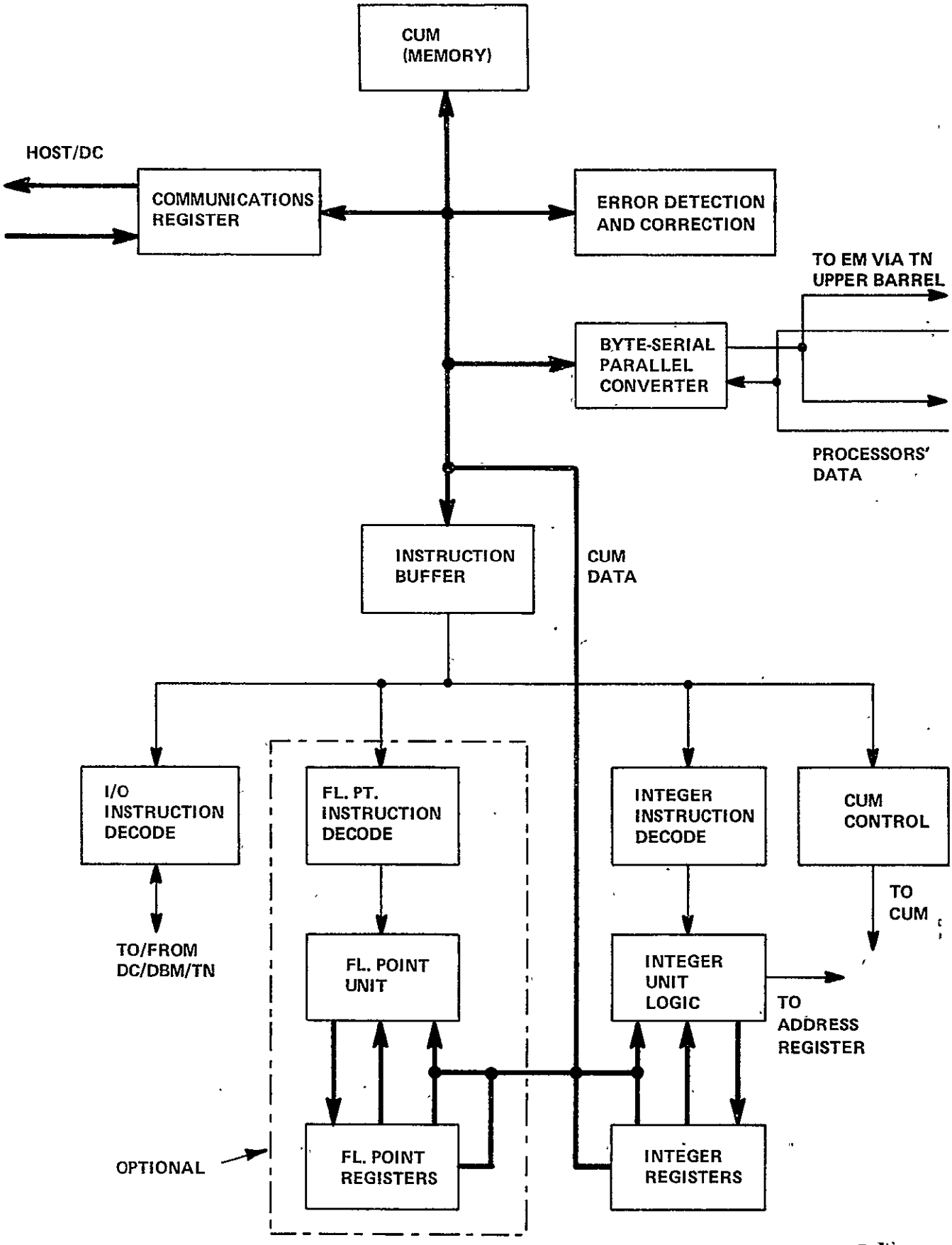


Figure 2-8. CU Block Diagram

In addition to the portion shown in Figure 2-8, the control unit also contains a section which resolves conflicts for EM between the instructions of the NSS and the needs of the DBM controller.

The control unit has four semi-independent execution stations, just as the processor has three. The degree to which the execution of the independent sections is to be overlapped is a subject for study during simulations in future work. Using the two aerodynamic flow models as benchmarks tells us that no overlap is required, therefore specifying an exact mechanism of overlap has been deferred. The four units are:

* Integer Unit
* Memory Control
* Floating Point Unit (optional, can be omitted if it is determined that so called scalar processor capability is not required for the contemplated applications. See Section 6.5)
* Interface to host and DBM controller

Instruction timing is given in the next section, 2.5. Table 2-4 lists the features of the CU.

2.4.5.2 Scalar Processor

Floating point scalars are an item of concern in some applications. In the baseline system, an optional design feature to handle floating-point scalars is a floating-point arithmetic capability in the control unit. For a discussion of other options for attaching scalar capability to the FMP, see section 6.16. Scalar floating point capability is not be be confused with the "scalar unit" found in some other designs. The addressing and control functions of such a "scalar unit" are included in the control unit here whether or not the floating-point option is included.

TABLE 2-4
CONTROL UNIT CHARACTERISTICS

UNIT: Control Unit: (CU) No. In System: 1

FUNCTIONAL CHARACTERISTICS:

Function: Executes the non-array portion of the FMP program. Executes the FMP resident portion of the system software.

Source of Control; During User Program: Program stream contained in Control Unit Memory
During System Startup and Diagnostics: Same plus commands issued from Diagnostic Controller

---

Storages; Capacity: Integer Register file, perhaps 16 words, exact number to be determined by simulation. Floating point register file of 16 words.
Speed: Single-clock access to two registers per file. 40 ns clock.

Connectivity to Other Elements:

#	Path	To or From	Sig.	Timing	Primary Use
1	control	To DBM Controller	TBD	TBD	Control of DBM-EM transfers
2	return	From DBM Controller	TBD	TBD	Completion, error, EM conflict resolution
3	control	To EM	TBD	TBD	Control of EM
4	return	From EM	TBD	TBD	Monitoring, errors, interrupt
5	control	To TN	13	TBD	Control of TN
6	STORCU	To TN	9	byte/20ns	Data to be stored in EM
7	LOADCU	From TN	9	byte/20ns	Data fetched from EM to CU
8	command	To Processor	4	TBD	Diagnostic commands to the processor
9	sync	To Processor	4	edge	Synchronization of array
10	sync	From Processor	4	edge	Synchronization of array
11	BDCST	To Processor	8	byte/20ns	Broadcast data
12	HVST	To Processor	8	byte/20ns	Data (such as global max) to CU

RELIABILITY/REPAIRABILITY/TRUSTWORTHINESS

Error Control Methods: TBD
Repair Method: TBD. Repair in place; FMP is down until CU repaired
MTBF of Unit: See Chapter 5
Degraded Modes Available: None.

PHYSICAL

Chip Count; 1980 Projection: 3,000 chips (a coarse estimate) If use 1977 parts: 4,000 chips (100k ECL, etc.)
Physical Size: 1980 1977:
Power Drain: 1980 1977:

The FORTRAN language and compiler of chapter 3 makes no use of the floating-point option in the CU, as there was no use for it in the four codes used for benchmarking.

2.4.6 Control Unit Memory (CUM)

The control unit memory holds both program and data for the control unit. It is addressible only from the control unit, and sends all data into the central data bus of the control unit.

The control unit memory is identical in electrical design and uses the same 16k-bit RAM chips as the processor memories. Its size is subject to verification via simulation. The size resulting from considerations of the flow-model matching study is 32,768 words.

The control unit memory is initially loaded from DBM at the beginning of each run using a routine which is itself resident in CUM and executes on the CU. The routine transfers data and program from DBM to CUM via EM.

Data on the control unit memory is found in Table 2-5.

2.4.7 Extended Memory Module

Extended memory (EM) is the "main" memory of the FMP, in that it holds the data base for the program during program execution. Temporary variables, or work space, can be held in either EM or PDM, as appropriate to the problem. All I/O to and from the FMP is to and from EM via DBM. Control of the EM is from two sources, the first is instructions executed in the CU, the second is the DBM controller which handles the DBM-EM transfers. In the baseline system design, the DBM-EM rate is such that the CU can be given first priority into EM without losing any of the DBM-EM transfers, therefore, the CU instructions have priority in the EM.

TABLE 2-5
CHARACTERISTICS OF CONTROL UNIT MEMORY

UNIT: Control Unit Memory (CUM) No. In System: 1

FUNCTIONAL CHARACTERISTICS

Function: Contains data local to the CU, and CU's program. Also contains processor program as source for overlay during runs. Holds mailbox for host-FMP communication. Holds copy of DBM allocation map.

Source of Control; During User Program: CU
During System Startup and Diagnostics: Same plus may be accessed by DC if CU not running

Storages; Capacity: 32,768 words.
Speed: 120 ns cycle

Connectivity to Other Elements:

#	Path	To or From	No. Sig.	Timing	Primary Use
1	data	To/from CU	55	static	Fetch and store data
2	address	From CU	16	static	Address
3	command	From CU	2	edge or static	Command

RELIABILITY/REPAIRABILITY/TRUSTWORTHINESS

Error Control Methods: SECDED
Repair Method: FMP is down while CUM is down. Must replace failed modules for FMP to recover.
MTBF of Unit: Dominated by control logic because of SECDED
Degraded Modes Available: Error correction allows program to continue at reduced reliability; in single-bit failure cases.

PHYSICAL

Chip Count; 1980 Projection: 175 chips (110 mem + 15 control)
If use 1977 parts: 470 (100 ECL, etc.) 440 mem + 30 control)
Physical Size; 1980: TBD
1977: TBD
Power Drain; 1980:
1977:

EM consists of 521 identical modules, which are accessed in parallel. 521 is a prime number for the sake of allowing efficient parallel fetching for all vectors of any length (with the minor exception of any vectors that happen to have elements spaced apart in memory by exactly 521).

From each EM module we need a transfer rate and access time consistent with the most economical implementation. For the baseline system, an implementation in 64k-bit dynamic RAM was chosen, as being the most economical implementation available by 1980. The low chip count also enhances reliability. Projections say that a 64k-bit chip will have 250 ns cycle time by that date. The 280 ns cycle time of the memory is compatible with the 140 ns per word transfer rate through the transposition network. Each word carries single- error-correction-double-error-detection code, which is generated at the source (DBM, CU, or processor) and also checked there, so that transfer paths are covered by the same error control as the contents of EM.

Having decided on a TN that is almost twice as fast as the EM module, it would be possible to build the EM module in two interlaced submodules, if it the streaming mode of fetching were to see much use. Section 6.10 discusses the tradeoff between implementing or not implementing this streaming mode of access. The baseline system as described in this document avoids the complexities of a design suitable for streaming, which includes among other things, a capability of incrementing the address in the EM module by nonunity increments. The chip count of table 2-6 does not include any incrementer.

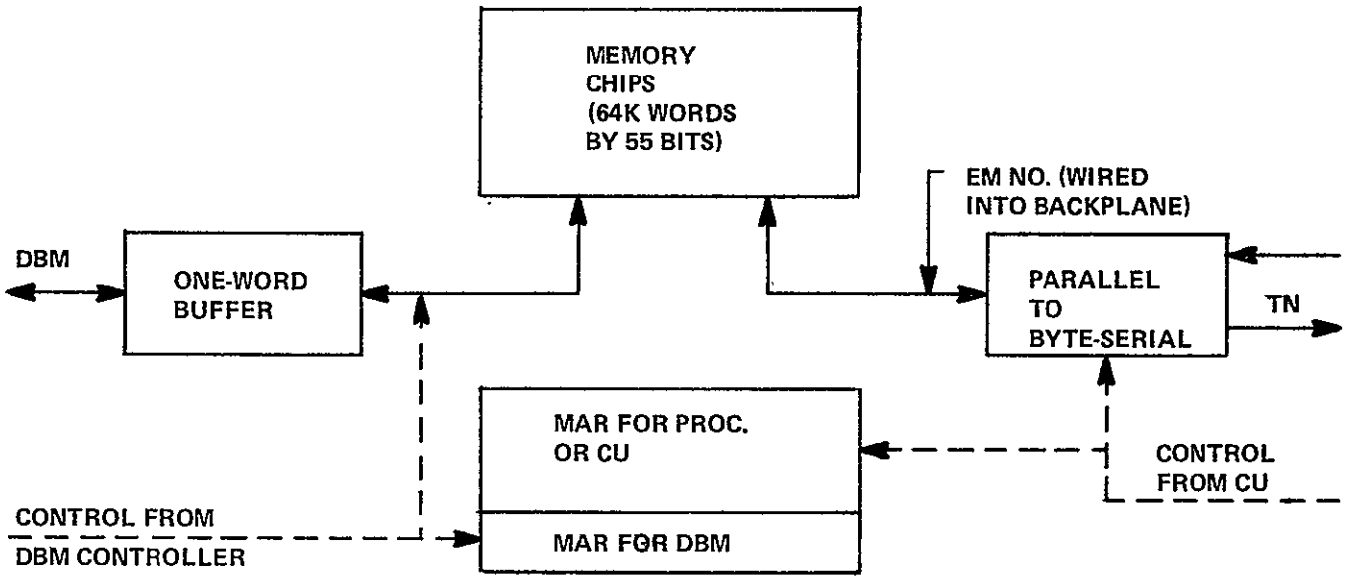


Figure 2-9. EM Module

TABLE 2-6
EM MODULE CHARACTERISTICS

UNIT: EM Module No. in System: 521

FUNCTIONAL CHARACTERISTICS

Function: Stores problem data base during program executions. Most nearly corresponds to "core" of conventional processor.

Source of Control; During User Program: Receives commands from CU
During System Startup and Diagnostics: Same

Storages; Capacity: 65,536 words
Speed: Access time 200-250 ns, interlaced for 140 ns/word block transfer

Connectivity to Other Elements:

#	Path	To or From	No. Sig.	Timing	Primary Use
1	LOADEM	To TN	9	byte/20ns	Fetching data to processors and CU
2	STOREM	From TN	9	byte/20ns	Storing data from processors and CU
3	---	To DBM	9	full word in 400 ns	Results back to DBM
4	---	From DBM	9	full word in 400 ns	Initial data (and eventually, overlay) from DBM
5	No	From backplane	10	D.C. level	Module's own number
6	Control	From CU	TBD	TBD	Controls EM operations

RELIABILITY/REPAIRABILITY/TRUSTWORTHINESS

Error Control Methods: SECDED (providing acceptable error rates are demonstrated)
Repair Method: Remove and replace
MTBF of Unit: Control dominates failure modes because of SECDED.
Degraded Modes Available: Data continues to be corrected even when there is one hard error, allowing the current program to complete before repairs are undertaken.

PHYSICAL

Chip Count; 1980 Projection: 86 (55 memory + 30 control)
If use 1977 parts: 274 (100K ECL, etc.) (224 mem. + 50 control)
Physical Size; 1980: One medium sized p.c. board
1977:
Power Drain: 1980
1977:

Figure 2-8 shows the EM module, including two address registers, a one-word buffer for DBM transfers, and an access path to the EM modules own number, wired into the backplane. Table 2-6 gives the data on the EM module.

2.4.8 Fanout Tree

A series of fanout boards is supplied to provide the CU to processor connection. From CU to processor,s signals fan out to a final 512 destinations. From the processors, the signals are combined, so that, within the CU, a single result appears in response to 512 signals emitted by the processors. For example, the "all processors ready" signal becomes true at the clock that the last enabled processor emits "I got here". Another such signal is the 512-input OR of "enabled".

At the processor, some signals are wired per-processor directly to the last level of fanout board; others are daisy-chained to eight processors from a single signal pin on the last board. The fanout boards are pin-limited. Simple buffers with one input pin and one output pin per signal dominate the circuit count, so hex buffers, easily available today, will not be improved upon by 1979-1980.

Data on the fanout tree is in Table 2-7. The figure demonstrating the fanout tree is Figure 2-10.

2.4.9 Transposition Network

The transposition network allows the fully parallel, 512-wide, fetching of sets of variables that are to be processed in parallel. Up to 512 elements in one-dimensional vectors of any type can be fetched at full speed in parallel. When DOALL loops have two index variables, two-dimensional subsets of multidimentional arrays can also be fetched in parallel. For details, see Ref 1, and Chapter Three.

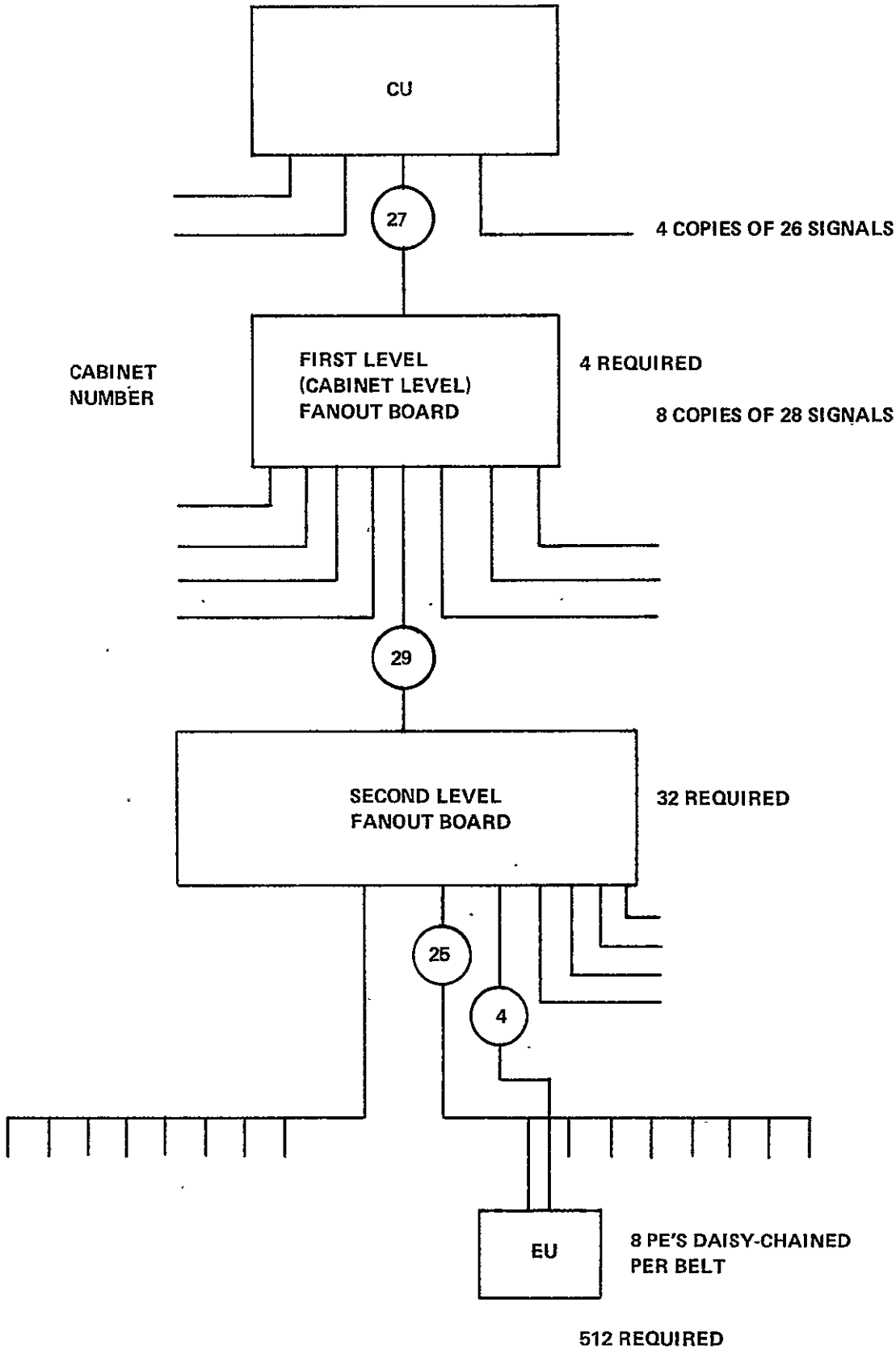


Figure 2-10. Fanout Tree

TABLE 2-7
FANOUT TREE CHARACTERISTICS

UNIT: Fanout Tree, CU to Processors No. In System: 1

FUNCTIONAL CHARACTERISTICS

Function: Provides fanout for signals from CU to the 512 processors; accepts signals from the 512 processors and combines them appropriately for the CU. Consists of 36 boards.

Source of Control; During User Program: No control; all passive logic.
During System Startup and Diagnostics: Same

Connectivity to Other Elements:

#	Path	To or From	No. Sig.	Timing	Primary Use
1	command	From CU	4	TBD	Diagnostic
2	sync	From CU	4	edge	Synchronization of array
3	sync	To CU	4	edge	Synchronization of array
4	BDCHT	From CU	8	byte/20ns	Broadcast data
5	HVST	To CU	8	byte/20ns	Data to CU (such as global MAX)
6	command	To proc. 8's	4(x 64)	TBD	Diagnostic
7	sync	To proc. 8's	4(x 64)	edge	Synchronization of array
8	sync	From proc.	4(x 512)	edge	Synchronization of array
9	BDCST	To proc. 8's	8(x 64)	byte/20ns	Broadcast data
10	HVST	From proc. 8's	8(x 64)	byte/20ns	512-input OR of data from processor to CU. 1st 8-way OR done on proc. wiring

RELIABILITY/REPAIRABILITY/TRUSTWORTHINESS

Error Control Methods: SECDED on broadcast and harvest data.
Repair Method: Remove and replace of defective boards.
MTBF of Unit: See Chapter 5.
Degraded Modes Available: None

PHYSICAL

Chip COunt; 1980 Projection: 2,000 chips all small scale integration. Dominated by 1,504 hex buffers.	If use 1977 parts: 2,000 chips (100K ECL, etc.)	
Physical Size; 1980: 32 cards of 60-80 chips each	1977: Same	
Power Drain; 1980: 1.6 kw	1977: Same	

The transposition network consists of 521 switchable data paths from EM to processor, and another 521 data paths from processor to EM. There are two 10-bit control registers, one for offset of the starting element, and one for skip distance. Since there are two sets of data paths, the first from processor to EM module, and the second from EM module to processor, the settings of the two paths could be separately controlled. There is just one instruction that would go faster if both paths are used simultaneously with different settings, namely SHIFTN (see Table 2-10 and 2-11 for a description). SHIFTN is used in functions that operate "horizontally" across the parallelism of the array, such as global sum, global maximum, or global product. SHIFTN would also be used to implement a Fast Fourier transform on the FMP. In the aero codes used as benchmarks, there is very little use of SHIFTN, so there is no justification for having separate settings for the first and second data paths, and bidirectional data paths would serve as well.

A three-bit command register enables the following commands:

1. Enable transfers between processor and EM. The presence or absence of actual transfer is signified by the presence or absence of a signal on the strobe line that accompanies each byte-wide signal path.
2. Enable transfers between CU port and EM.
3. Enable transfers between the remaining eight paths and EM (built into the design to allow these eight ports to service the scalar processor).
4. Broadcast from selected EM module to all processors.

Table 2-8 gives the characteristics of the transposition network. Figure 2-11 shows the barrel switches that implement it.

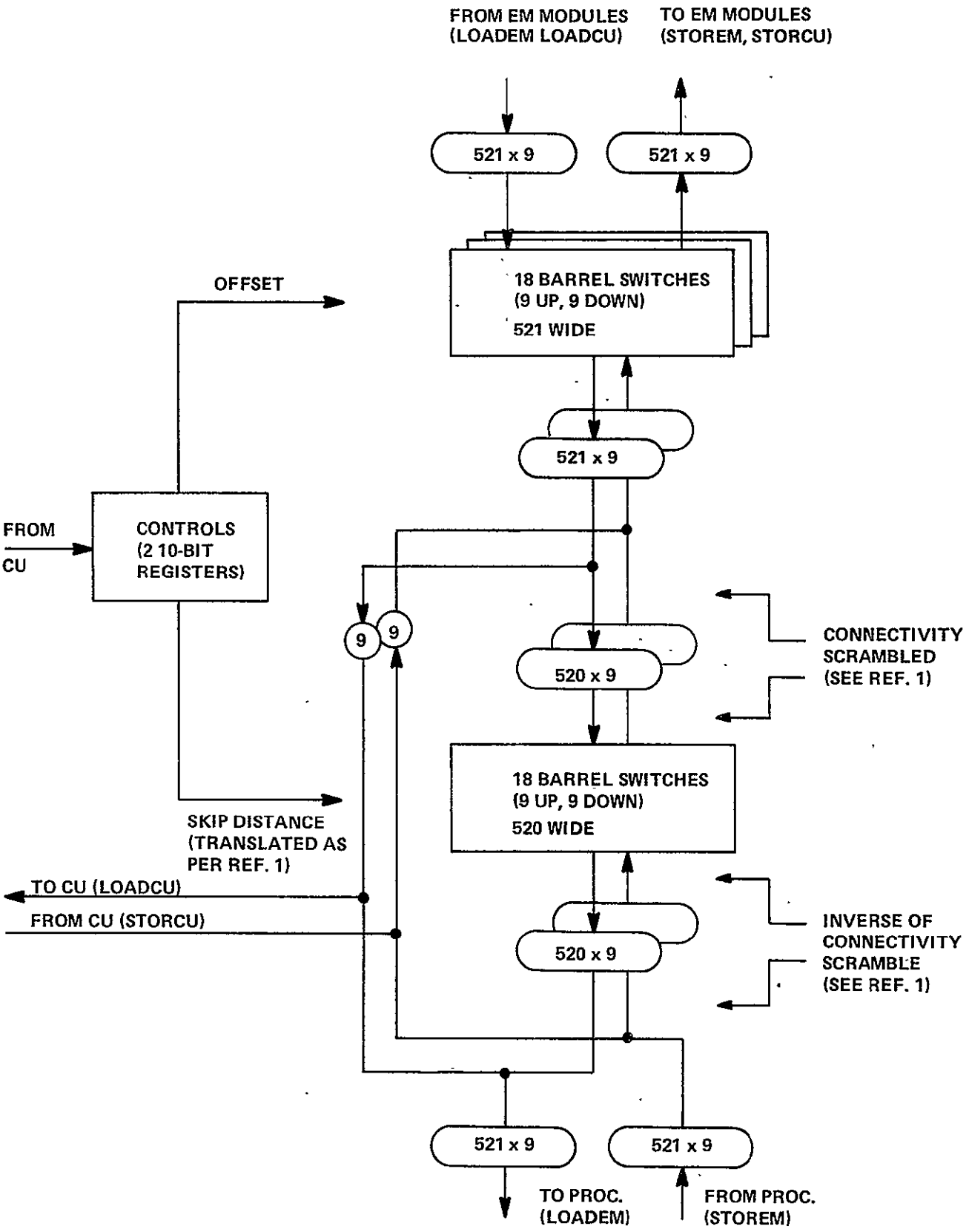


Figure 2-11. Transposition Network

TABLE 2-8
TRANSPOSITION NETWORK CHARACTERISTICS

UNIT: Transposition Network (TN) No. In System: 1

FUNCTIONAL CHARACTERISTICS

Function: Provides 521 data paths for fetching in parallel from all EM modules to all processors; provides 521 data paths for storing in parallel from all processors to 512 EM modules. Provides path from any one EM module to all processors. Provides data path to any EM module from CU, also path from any EM module to CU.

Source of Control; During User Program: Commands from CU.
During System Startup and Diagnostics: Same

Storages; Capacity: None. Command register 10 bits offset, 10 bits skip distance, about 3 bits of command.
Speed:

Connectivity to Other Elements:

#	Path	To or From	No. Sig.	Timing	Primary Use
1	LOADEM	To Processor	9(x 512)	20ns/byte	Data to processor during LOADEM
2	STOREM	From Processor	9(x 512)	byte/20ns	EM addresses and STOREM data from proc.
3	LOADCU	To CU	9	byte/20ns	Data to CU during LOADCU
4	STORCU	From CU	9	byte/20ns	Data and address from CU
5	---	To EM modules	9(x 521)	byte/20ns	Data and address to EM modules
6	---	From EM modules	9(x 521)	byte/20ns	Data from EM modules
7	control	From CU	13	TBD	Reset controls
8	spare	To TBD	9(x 8)	byte/20ns	Reserved for scalar processor
9	spare	From TBD	9(x 8)	byte/20ns	Reserved for scalar processor

RELIABILITY/REPAIRABILITY/TRUSTWORTHINESS

Error Control Methods: SECDED applied to EM word passes through TN. Detects hard failures, corrects transients.
Repair Method: TBD
MTBF of Unit: See chapter 5
Degraded Modes Available: Some portion of the TN can be bypassed by programs that are compiled for a less-than full complement of processors. Most, however, cannot.

PHYSICAL

Chip Count; 1980 Projection: 10,980 (10,480 shifter chips + 500 control)
If use 1977 parts: 17,270 (100K ECL, etc.) 16,770 F 100158 chips + 500 control)

Physical Size; 1980: About 200 boards if 200 signals allowed per board. Is pin limited.
1977: Same

Power Drain: 1980:
1977:

2.4.10 Data Base Memory (DBM)

Data Base Memory (DBM) is the window in the computational envelope of the FMP. All jobs to be run on the FMP are staged into DBM before running both program and data, all output from the FMP is staged through the DBM. At some future time (but not with the initial operating system) DBM could be used to back up EM for those problems whose data base is larger than EM. Control of the data base memory is from a DBM controller, which accepts commands both from the CU for transfers between DBM and EM, and from the host for transfers between DBM and the file system.

Many design options exist for the data base memory. Out of this set of options one particular design was chosen for the baseline system. This chosen design is a CCD memory built out of 256k-chips, which are projected to be available in the 1980 period. If data base memory were to be built before the appearance of sufficiently economical CCD chips, one would use some form of parallel-head rotating magnetic storage. The design described here is based on the existence of 256k-bit CCD chips each arranged in the form of 128 shift registers of 2,048 bits each.

With a projected shift rate of 2.5 MHz in the CCD chips, a desired transfer rate of 2.5 Mwd/s to and from EM, DBM is built 55 chips wide, for parallel emission of 55-bit words, by 512 chips deep. The natural block size with 2,048 bits in each shift register delivering a block of 2,048 words, is adopted. There are 64k blocks for a total of 134,217,728 words. Error correction is a SECDED, probably the modified Hamming-plus-parity implemented by Motorola's 10,163 chip.

Since the array of CCD chips is 512 x 55, the DBM is constructed in a number of physical modules, say each one 64 x 55 chips. The repair philosophy is to pull and replace individual modules, and the degraded mode of operation would be to run with one or more modules missing, and the operating system would have to know to avoid assigning any data to that space.

There are several (probably four) block-sized buffers, which stand between the CCD storage and the host interface, in order to reduce the interference with DBM-EM transfers produced by simultaneous DMB-host transfers. They can also serve as timing buffers to the host's disk packs. See Fig. 2-12.

After the transfer of a block to or from the CCD store, the shift registers rest at the starting position until shifting is required by the refresh requirements, or until the CCD store is again addressed, whichever occurs first. Therefore, whenever there are several requests for transfer pending at once, or when they occur with sufficient frequency, the access time is essentially zero to the first word of the block. For transfers arriving at random times, far enough apart in time so as not to interfere, the average access time is given by:

$$T_{av} = \tfrac{1}{2}(T_b 2/T_r)$$

where T_b is the transfer time of a single block (0.82 ms) and T_r is the time between refreshes. T_r will be in the specification of the device, and is expected to lie between 1 ms and 10 ms. Therefore, the average access time for random data at low usage, to the first word of the block, has an upper bound which is expected to lie between 0.67 ms and 0.067 ms. As traffic increases, the access time is mostly due to interference between competing accesses, while the contribution due to delay in the memory goes to zero.

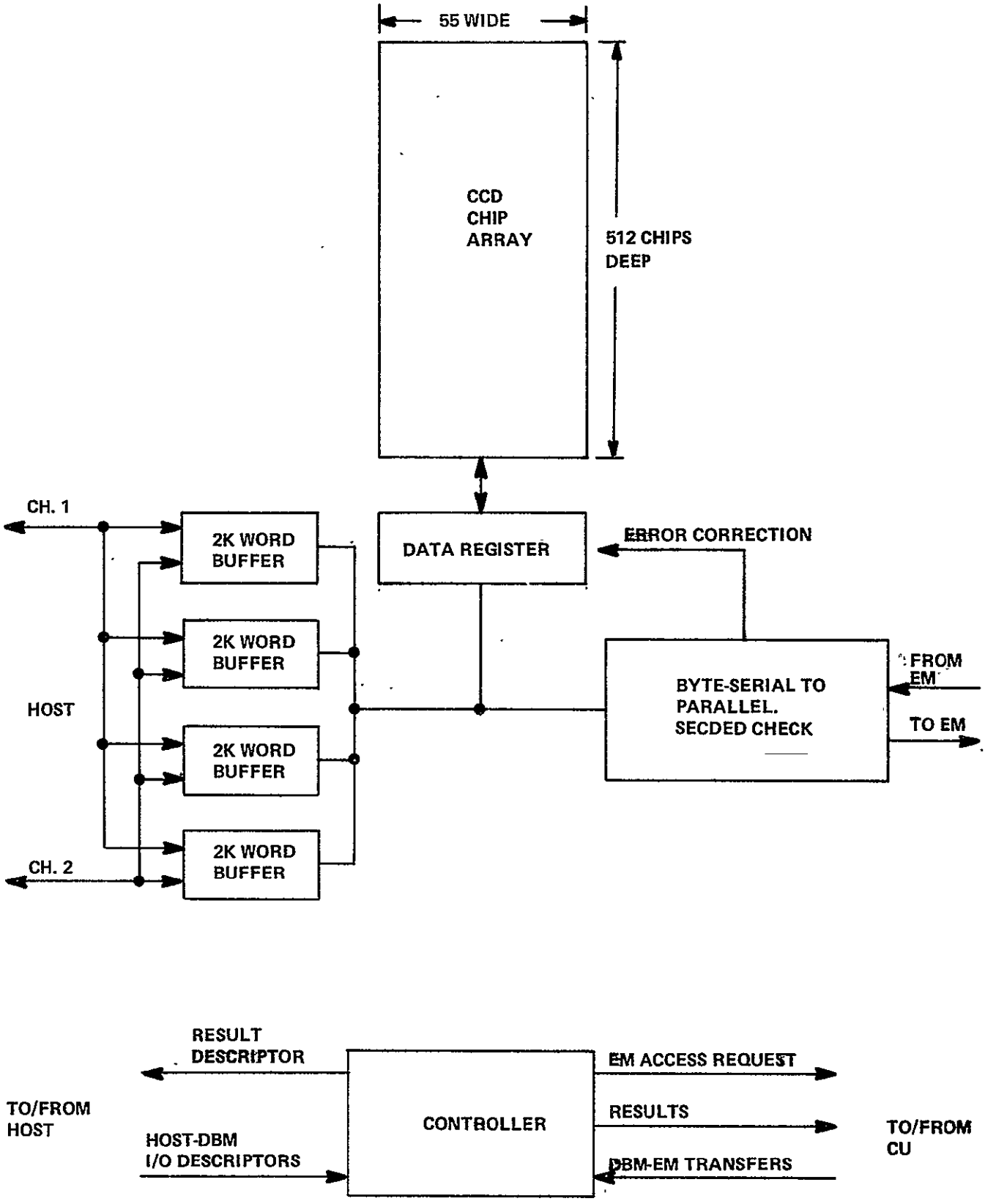


Figure 2-12. DBM Block Diagram

TABLE 2-9
DATA BASE MEMORY CHARACTERISTICS

UNIT: Data Base Memory (DBM) and its controller No. In System: 1

FUNCTIONAL CHARACTERISTICS

Function: In this memory, data is staged for FMP jobs not yet started, and results of FMP jobs are output from the FMP. Almost all communication between FMP and host goes through this memory, both data and program. CCD storage is postulated, although other options are available, including disk pack. Resolves host-CU conflicts.

Source of Control; During User Program: DBM-EM transfers controlled from CU, DBM-host transfers controlled from host.
During System Startup and Diagnostics: Same

Storages; Capacity: 134×10^6 words in blocks
Speed: 140 Mb/s (an easily adjustable parameter)

Connectivity to Other Elements:

#	Path	To or From	Sig.	Timing	Primary Use
1	---	To/from EM	8+8	words/40 ns	Loads EM at start of run, unloads results
2	---	To/from host	TBD, 2 paths min	rate matches host file system	Loading DBM, unloading results
3	control	From CU	TBD	TBD	Receives control from CU for DBM-EM transfers
4	result	To CU	TBD	TBD	---
5	control	From host	TBD	TBD	Receives control from host for DBM-file-system transfers
6	result	To host	TBD	TBD	Monitoring and error cases

RELIABILITY/REPARIABILITY/TRUSTWORTHINESS

Error Control Methods: TBD. SECDED may be adequate, and will be used if so. "Scrubbing" errors arising due to refresh will be needed in CCD memories.
Repair Method: TBD.
MTBF of Unit: Domniated by controls since SECDED on memory.
Degraded Modes Available: Error correction codes allow valid data to be fetched in spite of errors in memory. Can operate with failed modules removed.

PHYSICAL

Chip Count; 1980 Projection: 29,160 (28,160 mem + 1,000 control)
Physical Size: 1980: about 150 large boards
Power Drain; 1980:

If use 1977 parts: (100K ECL, etc.) use disk pack
1977: eight disk pack drives
1977:

As a background job, the DBM controller periodically initiates an access for the purpose of reading the contents of a block and rewriting that same block with all detectable errors corrected, since errors are spontaneously created in CCD memories at a low rate during the refresh operation. It has been conjectured that these errors are caused by cosmic ray bombardment of the CCD chips, discharging the little capacitors by temporarily ionizing the oxide. The rate of periodically initiating access can rationally be determined only after getting the vendor's specification on the number of refreshes per error. Preliminary Fairchild data, if it continues to be true, indicates that one should scrub through the entire DBM every seven minutes, or that this background task should occur at one eighth the normal bandwidth of the DBM. Therefore, this background access is initiated every 6.55 ms. Only one error-scrubbing access will be pending at a time, even if the delay in starting exceeds 6.55 ms. They are not queued.

The DBM has a number of channels into the file system of the host. The number is to be determined by simulation. Initial estimates are that two channels provide more channel capacity than needed for the aerodynamic flow models. At least two are needed for reasons of reliability. Two are assumed for the baseline system design.

No buffering is needed on the EM side beyond the one-word buffers in each EM module. The CU will guarantee the acceptance by the EM of a word coming from DBM is less than 400 ns. Likewise, when transferring from EM to DBM, the EM module has its one-word buffer loaded nominally 800 ns or more ahead of the DBM requirement, and this time will not slip by more than 400 ns from interference with array transfers.

DBM-EM transfers have priority in the EM controls. However, there is little interference with CU-initiated EM transfers. For example, when transferring from EM to DBM, one EM cycle loads 521 of the per-EM-module one-word buffers, and then waits for 208 microseconds before another EM cycle is required for the DBM transfer path.

A design decision, to be made with the aid of simulation in phase II, is whether the LOADEM and STOREM instructions should be limited to 512 words per execution, or whether they should transfer 512 x N words at a time. The description given above is concordant with a design in which LOADEM and STOREM are 512-word instructions, which are the only use made of LOADEM and STOREM in the FORTRAN compiler described in Chapter Three. In Chapter Six the implications of this choice are discussed at further length.

Use of DBM is as a staging area for jobs going into the FMP or coming out of the FMP. The hardware design also permits its use as a source for overlaying data and program into the FMP. It is possible to transfer less than a full block, but not to start any place other than the beginning of the block. A decision to make heavy use of the overlay capability would result in reevaluating the transfer rate between EM and DBM.

2.5 INSTRUCTION SET AND INSTRUCTION TIMING

This section lists the instruction set together with a list of numbers giving the execution times of each.

2.5.1 Tables

There are three tables. Table 2-10 contains the instructions and timing for the processor, of which there are 512. Table 2-11 contains instructions and timing for the control unit of the baseline system. Since no scalar unit is required for the aerodynamic equations, scalar unit timings are not specifiable on the basis of any known application. Rather arbitrarily, the floating-point instructions of table 2-12 are given the same timing as their processor counterparts. These instructions belong to the option for processing floating-point scalars in the control unit.

Instruction formats are easy to specify, and have been postponed until more difficult issues are resolved. See section 6.5.

2.5.2 Instruction Execution Timing

For the processor instructions there are three separate functional units involved. Each instruction has a starting time in each of the three units and an ending time or does not use that unit. The time of execution of each instruction is dependent on its time of occupancy (if any) in each of the independent execution units, namely: integer unit, floating point unit, and memory controls. The timing is described most easily with respect to the instruction fetching process, which determines the starting time of each successive instruction. A fourth function unit, to allow EM fetches and stores to transpire in parallel with other processing, is under consideration, but has not been included in this description.

Entries in the table have the following significance:

"No. of clock periods" is the number of clocks from when the instruction normally issues to a functional unit, to the termination of the instruction. The instruction will always have been decoded from out of the staging register for at least one clock prior to this.

"Unit busy" is of the form n-m, where n is the number of the latest clock that previous instruction is allowed to occupy this unit, and m is the last clock that this current instruction occupies this unit.

Some instructions merely stop the instruction fetching process for a while, until the control unit restarts it. The clock times given for these instructions represent the time from first decoding such an instruction in the staging register, until the start of decoding of the next instruction, under the most favorable circumstances. These instructions are in tables 2-10 and 2-11, and are WAIT, STOP, and HELP.

2.5.3 Instruction Fetch Timing

Timing of the instruction fetching mechanisms can be seen with respect to Figure 2-13. The next instruction is being held in a staging register. Out of the staging register is decoded the start times required for the functional units if this instruction were to start at this clock, and the time it will occupy the holding register. Out of the integer, the floating point, and the

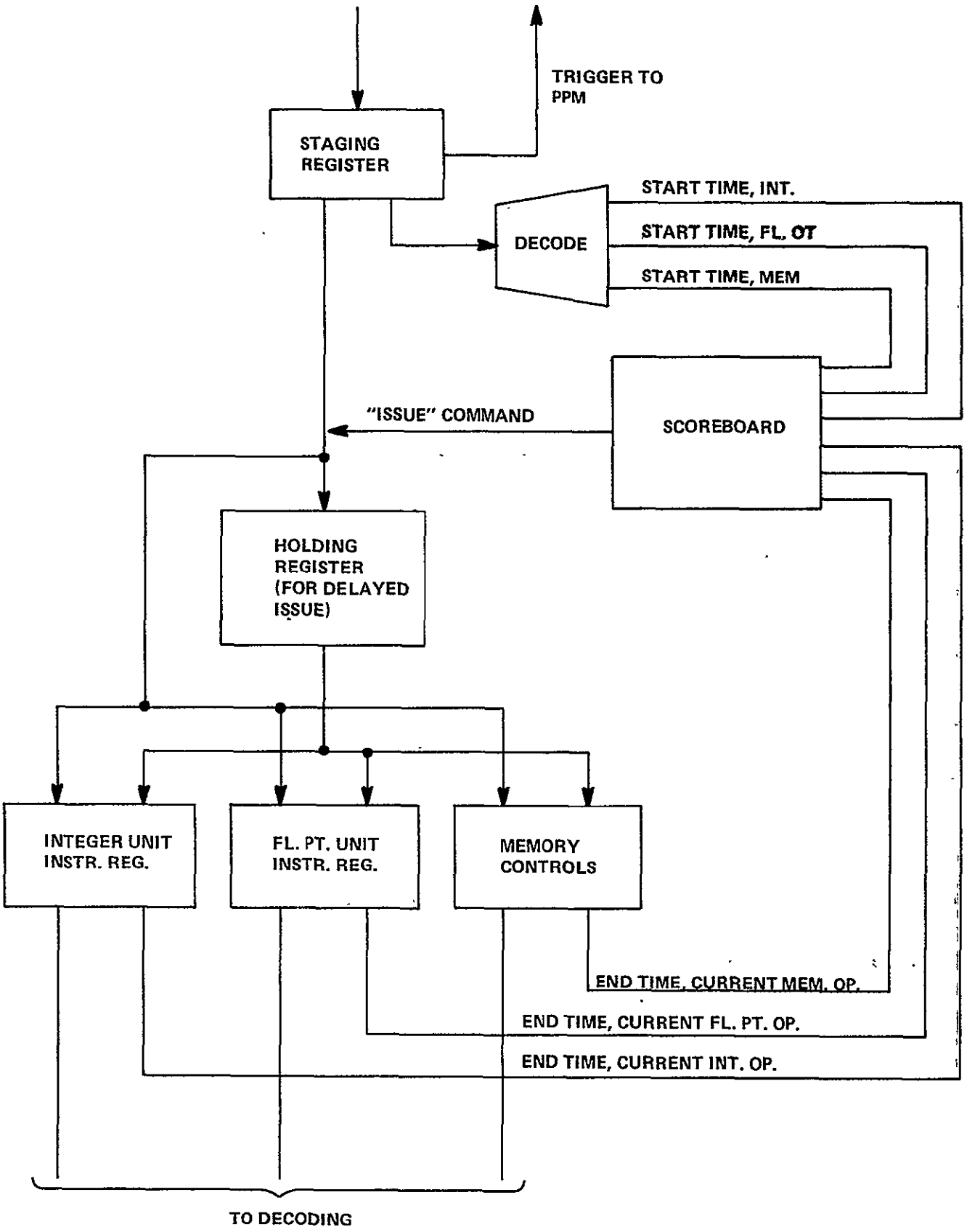


Figure 2-13. Instruction Fetching Mechanism

memory control functional unit is decoded the ending time associated with the currently executing instruction. The "scoreboard" compares all six times. When all four comparisons say the next instruction will not interfere with current instructions, the instruction is transferred from the staging register to the one or more functional unit instruction registers. If delayed starts in other functional units are part of this instruction, the instruction is passed to the holding register to free the staging register for the next instruction.

The program counter always points to the next word in memory after the staging register contents. Thus, normally the PPM will be holding the next instruction word statically at its output lines. Only when the staging register is unloaded in less than three clocks (the PPM cycle) will the next word not appear.

A complexity is the existence of half-word and full-word instructions. Empty halves of half-word instructions carry the first half of the next instruction, so full-word instructions may only have their first half present in the staging register. The first half is sufficient to determine the timing. However, the second half will contain any memory addresses, so when a fetch from memory is involved, the second half must also be fetched before the memory part of the operation can start.

In the baseline system, those instructions which contain a memory address (either for data or as a branch address), or a literal, are full-word 48-bit instructions. Others are 24 bits.

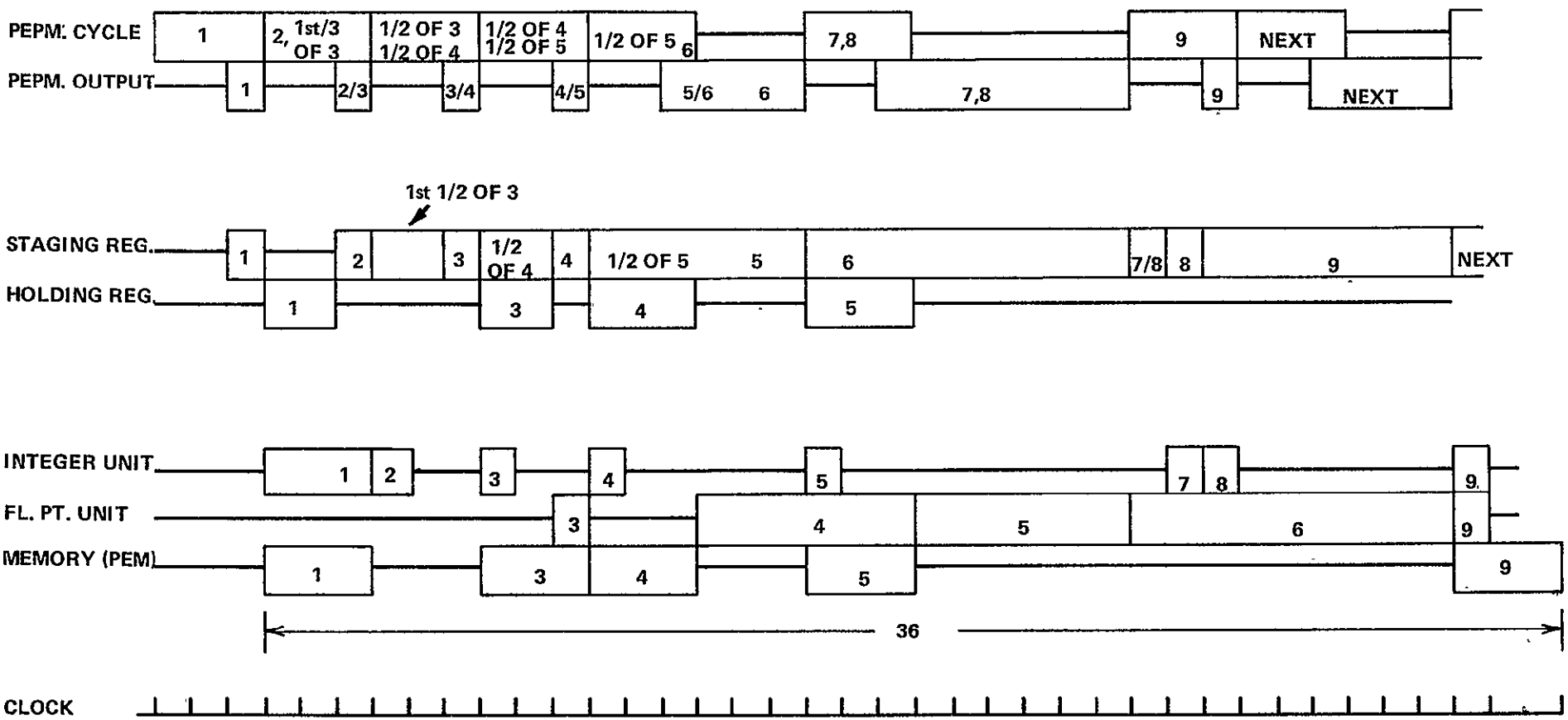


Figure 2-14. Timing Diagram

Jumps take an extra three clocks before the first instruction on the path branched-to can be started.

2.5.4 Example

For an example of how this works, take the sequence of instructions:

1. FETCH from memory to integer register
2. IADD reg. to reg.
3. FETCH from memory to floating point register
4. ADD from memory (indexed by integer reg.) to fl. pt. reg.
5. ADD from mem. (indexed by integer reg.) to fl. pt. reg.
6. MUL from fl. pt. reg. to fl. pt. reg.
7. IADD int. reg. to reg.
8. IADD int. reg. to reg.
9. STORE from fl. pt. reg. to mem. (indexed by int. reg.)

Figure 12-14 shows the timing diagram for this sequence, according to the previous instructions. The instructions are given by number in Figure 12-13. Each clock is 40 ns.

The entire sequence of nine instructions takes 36 clocks, or 1,440 ns. The sum of the "no. of clocks" column in the timing table, for these same instructions is 40 clocks. Overlap between functional units gained little in this example. It is expected to gain more in examples which have a higher emphisis on computing addresses in the integer unit. In this present example, the timing would have come out the same if the holding register had not been there, if loading of the staging register were merely delayed. Simulation may tell us that the holding register gains nothing; that only the staging register is needed. Simulation during phase II will attempt to evaluate the gain given by the complexities here described. The final instruction fetching machinery will be the result of a tradeoff between simplicity and throughput.

2.5.5 Control Unit Timing

In the absence of a completely detailed design of the control unit, the internal structure and overlapping capabilities cannot be visualized with certainty. No overlap mechanism in the control unit is described in the table except for memory. Since there are four semi-independent instruction execution units, these times are pessimistic indeed. However, for aerodynamic flow problems used as benchmarks, the pessimistic assumption is expected not to matter. For aero flow problems, the interfering CU action will be address calculations, which will be a solid swatch of instructions all for the integer unit. Thus, we postpone designing the overlap and look-ahead capabilities within the CU until simulation in phase II tells us how much design effort we should spend on them.

It is assumed that memory fetches and stores will be overlapped. Fetches can be initiated before the previous instruction is started. Fetch and store are three clocks each. The fetch of the next instruction must follow the store of this one, when fetch follows store in the instruction sequence.

The diagnostic controller is not used during normal program running. It is used only for diagnostics and for system initialization when power first comes on, or for reinitializing the FMP system software.

Instruction fetching in the CU is overlapped with instruction execution, but is out of the same CUM that holds the CU data. The instruction execution unit will look ahead by an amount yet to be determined.

The scalar processor is here implemented by adding floating-point capability to the control unit and the entire repertoire of floating point processor type instructions is added to the control unit instruction set. See the discussion on "Scalar Processor", in Chapter 6. These instructions are:

> ADD, SUB, MUL, DIV, MAD, SSQ, ADDD, MULD, LT, LE, GT, GE, NEG, EQ, NE, INFL, FIX, FLOAT, INFZ, SETFL, SETZ, PAK2, ABS, UPF, and PENO (which yields either "0" or "512", to be determined)

A scalar capability resident in the control unit may require a faster control unit than the one described in the accompanying timing tables. The degree of speedup of the design required is a matter to be determined by simulation. Parallel operation of semi-autonomous units (as seen in the processor) is one of the ploys used to achieve increased speed, together with fast multiply algorithms and other logic speedups. A method of achieving faster CU memory operation also may be required. Several memory modules, either interlaced or dedicated to concurrent and overlappable functions, could be included in such a design. The times shown here ignore these additional design options, since they will not be needed for aero flow benchmarks.

2.5.6 Corresponding Times in Synchronizing Instructions

An additional detail is the relative timing of instructions that must be synchronized between CU and processors. For these instructions, execution will proceed when all enabled processors and the CU have reached the instruction. For each instruction there is a "CU lead time", T_L. The timing rules are as follows:

The "go" pulse is emitted from the control unit a time T_C after the start of the instruction, if the "All processors ready" signal does not delay it. The "go" pulse is effective at the processors no sooner than a time T_P after the start of the instruction in the processor. Thus, if both CU and processor arrive at this instruction at the correct time that both can execute it in the minimum time, there will be an offset of $(T_P - T_C)$ clocks between these two initiations. For various cooperating pairs of synchronizing instructions, Table 2-13 gives T_L $(=T_P - T_C)$.

Table 2-13 contains three columns. Column 1 is the CU name of the instruction. Column 2 is the processor name of the matching instruction. Column 3 is the CU lead time T_L. Negative T_L means that the CU can arrive at the instruction $-T_L$ clocks after the last processor without delaying the time of the instruction past its last-processor start time. T_L values tend to be negative because the "same" clock pulse at the CU and the processors is actually about 60 ns sooner at the CU. That is, $T_L=0$ implies that the CU is 60 ns ahead of the processor.

2.5.7 Exceptional Cases

Within the processor, all fault cases result in an interrupt to system software that is resident in the processor. It is possible to handle some interrupts without interrupting the CU. Floating-point out-of-range detection does not cause interrupts, but results in setting the floating-point variables into "infinity" or "infinitesimal". Any integer overflow causes an interrupt, on the theory that most integer operations are address calculations and overflow represents a faulty address. Attempting to insert a number outside the range $\pm 2^{15}-1$ into a 16-bit integer register causes an integer interrupt; likewise executing a FIXD (double-length integer) on a number outside the range $\pm 2^{31}-1$ results in interrupt. Any detection of error in the error-detection-correction logic results in processor interrupt. When the error is correctible, the interrupt merely logs its occurrence and returns to user processing.

TABLE 2-10
PROCESSOR INSTRUCTIONS

	Description	No. Clock Periods	Unit Busy Int	Unit Busy Flt'g Point	Unit Busy Mem	Instr. Length
ADD, SUB*	Floating point add/subtract. Result to fl. pt. reg.					
	Case 1. Reg. + Reg. to Reg.	6		0-6		24
	Case 2. Reg. + Lit. to Reg.	6		0-6		48
	Case 3. Reg. + Mem. to Reg.	9	0-1	3-9	0-3	48
MUL*	Floating point multiply					
	Case 1. Reg. x Reg. to Reg.	9		0-9		24
	Case 2. Reg. x Lit. to Reg.	9		0-9		48
	Case 3. Reg. x Mem. to Reg.	12	0-1	3-12	0-3	48
DIV*	Floating point divide					
	Case 1. Reg./Reg.	44		0-44		24
	Case 2. Reg./Lit. to Reg.	44		0-44		48
	Case 3. Reg./Mem. to Reg.	47	0-1	3-47	0-3	48
DIVR	Same as DIV except the second operand is divided by the 1st.					
	Case 1. 2d operand in reg. not implemented					
	Case 2. Lit./Reg. to Reg.	44		0-44		48
	Case 3. Mem./Reg. to Reg.	47	0-1	3-47	0-3	48
MAD	Floating point add product of two operands to third operand. Result to same register in which third operand was found.					
	Case 1. Reg. x Reg. + Reg. to Reg.	11		0-11		24
	Case 2. Reg. x Lit. + Reg. to Reg.	11		0-11	48	
	Case 3. Reg. x Mem. + Reg. to Reg.	14	0-1	3-14	0-3	48
SSQ	Floating point sum of squares					
	Case 1. $Reg.^2 + Reg.^2$ to Reg.	21		0-21		24
	Case 2. $Mem.^2 + Reg.^2$ to Reg.	24	0-1	3-24	0-3	48
ADDD, SUBD	Floating point sum (or difference) of two registers is kept in double length form and kept in two successive fl. pt. reg. The exponents of the two results differ by at least 38.	13		0-13		24
MULD	Floating point multiply, with the full double length result put into two successive fl. pt. registers in the form of two normalized flt. pt. words with an exponent different of 36 or more. Inputs are from registers	17		0-17		24

*If non-rounding versions of these instructions are supplied, the nexecution times will not differ from those given for the rounding version.

TABLE 2-10 (cont.)

	Description	No. Clock Periods	Unit Busy Int	Flt'g Point	Mem	Instr. Length
FLIT	Transfer the 32-bit literal to the leading 32 bits of the fl. pt. reg.	2		0-2		48
IADD, ISUB	Integer add and subtract. Both input operands are from integer registers, result goes to a third register. One input may be litera.					
	Case 1. Reg. ± Reg. or literal	1	0-1			24 (48 if lit.)
	Case 2. Reg. ± memory	4	0-4		0-3	24
IADM, ISBM	Same as IADD, ISUB, except the first operand and result are double-length (from concatenation of int. reg. with next it. reg.)					
	Case 1. 2d operand int. reg.	2	0-2			24
	Case 2. 2d operand lit.	2	0-2			48
	Case 3. 2d operand from mem.(16 bits)	5	0-5	2-3	0-3	48
IADDD, ISBD	Double-length integer add, oneoperand in two successive registers, second from two successive integer register, result to two successive integer registers	2	0-4			24
IADDD, ISBD	Second (32-bit) operand from memory	5	0-5		0-3	48
IMUL	Integer multiply					
	Case 1 reg. x reg. or literal	9	0-9			24 (48 if lit)
	Case 2 reg. x memory	12	0-12		0-3	48
IDIV	Integer divide. Register or literal divided by register, result to register					
	Case 1 reg./reg. or literal	16	0-16			24 (48 if lit)
	Case 2 reg./memory	19	0-19		0-3	
IMLD	Multiply double-length integer in two successive registers by single-length integer, result to two successive registers	17	0-17			24 (48 if lit)
IDVD	Divide double length integer in one pair of register by single length integer. Result to single-length register	32	0-32			24

TABLE 2-10 (cont.)

	Description	No. Clock Periods	Unit Busy Int	Flt'g Point	Mem	Instr. Length
ID521	Divide double length integer in register by 521, leave result in double-length register	13	0-13			24
IMOD	Saved remainder instead of quotient from IDIV	16	0-16			24 (48 if lit)
ILIT	Transfer 16-bit literal to int.reg.	1	0-1			48
ILITT	Transfer 32-bit literal to double-2 length integer register formed by the concatenation of two single-length int. reg.	2	0-2			48
IALIT	Add the 32-bit literal to the designated double-length int. reg.	2	0-2			48
SB	Set least significant bit of integer equal to the result of the proceding test (excecuted prior to the actual jump)	1	0-1			24
IADD1,ISUB1	Add (Subtract) 1 from content of int.reg.	1	0-1			24
IMDD	Same as IDVD, except result is remainder not quotient	32	0-32			24
ILT,ILE,IGT IGE,IEQ,INE	Test first integer register against second int. reg., if true, branch to location in branch address field.	2		0-2		48
	If fall thru:	2	0-2			48
	If branch,	4	0-4			48
SHF	Shift index register right end-around by the number of places found in second register	2	0-2			24
LT, LE, GT, GE, EQ, NE	Test operand in first fl. pt. register for compliance with condition with expressed condition with request to 2nd reg. new PCR address in address field	2		0-2		48
		4	2-4	0-4		fall-thru
						if jump
TIX	Test integer in one register against integer in second register, increment by content of third reg. Single length only.	2	0-2			48
		4	0-4	0-2		fall-thru
						if jump
AND,OR	Logic combination of one integer register with another, result to a third	1	0-1			24

TABLE 2-10 (cont.)

	Description	No. Clock Periods	Unit Busy Int	Unit Busy Flt'g Point	Unit Busy Mem	Instr. Length
NOT	Complement of one integer register, result to a second	1	0-1			24
BIT	If Nth bit of integer register is ONE, fall through, else jump to address contained in second index register. N is in register or literal	2 4	0-2 0-4		2-4	24 (48 if lit)
JUMP	Set program counter to value found in reg.	2	0-2		1-2	24
CALL	Subroutine entry. Involves automatic handling of stack of return information, and parameter passing	to be determined, up to thirty clocks				48
RETURN	Subroutine return. Stack cut-back	to be determined, up to thirty clocks				24
INFY	Test fl. pt. reg for equal to infinity	2			0-2	24
		4	2-4		0-4	if jump
INFL	Test Fl. pt. reg. for infinitesimal	2			0-2	24
		4	2-4		0-4	if jump
POP	Execute stack action of RETURN, but do not change program counter setting	to be determined				48
TOS	Set stack pointer to new value, value found in register	1	0-1			24
FIX	Convert operand found in fl. pt. reg to integer. Result to integer register.	4	3-4		0-4	24
FLOAT	Current operand in int. reg. to floating, result to fl. pt. reg.	4	0-4		1-4	24
FIXD	Convert operand found in fl. pt. register to integer, result to two successive integer registers	5	3-5		0-5	24
INFZ	Convert operand in fl. pt. reg. to zero if infinitesimal	1			0-1	24
SETFL	Set infintesimal control bit. Underflow will thereafter create infinitesimals	1			0-1	24

TABLE 2-10 (cont.)

	Description	No. Clock Periods	Unit Busy Int	Flt'g Point	Mem	Instr. Length
SETZ	Reset infinitesimal control but, U'flow will thereafter create zeroes	1		0-1		24
PAK2	Take two floating point registers, round the value found in each to 24 bits length, concatenate the result, store in memory. The original operands are saved as long as the third register is distinct	9	6-7	0-6	6-9	48
PAKI	Take two integer registers, move one to the first half, and the other to the second half of a 48-bit word which is then stored in memory	2	0-2	1-4	1-4	48
PAKID	Same, except that two pairs of integer registers hold 32-bit integers each, which are truncated (off left end) to 24 bit integers before packing	4	0-4	2-7	4-7	48
PAKI3	Pack three 16-bit integer registers in a single word which is then stored to memory	5	0-5	2-8	5-8	48
UPI	Move the two 24-bit halves of a word fetched from memory to the pairs of registers indicated by the two integer reg. addresses	5	3-5	2-4	0-3	48
UPI3	Move the three 16-bit fields of a word fetched from memory to the three int. registers addressed. Like PAKI3, may be used to keep an index value, its increment and its limit packed into a single memory word	6	3-6	2-5	0-3	48
UPF	Move the 24-bit havles of a word fetched from memory to the leading 24 bits of the two fl. pt. registers addressed, with zero fill	5	0-1	2-5	0-3	48
BDCST	Broadcast. Receive byte serial word from the CU and insert it into the processor. Timing varies with the destination.					
	Case 1. Fl. Pt. register	7	7-8	4-7		24
	Case 2. Single Int. register	8	7-9	4-7		24
	Case 3. Double (pair of) Int. reg.	9	7-9	4-8		24
	Case 4. PEM	9		4-7	6-9	48
HVST	"Unbroadcast", send word to the control unit. From fl. pt. register only.	7		4-7		24

TABLE 2-10 (cont.)

	Description	No. Clock Periods	Unit Busy Int	Unit Busy Flt'g Point	Unit Busy Mem	Instr. Length
FETCH	Move literal or register to register					
	Case 1. Literal or fl. pt. reg. to fl. pt.	1		0-1		24 (48 if lit)
	Case 2. Literal or int. reg. to int. reg.	1	0-1			24 (48 if lit)
	Case 3. Lit. to fl. pt. or vice versa	1	0-1	0-1		24
	Case 4. Memory to fl. pt. reg.	3	0-1	2-3	0-3	48
	Case 5. Memory to int. reg.	3	0-3		0-3	48
	All integers above are 16-bit integers. For fetching to pairs of integer registers, fetching double-length integers, times are:					
	Case 6. Flt. pt. to double integer reg's or vice versal	2	0-2	0-2		24
	Case 7. Double int. to double int.	2	0-2			24
	Case 8. Memory to double int.	4	0-4		0-3	48
STORE	Store from source to PDM					
	Case 1. Fl. pt. to memory	3	0-1	0-3	0-3	48
	Case 2. 16-bit integer to memory	4	0-1	1-4	1-4	48
	Case 3. Double length (32-bit) int. to mem	5	0-2	2-5	2-5	48
WAIT	Cease operations until CU emits "go". Takes one clock (at the instruction fetch unit), before transmitting the "I got here" signal. Takes three clocks for "I got here" to echo back from the CU as a new setting for the program counter, takes 5 clocks after that for the first instruction to get decoded. Takes only 4 clocks if PCR not changed.	9				24
STOP	Same as WAIT plus reset "enable". The 9 clocks include the time to restart the program after starting but do not include any new setting of the program counter.	9				24
HELP	Same as STOP, plus sends interrupt to CU	9				24
PNO	Read processor no. from backplane into integer register	1	0-1			24
	If processor is above the swithced-out spare, add 1 to the number.	2	0-2			24

In all of the following TN instructions, an option is that the execution may be conditional on an additional integer register's last bit. Thus, participation of a given processor in a LOADEM or STOREM need not use the much slower mechanism of executing STOP followed by a subsequent turn on.

TABLE 2-10 (cont.)

	Description	No. Clock Periods	Unit Busy Int	Flt'g Point	Mem	Instr. Length
LOADEM	Fetch 1 word from EM, address in pair of int. registers, to fl. pt. register. After first clock, test "ready" line from CU before continuing to count clocks	13	0-13	12-13		24
LOADEMM	Fetch N words from EM address in pair of int. registers, to PEM. test "CU ready" line as above. Memory cycles N times. Memory address found in int. reg. not in instruction (Note 1).	13+ 4N (Note 1)	0-13		13- 13+ 4N	24
STOREM	Store 1 word from fl. pt. register to EM. EM address in double int. register.	5	0-2	1-5		24
STOREMM	Store N words from PEM to EM. PEM address is in integer register (Note 1)	5+4N		5-5+ 4N		
SHIFTN	Transmit one word from fl. pt. register out onto TN after testing "CU ready" line. After transmission, test for a new turn-on of "CU ready", and receive from the line. The time given includes the 4 clocks the PE waits while the CU sets the TN to a new setting.	12		0-12		24
EMNO	Read EM module number into the processor. Wait for "CU ready", then transmit to int. register. Delays through the wire of the PE-to-CU-to-EM-to-PE path are included	8	7-8	6-7		24
OFF	Test bit of int. reg., if ZERO, halt and reset "enable" bit	2	0-1			24
ABS	Make sign bit of fl. pt reg. positive.					
	Case 1. Operand in fl. pt. reg.	1		0-1		24
	Case 2. Operand from memory	3	0-1	2-3	0-3	48
NEG	Change sign of fl. pt. reg.	1		0-1		24

Note 1: These EM instructions, with a streaming of N words per instruction are included to assist in evaluating the tradeoff between allowing such an N-word streaming of data, and restricting the EM instructions to 1 word each. A number of advantages accrue to the limitation to N=1. All of these instructions are implemented, but, in the baseline design here presented we have limited the machine to N=1. A design option exists to implement other N up to some large limit. See Chapter Six.

TABLE 2-11
CONTROL UNIT INSTRUCTIONS

	Description	No. CU Clocks	Memory	Instr. Length
CADD, CSUB	Add, subtract integers within the CU (32 bits)			
	Case 1. Literal or reg. to reg.	1		24 (48 if lit
	Case 2. Memory to register	1	Fetch	48
CDV521	Integer div. of register by 521, result to a second register	9		24
CMD521	Similar to CDV521 except that original number MOD 521 is left in a third register.	10		24
CDVMD521	Produces both quotient and remainder for 521	11		24
CMD512	Save last 9 bits of one reg. in second reg.	1		24
CDV512	Shift right 9 places end-off into 2nd reg.	1		24
CMUL	Multiply two operands together			
	Case 1. Literal or reg. by reg.	3+½N		24 (48 if lit.
	Case 2. Memory by register	3+½N	Fetch	48
	N is the bit position of the most significant ONE in the multiplier. Thus, multiplying by small positive integers is fast.			
CDIV	Divide register by register or literal	5+N		24 (48 if lit.
	Divide register by memory	5+N	Fetch	48
	A preliminary shift, controlled by the number of leading zeroes in divisor and dividend, produces all or all but one of the zeroes in the quotient before the N successive subtractions.			
CMOD	Save remainder from CDIV			
	Case 1. Divisor from register	6+N		24
	Case 2. Divisor from memory	6+N	Fetch	48
INT	Test bit n of interrupt register, reset it	10		24
MASK	Set/reset nth bit of mask register	10		24

TABLE 2-11 (cont.)

	Description	No. CU Clocks	Memory	Instr. Length
CIAD1, CISB1	Add (subtract) from register	1		24
CSHFD	Shift reg. by the shift distance (literal, or found in 2d reg.) end-off	1	24	
CSHF	Shift end-around	1	24	
CSHFN	Shift numeric. If a right shift, fill the left with copies of the sign bit. If left, the shifted-off bits must all equal the retained sign bit, or integer overflow is declared.	3	24	
TIOM	Transmit content of two or three registers to DBM-EM controller	2	Fetch	24
CFCH	Fetch from CU memory to register	1	Fetch	48
CSTR	Store to CUM from register	1	Store	48
CTIX	Text index in register, and increment Case 1. Fall-through Case 2. Jump	 3 7	 24	
TIOH	Read or write 2 words into 48-bit host-readable register, interrupt host	2	24	
CGT, CGE CLS, CLE CEQ, CNE	Test register against register Case 1. Fall-through Case 2. Jump	 3 8	 24	
CCALL	Enter subroutine, ignore processors	20	24	
CCALLS	Enter subroutine, synch	23	24	
CRET	Return from subroutine, ignore processors	30	24	
CRETS	Return from subroutine, synch	33	24	
UBSCST	Unconditionally force the processor to accept a stream of N words for PEM or PEPM with starting address in CU register	6+4N	Fetch during inst.	48
UBDCSTE	Same except only enabled processors are loaded	6+4N	Fetches	48

TABLE 2-11 (cont.)

	Description	No. CU Clocks	Memory	Instr. Length
USETP	Unconditionally force the content of CUM into designated processor register. CUM address is in instruction stream with index option	4	Fetch	48
USETPO	Same, plus turn on "enable" bit of the processor		4	Fetch
CHALTP	Halt PE's at end of next PE instruction, Wait for all PE's to finish. Can restart	4		24
CSTOPP	Stop processors in second clock of this instruction. Cannot restart processors, until reinitialized	3		24
LOADCU	Fetch to CUM from EM via TN. EM address in CU register is DIV 521 to make address-within-module, and MOD 521 to form module no. (which sets the barrel part of the TN). The DIV and MOD are computationally expensive, therefore, we stream N words. (Note 1)	26+ 4N (Note 1)	Series of Stores	48
STORCU	Store from CUM to EM. Address calculation like LOADCU. N words (Note 1)	26+ 4N (note 1)	Series of Fetches	48
LOADRCU	Same as LOADCU except the destination is the register, rather than memory pointed to by the register	23		48
STORRCU	Same as STORCU except the data is taken from the reg. rather than memory	23		48
CFETCH	Fetch from CUM to address indexable by register	1	Fetch	48
CSTORE	Store to CUM from register	1	Store	48
CJUMP	Change PCR setting	1		24
LOADEM	Set TN to settings found in register (ROM for $\log_3$ (skip-distance) is in hardware). Send "CU ready" bit to processor. When "all processors ready" comes back, send N successive "read" commands to EM at 4 clock spacing. (See Note 1) Includes TN setting for broadcasting to all processors for one EM module.	4+4N		24
STOREM	Same, except "write" command sent to EM.	8		24

TABLE 2-11 (cont.)

	Description	No. CU Clocks	Memory	Instr. Length
SHIFTN	Set TN setting and send "CU ready". When "all processors ready" comes back, wait 1 clock, set TN to 2d setting, and send "go".	8	24	
EMNO	Set TN setting and send "CU ready". When "all processors ready" comes back, send "read module no." to EM and "go" to processor, appropriately timed.	6	24	
CGTS, CGES, CLSS, CLES, CEQS, CNES	Perform indicated test and wait for "all processors ready". Then send command to processors to load PCR to either first or second address depending on the test result. Also branch in CU if test succeeds.	6	24	
CTIXS	Test index against liiit and wait for "all processors ready". Then jam			
CILIT	16-bit literal to int. reg.	1	24	
CLITT	Transfer 32-bit literal to CU. reg.	2	48	
CALIT	Add 32-bit literal to CU reg.	2	48	
SETTN	Set TN controls. No synchronization or processor interaction occurs	4	24	
LOOP	Wait till "all processors ready". If any are enabled issue "go". If none are enabled, jam processor PCR to new setting found in address field. Used for synchronized execution of loops whose loop control is in a processor variable, and may be data dependent per processor.	2	24	
SYNCH	Wait for "all processors ready". Issue "go"	2	24	

TABLE 2-11 (cont.)

	Description	No. CU Clocks	Memory	Instr. Length
BDCST	Wait for "all processors ready", then transmit byte-serial word and "go".			
	Case 1. Word comes from CU register	5	24	
	Case 2. Word comes from CUM	5	Fetch	48
HVST	Wait for "all processors ready" then transmit "go", receive 48-bit word (If PE is transmitting an integer, later bytes may be empty except for the check bits)	9	24	
CAND, COR	Logic combination of two CU words, result to register.			
	Case 1. Both operands in registers or lit.	2	24	
	Case 2. One operand from CUM	2	Fetch	48
CNOT	Bit complement of CU register	2	24	
CIMP	A and not B. Logic			
	Case 1. Both operands register or literal	2	24	
	Case 2. One operand from CUM	2	Fetch	48
MOVE	Register-to-register move	1	24	
CBIT,CBITS	Jump if any bit of register, ANDed with 2nd register or literal is ON	6	24	

Note 1: These EM instructions, with a streaming of N words per instruction are included to assist in evaluating the tradeoff between allowing such an N-word streaming of data, and restricting the EM instructions to 1 word each. A number of advantages accure to the limitation to N=1. All of these instructions are implemented, but, in the baseline design here presented we have limited the machine to N=1. A design option exists to implement other N up to some large limit. See Chapter Six.

TABLE 2-12
FLOATING POINT SCALAR INSTRUCTIONS

	Description	Clocks	Memory	Instr. Length
ADD, SUB	Case 1. Reg. or lit. + reg. to reg.	6		24 (48 if lit.)
	Case 2. Reg. + mem. to reg.	6	Fetch	48
MUL	Case 1. Reg. x reg. or lit. to reg.	9		24 (48 if lit.)
	Case 2. Reg. x mem. to reg.	9	Fetch	48
DIV	Case 1. Reg. or reg./lit to reg.	44		24 (48 if lit.)
	Case 2. Reg./mem. to reg.	44	Fetch	48
DIVR	same as DIV with operands reversed, Case 2 only.	44	Fetch	48
MAD	Case 1. Reg. x reg. or lit. + reg. to reg.	11		24 (48 if lit)
SSQ	Case 1. $Reg.^2 + Reg.^2$ to reg.	21		24
	Case 2. $Mem.^2 + reg.^2$ to reg.	21	Fetch	48
ADDD	Floating point double length addition	13		24
MULD	Floating point double length multiply capability (single length inputs)	17		24
LT, LE, GT, GE, EQ, NE, INFY, INFL	Tests on floating point registers	2 4		48 if fall thru if jump
FIX, FLOAT	Convert data type	4		24
INFX	Convert infinitesimal to zero	1	24	
SETFL, SETZ	Set response to underflow to infintesimal or zero	1		24
PAK2	Pack two truncated fl. pt. words in mem. word.	6	Store	48
UPF	Unpack two truncated fl. pt. words	2	Fetch	48
PENO	Load CU register with predetermined lit. Supplied only to permit symmetry with processors' code stream.	1	24	
ABS	Take absolute value,			
	Case 1./ reg./	1		24
	Case 2./mem./	1	Fetch	48
NEG	Change Sign	1		24

TABLE 2-13

OFFSET TIMES OF PROCESSOR-CU SYNCHRONIZED INSTRUCTIONS

CU INSTRUCTION OR ACTION	PROCESSOR INSTRUCTION	T_L
Interrupt	HELP	-3
LOADEM	LOADEM	1
STOREM	STOREM	1
SHIFTN	SHIFTN	3
EMNO	EMNO	1
BDCAST	BDCAST	-3
HVST	HVST	-3
SYNC	WAIT	-3
CGTS, OGES, CLSS CLES, CEQS, CNES, CTIXS, CJUMPS CBITS	WAIT	-3
CCALLS	STOP or WAIT	-3
CRETS	STOP or WAIT	-3
LOOP	WAIT	-3

Ref. 1. Burroughs Corporation, "Final Report, Numerical Aerodynamic Simulation Facility, Preliminary Study", Dec. 1977.

CHAPTER 3
SOFTWARE ISSUES

3.1 EXTENDED FORTRAN FOR THE FMP

3.1.1 INTRODUCTION

This chapter describes the extensions and restrictions on the FMP FORTRAN language and compiler at the functional level. The overall functional view of this piece of software is stated below, and is sketched in Figure 3-1.

1. NSS FORTRAN will be as compatible with ANSI FORTRAN (X3J3/90) and B7800 FORTRAN as the architecture permits. Differences from these standards will be indicated in this document and in detail in the later detailed design specification.

2. The compilation process will be performed on the B7800 front end and will produce code to be executed on the FMP system.

3. FMP FORTRAN will have array operations designed to allow the explicit expression of parallel operations available with the architecture.

4. The compiler will be designed in a modular fashion with an internal representation between components which is identical so that addition modules can be added if desired. The components as envisioned at this time are:

   a. A parser
   b. A preliminary optimizer which performs standard serial optimization techniques.
   c. A secondary optimizer which may reorder code to obtain maximum overlap of functional units.

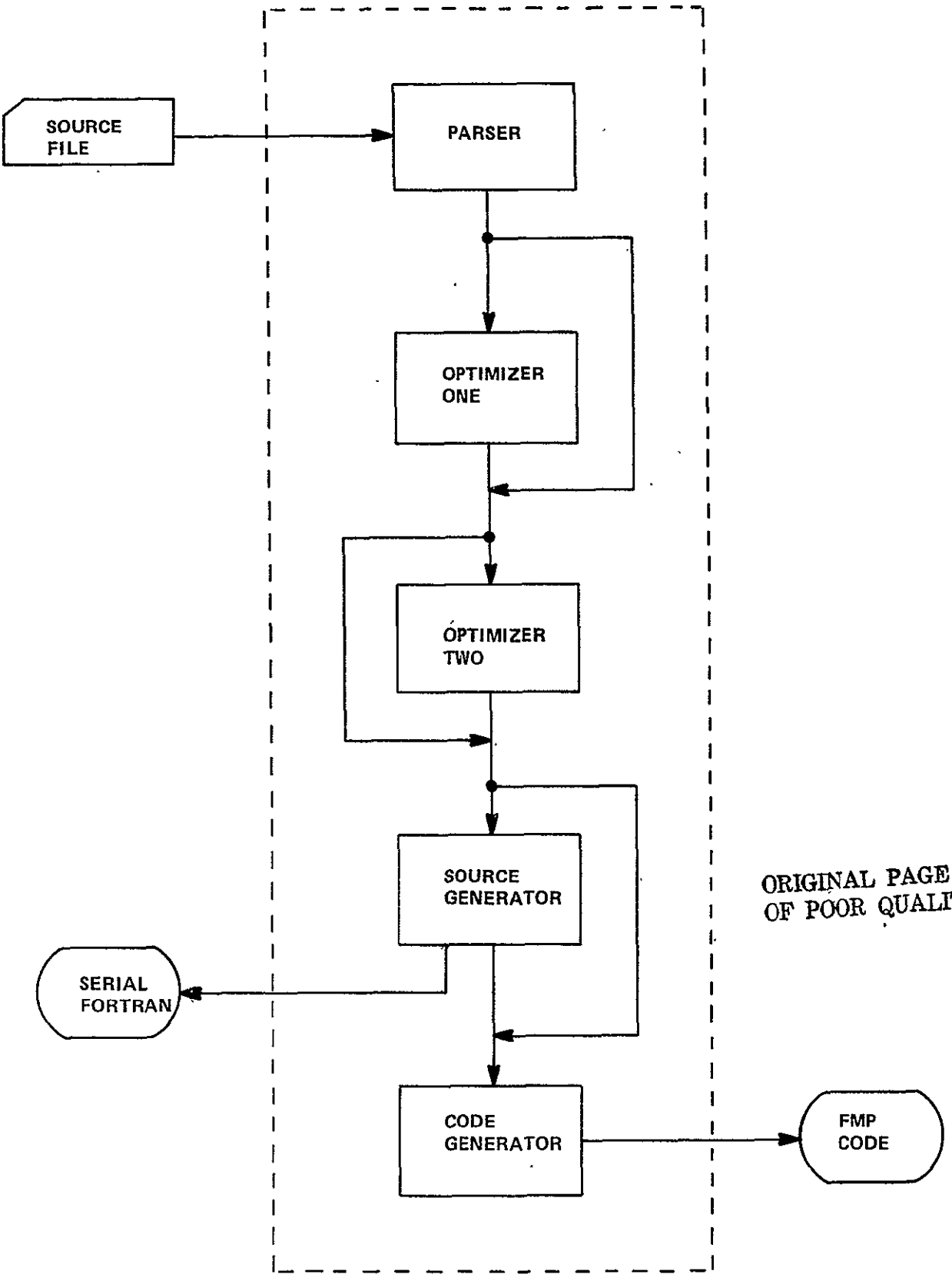


Figure 3-1. FMP Compiler Components

d. A code generator
e. A source regenerator which will regenerate serial FORTRAN as a method of enhancing portability and providing the user with a programming tool during the early phases of using the machine.

3.1.2 Functional Objectives of Language Development

In the development of the FMP language and the FMP compiler the following goals were set which are listed below:

1. Allow the user to access features of the machine in a simple straight forward manner.
2. Add a small number of extensions which are general in nature rather than a host of specific cases.
3. As much as is possible keep both the syntax and semantics of the extensions isolated from those employed in serial FORTRAN constructs.
4. Provide easily understood and recognizable constructs which yields programs which the user can understand and recognize without translation back to serial constructs.

3.1.3 Major Extensions to FORTRAN

There are only two primary extensions to the ANSI FORTRAN. All other additions and restrictions to the language follow from these primary extensions. The two consist of a modification to the normal set of non-executable specification statements and the addition of a parallel construct.

The modifications in the specification statements are made to allow the user to control the memory allocation to maximize efficient utilization of the machine. These memory resident specifications allow the user to explicity control the allocation of his data among the Control Unit Memory (CUM), the Extended Memory (EM), and Processor Memory (PM). The second construct is a parallel construct put in the language to aid the user in obtaining a simple way in which to express the parallel aspects of his problem. With both constructs equivalences can be made to ANSI FORTRAN so that a serial FORTRAN can be regenerated.

3.1.4 Specification Statements

The modifications to FORTRAN will permit the following specifications:

1. DIMENSION
2. EXTENDED
3. LOCAL
4. GLOBAL

For the present the following statements will be disallowed:

1. EQUIVALENCE
2. COMMON (Blank or named)

3.1.4.1 The DIMENSION statement retains its ANSI FORTRAN meaning. The DIMENSION statement is used to specify the sumbolic names and dimension specifications (extents) of arrays.

3.1.4.2 The EXTENDED specification statement declares that the variables specified in the statement are resident in the Extended Memory. The form of declaration is:

EXTENDED /cb/ nlist (, /cb/ nlist).....

or

EXTENDED nlist

where cb is an extended block name

nlist is a list of variable names or array declarators. Only one appearance of a symbolic name as a variable name or array declarator is permitted in all such a symbolic name as a variable name or array declarator is permitted in all suchlists in a program unit. The ellipses represent repetition.

This construct is similar to blank COMMON in the sense that execution of a RETURN or END statement never causes these quantities to become undefined. (See Specification FORTRAN X3J3/90 page 8-3)

3.1.4.3 The LOCAL specification statement declares that the variables specified in the statement are resident in Processor Memory. The form of the declaration is:

LOCAL /cb/ nlist (, /cb/ nlist).....

or

LOCAL nlist

where cb and nlist are defined as above.

This construct is similar to named COMMON in FORTRAN in the sense that execution of a RETURN or END may cause the quantities to be undefined. Note however that execution of a RETURN or END within a subprogram will not cause entries to become undetermined in a LOCAL block that appears in the subprogram and appears in at least one other program unit that is referencing it either directly or indirectly. (See Specification FORTRAN X3J3/90 page 15-15)

3.1.4.4 The GLOBAL specification statement declares that variables specified in the statement are controlled by the Control Unit and are broadcast automatically to the Processor Memory on Program initiation or if they modified during the execution of a program. The form of the declaration is:

```
GLOBAL /cb/ nlist (, /cb/ nlist).....
          or
GLOBAL nlist
```

where cb and nlist are defined as above.

3.1.5 The Parallel Construct

The executable DOALL construct is a control statement provided to permit concurrent execution of segments of a program.

The DOALL statement is used in conjunction with a terminal statement ENDDO to form together a loop called the DOALL loop. The form of these two statements is

DOALL, $I=I_1$, I_2 ($,I_3$) ($;J=J_1$, J_2($,J_3$)) ($;K=K_1$, K_2($;K_3$))

ENDDO

I is the name of an integer variable. I_1, I_2, I_3 are each integers.

3.1.5.1 Range of a DOALL loop. The range of a DOALL loop consists of all executable statements that appear following the DOALL statement including the terminal ENDDO statement.

No additional DOALL statements may occur within the range of a DOALL.

If a DO statement appears within the range of a DOALL statement it must be fully contained within the range of the DOALL statement.

If a arithmetic or logical IF statement occurs within a DOALL statement, it may not transfer control out of the range of the DOALL statement. Transfer into the range of a DOALL is prohibited.

3.1.5.2 Active and inactive DOALL-loops. A DOALL loop is either active or inactive. Initially inactive, a DOALL becomes active only when its DOALL statement is executed.

Once active, the DOALL-loop becomes inactive only when the iteration count (3.1.5.4) for each of its increment parameters becomes zero.

Execution of a FUNCTION reference or a CALL statement that appears in the range of a DOALL statement does not cause the DOALL to become inactive. Note specification of an alternative return specifier outside the range of the DOALL is disallowed.

3.1.5.3 Incrementation Parameters. Specified in the DOALL statement are at least one set of parameters which are to control the execution of the statements within the range of the DOALL loop. These are called the incrementation parameter set and there may be a total of three sets of them. Each parameter set consists of three (four) integers known as the DOALL variable, the initial parameter, the terminal parameter, and (the increment parameter).

3.1.5.4 Referencing the DOALL variable within the DOALL loop. References to the DOALL variable, I, (J) or (K) within the DOALL-loop is permitted for the following references:

1. Any reference to array subscripts for arrays declared to be in Extended Memory, however, the DOALL variable may not reference outside the declared array.
2. Any reference to the value of the DOALL variable within an expression of an IF statement if control is not transferred beyond the range.
3. The DOALL variable may be used in the evaluation of an assignment statement, however, not to form forbidden array reference.

The utilization of the DOALL variable is specifically prohibited for the following:

1. Any reference to array subscripts for variables declared to be LOCAL or which appear in a DIMENSION statement either explicitly or implicitly.
2. The DOALL variable may not be reassigned within the range of the DOALL-loop except by the DOALL statement.
3. Transfer of control into the range of a DOALL-loop is prohibited.

3.1.5.5 Execution of the DOALL construct. The effect of executing a DO-ALL-loop construct is to execute all body statements, those following after the DOALL statement and preceding the ENDDO statement, in a serial fashion for those determined incrementation parameters set in the DOALL statement. The initial parameter M_1 the terminal parameter M_2, and the incrementation parameter M_3 are determined for each incrementation set, I_1, I_2, I_3. This determines the allowable values of the DOALL variables I(J and K) equal to N_I.

The DOALL variable I with its N_I allowed values is paired with the first allowed variable of J. Next the DOALL variable of I with its N_I allowed values is paired with the second allowed variable of J. This continues until all possible combinations occur. The total number of combinations is:

N_I for a single DOALL-loop incrementation set

$N_I * N_J$ for a double DOALL-loop incrementation set

$N_I * N_J * N_K$ for a triple DOALL-loop incrementation set

Hence the body statements are executed in serial fashion for each given set of DOALL variables allowed, either I, I & J, or I, J, & K in a strictly parallel sense.

3.1.6 Subroutines & Procedures as Program Subunits (to be resolved in Phase II)

3.1.7 Other Constructs

3.1.7.1 ASSIGN Statement. The ASSIGN statement has been dropped as a possible candidate for a FMP extension. It was found that the access to Extended Memory could be handled by simple compiler algorithms through the EXTENDED declaration. It was found that in complex control structures the programmer was more likely to make mistakes and cause ARRAY bound errors than if the compiler was to perform all the necessary accessing. Some details of this will be shown in later examples. (See 3.2.2.2 discussion and Fig. 3.4).

3.1.7.2 I/O. All I/O for NSS FORTRAN must be performed on variables assigned to Extended or Control Unit Memory. If variables in Processor Memory are referenced in an I/O statement a syntactical error will result.

3.1.8 Examples of Constructs in FMP FORTRAN

3.1.8.1 VALID Triply Nested DOALL-Loop

```
EXTENDED Q(100, 100, 100), S(100, 100, 100)
DOALL, I = 2, 99; J = 2, 99; K = 2, 99

RR = 1.0/Q(I, J+1, K-1)

R1 = Q(I+1, J, K) - Q(I-1, J, K)
R2 = Q(I, J, K+1) - Q(I, J, K-1)
S(I, J, K) = RR * R1 * R2

ENDDO;
```

2. INVALID

```
EXTENDED  Q(100, 100, 100), S(100, 100, 100)
DIMENSION  R1(100), R2(100)

DOALL, I = 2, 99; J = 2, 99; K = 2, 99

RR = 1.0/Q(I, J+1, K-1)

R1(I) = Q (I+1, J, K) - Q(I-1, J, K)
R2(I) = Q(I, J, K+1) - Q(I, J, K-1)
S(I, J, K) = RR * R1(I-1) * R2(I+1)

ENDDO;
```

This construct is invalid because the arrays R_1 and R_2 declared in the DIMENSION statement are referenced by the DOALL variable I. If it is necessary to so reference the arrays R_1 and R_2 arrays the doubly nested DOALL construct should be used (See 3.1,8.2).

3. VALID

```
      EXTENDED Q(100, 100, 100), S(100, 100, 100)

      DOALL, I = 25, 50, 2; J = 1, 99; K = 2, 100

      RR = 1.0/Q(I, J+1, K-1)

      IF (I. GT. 30) GO TO 1

      R1 = Q(I+1, J, K) - Q(I-1, J, K)
      S(I, J, K) = RR * R1
      GO TO 2
    1 R1 = Q(I-1, J, K) - Q(I+1, J, K)
      S(I, J, K) = RR * R1
    2 CONTINUE
      ENDDO;
```

4. INVALID

```
      EXTENDED Q(100, 100, 100), S(100, 100, 100)
      DOALL, I = 25, 50, 2; J = 1, 99; K = 2, 100

      RR = 1.0/Q(I, J+1, K-1)
      IF (I. GT. 30) GO TO 1
      R1 = Q(I+1, J, K) - Q(I-1, J, K)
      S(I, J, K) = RR * R1

      ENDDO;
    1 CONTINUE
```

This DOALL-loop construct is invalid because it transfers control out of the range of the DOALL.

5. INVALID

```
      EXTENDED Q(100, 100, 100), S(100, 100, 100)
      DIMENSION R1(100), R2(100)
      GLOBAL JL, KL
      DOALL J=2, JL; K=2, KL
      R1 (I) = 6.7
      If (J  30) GO TO 3
      If (K  30) GO TO 4
      DO 1  I = 2, 99
      RR = 1.0/Q(I, J, K)
      GO TO 5
   3  RR = 1.0/Q(I, J-1, K)
      GO TO 5
   4  RR = 1.0/Q(I, J, K-1)
   5  R1(I) = Q(I+1, J, K) - Q(I-1, J, K)
      R2(I) = Q(I, J, K+1) - Q(I, J, K-1)
      S(I, J, K) = RR * R1(I-1) * R2(I+1)
   1  CONTINUE
      ENDDO;
```

ANSI FORTRAN specifically prohibits transfer of control from outside a DO-loop to into the body statements of a DO-loop.

3.1.8.2 Doubly Nested Loops

1. VALID

```
EXTENDED Q(100, 100, 100), S(100, 100, 100)
DIMENSION R1(100), R2(100)
DOALL, J=2, 99; K=2, 99
R1(I)=6.7
DO1 I=2,99
RR=1.0/Q(I, J+1, K-1)
R1(I) = Q(I+1, J, K) - Q(I-1, J, K)
R2(I) = Q(I, J, K+1)
S(I, J, K) = RR * R1(I-1) * R2(I+1)
1 CONTINUE
ENDDO; This is the correct syntax for handling the
```

problem in Example 2. (3.1.8.1)

2. VALID

```
EXTENDED Q(100, 100, 100), S(100, 100, 100)
DIMENSION R1(100), R2(100)
GLOBAL JL, KL
DOALL, J=2, JL; K=2, KL
R1(I)=6.7
DO 1 I = 2, 99
If (J.GT.30) GO TO 3
If (K.LT.30) GO TO 4
RR=1.0/Q(I, J, K)
GO TO 5
3 RR=1.0/Q(J, J-1, K)
GO TO 5
```

```
      4  RR 1.0/Q(I, J, K-1)
      5  R1(I) = Q(I+1, J, K) - Q(I-1, J, K)
         R2(I) = Q(I, J, K+1) - Q(I, J, K-1)
         S(I, J, K) = RR * R1(I-1) * R2(I+1)
      1  CONTINUE
         ENDDO;
```

3.1.8.3 Use of the LOCAL Construct

```
      1.  VALID
          EXTENDED Q(100, 100, 100), S(100, 100, 100)
          LOCAL R1(100, R2(100), ·CONST
          GLOBAL JL, JK)
          DOALL, J=1,JL; K=1,KL
          R(1)=6.0
          R(100)=10.0
          DO 1 I = 2, 99
          RR=1.0/Q(I, J, K)
          R1(I) = Q(I+1, J, K) - Q(I-1, J, K)
          R2(I) = Q(I, J, K+1) - Q(I, J, K-1)
          CALL TEST (I)
          S(I, J, K) = RR * R(I-1) * R2(I+1) * CONST
       1  CONTINUE
          ENDDO;

          SUBROUTINE TEST(I)
          LOCAL R1(100), R2(100), CONST
          IF (R1(I). GT. R2(I)) CONST=R1(I)
          RETURN
          END
```

2. INVALID

```
      EXTENDED Q(100, 100, 100), S(100, 100, 100)
      LOCAL R1(100), R2(100)
      GLOBAL JL, JK
      DOALL, J=1,JL; K=1,KL
      R(1) = 6.0
      R(100) = 10.0
      DO 1 I = 2, 99
      RR=1.0/Q(I, J, K)
      R1(I) = Q(I+1, J, K) - Q(I-1, J, K)
      R2(I) = Q(I, J, K+L) - Q(I, J, K-1)
      CALL TEST(I)
      S(I,J,K) = RR*R1(I-1)*R2(I+1)*CONST
    1 CONTINUE
      ENDDO;
```

Using the identical SUBROUTINE TEST above would cause an undefined reference to CONST because the LOCAL declaration does not contain the variable CONST. Naturally, TEST could have been defined with two parameters I and CONST. which would have been valid.

3.2 HAND COMPILATION FOR SAM

3.2.1 Overview

The methodology of hand compilation for the SAM will be described through a series of examples each of which will be transformed in a series of stages from original FORTRAN to ASSEMBLER CODE. References will be made to Appendix (A) which discusses preliminary compiler alogrithms for setting the transposition network.

In each example the following code steps will be taken:

1. Original NASA-AMES FORTRAN
2. Extended FORTRAN for SAM
3. Compiler output including code reorganization (written in a Pseudo FORTRAN
4. Compile output showing Transposition Network and Memory Module computations (again in a pseudo FORTRAN or META ASSEMBLER)
5. ASSEMBLER CODE

The example chosen from the Explicit Code was the SUBROUTINE TURBDA because it demonstrates the ability of SAM to operate in a concurrent manner and provides a vehicle for demonstrating the compiler's ability to handle control statements through a "mimicking" technique and also provides an example of why it is felt that an ASSIGN statement could cause programmer error. The second example is the major LOOPS of the SUBROUTINE STEP including the subroutine calls and the called SUBROUTINES BTRI and XXM. One loop (DO 20) will be discussed in detail while the other two (DO 30) and (DO 40) will show the differences in the transposition network settings and the memory module accesses for the different memory accessing. (DO30 & DO40 discussion to be supplied later).

3.2.2 SUBROUTINE TURBDA

3.2.2.1 Original Code and SAM Extended FORTRAN

In Figure 3-2 the original NASA-AMES version of the SUBROUTINE is shown. The FMP Extended FORTRAN as written by the programmer is given in Figure 3.3. In both cases the common declarations were modified slightly to remove extraneous variables from this specific example. As you will note, the programmer wrote a two dimensional DOALL-loop with a serial inner DO loop. Because there is no data depending on I it could have been written as a three dimensional DOALL.

3.2.2.2 Preliminary Code Analysis

Figure 3-4 shows the preliminary compiler code analysis. Within the DO 1 loop the compiler determines what array elements stored in Extended Memory must be fetched through the Transposition Network. For a given I, J, K, EI(I, J, K) must be fetched. However, only for J=1 must the element EI(I, J+1, K) and for K=1 must the element EI(I, J, K+1). The compiler will be capable of recognizing these accesses to extended memory and will "mimic" the branch structure. It also will be able with this mirroring of the original structure be able to access only the requisite elements and prohibit out of bounds access of the array even if those elements are not subsequently used. This protection is even more critically necessary when accesses occur in the negative sense rather than the positive one as in this example.

```
      SUBROUTINE TURBDA
      COMMON/A12/ RHOW(31,31,31),E(31,31,31),E1(31,31,31)
      COMMON/A5/ IL,JL,KL,CV
      COMMON/A6/ RMUL(31,31,31)
      CV1=1./CV
      DO 1 K=1,KL
      DO 1 J=1,JL
      DO 1 I=1,IL
      TEMP=ABS(EI(I,J,K))*CV1
      IF(K.EQ.1) TEMP=.5*ABS(EI(I,J,1)+EI(I,J,2))*CV1
      IF(J.EQ.1) TEMP=.5*ABS(EI(I,1,K)+EI(I,2,K))*CV1
      RMUL(I,J,K)=2.207E-08*SQRT(TEMP**3)/TEMP+198.6)
    1 CONTINUE
      RETURN
      END
```

Figure 3-2. Original NASA-AMES FORTRAN

```
      SUBROUTINE TURBDA
      EXTENDED/A12/ RHOW(31,31,31),E(31,31,31),E1(31,31,31)
      GLOBAL/A5/ IL,JL,KL,CV
      EXTENDED/A6/ RMUL(31,31,31)
      CV1=1./CV
      DOALL, J=1,JL;K=1,KL
      DO 1 I=1,IL
      TEMP=ABS(EI(I,J,K))*CV1
      IF(K.EQ.1) TEMP=.5*ABS(EI(I,J,1)+EI(I,J,2))*CV1
      IF(J.EQ.1) TEMP=.5*ABS(EI(I,1,K)+EI(I,2,K))*CV1
      RMUL(I,J,K)=2.270E-08*SQRT(TEMP**3)/TEMP+198.6)
    1 CONTINUE
      ENDDO;
      RETURN
      END
```

Figure 3-3. Extended FORTRAN for SAM

```
      SUBROUTINE TURBDA
      EXTENDED EI(31,31,31),RMUL(31,31,31)
      GLOBAL CV,JL,KL,IL
      DOALL, J=1,JL;K=1,KL
      CV1 = 1.0/CV
      DO 1 I=1,IL
      E1 =EI(I,J,K)
      FOR(J,NEQ.1) null fetch next line
      E2 =EI(I,J+1,K)
      FOR(K.NEQ.1) null fetch next line
      E3 =EI(I,J,K+1)
      IF(J.EQ.1) GO TO 3
      IF(K.EQ.1) GO TO 2
      TEMP=ABS(E1 )*CV1
      GO TO 4
    2 TEMP= 0.5*ABS(E1+E3 )*CV1
      GO TO 4
    3 TEMP=0.5*ABS(E1 + E2 )*CV1
    4 RMUL(I,J,K) = 2.270E-08*SQRT(TEMP**3)/(TEMP+198.6)
    1 CONTINUE
      ENDDO
      RETURN
      END
```

Note: The expression "Null fetch next line" implies that the transposition network will be set to fetch all the elements for EI(I,J+1,K) for given I. However only those for which J=1 will in fact be passed from Extended Memory to the Processors.

Figure 3-4. Compiler Code Analysis

As one can see in this example all processors for which J≠1 & K≠1 all execute TEMP=ABS(E1*CV1. All processors for which J=1 (including K=1) compute TEMP=0.5* ABS(E1+E2)*CV1. All processors for which K=1 and J=1 form TEMP=0.5*ABS(E1+E3)* CV1. These three cases occur for a given I concurrently.

3.2.2.3 Computer Programmatic Transformations Including Transposition Network Calculations

Figure 3-5 shows the Control Unit and Processor Element code streams in a FORTRAN like language or META ASSEMBLER. The compiler recognizing the two dimensional DOALL on J,K, which are the second and third indices of Extended arrays EI and RMUL and calculates the number of cycles to be performed (the DO 10 loop)

$$\text{i.e. } NMAX = \frac{(ISECONDSIZE*THIRDSIZE + Nprocessors-1)}{Nprocessors}$$

$$= \frac{(31*31 + 512-1)}{512} = 2$$

Similiarly the compiler recognizes that ISKIP=IFIRSTSIZE=31. Note that all accesses to EI and RMUL are of type 1 as described in Appendix A.

```
 CU INSTRUCTIONS                                    PE INSTRUCTIONS

ENTER TURBDA                          1          ENTER TURBDA
                                      2            CV1=1.0/CV
   DO 10 N=1,2                        3            DO 10 N=1,2
   IVV=512*N-512                      4            IVV=512*N-512
                                      5            IV= IVV+PENO
                                      6            KM1=IV/31
                                      7            K = KM1+1
                                      8            J = IV-KM1*31+1
   IN= IVV*31                         9            IN= IV*31
   IAØ1=IBSET+IN                     10            IA01= IBSEI+IN
   IAØ2=IAØ1+31                      11            IA02- IAØ1+31
   IAØ3=IAØ+961                      12            IA03=IAØ+961
   IAØ4=IBSRM+IN                     13            IA04= IBSRM+IN
   DO 1 I=1, IL                      14            DO 1 I=1,IL
   II=I-1                            15            II=I-1
   OFFSET1=MOD(IAØ1+II,521)          16            MADD1= (IAØ1+II)/521
                                     17 SYNCH
                                     18            FOR (J.NE.1) MODE=0
   OFFSET2=MOD(IAØ1+II,521)          19            MADD2= (IA02+II)/521
                                     20 SYNCH
                                     21            FOR (K.NE.1) MODE=0
   OFFSET3=MOD(IAØ3+II,521)          22            MADD3= (IAØ3+II)/521
                                     23 SYNCH
                                     24            IF (J.GT,JL) GO TO 8
                                     25            IF (K.GT,KL) GO TO 8
                                     26            IF (J.EQ.1) GO TO 2
                                     27            IF (K.EQ.1) GO TO 3
                                     28            TEMP=ABS(E1)*CV1
                                     29            GO TO 4
                                     30          2 TEMP=0.5*ABS(E1+E3)*CV1
                                     31            GO TO 4
                                     32          3 TEMP=0.5(ABS(E1+E2)*CV1
                                     33          4 R=2.27OE-08*TEMP
                                     34            *SQRT(TEMP)/(TEMP+198.6)
   OFFSET4=MOD(IAØ4+II,521)          35            MADD4=(IA04+II)/521
                                     36          8 CONTINUE
                                     37 SYNCH
  1 CONTINUE                         38          1 CONTINUE
 10 CONTINUE                         39         10 CONTINUE
EXIT                                 40        EXIT
```

Note: The Expression Mode ≠0 is merely a device used to imply that for those values of the variable not equal to 1 fetches through the Transposition Network do not occur.

Figure 3-5. Compiler Output with Transposition Calculations

On entering the subroutine (line 1) of Figure 3.5 each processing element calculates CV1 (line 2). Loop 10 is then initiated which represents the number of times the array must be cycled as mentioned above (line 3). Next IVV is calculated which represents the number of processors that have been utilized to that cycle number. Obviously the compiler does not perform 512*N-512 but rather start from zero and increment by 512, however, FORTRAN usage was utilized here. The processing elements then perform a number of calculations (line 4 - line 8). IV=IVV+IPENO represents the address in J,K space that each processing element has. From that number its J and K value is determined (line 7 and line 8). KM1 (line 6) which represents the K value minus 1 which is used in the J calculation is calculated separately.

Lines 10 thru 13 represent address calcuations. For the control unit one is calculating the address of the array element which is to go into processing element Ø for each transposition network setting, i.e. THE OFFSET. The processing element it is performing and address calculation on the specific array element. This is why line 9 has different determinations for IN. Lines 10 thru 13 are address calculations for EI(I,J,K) (line 10) EI(I,J+1,K) (line 11), EI(I,J,K+1) (line 12) and RMUL(I,J,K) (line 13). Note line 10 and 13 start from the base address IBSET of EI and IBSRM of RMUL. The CU instructions are computing the address calculation for the array element which is to go to processor =Ø while the processors are calculating the address of the array element to go to Processor = IPENO.

Note all these index computations are performed only for the outer loop. They do not occur for the inner DO1I=1,IL loop (line 14).

Next the I index is decremented by 1 (line 15), again a FORTRAN antifact, which would not occur in the ASSEMBLER code but this is FORTRAN. The memory module address, MADD1 (line 16) is computed in the processing element while the offset, IFSET1 (line 16) is computed by the mod function in the control unit. The array and the control unit now SYNCHRONIZE. In a similar fashion in the offset and memory module address are calculated for each of the next two array access and synchronized accordingly (lines 18 thru 28). Note that for (J.NE.1) (line 18) a mode bit is set which turns off the array fetch. Similarily for (K.NE.1) (line 21).

The next step the compiler takes is to skip computations for those values of J between JL+1 and 31, the value declared for the array in the EXTENDED declaration (line 24). This is the way the preliminary compiler is going to handle the one dimensional vector length/declared extent problem at this juncture. Alternative algorithm are known; however teaching the algorithms and subsequent hand compilation would require Burroughs more effort than the possible machine performance degradation that might occur during simulation. For (K.GT.KL) a similar branch is performed (line 25). Note that 8 CONTINUE must be above the next synchronization point. Next the branches for sections of code which will be computed for (J.EQ.1), ((K.EQ.1). AND (J.NEQ.1)) and for all other J and K values less than JL and KL. (lines 26 thru 32) All processors except those that have J or K values greater than JL or KL then process lines (33,34). The OFFSET calculation for RMUL is then made in the Control Unit and the Memory Module address in the processors (line 35). Synchronization occurs and the transfer of RMUL (I,J,K) from Processor to Extended Memory occurs. Lines 14 to 37 are looped until IL is reached and then the second cycle, line 3 to 38 are executed before the subroutine is EXITed.

Earlier it was mentioned that this piece of code could have been executed as a three dimensional DOALL loop. As can now be seen, this would probably not be advantageous in terms of performance for two reasons. First, due to the branches on J and K (lines 24 thru 27) each processor would have to perform the index calculations of lines 6, 7, and 8 for all I values if one did a 3-D DOALL-loop. Second, since IL< 31 one only needs to execute this loop with the preliminary compiler IL times with a 2-D DOALL-loop. In a 3-D DOALL loops I would have to be computed and a branch similar to lines 24 and 25 would also have to be made. At this time this appears less efficient in highly branched code and where the array fit is good - i.e., on cycle 1, all 512 processors are utilized while in cycle 2, 88% of the processors are utilized. If the array size were instead EI(25,25,25) then 100% would be used on cycle 1 while only 113 or 22% would be used on cycle 2. With a 3-D DOALL one would have 31 cycles of which 30 would be 100% busy and 1 cycle of 50% busy. In that case the additional indexing computations would be masked in the total execution time.

3.2.2.4 Assembler Code for TURBDA

This code is shown in Figures 3-6 and 3-7.

```
L
1000            IDENT     CU/IMPLICIT/TURBDA
1001            CODESEG
1002            ENT       START
1003 START      CILIT     CR1,0
1004            CILIT     CR2,1
1005 L3         CTIX      CR1,CR2,L4
1006            CSHFM     CR3,CR1,-9
1007            CMULL     CR6,CR3,31
1008            CFETCH    CR8,IL
1009            CILIT     CR7,1
1010 L1         CTIX      CR7,CR8,L1
1011            CIADDL    CR9,CR6,IBSEI1
1012            CIADDP    CR9,CR7,CR9
1013            MOD521    CR9
1014            CILIT     CR10,31
1015            LOADEM    CR9,CR10
1016            CIADDL    CR9,CR6,IBSEI2
1017            CIADDR    CR9,CR7,CR9
1018            MOD521    CR9
1019            LOADEMC   CR9,CR10
1020            CIADDL    CR9,CR6,IBSEI3
1021            CIADDP    CR9,CR7,CR9
1022            MOD521    CR9
1023            LOADEMC   CR9,CR10
1024            CIADDL    CR9,CR6,IBSRM1
1025            CIADDP    CR9,CR7,CR9
1026            MOD521    CR9
1027            STOPEM    CR9,CR10
1028            JUMP      L1
1029 L1         JUMP      L3
1030 L4         RETURN
1031            END
*
```

Figure 3-6. Handcompiled Control Unit Code Subroutine TURBDA

```
L
1000            INDENT    PE/IMPLICIT TURBDA
1010            CODESEG
1020            ENT       START
1030 START      FLIT      FP1,1.0
1040            FDIVL     FP1,FP1,CV1
1050            ILIT      IR2,1
1060            ILIT      IR1,0
1070 L3         ITIX      IR1,IR2,L4
1080            SHFL      IR3,IR2,-9
1090            PENO      IP4
1100            IADD      IP4,IP3,IR4
1110            IDIVL     IR5,IR4,31
1120            ISTORE    IR5,AM1
1130            IMULL     IP6,IR5,31
1140            ISUB      IP6,IR4,IR6
1150            ISTORE    IR6,JM1
1160            IMULL     IP6,IP4,31
1170            IFETCH    IP8,IL
1180            ILIT      IR7,1
1190 L14        ITIX      IR7,IP8,L1
1200            IADDM     IR8,IR6,IBSE11
1205            IADD      IP8,IP8,IR7
1210            ID521     IR8
1220            LOADEM    IR8,E1
1230            IADDM     IR8,IR6,IBSE12
1235            IADD      IP8,IR8,IR7
1240            ID521     IR8
1250            ILIT      IR10,1
1260            IFETCH    IP11,JM1
1270            IEQ       IP11,0,L100
1280            ILIT      IR10,0
1290 L100       LOADEMC   IP8,E2,IR10
1300            IADDM     IR8,IR6,IBSE13
1305            IADD      IP8,IP8,IR7
1310            ID521     IR8
1320            ILIT      IR10,1
1330            IFETCH    IR11,AM1
1340            IEQ       IR11,0,L200
1350            ILIT      IR10,0
1360 L200       LOADEMC   IR8,E3,IR10
1370            IFETCH    IP12,JM1
1375            IFETCH    IP13,JL
1376            ISUBL     IR13,IR13,1
1380            IGT       IR12,IR13,L80
1390            IFETCH    IP13,AM1
1400            IFETCH    IP14,AL
1405            ISUBL     IR14,IP14,1
1410            IGT       IP13,IP14,L80
1420            IEQ       IP11,0,L30
1430            IEQ       IP13,0,L20
1440            FFETCH    FP2,E1
1450            ABS       FP2
1460            FMUL      FP2,FP1,FP2
1470            FSTORE    FP2,TEMP
1480            JUMP      L40
```

Figure 3-7. Handcompiled Execution Unit Code Subroutine TURBDA

3.2.3 SUBROUTINE STEP (LOOP DO 20)

The next portion of code to be examined is STEP (loop DO 20) which includes CALLS to BTRI and XXM. A number of Figures have been made of the code and they are listed below with a brief description.

Figure 3-8	The original NASA-AMES FORTRAN of Subroutine STEP.
Figure 3-9	SAM Extended FORTRAN for Subroutine STEP
Figure 3-10	A comparison file of Figures 3-8 and 3-9 showing R(Replacements), I(Insertions) - (Deletions)
Figure 3-11	Preliminary Compiler Code Reorganization for Subroutine STEP
Figure 3-12	A comparison of the Figures 3-9 and 3-12
Figure 3-13	Compiler programatic transformations including Transposition Network Settings for Control Unit Subroutine STEP
Figure 3-14	Same as above for Processor - Subroutine STEP
Figure 3-15	Implicit/Steppiece NSS3CU Assembler Code
Figure 3-16	Implicit/Steppiece NSS3PE Assembler Code

Additionally the SUBROUTINES BTRI and XXM are examined. The related Figures are:

Figure 3-17	Original NASA-AMES Code for Subroutine BTRI
Figure 3-18	SAM Extended FORTRAN for Subroutine BTRI
Figure 3-19	Comparison of Figures 3-17 and 3-18
Figure 3-20	Original NASA-AMES Code for Subroutine XXM
Figure 3-21	A modified version of XXM1 which will produce improved performance on the CDC7600 and SAM
Figure 3-22	SAM Extended FORTRAN for SUBROUTINE XXM1
Figure 3-23	Comparison of Figures 3-21 and 3-22

```
184100          SUBROUTINE STEP
184200          COMMON/BASE/NMAX,JMAX,KMAX,LMAX,JM,KM,LM,DT,GAMMA,GAMI,SMU,FSMACH
184300         1  ,DX1,DY1,DZ1,ND,ND2,V(5),FD(5),HD,ALP,GD,OMEGA,HDX,HDY,HDZ
184400         2,RM,CNBR,PI,ITR,INVISC,LAMIN,NP,INT1,INT2,INIT
184500          COMMON/GEO/NB1,NB2,RFRONT,RMAX,XR,XMAX,DRAD,DXC
184600          COMMON/READ/IREAD,IWRIT,NGRI
184700          COMMON/VIS/RE,PR,RMUE,RK
184800          COMMON/VARS/Q(720,6,30)
184900          COMMON/VARQ/S(720,5,30)
185000          COMMON/VAR1/X(720,30),Y(720,30),Z(720,30)
185100          COMMON /VAR3/P(120,30),XX(60,4),YY(60,4),ZZ(60,4)
185200    C     LEVEL 2,Q,S,X,Y,Z
185300          COMMON/COUNT/NC,NC1
185400          COMMON/BTRID/A(60,5,5),B(60,5,5),C(60,5,5),D(60,5,5),F(60,5)
185500    C
185600    C
188200    C
188300          RM = SMU
188400          C8 = 1.+2.*RM
188500          GAM2 = 2.-GAMMA
188600          DO 20 L = 2,LM
188700          DO 20 K = 2,KM
188800    C
188900    C***FILTRX
189000    C
189100          KL = (L-1)*ND+K
189200          JA=2
189300          JB=JMAX-1
189400          CALL XXM(K,L,1,JMAX)
189500          DO 12 J=1,JMAX
189600          R1 =XX(J,1)*HDX
189700          R2 =XX(J,2)*HDX
189800          R3 =XX(J,3)*HDX
189900          R4 =XX(J,4)*HDX
190000    C
190100    C*******AMATRX
190200    C
190300          RR= 1./Q(KL,1,J)
190400          U = Q(KL,2,J)*RR
190500          V = Q(KL,3,J)*RR
190600          W = Q(KL,4,J)*RR
190700          UU = U*R1+V*R2+W*R3
190800          UT = U**2+V**2+W**2
190900          C1 = GAMI*UT*.5
191000          C2 = Q(KL,5,J)*RR*GAMMA
191100          C3=C2-C1
191200          C4=R4+UU
191300          C5=GAMI*U
191400          C6=GAMI*V
191500          C7=GAMI*W
191600          D(J,1,1) = R4
191700          D(J,1,2) = R1
191800          D(J,1,3) = R2
191900          D(J,1,4) = R3
192000          D(J,1,5) = 0.
192100          D(J,2,1) = R1*C1-U*UU
192200          D(J,2,2) = C4+R1*GAM2*U
192300          D(J,2,3) = -R1*C6+R2*U
192400          D(J,2,4) = -R1*C7+R3*U
192500          D(J,2,5) = R1*GAMI
192600          D(J,3,1) = R2*C1-V*UU
192700          D(J,3,2) = R1*V-R2*C5
192800          D(J,3,3) = C4+R2*GAM2*V
192900          D(J,3,4) = -R2*C7+R3*V
193000          D(J,3,5) = R2*GAMI
193100          D(J,4,1) = R3*C1-W*UU
193200          D(J,4,2) = R1*W-R3*C5
193300          D(J,4,3) = R2*W-R3*C6
193400          D(J,4,4) = C4+R3*GAM2*W
193500          D(J,4,5) = R3*GAMI
```

Figure 3-8. Original Piece of Subroutine STEP

```
193600          D(J,5,1) = (-C2+2.*C1)*UU
193700          D(J,5,2) = R1*C3-C5*UU
193800          D(J,5,3) = R2*C3-C6*UU
193900          D(J,5,4) = R3*C3-C7*UU
194000          D(J,5,5) = R4+GAMMA*UU
194100    C
194200    C******END OF AMATRX
194300    C
194400     12   CONTINUE
194500          DO 25 J=JA,JB
194600          RJ = 1./Q(KL,6,J)
194700          RMJ=RM*RJ
194800          RR = RMJ*Q(KL,6,J-1)
194900          RF = RMJ*Q(KL,5,J+1)
195000          DO 23 N=1,5
195100          A(J,N,1) = -D(J-1,N,1)
195200          A(J,N,2) = -D(J-1,N,2)
195300          A(J,N,3) = -D(J-1,N,3)
195400          A(J,N,4) = -D(J-1,N,4)
195500          A(J,N,5) = -D(J-1,N,5)
195600          B(J,N,1) = 0.0
195700          B(J,N,2) = 0.0
195800          B(J,N,3) = 0.0
195900          B(J,N,4) = 0.0
196000          B(J,N,5) = 0.0
196100          C(J,N,1) = D(J+1,N,1)
196200          C(J,N,2) = D(J+1,N,2)
196300          C(J,N,3) = D(J+1,N,3)
196400          C(J,N,4) = D(J+1,N,4)
196500          C(J,N,5) = D(J+1,N,5)
196600          A(J,N,N) = A(J,N,N)-RR
196700          B(J,N,N) = C8
196800          C(J,N,N) = C(J,N,N)-RF
196900     23   F(J,N)=S(KL,N,J)
197000     25   CONTINUE
197100    C
197200    C*****END OF FILTRX
197300    C
197400    C
197500    C   S MUST BE ZERO ON B.C.
197600    C
197700          CALL BTRIX(2,JM)
197800          DO 21 J = 2,JM
197900          S(KL,1,J) = F(J,1)
198000          S(KL,2,J) = F(J,2)
198100          S(KL,3,J) = F(J,3)
198200          S(KL,4,J) = F(J,4)
198300     21   S(KL,5,J) = F(J,5)
198400        20 CONTINUE
218600          RETURN
218700          END
```

Figure 3-8. Original Piece of Subroutine STEP (Cont)

```
184100           SUBROUTINE STEP
184200           GLOBAL/BASE/NMAX,JMAX,KMAX,LMAX,JM,KM,LM,GAMMA,GAMI,SMU,FSMACH
184300          1   ,DX1,DY1,DZ1,ND,ND2,PV(5),PD(5),HD,ALP,GO,OMEGA,HDX,HDY,HDZ
184400          2,RM,CNBR,PI,INVISC,LAMIN,NP
184500           GLOBAL/GEO/NB1,NB2,RFRNT,RMAX,XR,XMAX,DRAD,DXC
184600           GLOBAL/READ/IREAD,IWRIT,NGRI
184700           GLOBAL/VIS/RE,PR,RMUE,RK
184800           EXTENDED/VARS/Q(720,30,6)
184900           EXTENDED/VARO/S(720,30,5)
185000           EXTENDED/VAR1/X(720,30),Y(720,30),Z(720,30)
185100           LOCAL/VAR3/P(120,30),XX(60,4),YY(60,4),ZZ(60,4)
185200     C     LEVEL 2,Q,S,X,Y,Z
185300           CONTROL/COUNT/NC,NC1,DT
185400           LOCAL/BTRID/A(60,5,5),B(60,5,5),C(60,5,5),D(60,5,5),F(60,5)
185500     C
185600     C
188200     C
188300           RM = SMU
188400           C8 = 1.+2.*RM
188500           GAM2 = 2.*GAMMA
188600           DOALL,K=2,KM,L=2,LM
188900     C
188900     C***FILTRX
189000     C
189100           KL = (L-1)*ND+K
189200
189300
189400           INCLUDE XXM1(K,L,',JMAX)
189500           DO 12 J=1,JMAX
189501           Q1=Q(KL,J,1)
189502           Q2=Q(KL,J,2)
189503           Q3=Q(KL,J,3)
189504           Q4=Q(KL,J,4)
189505           Q5=Q(KL,J,5)
189600           R1 =XX(J,1)*HDX
189700           R2 =XX(J,2)*HDX
189800           R3 =XX(J,3)*HDX
189900           R4 =XX(J,4)*HDX
190000     C
190100     C*******AMATRX
190200     C
190300           RR= 1./Q1
190400           U  = Q2*RR
190500           V  = Q3*RR
190600           W  = Q4*RR
190700           UU  = U*R1+V*R2+W*R3
190800           UT  = U**2+V**2+W**2
190900           C1  = GAMI*UT*.5
191000           C2  = Q5*RR*GAMMA
191100           C3=C2-C1
191200           C4=R4+UU
191300           C5=GAMI*U
191400           C6=GAMI*V
191500           C7=GAMI*W
191600           D(J,1,1)  =  R4
191700           D(J,1,2)  =  R1
191800           D(J,1,3)  =  R2
191900           D(J,1,4)  =  R3
192000           D(J,1,5)  =  0.
192100           D(J,2,1)  =  R1*C1-U*UU
192200           D(J,2,2)  =  C4+R1*GAM2*J
192300           D(J,2,3)  =  -R1*C6+R2*U
192400           D(J,2,4)  =  -R1*C7+R3*U
192500           D(J,2,5)  =  R1*GAMI
192600           D(J,3,1)  =  R2*C1-V*UU
192700           D(J,3,2)  =  R1*V-R2*C5
192800           D(J,3,3)  =  C4+R2*GAM2*/
192900           D(J,3,4)  =  -R2*C7+R3*V
193000           D(J,3,5)  =  R2*GAMI
193100           D(J,4,1)  =  R3*C1-W*UU
```

Figure 3-9. Identical Piece of Subroutine STEP in SAM Extended FORTRAN

```
00193200          D(J,4,2) = R1*W-R3*C5
00193300          D(J,4,3) = R2*W-R3*C6
00193400          D(J,4,4) = C4+R3*GAM2**4
00193500          D(J,4,5) = R3*GAMI
00193600          D(J,5,1) = (-C2+2.*C1)*UU
00193700          D(J,5,2) = R1*C3-C5*UU
00193800          D(J,5,3) = R2*C3-C6*UU
00193900          D(J,5,4) = R3*C3-C7*UU
00194000          D(J,5,5) = R4+GAMMA*UU
00194100    C
00194200    C******END OF AMATRX
00194300    C
00194400     12   CONTINUE
00194500          DO 25 J=2,JMAX-1
00194501          IF (J.GT.2) GO TO 777
00194502          Q6=Q(KL,J,6)
00194503          Q6M=Q(KL,J-1,6)
00194504          GO TO 778
00194505    777   Q6M = RX
00194506          Q6= RY
00194510    778   Q6P=Q(KL,J+1,6)
00194511          RX  = Q6
00194512          RY  = Q6P
00194600          RJ  = 1./Q6
00194700          RMJ=RM*RJ
00194800          RR  = RMJ*Q6M
00194900          RF  = RMJ*Q6P
00194901           S1 = S(KL,J,1)
00194902           S2 = S(KL,J,2)
00194903           S3 = S(KL,J,3)
00194904           S4 = S(KL,J,4)
00194905           S5 = S(KL,J,5)
00195000          DO 23 N=1,5
00195100          A(J,N,1) = -D(J-1,N,1)
00195200          A(J,N,2) = -D(J-1,N,2)
00195300          A(J,N,3) = -D(J-1,N,3)
00195400          A(J,N,4) = -D(J-1,N,4)
00195500          A(J,N,5) = -D(J-1,N,5)
00195600          B(J,N,1) = 0.0
00195700          B(J,N,2) = 0.0
00195800          B(J,N,3) = 0.0
00195900          B(J,N,4) = 0.0
00196000          B(J,N,5) = 0.0
00196100          C(J,N,1) = D(J+1,N,1)
00196200          C(J,N,2) = D(J+1,N,2)
00196300          C(J,N,3) = D(J+1,N,3)
00196400          C(J,N,4) = D(J+1,N,4)
00196500          C(J,N,5) = D(J+1,N,5)
00196600          A(J,N,N) = A(J,N,N)-RR
00196700          B(J,N,N) = C8
00196800          C(J,N,N) = C(J,N,N)-RF
00196900       23  CONTINUE
00196901           F(J,1) = S1
00196902           F(J,2) = S2
00196903           F(J,3) = S3
00196904           F(J,4) = S4
00196905           F(J,5) = S5
00197000     25   CONTINUE
00197100    C
00197200    C*****END OF FILTRX
00197300    C
00197400    C
00197500    C  S MUST BE ZERO ON B.C.
00197600    C
00197700          CALL BTRI(2,JM)
00197800          DO 21 J = 2,JM
00197900          S1 = F(J,1)
00198000          S2 = F(J,2)
00198100          S3 = F(J,3)
00198200          S4 = F(J,4)
00198300          S5 = F(J,5)
00198301          S(KL,J,1) = S1
00198302          S(KL,J,2) = S2
00198303          S(KL,J,3) = S3
00198304          S(KL,J,4) = S4
00198305          S(KL,J,5) = S5
00198306     21     CONTINUE
00198400           ENDDO;ENDDO
00218600          RETURN
00218700          END
```


Figure 3-9. Identical Piece of Subroutine STEP in SAM Extended FORTRAN (Cont)

3.2.3.1 Sam Extended FORTRAN for SUBROUTINE STEP (LOOP DO 20).

Figure 3-10 shows the changes made in the original NASA-AMES program to produce SAM Extended FORTRAN. As can be seen, the greatest number of changes occur in the declarations. Only the named COMMON blocks, VARS, VARØ, and VAR1 need to be put in Extended Memory. Note for simplicity in accessing the last two extents on the S and Q matrices were interchanged.

The Named Common Blocks VAR3 and BTRID are put in LOCAL Memory. It should be noted that in another portion of the program, SUBROUTINE METOUT, the arrays XX, YY, and ZZ are written out after the subroutine calls. This would not be permitted and an additional copy to Extended Memory Arrays, say XX1, YY1, and ZZ1 would be needed. Also, the P array is used in a variety of ways including an EQUIVALENCE statement in other portions of the code. However, for this specific portion of the code the P array is not accessed in any way and so for convenience was left in LOCAL for the example. Copies of all data in GLOBAL memory are assumed to be in Processor Memory.

The only other changes to the program were the replacement of the DO 20 loops with the two dimensional DOALL loop (and ENDDO statement) and the replacement of the CALL statement in line 189700 to an INCLUDE since the PROCEDURE XXM1 has Extended Memory References. (Further discussion of this will be supplied later.)

```
1    R 184200        GLOBAL/BASE/MAX,JMAX,KMAX,LMAX,JM,KM,LM,GAMMA,GAMI,SML,FSMACH
2    R 184400       2,RM,CNBR,PI,INVISC,LAMIN,NP
3    R 184500        GLOBAL/GEO/NB1,NB2,RFRONT,PMAX,XR,XMAX,DRAD,DXC
4    R 184600        GLOBAL/READ/IREAD,IWRIT,NGRI
5    R 184700        GLOBAL/VIS/RE,PR,RMUE,RK
6    R 184800        EXTENDED/VARS/G(720,30,6)
7    R 184900        EXTENDED/VARO/S(720,30,5)
8    R 185000        EXTENDED/VAR1/X(720,30),Y(720,30),Z(720,30)
9    R 185100        LOCAL/VAR3/P(120,30),XX(60,4),YY(60,4),ZZ(6),4)
10   R 185300        CONTROL/COUNT/NC,NC1,DT
11   R 185400        LOCAL/BTRID/A(60,5,5),B(60,5,5),C(60,5,5),D(60),5,5),F(60,5)
12   R 188600        DOALL,K=2,KMX-2,LM
13   - 188700
14   R 189400         INCLUDE XXM1((,L,1,JMAX)
15   R 198400          ENDDO;ENDDO
```

Figure 3-10. Comparison of Original and SAM Extended FORTRAN - Subroutine STEP

3.2.3.2 Preliminary Code Analysis and Code Reorganization for STEP.

Figure 3-11 shows the preliminary code reorganization that would be performed by the compiler. The DO loop variables in line 194500 have been modified so that they now read DO 25 J=2, JMAX-1. This was done so that the initial and terminal values are composed of literals or Global variables that would exist both in Processor and Central Memory.

The code only accesses the arrays Q and S from Extended Memory. The accessing of the Q array is shown in lines 189501-189505 and in lines 194501-194510. The notation for this data movement from Extended Memory to Processor Memory is with the FORTRAN statement Q1=Q(KL, J, 1). (This notation is used for clarity and is not meant to be an implied ASSIGN statement.) The accessing of Q(KL, J-1, 6) is only necessary of J=2 for the other values exist in Processor Memory, hence, the IF test and branch at line 194501. Since the DO 25 loop exists in both the Processor and Control Unit Code the execution pattern is:

1. Set J=2
2. Synch for fetch Q(KL, 2, 6)
2. Synch for fetch Q(KL, 1, 6)
3. Synch for fetch Q(KL, 3, 6)
4. Set J=3
5. Synch for fetch Q(KL, 4, 6) (2 and 3 already in Processor Memory)
6. Set J=4
7. Synch for fetch Q(KL, 5, 6) (3 and 4 already in Processor Memory)

```
IMPLICIT/STEPPIECENSS1 (12/22/77)

 184100           SUBROUTINE STEP
 184200           GLOBAL/BASE/NMAX,JMAX,KMAX,LMAX,JM,KM,LM,GAMMA,GAMI,SMU,FSMACH
 184300          1   ,DX1,DY1,DZ1,ND,ND2,FV(5),FD(5),HD,ALP,GD,OMEGA,HDX,HDY,HDZ
 184400          2,RM,CNBR,PI,INVISC,LAMIN,NP
 184500           GLOBAL/GEO/NB1,NB2,RFRONT,RMAX,XR,XMAX,DRAD,DXC
 184600           GLOBAL/READ/IREAD,IWRIT,NGRI
 184700           GLOBAL/VIS/RE,PR,RMUE,RK
 184800           EXTENDED/VARS/Q(720,30,6)
 184900           EXTENDED/VARO/S(720,30,5)
 185000           EXTENDED/VAR1/X(720,30),Y(720,30),Z(720,30)
 185100           LOCAL/VAR3/P(120,30),XX(60,4),YY(60,4),ZZ(60,4)
 185200     C     LEVEL 2,Q,S,X,Y,Z
 185300           CONTROL/COUNT/NC,NC1,DT
 185400           LOCAL/BTRID/A(60,5,5),B(60,5,5),C(60,5,5),D(60,5,5),F(60,5)
 185500     C
 185600     C
 188200     C
 188300           RM = SMU
 188400           C8 = 1.+2.*RM
 188500           GAM2 = 2.-GAMMA
 188600           DOALL,K=2,KM,L=2,LM
 188800     C
 188900     C***FILTRX
 189000     C
 189100           KL = (L-1)*ND+K
 189200
 189300
 189400           INCLUDE XXM1(K,L,1,JMAX)
 189500           DO 12 J=1,JMAX
 189501           Q1=Q(KL,J,1)
 189502           Q2=Q(KL,J,2)
 189503           Q3=Q(KL,J,3)
 189504           Q4=Q(KL,J,4)
 189505           Q5=Q(KL,J,5)
 189600           R1 =XX(J,1)*HDX
 189700           R2 =XX(J,2)*HDX
 189800           R3 =XX(J,3)*HDX
 189900           R4 =XX(J,4)*HDX
 190000     C
 190100     C*******AMATRX
 190200     C
 190300           RR= 1./Q1
 190400           U  = Q2*RR
 190500           V  = Q3*RR
 190600           W  = Q4*RR
 190700           UU = U*R1+V*R2+W*R3
 190800           UT = U**2+V**2+W**2
 190900           C1 = GAMI*UT*.5
 191000           C2 = Q5*RR*GAMMA
 191100           C3=C2-C1
 191200           C4=R4+UU
 191300           C5=GAMI*U
 191400           C6=GAMI*V
 191500           C7=GAMI*W
 191600           D(J,1,1) = R4
 191700           D(J,1,2) = R1
 191800           D(J,1,3) = R2
 191900           D(J,1,4) = R3
 192000           D(J,1,5) = 0.
 192100           D(J,2,1) = R1*C1-U*UU
 192200           D(J,2,2) = C4+R1*GAM2*U
 192300           D(J,2,3) = -R1*C6+R2*U
 192400           D(J,2,4) = -R1*C7+R3*U
 192500           D(J,2,5) = R1*GAMI
 192600           D(J,3,1) = R2*C1-V*UU
 192700           D(J,3,2) = R1*V-R2*C5
 192800           D(J,3,3) = C4+R2*GAM2*V
 192900           D(J,3,4) = -R2*C7+R3*V
 193000           D(J,3,5) = R2*GAMI
 193100           D(J,4,1) = R3*C1-W*UU
 193200           D(J,4,2) = R1*W-R3*C5
 193300           D(J,4,3) = R2*W-R3*C6
 193400           D(J,4,4) = C4+R3*GAM2*W
 193500           D(J,4,5) = R3*GAMI
 193600           D(J,5,1) = (-C2+2.*C1)*UU
 193700           D(J,5,2) = R1*C3-C5*UU
 193800           D(J,5,3) = R2*C3-C6*UU
 193900           D(J,5,4) = R3*C3-C7*UU
```


Figure 3-11. Preliminary Compiler Code Reorganization Subroutine STEP

```
194000            D(J,5,5) = R4+GAMMA*UU
194100      C
194200      C******END OF AMATRX
194300      C
194400       12   CONTINUE
194500            DO 25 J=2,JMAX-1
194501            IF (J.GT.2) GO TO 777
194502            Q6=Q(KL,J,6)
194503            Q6M=Q(KL,J-1,6)
194504            GO TO 778
194505      777   Q6M = RX
194506              Q6= RY
194510      778   Q6P=Q(KL,J+1,6)
194511            RX  = Q6
194512            RY = Q6P
194600            RJ = 1./Q6
194700            RMJ=RM*RJ
194800            RR = RMJ*Q5M
194900            RF = RMJ*Q6P
194901              S1 = S(KL,J,1)
194902              S2 = S(KL,J,2)
194903              S3 = S(KL,J,3)
194904              S4 = S(KL,J,4)
194905              S5 = S(KL,J,5)
195000            DO 23 N=1,5
195100            A(J,N,1)  = -D(J-1,N,1)
195200            A(J,N,2)  = -D(J-1,N,2)
195300            A(J,N,3)  = -D(J-1,N,3)
195400            A(J,N,4)  = -D(J-1,N,4)
195500            A(J,N,5)  = -D(J-1,N,5)
195600            B(J,N,1)  = 0.0
195700            B(J,N,2)  = 0.0
195800            B(J,N,3)  = 0.0
195900            B(J,N,4)  = 0.0
196000            B(J,N,5)  = 0.0
196100            C(J,N,1)  = D(J+1,N,1)
196200            C(J,N,2)  = D(J+1,N,2)
196300            C(J,N,3)  = D(J+1,N,3)
196400            C(J,N,4)  = D(J+1,N,4)
196500            C(J,N,5)  = D(J+1,N,5)
196600            A(J,N,N)  = A(J,N,N)-RR
196700            B(J,N,N)  = C8
196800            C(J,N,N)  = C(J,N,N)-RF
196900       23     CONTINUE
196901              F(J,1) = S1
196902              F(J,2) = S2
196903              F(J,3) = S3
196904              F(J,4) = S4
196905              F(J,5) = S5
197000       25   CONTINUE
197100      C
197200      C*****END OF FILTRX
197300      C
197400      C
197500      C     S MUST BE ZERO ON B.C.
197600      C
197700            CALL BTRI(2,JM)
197800            DO 21 J = 2,JM
197900            S1 = F(J,1)
198000            S2 = F(J,2)
198100            S3 = F(J,3)
198200            S4 = F(J,4)
198300            S5 = F(J,5)
198301            S(KL,J,1) = S1
198302            S(KL,J,2) = S2
198303            S(KL,J,3) = S3
198304            S(KL,J,4) = S4
198305            S(KL,J,5) = S5
198400              ENDDO;ENDDO
218600            RETURN
218700            END
```

ORIGINAL PAGE IS
OF POOR QUALITY

Figure 3-11. Preliminary Compiler Code Reorganization
Subroutine STEP (Cont)

In SUBROUTINE TURBDA branches on the DOALL variable were demonstrated. This example demonstrates branching capability in fetching on inner nested DO loop variables.

Finally the fetching and storing of the array S is shown in lines 194901-194905 and 198301-198305. Because of the notation chosen, i.e., Si = S(KL, J, i) the statements were removed from the DO LOOP (23) on N. This is not a requirement. An array, say SS with subscripts could have been declared with a simple DIMENSION statement.

Figure 3-12 shows the lines of code that have been replaced (R), inserted (I), or deleted (-).

3.2.3.3 Programmatic Transformations by the Compiler and Transposition Network Calculations for STEP Portion

Figure 3-13 and 3-14 shows explicitly the address calculations for setting the Transposition Network Offset (3-13) and the Memory Module address (3-14) for each access from Extended Memory.

Considering the Control Unit Code first in a line by line basis:

188600 Hidden loop N has 2 cycles

188601 Calculation of # of PE's used to that cycle

188601 Address of Q(IVV+1,1,1) in memory which is in PE#=Ø. i.e., on cycle 1 the address of Q(1,1,1) is equal to the base address of Q in memory. On cycle 2 the address of Q(513,1,1) is the base address of Q plus 512.

188602 Address of Q(IVV+1,1,2) is 42,600 greater than Q(IVV+1,1,1)

188603 - 188639 Similar other calculations for S and Q

```
 1    R 189200
 2    R 189300
 3    I 189501          Q1=Q(KL,J,1)
 4    I 189502          Q2=Q(KL,J,2)
 5    I 189503          Q3=Q(KL,J,3)
 6    I 189504          Q4=Q(KL,J,4)
 7    I 189505          Q5=Q(KL,J,5)
 8    R 190300          RR= 1./Q1
 9    R 190400          U = Q2*RR
10    R 190500          V = Q3*RR
11    R 190600          W = Q4*RR
12    R 191000          C2 = Q5*RR*GAMMA
13    R 194500          DO 25 J=2,JMAX-1
14    I 194501          IF (J.GT.2) GO TO 777
15    I 194502          Q6=Q(KL,J,6)
16    I 194503          Q6M=Q(KL,J-1,5)
17    I 194504          GO TO 778
18    I 194505  777     Q6M = RX
19    I 194506            Q6= RY
20    I 194510  778     Q6P=Q(KL,J+1,5)
21    I 194511          RX  = Q6
22    I 194512          RY  = Q6P
23    R 194600          RJ = 1./Q6
24    R 194800          RR = RMJ*Q6M
25    R 194900          RF = RMJ*Q6P
26    I 194901            S1 = S(KL,J,1)
27    I 194902            S2 = S(KL,J,2)
28    I 194903            S3 = S(KL,J,3)
29    I 194904            S4 = S(KL,J,4)
30    I 194905            S5 = S(KL,J,5)
31    R 196900    23      CONTINUE
32    I 196901            F(J,1) = S1
33    I 196902            F(J,2) = S2
34    I 196903            F(J,3) = S3
35    I 196904            F(J,4) = S4
36    I 196905            F(J,5) = S5
37    R 197900          S1 = F(J,1)
38    R 198000          S2 = F(J,2)
39    R 198100          S3 = F(J,3)
40    R 198200          S4 = F(J,4)
41    R 198300          S5 = F(J,5)
42    I 198301          S(KL,J,1) = S1
43    I 198302          S(KL,J,2) = S2
44    I 198303          S(KL,J,3) = S3
45    I 198304          S(KL,J,4) = S4
46    I 198305          S(KL,J,5) = S5
47    I 198306   21       CONTINUE
```

Figure 3-12. Comparison of SAM Extended FORTRAN and Compiler Reorganized Code - Subroutine STEP

```
00184001    C  THE COMPILER WILL HAVE DETERMINED THE NUMBER OF CYCLES
00184002    C  OF THE HIDDEN LOOP
00184003    C
00184004    C   IE. NHIDDEN = (720/ND*LMAX+N-1)/N = 2 CYCLES
00184005    C
00184006    C   ALSO THAT ISKIP=1
00184007    C
00184100          SUBROUTINE ST P
00184200          GLOBAL/BASE/NMAX,JMAX,(MAX,LMAX,JM,KM,LM,GAMMA,GAMI,SML,FSMACH
00184300         1  ,DX1,DY1,DZ1,ND,ND2,V(5),FD(5),HD,ALP,GD,OMEGA,HDX,HDY,HDZ
00184400         2,RM,CNBR,PI,INVISC,LAMIN,NP
00184500          GLOBAL/GEO/NB1,NB2,RFRJNT,RMAX,XR,XMAX,DRAD,DXC
00184600          GLOBAL/READ/IREAD,INRIT,NGRI
00184700          GLOBAL/VIS/RE,PR,PRMUE,RK
00184800          EXTENDED/VARS/Q(720,30,6)
00184900          EXTENDED/VARO/S(720,30,5)
00185000          EXTENDED/VAR1/X(720,30),Y(720,30),Z(720,30)
00185100          LOCAL/VAR3/P(120,30),XX(60,4),YY(60,4),ZZ(60,4)
00185200    C     LEVEL 2,Q,S,X,Y,Z
00185300          CONTROL/COUNT/NC,NC1,DT
00185400          LOCAL/BTRID/A(60,5,5),B(60,5,5),C(60,5,5),D(60,5,5),F(60,5)
00185500    C
00185600    C
00188200    C
00188300
00188400
00188500
00188600          DO 20 N=1,2
00188601          IVV=512*N-512
00188630          IAOQ1=IBSQ1  + IVV
00188631          IAOQ2=IAOQ1  + 42600
00188632          IAOQ3=IAOQ2  + 42600
00188633          IAOQ4=IAOQ3  + 42600
00188634          IAOQ6=IAOQ5  + 42600
00188635          IAOS1=IBSS1  + IVV
00188636          IAOS2=IAOS1  + 42600
00188637          IAOS3=IAOS2  + 42600
00188638          IAOS4=IAOS3  + 42600
00188639          IAOS5=IAOS4  + 42600
00188640          IAOXM =  IBSX +IVV-1
00188641          IAOXP =  IBSX + IVV + 1
00188642          IAOXMN=  IBSX +IVV-ND
00188643          IAOXPN=  IBSX +IVV+ND
00188644          IAOYM =  IBSY +IVV-1
00188645          IAOYP =  IBSY + IVV + 1
00188646          IAOYMN=  IBSY +IVV-ND
00188647          IAOYPN=  IBSY +IVV+ND
00188648          IAOZM =  IBSZ +IVV-1
00188649          IAOZP =  IBSZ + IVV + 1
00188650          IAOZMN=  IBSZ +IVV-ND
00188651          IAOZPN=  IBSZ +IVV+ND
00188800    C
00188900    C***FILTRX
00189000
00189100
00189200    C
00189300
00189404          KL = (L-1)*ND+K
00189405          DO 10 J = 1,JMAX
00189406          JJ= (J-1)*720
00189407          IFSETQ6 = MOD((IAOQ6+JJ),521)
00189408    SYNCH
00189409          IFSETXM = MOD((IAOXM+JJ),521)
00189410    SYNCH
00189411          IFSETXP = MOD((IAOXP+JJ),521)
00189412    SYNCH
00189413          IFSETXMN= MOD((IAOXMN+JJ),521)
00189414    SYNCH
00189415          IFSETXPN= MOD((IAOXPN+JJ),521)
00189416    SYNCH
```

Figure 3-13. Control Unit Code for SAM - Subroutine STEP (META Assembler)

```
189417          IFSETYM  = MOD((IAOYM+JJ),521)
189418    SYNCH
189419          IFSETYP  = MOD((IAOYP+JJ),521)
189420    SYNCH
189421          IFSETYMN= MOD((IAOYMN+JJ),521)
189422    SYNCH
189423          IFSETYPN= MOD((IAOYPN+JJ),521)
189424    SYNCH
189425          IFSETZM  = MOD((IAOZM+JJ),521)
189426    SYNCH
189427          IFSETZP  = MOD((IAOZP+JJ),521)
189428    SYNCH
189429          IFSETZMN= MOD((IAOZMN+JJ),521)
189430    SYNCH
189431          IFSETZPN= MOD((IAOZPN+JJ),521)
189432    SYNCH
189450
189451
189453
189454
189455
189457
189458
189459
189460
189461       10 CONTINUE
189500          DO 12 J=1,JMAX
189501          JJ=(J-1)*720
189502          IFSETQ1 =  MOD((IAOQ1+JJ),521)
189503    SYNCH
189504          IFSETQ2 =  MOD((IAOQ2+JJ),521)
189505    SYNCH
189506          IFSETQ3 =  MOD((IAOQ3+JJ),521)
189507    SYNCH
189508          IFSETQ4 =  MOD((IAOQ4+JJ),521)
189509    SYNCH
189510          IFSETQ5 =  MOD((IAOQ5+JJ),521)
189600
189700
189900
190000
190100
190200
190300
190400
190500
190600
190700
190800
191000
191100
191200
191300
191400
191500
191600
191700
191800
191900
192000
192100
192200
192300
192400
192500
192600
192700
192800
192900
193000
193100
193200
193300
193400
193500
193600
193700
193800
193900
194000
194100
```

Figure 3-13. Control Unit Code for SAM - Subroutine STEP (META Assembler) (Cont)

```
 12    CONTINUE
       DO 25 J=2,JMAX-1
        JJ= (J-1)*720
       IF(J.GT.2) GO TO 777
       IFSETQ6 =  MOD((IAOQ6+JJ),521)
SYNCH
       IFSETQ6M = MOD((IAOQ6+JJ-720),521)
SYNCH
       GO TO 778
 777   CONTINUE
 778   IFSETQ6P =MOD((IAOQ6+JJ+720),521)
SYNCH
       IFSETS1 = MOD((IAOS1+JJ),521)
SYNCH
       IFSETS2 = MOD((IAOS2+JJ),521)
SYNCH
       IFSETS3 = MOD((IAOS3+JJ),521)
SYNCH
       IFSETS4 = MOD((IAOS4+JJ),521)
SYNCH
       IFSETS5 = MOD((IAOS5+JJ),521)
```



```
 25    CONTINUE
C
C*****END OF FILTRX
C
C
C   S MUST BE ZERO ON B.C.
C
       CALL BTRI(2,JM)
        DO 21 J=2,JM
        JJ= (J-1)*720
```

```
       IFSETS1 = MOD((IAOS1+JJ),521)
SYNCH
       IFSETS2 = MOD((IAOS2+JJ),521)
SYNCH
       IFSETS3 = MOD((IAOS3+JJ),521)
SYNCH
       IFSETS4 = MOD((IAOS4+JJ),521)
SYNCH
       IFSETS5 = MOD((IAOS5+JJ),521)
 21      CONTINUE
 20    CONTINUE
       RETURN
       END
```

Figure 3-13. Control Unit Code for SAM - Subroutine STEP (META Assembler) (Cont)

```
184001   C  THE COMPILER WILL HAVE DETERMINED THE NUMBER OF CYCLES
184002   C  OF THE HIDDEN LOOP
184003   C
184004   C    IE. NHIDDEN = (720/ND*LMAX+N-1)/N = 2 CYCLES
184005   C
184006   C    ALSO THAT ISKIP=1
184007   C
184008   C     *******
184009   C     ******* NOTE ALL SYNCHS IN THE PE CODE ARE WITH THE
184010                 PROVISO THAT THEY DO NOT FETCH FOR K.LT. 2 OR
184011   C             K.GT.KM OR L.LT. 2, OR L.GT.LM
184012   C
184013   C             IE SYNCH SHOULD BE REPLACED WITH AN EXPRESSION
184014   C
184015                 SYNCH WITH MODE=0 FOR K.LT.2,K.GT.KM,L.LT.2,L.GT.LM
184016                 NOTE THE IF BRANCHES IN EACH DO LOOP AFTER THE
184017   C             SYNCH CODE
184100          SUBROUTINE STEP
184200          GLOBAL/BASE/NMAX,MMAX,SMAX,LMAX,JM,KM,LM,GAMMA,GAMI,SMU,FSMACH
184300         1   ,DX1,DY1,DZ1,ND,ND2,PV(5),FD(5),HD,ALP,GD,OMEGA,HDX,HDY,HDZ
184400         2,RM,CNBR,PI,INVISC,LAMIN,NP
184500          GLOBAL/GEO/NB1,NB2,REFINT,RMAX,XR,XMAX,DRAD,DXC
184600          GLOBAL/READ/IREAD,IWRIT,NGRI
184700          GLOBAL/VIS/RE,PR,RMUE,RK
184800          EXTENDED/VARS/Q(720,30,6)
184900          EXTENDED/VARO/S(720,30,5)
185000          EXTENDED/VAR1/X(720,30),Y(720,30),Z(720,30)
185100          LOCAL/VAR3/P(120,30),XX(60,4),YY(60,4),ZZ(60,4)
185200   C      LEVEL 2,Q,S,X,Y,Z
185300          CONTROL/COUNT/NC,NC1,DT
185400          LOCAL/BTRID/A(60,5,5),B(60,5,5),C(60,5,5),D(60,5,5),F(60,5)
185500   C
185600   C
188200   C
188300          RM = SMU
188400          C8 = 1.+2.*RM
188500          GAM2 = 2.-GAMMA
188600          DO 20 N=1,2
188601          IVV=512*N-512
188602          IV = IVV + IPEND
188603          LM1 = IV/ND
188604          L   = LM1 + 1
188605          KM1 = IV-LM1*ND
188606          K   = KM1 + 1
188607          IAOQ1=IBSQ1 + IV
188608          IAOQ2=IAOQ1 + 42600
188609          IAOQ3=IAOQ2 + 42600
188610          IAOQ4=IAOQ3 + 42600
188611          IAOQ6=IAOQ5 + 42600
188612          IAOS1=IBSS1 + IV
188613          IAOS2=IAOS1 + 42600
188614          IAOS3=IAOS2 + 42600
188615          IAOS4=IAOS3 + 42600
188616          IAOS5=IAOS4 + 42600
188617          IAOXM = IBSX +IV-1
188618          IAOXP = IBSX + IV + 1
188619          IAOXMN= IBSX +IV-ND
188620          IAOXPN= IBSX +IV+ND
188621          IAOYM = IBSY +IV-1
188622          IAOYP = IBSY + IV + 1
188623          IAOYMN= IBSY +IV-ND
188624          IAOYPN= IBSY +IV+ND
188625          IAOZM = IBSZ +IV-1
188626          IAOZP = IBSZ + IV + 1
188627          IAOZMN= IBSZ +IV-ND
188628          IAOZPN= IBSZ +IV+ND
188800   C
188900   C***FILTRX
199000   C
199100          KL = (L-1)*ND+K
```

ORIGINAL PAGE IS
OF POOR QUALITY

Figure 3-14. Processor Code for SAM - Subroutine STEP (META Assembler)

```
192200           D(J,2,2) = C4+R1*GAM2*U
192300           D(J,2,3) = -R1*C6+R2*U
192400           D(J,2,4) = -R1*C7+R3*U
192500           D(J,2,5) = R1*GAMI
192600           D(J,3,1) = R2*C1-V*UU
192700           D(J,3,2) = R1*V-R2*C5
192800           D(J,3,3) = C4+R2*GAM2*U
192900           D(J,3,4) = -R2*C7+R3*V
193000           D(J,3,5) = R2*GAMI
193100           D(J,4,1) = R3*C1-W*UU
193200           D(J,4,2) = R1*W-R3*C5
193300           D(J,4,3) = R2*W-R3*C6
193400           D(J,4,4) = C4+R3*GAM2*U
193500           D(J,4,5) = R3*GAMI
193600           D(J,5,1) = (-C2+2.*C1)*UU
193700           D(J,5,2) = R1*C3-C5*UU
193800           D(J,5,3) = R2*C3-C6*UU
193900           D(J,5,4) = R3*C3-C7*UU
194000           D(J,5,5) = R4+GAMMA*UU
194100     C
194200     C******END OF AMATRX
194300     C
194400      12   CONTINUE
194500           DO 25 J=2,JMAX-1
194501           JJ= (J-1)*720
194502           IF (J.GT.2) GO TO 777
194512           MADDQ6 =  (IAQ06+JJ)/521
194513     SYNCH
194514           MADDQ6M = (IAQ06+JJ-720))/521
194515     SYNCH
194516           GO TO 778
194517       777 Q6M = RX
194518           Q6 = RY
194521       778 MADDQ6P =(IAQ06+JJ+720)/521
194522     SYNCH
194523           MADDS1 = (IAQS1+JJ)/521
194524     SYNCH
194525           MADDS2 = (IAQS2+JJ)/521
194526     SYNCH
194527           MADDS3 = (IAQS3+JJ)/521
194528     SYNCH
194529           MADDS4 = (IAQS4+JJ)/521
194530     SYNCH
194531           MADDS5 = (IAQS5+JJ)/521
194532            IF((K.LT.2).OR.(K.GT.(M).OR.(L.LT.2).OR.(L.GT.LM)))  GO TO25
194538           RX=Q6
194539           RY = Q6P
194540           RJ = Q(KL,6,J)
194600           RJ = 1./Q6
194700           RMJ=RM*RJ
194800           RR = RMJ*Q6M
194900           RF = RMJ*Q6P
195000           DO 23 N=1,5
195100           A(J,N,1) = -D(J-1,N,1)
195200           A(J,N,2) = -D(J-1,N,2)
195300           A(J,N,3) = -D(J-1,N,3)
195400           A(J,N,4) = -D(J-1,N,4)
195500           A(J,N,5) = -D(J-1,N,5)
195600           B(J,N,1) = 0.0
195700           B(J,N,2) = 0.0
195800           B(J,N,3) = 0.0
195900           B(J,N,4) = 0.0
196000           B(J,N,5) = 0.0
196100           C(J,N,1) = D(J+1,N,1)
196200           C(J,N,2) = D(J+1,N,2)
196300           C(J,N,3) = D(J+1,N,3)
196400           C(J,N,4) = D(J+1,N,4)
196500           C(J,N,5) = D(J+1,N,5)
196600           A(J,N,N) = A(J,N,N)-RR
196700           B(J,N,N) = C8
196800           C(J,N,N) = C(J,N,N)-RF
196900        23  CONTINUE
196901           F(J,1) = S1
196902           F(J,2) = S2
196903           F(J,3) = S3
196904           F(J,4) = S4
196905           F(J,5) = S5
197000        25 CONTINUE
197100     C
197200     C*****END OF FILTRX
197300     C
```

Figure 3-14. Processor Code for SAM - Subroutine STEP (METÀ Assembler) (Cont)

```
189200
189300
189405          DO 10 J = 1,JMAX
189406          JJ= (J-1)*720
189407          MADDQ6 =  (IAOQ6+JJ)/521
189408    SYNCH
189409          MADDXM =  (IAOXM+JJ)/521
189410    SYNCH
189411          MADDXP =  (IAOXP+JJ)/521
189412    SYNCH
189413          MADDXMN=  (IAOXMN +JJ)/521
189414    SYNCH
189415          MADDXPN=  (IAOXPN +JJ)/521
189416    SYNCH
189417          MADDYM =  (IAOYM+JJ)/521
189418    SYNCH
189419          MADDYP =  (IAOYP+JJ)/521
189420    SYNCH
189421          MADDYMN=  (IAOYMN +JJ)/521
189422    SYNCH
189423          MADDYPN=  (IAOYPN +JJ)/521
189424    SYNCH
189425          MADDZM =  (IAOZM+JJ)/521
189426    SYNCH
189427          MADDZP =  (IAOZP+JJ)/521
189428    SYNCH
189429          MADDZMN=  (IAOZMN +JJ)/521
189430    SYNCH
189431          MADDZPN=  (IAOZPN +JJ)/521
189432    SYNCH
189433           IF((K.LT.2).OR.(K.GT.(M).OR.(L.LT.2).OR(L.GT.LM) GO TO 10
189450          RJ = Q(KL,6,J)
189451          XK = (XP-XM)*DY2
189452          YK = (YP-YM)*DY2
189453          ZK = (ZP-ZM)*DY2
189454          XL = (XPN-XMN)*DZ2
189455          YL = (YPN-YMN)*DZ2
189456          ZL = (ZPN-ZMN)*DZ2
189457          XX(J,1) = (YK*ZL-ZK*YL)*RJ
189458          XX(J,2) = (ZK*XL-XK*ZL)*RJ
189459          XX(J,3) = (XK*YL-YK*XL)*RJ
189460          XX(J,4) = -OMEGA*(Z(KL,J)*XX(J,2)-Y(KL,J)*XX(J,3))
189461       10 CONTINUE
189500          DO 12 J=1,JMAX
189501          JJ=J-1
189502          MADDQ1 =  (IAOQ1+JJ)/521
189503    SYNCH
189504          MADDQ2 =  (IAOQ2+JJ)/521
189505    SYNCH
189506          MADDQ3 =  (IAOQ3+JJ)/521
189507    SYNCH
189508          MADDQ4 =  (IAOQ4+JJ)/521
189509    SYNCH
189510          MADDQ5 =  (IAOQ5+JJ)/521
189550           IF((K.LE.2).OR.(K.GT.(M).OR.(L.LE.2).OR.(L.GT.LM) GO TO 12
189600          R1 =XX(J,1)*HDX
189700          R2 =XX(J,2)*HDX
189800          R3 =XX(J,3)*HDX
189900          R4 =XX(J,4)*HDX
190000    C
190100    C*******AMATRX
190200    C
190300          RR= 1./Q1
190400          U = Q2*RR
190500          V = Q3*RR
190600          W = Q4*RR
190700          UU = U*R1+V*R2+W*R3
190800          UT = U**2+V**2+W**2
190900          C1 = GAMI*UT*.5
191000          C2 = Q5*RR*GAMMA
191100          C3=C2-C1
191200          C4=R4+UU
191300          C5=GAMI*U
191400          C6=GAMI*V
191500          C7=GAMI*W
191600          D(J,1,1) = R4
191700          D(J,1,2) = R1
191800          D(J,1,3) = R2
191900          D(J,1,4) = R3
192000          D(J,1,5) = 0.
192100          D(J,2,1) = R1*C1-U*UU
```

Figure 3-14. Processor Code for SAM - Subroutine STEP (META Assembler) (Cont)

```
197400    C
197500    C  S MUST BE ZERO ON B.C.
197600          IF((K.LE.2).OR.(K.GT.KM).OR.(L.LE.2).OR.(L.GT.LM)GO TO666
197700          CALL BTRI(2,JM)
197800      666 DO 21 J = 2,JM
197900          S1 = F(J,1)
198000          S2 = F(J,2)
198100          S3 = F(J,3)
198200          S4 = F(J,4)
198300          S5 = F(J,5)
198301          MADDS1 = (IAOS1+JJ)/521
198302    SYNCH
198303          MADDS2 = (IAOS2+JJ)/521
198304    SYNCH
198305          MADDS3 = (IAOS3+JJ)/521
198306    SYNCH
198307          MADDS4 = (IAOS4+JJ)/521
198308    SYNCH
198309          MADDS5 = (IAOS5+JJ)/521
198311       21   CONTINUE
198400       20    CONTINUE
218600          RETURN
218700          END
```

Figure 3-14. Processor Code for SAM - Subroutine STEP (META Assembler) (Cont)

188640 Since the PROCEDURE XXM1 has been INCLUDED it is necesary to perform address calculations for the X, Y, Z arrays. In a similar fashion IAØXM represents the address of X(KL-1,J) or rather X(Ø1) on cycle 1 and X(511,1) on cycle 2. It appears that at this juncture that one is accessing outside of array bounds. Note that in the original FORTRAN (Figure 3-6) the L and K loops go from 2 to LM and KM respectively while the hidden N loop of this Figure does not indicate this. Line 189433 of Figure 3-12 is an IF branch meant to indicate that the code will not be executed. In fact a transposition network calculation will be made for PE#=0 on an address one less than the base address in order to calculate the OFFSET. However, because of the K,L calculations done in the PE code those specific accesses are not performed. i.e., for this case those PE's whose K or L value is less than 2 or greater than KM or LM will not perform the computation.

188641-18850 Similar computations for X(KL+ND,J) etc. with J always set equal to 1.

189405 First inner J loop which has been included from procedure XXM1.

189407-189432 Synchronizations and OFFSET computations by MOD (Address,521)

189500-19440 DO 12 loop with attended accesses of Q matrix values 1-5.

194500-197000 DO 25 loop with the fetches to Q(KL,J,6), Q(KL,J=1,6) and Q(KL,J+1),6). For simplicity additional computations were not made in the N loop initiation to specify an OAØQ6P(plus) + IAØQ6M(minus) equivalent to the J+1 and J-1 but rather left the addition and subtraction to be done in the MOD function expressions of line 194527 and 194523. This would infact be inefficient as it would be performed for each J. The IF branch spanning 194503-194527 has been explained in 3.2.3.2.

197800-198310 DO 21 loop

198400 End of DO 20 loop - The cycle loop

In an early analogous manner the Processor Element Code is generated. In this case however each processor performs a calculation to determine relative address as a function of cycle and PE#.

188602 Calculation of relative address

188604 Calculation of L value

188606 Calculation of K value

188607-188628 Calculation of array addresses in Extended Memory

189405-189461 DO 10 loop included from XXM1 procedure. Note as the J index increases the array address increases by 720. Also line 189433 indicates the "non-computation" for undesirable K and L values

189500-194400 DO 12 loop

194500-197000 DO 25 loop

197700 CALL BTRI a SUBROUTINE in the normal FORTRAN sense. Its modification into SAM Extended FORTRAN is shown in Figure 3-21 to be few indeed. (A branch around BTRI should be explicitly shown similar to line 189433)

197800-198311 DO 21 loop

198400 End of N loop for number of cycles

3.2.3.4 Assembler Code for STEP

To be supplied in Phase II

3.2.3.5 Subroutine BTRI - SAM Extended FORTRAN

As can be seen in Figure 3-19 the comparision of the original FORTRAN (Figure 3-17) and the SAM Extended FORTRAN (Figure 3-18) only one change had to be made in the code. This was the LOCAL declaration for Named COMMON/BTRID/. Since no extended variables are fetched or stored in this piece of code it runs entirely internal to the processor as written.

3.2.3.6 SUBROUTINE XXM and XXM1.

It was noted in examining the IMPLICIT code that the majority of calls to the SUBROUTINE XXM occurred in loops whose initial and terminal members precluded taking the branches which occurred in this code. (Lines 245800, 245900, 247500, and 247600.) Since this reduces the performance of the whole code on both the CDC7600 and on SAM the code was modified into two SUBROUTINES. One, XXM, to be used when the calling loop had initial and terminal values and XXM1 for those calling loops in which K never equal to 1 or KMAX and L never equal to 1 or LMAX. See Figures 3-20 and 3-21.

Figure 3-22 shows XXM1 written in SAM Extended FORTRAN and 3-23 shows the differences.

Since this code was brought into STEP via the INCLUDE statement, further discussion is not necessary.

```
42200           SUBROUTINE BTRI(ILA,IUA)
42300           LOCAL/BTRID/A(60,5,5),B(60,5,5),C(60,5,5),D(60,5,5),F(60,5)
42400           DIMENSION H(5,5)
42500           REAL L11,L21,L22,L31,L32,L33,L41,L42,L43,L44,L51,L52,L53,L54,L55
42600           IL=ILA
42700           IU=IUA
42800           IS=IL+1
42900           IE=IU-1
43000     C     INSERT LUDEC
43100           L11=1./B(IL,1,1)
43200           L21=B(IL,2,1)
43300           U12=B(IL,1,2)*L11
43400           L22=1./(B(IL,2,2)-L21*U12)
43500           U13=B(IL,1,3)*L11
43600           U14 = B(IL,1,4)*L11
43700           U15=B(IL,1,5)*L11
43800           L31=B(IL,3,1)
43900           L32=B(IL,3,2)-L31*U12
44000           U23=(B(IL,2,3)-L21*U13)*L22
44100           L33=1./(B(IL,3,3)-U13*L31-U23*L32)
44200           U24=(B(IL,2,4)-L21*U14)*L22
44300           U25=(B(IL,2,5)-L21*U15)*L22
44400           L41=B(IL,4,1)
44500           L42=B(IL,4,2)-L41*U12
44600           L43=B(IL,4,3)-L41*U13-L42*U23
44700           U34=(B(IL,3,4)-L31*U14-L32*U24)*L33
44800           L44=1./(B(IL,4,4)-U14*L41-U24*L42-U34*L43)
44900           U35=(B(IL,3,5)-L31*U15-L32*U25)*L33
45000           L51=B(IL,5,1)
45100           L52=B(IL,5,2)-L51*U12
45200           L53=B(IL,5,3)-L51*U13-L52*U23
45300           L54=B(IL,5,4)-L51*U14-L52*U24-L53*U34
45400           U45=(B(IL,4,5)-L41*U15-L42*U25-L43*U35)*L44
45500           L55=1./(B(IL,5,5)-L51*U15-L52*U25-L53*U35-L54*U45)
45600     C     COMPUTE LITTLE R S
45700           D1=L11*F(IL,1)
45800           D2=L22*(F(IL,2)-L21*D1)
45900           D3=L33*(F(IL,3)-L31*D1-L32*D2)
46000           D4=L44*(F(IL,4)-L41*D1-L42*D2-L43*D3)
46100           D5=L55*(F(IL,5)-L51*D1-L52*D2-L53*D3-L54*D4)
46200     C     COMPUTE BIG R S
46300           F(IL,5)=D5
46400           F(IL,4)=D4-U45*D5
46500           F(IL,3)=D3-U34*F(IL,4)-U35*D5
46600           F(IL,2)=D2-U23*F(IL,3)-U24*F(IL,4)-U25*D5
46700           F(IL,1)=D1-U12*F(IL,2)-U13*F(IL,3)-U14*F(IL,4)-U15*D5
46800     C     COMPUTE C PRIME FOR FIRST ROW
46900           DO 12 M=1,5
47000           D1=L11*C(IL,1,M)
47100           D2=L22*(C(IL,2,M)-L21*D1)
47200           D3=L33*(C(IL,3,M)-L31*D1-L32*D2)
47300           D4=L44*(C(IL,4,M)-L41*D1-L42*D2-L43*D3)
47400           D5=L55*(C(IL,5,M)-L51*D1-L52*D2-L53*D3-L54*D4)
47500           B(IL,5,M)=D5
47600           B(IL,4,M)=D4-U45*D5
47700           B(IL,3,M) = D3-U34*B(IL,4,M)-U35*D5
47800           B(IL,2,M) = D2-U23*B(IL,3,M)-U24*B(IL,4,M)-U25*D5
47900     12    B(IL,1,M) = D1-U12*B(IL,2,M)-U13*B(IL,3,M)-U14*B(IL,4,M)-U15*D5
48000           DO 13 I=IS,IE
48100     C       COMPUTE B PRIME*BIGR
48200           DO 14 N=1,5
48300      14   F(I,N)=F(I,N)-A(I,N,1)*F(I-1,1)-A(I,N,2)*F(I-1,2)-A(I,N,3)*F(I-1,
48400          *)-A(I,N,4)*F(I-1,4)-A(I,N,5)*F(I-1,5)
48500     C       COMPUTE B PRIME
48600           DO 11 N=1,5
48700           DO 11 M=1,5
48800      11   H(N,M)=B(I,N,M)-A(I,N,1)*B(I-1,1,M)-A(I,N,2)*B(I-1,2,M)-A(I,N,3)*
48900          *B(I-1,3,M)-A(I,N,4)*B(I-1,4,M)-A(I,N,5)*B(I-1,5,M)
49000     C     INSERT LUDEC AGAIN
49100           L11=1./H(1,1)
```

Figure 3-17. Original FORTRAN - Subroutine BTRI

```
49200           L21=H(2,1)
49300           U12=H(1,2)*L11
49400           L22=1./(H(2,2)-L21*U12)
49500           U13=H(1,3)*L11
49600           U14=H(1,4)*L11
49700           U15=H(1,5)*L11
49800           L31=H(3,1)
49900           L32=H(3,2)-L31*U12
50000           U23=(H(2,3)-L21*U13)*L22
50100           L33=1./(H(3,3)-U13*L31-U23*L32)
50200           U24=(H(2,4)-L21*U14)*L22
50300           U25=(H(2,5)-L21*U15)*L22
50400           L41=H(4,1)
50500           L42=H(4,2)-L41*U12
50600           L43=H(4,3)-L41*U13-L42*U23
50700           U34=(H(3,4)-L31*U14-L32*U24)*L33
50800           L44=1./(H(4,4)-U14*L41-U24*L42-U34*L43)
50900           U35=(H(3,5)-L31*U15-L32*U25)*L33
51000           L51=H(5,1)
51100           L52=H(5,2)-L51*U12
51200           L53=H(5,3)-L51*U13-L52*U23
51300           L54=H(5,4)-L51*U14-L52*U24-L53*U34
51400           U45=(H(4,5)-L41*U15-L42*U25-L43*U35)*L44
51500           L55=1./(H(5,5)-L51*U15-L52*U25-L53*U35-L54*U45)
51600     C     COMPUTE LITTLE RTS
51700           D1=L11*F(I,1)
51800           D2=L22*(F(I,2)-L21*D1)
51900           D3=L33*(F(I,3)-L31*D1-L32*D2)
52000           D4=L44*(F(I,4)-L41*D1-L42*D2-L43*D3)
52100           D5=L55*(F(I,5)-L51*D1-L52*D2-L53*D3-L54*D4)
52200     C     COMPUTE BIG RTS
52300           F(I,5)=D5
52400           F(I,4)=D4-U45*D5
52500           F(I,3)=D3-U34*F(I,4)-U35*D5
52600           F(I,2)=D2-U23*F(I,3)-U24*F(I,4)-U25*D5
52700           F(I,1)=D1-U12*F(I,2)-U13*F(I,3)-U14*F(I,4)-U15*D5
52800     C     COMPUTE C PRIMES
52900           DO 15 M=1,5
53000           D1=L11*C(I,1,M)
53100           D2=L22*(C(I,2,M)-L21*D1)
53200           D3=L33*(C(I,3,M)-L31*D1-L32*D2)
53300           D4=L44*(C(I,4,M)-L41*D1-L42*D2-L43*D3)
53400           D5=L55*(C(I,5,M)-L51*D1-L52*D2-L53*D3-L54*D4)
53500           B(I,5,M)=D5
53600           B(I,4,M)=D4-U45*D5
53700           B(I,3,M) = D3-U34*B(I,4,M)-U35*D5
53800           B(I,2,M) = D2-U23*B(I,3,M)-U24*B(I,4,M)-U25*D5
53900     15    B(I,1,M) = D1-U12*B(I,2,M)-U13*B(I,3,M)-U14*B(I,4,M)-U15*D5
54000     13    CONTINUE
54100           I=IU
54200     C      COMPUTE B PRIME*BIG R FOR LAST ROW
54300           DO 17 N=1,5
54400      17   F(I,N)=F(I,N)-A(I,N,1)*F(I-1,1)-A(I,N,2)*F(I-1,2)-A(I,N,3)*
54500          * F(I-1,3)-A(I,N,4)*F(I-1,4)-A(I,N,5)*F(I-1,5)
54600     C      COMPUTE B PRIME
54700           DO 18 N=1,5
54800           DO 18 M=1,5
54900      18   H(N,M)=B(I,N,M)-A(I,N,1)*B(I-1,1,M)-A(I,N,2)*B(I-1,2,M)-A(I,N,3)*
55000          *B(I-1,3,M)-A(I,N,4)*B(I-1,4,M)-A(I,N,5)*B(I-1,5,M)
55100     C     INSERT LUDEC AGAIN
55200           L11=1./H(1,1)
55300           L21=H(2,1)
55400           U12=H(1,2)*L11
55500           L22=1./(H(2,2)-L21*U12)
55600           U13=H(1,3)*L11
55700           U14=H(1,4)*L11
55800           U15=H(1,5)*L11
55900           L31=H(3,1)
56000           L32=H(3,2)-L31*U12
56100           U23=(H(2,3)-L21*U13)*L22
56200           L33=1./(H(3,3)-U13*L31-U23*L32)
56300           U24=(H(2,4)-L21*U14)*L22
56400           U25=(H(2,5)-L21*U15)*L22
56500           L41=H(4,1)
56600           L42=H(4,2)-L41*U12
56700           L43=H(4,3)-L41*U13-L42*U23
56800           U34=(H(3,4)-L31*U14-L32*U24)*L33
56900           L44=1./(H(4,4)-U14*L41-U24*L42-U34*L43)
57000           U35=(H(3,5)-L31*U15-L32*U25)*L33
57100           L51=H(5,1)
57200           L52=H(5,2)-L51*U12
```

ORIGINAL PAGE IS
OF POOR QUALITY

Figure 3-17. Original FORTRAN - Subroutine BTRI (Cont)

```
57300           L53=H(5,3)-L51*U13-L52*U23
57400           L54=H(5,4)-L51*U14-L52*U24-L53*U34
57500           U45=(H(4,5)-L41*U15-L42*U25-L43*U35)*L44
57600           L55=1./(H(5,5)-L51*U15-L52*U25-L53*U35-L54*U45)
57700     C     COMPUTE LITTLE RIS
57800           D1=L11*F(I,1)
57900           D2=L22*(F(I,2)-L21*D1)
58000           D3=L33*(F(I,3)-L31*D1-L32*D2)
58100           D4=L44*(F(I,4)-L41*D1-L42*D2-L43*D3)
58200           D5=L55*(F(I,5)-L51*D1-L52*D2-L53*D3-L54*D4)
58300     C     COMPUTE BIG RIS
58400           F(I,5)=D5
58500           F(I,4)=D4-U45*D5
58600           F(I,3)=D3-U34*F(I,4)-U35*D5
58700           F(I,2)=D2-U23*F(I,3)-U24*F(I,4)-U25*D5
58800           F(I,1)=D1-U12*F(I,2)-U13*F(I,3)-U14*F(I,4)-U15*D5
58900           I=IU
59000     20    I=I-1
59100           DO 19 N=1,5
59200      19   F(I,N)=F(I,N)-F(I+1,1)*B(I,N,1)-F(I+1,2)*B(I,N,2)-F(I+1,3)*B(I,N,3
59300          *  )-F(I+1,4)*B(I,N,4)-F(I+1,5)*B(I,N,5)
59400           IF (I.GT.IL)GOTO20
59500           RETURN
59600           END
```

Figure 3-17. Original FORTRAN - Subroutine BTRI (Cont)

```
42200           SUBROUTINE BTRI(ILA,IUA)
42300           COMMON/BTRID/A(60,5,5),B(60,5,5),C(60,5,5),D(60,5,5),F(60,5)
42400           DIMENSION H(5,5)
42500           REAL L11,L21,L22,L31,L32,L33,L41,L42,L43,L44,L51,L52,L53,L54,L55
42600           IL=ILA
42700           IU=IUA
42800           IS=IL+1
42900           IE=IU-1
43000     C     INSERT LUDEC
43100           L11=1./B(IL,1,1)
43200           L21=B(IL,2,1)
43300           U12=B(IL,1,2)*L11
43400           L22=1./(B(IL,2,2)-L21*U12)
43500           U13=B(IL,1,3)*L11
43600           U14 = B(IL,1,4)*L11
43700           U15=B(IL,1,5)*L11
43800           L31=B(IL,3,1)
43900           L32=B(IL,3,2)-L31*U12
44000           U23=(B(IL,2,3)-L21*U13)*L22
44100           L33=1./(B(IL,3,3)-U13*L31-U23*L32)
44200           U24=(B(IL,2,4)-L21*U14)*L22
44300           U25=(B(IL,2,5)-L21*U15)*L22
44400           L41=B(IL,4,1)
44500           L42=B(IL,4,2)-L41*U12
44600           L43=B(IL,4,3)-L41*U13-L42*U23
44700           U34=(B(IL,3,4)-L31*U14-L32*U24)*L33
44800           L44=1./(B(IL,4,4)-U14*L41-U24*L42-U34*L43)
44900           U35=(B(IL,3,5)-L31*U15-L32*U25)*L33
45000           L51=B(IL,5,1)
45100           L52=B(IL,5,2)-L51*U12
45200           L53=B(IL,5,3)-L51*U13-L52*U23
45300           L54=B(IL,5,4)-L51*U14-L52*U24-L53*U34
45400           U45=(B(IL,4,5)-L41*U15-L42*U25-L43*U35)*L44
45500           L55=1./(B(IL,5,5)-L51*U15-L52*U25-L53*U35-L54*U45)
45600     C     COMPUTE LITTLE R S
45700           D1=L11*F(IL,1)
45800           D2=L22*(F(IL,2)-L21*D1)
45900           D3=L33*(F(IL,3)-L31*D1-L32*D2)
46000           D4=L44*(F(IL,4)-L41*D1-L42*D2-L43*D3)
46100           D5=L55*(F(IL,5)-L51*D1-L52*D2-L53*D3-L54*D4)
46200     C     COMPUTE BIG R S
46300           F(IL,5)=D5
46400           F(IL,4)=D4-U45*D5
46500           F(IL,3)=D3-U34*F(IL,4)-U35*D5
46600           F(IL,2)=D2-U23*F(IL,3)-U24*F(IL,4)-U25*D5
46700           F(IL,1)=D1-U12*F(IL,2)-U13*F(IL,3)-U14*F(IL,4)-U15*D5
46800     C     COMPUTE C PRIME FOR FIRST ROW
46900           DO 12 M=1,5
47000           D1=L11*C(IL,1,M)
47100           D2=L22*(C(IL,2,M)-L21*D1)
47200           D3=L33*(C(IL,3,M)-L31*D1-L32*D2)
47300           D4=L44*(C(IL,4,M)-L41*D1-L42*D2-L43*D3)
47400           D5=L55*(C(IL,5,M)-L51*D1-L52*D2-L53*D3-L54*D4)
47500           B(IL,5,M)=D5
47600           B(IL,4,M)=D4-U45*D5
47700           B(IL,3,M) = D3-U34*B(IL,4,M)-U35*D5
47800           B(IL,2,M) = D2-U23*B(IL,3,M)-U24*B(IL,4,M)-U25*D5
47900     12    B(IL,1,M) = D1-U12*B(IL,2,M)-U13*B(IL,3,M)-U14*B(IL,4,M)-U15*D5
48000           DO 13 I=IS,IE
48100     C       COMPUTE B PRIME*BIGR
48200           DO 14 N=1,5
48300      14   F(I,N)=F(I,N)-A(I,N,1)*F(I-1,1)-A(I,N,2)*F(I-1,2)-A(I,N,3)*F(I-1,
48400          *)-A(I,N,4)*F(I-1,4)-A(I,N,5)*F(I-1,5)
48500     C       COMPUTE B PRIME
48600           DO 11 N=1,5
48700           DO 11 M=1,5
48800      11   H(N,M)=B(I,N,M)-A(I,N,1)*B(I-1,1,M)-A(I,N,2)*B(I-1,2,M)-A(I,N,3)*
48900          *B(I-1,3,M)-A(I,N,4)*B(I-1,4,M)-A(I,N,5)*B(I-1,5,M)
49000     C       INSERT LUDEC AGAIN
49100           L11=1./H(1,1)
```

Figure 3-18. SAM Extended FORTRAN Subroutine BTRI

```
      L21=H(2,1)
      U12=H(1,2)*L11
      L22=1./(H(2,2)-L21*U12)
      U13=H(1,3)*L11
      U14=H(1,4)*L11
      U15=H(1,5)*L11
      L31=H(3,1)
      L32=H(3,2)-L31*U12
      U23=(H(2,3)-L21*U13)*L22
      L33=1./(H(3,3)-U13*L31-U23*L32)
      U24=(H(2,4)-L21*U14)*L22
      U25=(H(2,5)-L21*U15)*L22
      L41=H(4,1)
      L42=H(4,2)-L41*U12
      L43=H(4,3)-L41*U13-L42*U23
      U34=(H(3,4)-L31*U14-L32*U24)*L33
      L44=1./(H(4,4)-U14*L41-U24*L42-U34*L43)
      U35=(H(3,5)-L31*U15-L32*U25)*L33
      L51=H(5,1)
      L52=H(5,2)-L51*U12
      L53=H(5,3)-L51*U13-L52*U23
      L54=H(5,4)-L51*U14-L52*U24-L53*U34
      U45=(H(4,5)-L41*U15-L42*U25-L43*U35)*L44
      L55=1./(H(5,5)-L51*U15-L52*U25-L53*U35-L54*U45)
C     COMPUTE LITTLE RIS
      D1=L11*F(I,1)
      D2=L22*(F(I,2)-L21*D1)
      D3=L33*(F(I,3)-L31*D1-L32*D2)
      D4=L44*(F(I,4)-L41*D1-L42*D2-L43*D3)
      D5=L55*(F(I,5)-L51*D1-L52*D2-L53*D3-L54*D4)
C     COMPUTE BIG RIS
      F(I,5)=D5
      F(I,4)=D4-U45*D5
      F(I,3)=D3-U34*F(I,4)-U35*D5
      F(I,2)=D2-U23*F(I,3)-U24*F(I,4)-U25*D5
      F(I,1)=D1-U12*F(I,2)-U13*F(I,3)-U14*F(I,4)-U15*D5
C     COMPUTE C PRIMES
      DO 15 M=1,5
      D1=L11*C(I,1,M)
      D2=L22*(C(I,2,M)-L21*D1)
      D3=L33*(C(I,3,M)-L31*D1-L32*D2)
      D4=L44*(C(I,4,M)-L41*D1-L42*D2-L43*D3)
      D5=L55*(C(I,5,M)-L51*D1-L52*D2-L53*D3-L54*D4)
      B(I,5,M)=D5
      B(I,4,M)=D4-U45*D5
      B(I,3,M) = D3-U34*B(I,4,M)-U35*D5
      B(I,2,M) = D2-U23*B(I,3,M)-U24*B(I,4,M)-U25*D5
15    B(I,1,M) = D1-U12*B(I,2,M)-U13*B(I,3,M)-U14*B(I,4,M)-U15*D5
13    CONTINUE
      I=IU
C       COMPUTE B PRIME*BIG R FOR LAST ROW
      DO 17 N=1,5
17    F(I,N)=F(I,N)-A(I,N,1)*F(I-1,1)-A(I,N,2)*F(I-1,2)-A(I,N,3)*
     * F(I-1,3)-A(I,N,4)*F(I-1,4)-A(I,N,5)*F(I-1,5)
C       COMPUTE B PRIME
      DO 18 N=1,5
      DO 18 M=1,5
18    H(N,M)=B(I,N,M)-A(I,N,1)*B(I-1,1,M)-A(I,N,2)*B(I-1,2,M)-A(I,N,3)*
     *B(I-1,3,M)-A(I,N,4)*B(I-1,4,M)-A(I,N,5)*B(I-1,5,M)
C     INSERT LUDEC AGAIN
      L11=1./H(1,1)
      L21=H(2,1)
      U12=H(1,2)*L11
      L22=1./(H(2,2)-L21*U12)
      U13=H(1,3)*L11
      U14=H(1,4)*L11
      U15=H(1,5)*L11
      L31=H(3,1)
      L32=H(3,2)-L31*U12
      U23=(H(2,3)-L21*U13)*L22
      L33=1./(H(3,3)-U13*L31-U23*L32)
      U24=(H(2,4)-L21*U14)*L22
      U25=(H(2,5)-L21*U15)*L22
      L41=H(4,1)
      L42=H(4,2)-L41*U12
      L43=H(4,3)-L41*U13-L42*U23
      U34=(H(3,4)-L31*U14-L32*U24)*L33
      L44=1./(H(4,4)-U14*L41-U24*L42-U34*L43)
      U35=(H(3,5)-L31*U15-L32*U25)*L33
      L51=H(5,1)
      L52=H(5,2)-L51*U12
```

Figure 3-18. SAM Extended FORTRAN Subroutine BTRI (Cont)

```
57300           L53=H(5,3)-L51*U13-L52*U23
57400           L54=H(5,4)-L51*U14-L52*U24-L53*U34
57500           U45=(H(4,5)-L41*U15-L42*U25-L43*U35)*L44
57600           L55=1./(H(5,5)-L51*U15-L52*U25-L53*U35-L54*U45)
57700     C     COMPUTE LITTLE RIS
57800           D1=L11*F(I,1)
57900           D2=L22*(F(I,2)-L21*D1)
58000           D3=L33*(F(I,3)-L31*D1-L32*D2)
58100           D4=L44*(F(I,4)-L41*D1-L42*D2-L43*D3)
58200           D5=L55*(F(I,5)-L51*D1-L52*D2-L53*D3-L54*D4)
58300     C     COMPUTE BIG RIS
58400           F(I,5)=D5
58500           F(I,4)=D4-U45*D5
58600           F(I,3)=D3-U34*F(I,4)-U35*D5
58700           F(I,2)=D2-U23*F(I,3)-U24*F(I,4)-U25*D5
58800           F(I,1)=D1-U12*F(I,2)-U13*F(I,3)-U14*F(I,4)-U15*D5
58900           I=IU
59000     20    I=I-1
59100           DO 19 N=1,5
59200      19   F(I,N)=F(I,N)-F(I+1,1)*B(I,N,1)-F(I+1,2)*B(I,N,2)-F(I+1,3)*B(I,N,
          *    )-F(I+1,4)*B(I,N,4)-F(I+1,5)*B(I,N,5)
59300           IF (I.GT.IL)GOTO20
59400           RETURN
59500           END
59600
```

Figure 3-18. SAM Extended FORTRAN Subroutine BTRI (Cont)

```
1    R 42300        LOCAL/BTRID/A(60,5,5),B(60,5,5),C(60,5,5),D(60,5,5),F(60,5)
```

Figure 3-19. Comparison of Original FORTRAN and SAM Extended FORTRAN - Subroutine BTRI

```
      SUBROUTINE XXM(M,LA,J1A,J2A)
      COMMON/BASE/NMAX,JMAX,KMAX,LMAX,JM,KM,LM,DT,GAMMA,GAMI,SML,ESMACH
     1   ,DX1,DY1,DZ1,ND,ND2,EV(5),FD(5),HD,ALP,GD,OMEGA,HDX,HDY,HDZ
     2,RM,CNBR,PI,ITR,IAVISC,LAMIN,NP,INTL,INT2,INT3
      COMMON/GEO/NB1,NB2,REFINT,RMAX,XR,XMAX,DRAD,DXC
      COMMON/READ/IREAD,IWRIT,NGRI
      COMMON/VIS/RE,PR,RMUE,RK
      COMMON/VARS/Q(720,6,30)
      COMMON/VARO/S(720,5,30)
      COMMON/VAR1/X(720,30),Y(720,30),Z(720,30)
      COMMON /VAR3/P(120,30),XX(60,4),YY(60,4),ZZ(60,4)
C     LEVEL 2,Q,S,X,Y,Z
      COMMON/COUNT/NC,NC1
      COMMON /FLSH/DX2,DY2,DZ2
C
C  XI METRICS FORMED FOR A K,L LINE IN J
C
C
C  SYMMETRY
C
      K = M
      L=LA
      J1=J1A
      J2=J2A
      KL = (L-1)*ND+K
      DO 10 J = J1,J2
      RJ = Q(KL,6,J)
       IF(K.EQ.1) GO TO 50
      IF(K.EQ.KMAX) GO TO 51
      XK = (X(KL+1,J)-X(KL-1,J))*DY2
      YK = (Y(KL+1,J)-Y(KL-1,J))*DY2
      ZK = (Z(KL+1,J)-Z(KL-1,J))*DY2
      GO TO 72
   50 CONTINUE
      XK = (-3.*X(KL,J)+4.*X(KL+1,J)-X(KL+2,J))*DY2
      YK = (-3.*Y(KL,J)+4.*Y(KL+1,J)-Y(KL+2,J))*DY2
      ZK = (-3.*Z(KL,J)+4.*Z(KL+1,J)-Z(KL+2,J))*DY2
      GO TO 72
   51 CONTINUE
      XK = (3.*X(KL,J)-4.*X(KL-1,J)+X(KL-2,J))*DY
      YK = (3.*Y(KL,J)-4.*Y(KL-1,J)+Y(KL-2,J))*DY
      ZK = (3.*Z(KL,J)-4.*Z(KL-1,J)+Z(KL-2,J))*DY
   72 CONTINUE
      IF(L.EQ.1) GO TO 52
      IF(L.EQ.LMAX) GO TO 53
      XL = (X(KL+ND,J)-X(KL-ND,J))*DZ2
      YL = (Y(KL+ND,J)-Y(KL-ND,J))*DZ2
      ZL = (Z(KL+ND,J)-Z(KL-ND,J))*DZ2
      GO TO 60
   52 CONTINUE
      XL = (-3.*X(KL,J)+4.*X(KL+ND,J)-X(KL+2*ND,J))*D
      YL = (-3.*Y(KL,J)+4.*Y(KL+ND,J)-Y(KL+2*ND,J))*D
      ZL = (-3.*Z(KL,J)+4.*Z(KL+ND,J)-Z(KL+2*ND,J))*D
      GO TO 60
   53 CONTINUE
      XL = (3.*X(KL,J)-4.*X(KL-ND,J)+X(KL-2*ND,J))*DZ
      YL = (3.*Y(KL,J)-4.*Y(KL-ND,J)+Y(KL-2*ND,J))*DZ
      ZL = (3.*Z(KL,J)-4.*Z(KL-ND,J)+Z(KL-2*ND,J))*DZ
   60 CONTINUE
      XX(J,1) = (YK*ZL-ZK*YL)*RJ
      XX(J,2) = (ZK*XL-XK*ZL)*RJ
      XX(J,3) = (XK*YL-YK*XL)*RJ
      XX(J,4) = -OMEGA*(Z(KL,J)*XX(J,2)-Y(KL,J)*XX(J,3))
   10 CONTINUE
      RETURN
      END
```

Figure 3-20. Original FORTRAN - Subroutine XXM

```
243200          SUBROUTINE XXM(M,LA,J1A,J2A)
243300          COMMON/BASE/NMAX,JMAX,KMAX,LMAX,JM,KM,LM,DT,GAMMA,GAMI,SMU,FSMACH
243400         1   ,DX1,DY1,DZ1,ND,ND2,FV(5),FD(5),HD,ALP,GD,OMEGA,HDX,HDY,HDZ
243500         2,RM,CNBR,PI,ITR,INVISC,LAMIN,NP,INT1,INT2,INT3
243600          COMMON/GEO/NB1,NB2,RFRNT,RMAX,XR,XMAX,ORAD,DXC
243700          COMMON/READ/IREAD,IWRIT,NGRI
243800          COMMON/VIS/RE,PR,RMUE,RK
243900          COMMON/VARS/Q(720,6,30)
244000          COMMON/VARO/S(720,5,30)
244100          COMMON/VAR1/X(720,30),Y(720,30),Z(720,30)
244200          COMMON /VAR3/P(120,30),XX(60,4),YY(60,4),ZZ(60,4)
244300  C       LEVEL 2,Q,S,X,Y,Z
244400          COMMON/COUNT/NC,NC1
244500          COMMON /FLSH/DX2,DY2,DZ2
244600  C
244700  C   XI METRICS FORMED FOR A K,L LINE IN J
244800  C
244900  C
245000  C   SYMMETRY
245100  C
245200          K = M
245300          L=LA
245400          J1=J1A
245500          J2=J2A
245600          KL = (L-1)*ND+K
245700          DO 10 J = J1,J2
245800          RJ = Q(KL,6,J)
245900          XK = (X(KL+1,J)-X(KL-1,J))*DY2
246000          YK = (Y(KL+1,J)-Y(KL-1,J))*DY2
246100          ZK = (Z(KL+1,J)-Z(KL-1,J))*DY2
246200          XL = (X(KL+ND,J)-X(KL-ND,J))*DZ2
246300          YL = (Y(KL+ND,J)-Y(KL-ND,J))*DZ2
246400          ZL = (Z(KL+ND,J)-Z(KL-ND,J))*DZ2
246500          XX(J,1) = (YK*ZL-ZK*YL)*RJ
246600          XX(J,2) = (ZK*XL-XK*ZL)*RJ
246700          XX(J,3) = (XK*YL-YK*XL)*RJ
246800          XX(J,4) = -OMEGA*(Z(KL,J)*XX(J,2)-Y(KL,J)*XX(J,3))
246900      10  CONTINUE
247000          RETURN
247100          END
```

Figure 3-21. Modified Version of Subroutine XXM1 for Improved Performance on Serial or Parallel Machine

```
243200          PROCEDURE XXM(M,LA,J1A,J2A)
243300          GLOBAL/BASE/NMAX,JMAX,KMAX,LMAX,JM,KM,LM,GAMMA,GAMI,SMU,FSMACH
243400         1  ,DX1,DY1,DZ1,ND,ND2,TV(5),FD(5),HD,ALP,GD,OMEGA,HDX,HDY,HDZ
243500         2,RM,CNBR,PI,INVISC,LAMIN,NP
243600          GLOBAL/GEO/NB1,NB2,RFRONT,RMAX,XR,XMAX,DRAD,DXC
243700          GLOBAL/READ/IREAD,IWRIT,NGRI
243800          GLOBAL/VIS/RE,PR,RMUE,RK
243900          EXTENDED/VARS/Q(720,30,6)
244000          EXTENDED/VARO/S(720,30,5)
244100          EXTENDED/VAR1/X(720,30),Y(720,30),Z(720,30)
244200          LOCAL/VAR3/P(120,30),XX(60,4),YY(60,4),ZZ(60,4)
244300  C       LEVEL 2,Q,S,X,Y,Z
244400          CONTROL/COUNT/NC,NC1,DT
244500          GLOBAL/FLSH/DX2,DY2,DZ2
244600  C
244700  C   XI METRICS FORMED FOR A K,L LINE IN J
244800  C
244900  C
245000  C   SYMMETRY
245100  C
245200              K = M
245300              L=LA
245400              J1=J1A
245500              J2=J2A
245600              KL = (L-1)*ND+K
245700              DO 10 J = J1,J2
245800              RJ = Q(KL,6,J)
245900              XK = (X(KL+1,J)-X(KL-1,J))*DY2
246000              YK = (Y(KL+1,J)-Y(KL-1,J))*DY2
246100              ZK = (Z(KL+1,J)-Z(KL-1,J))*DY2
246200              XL = (X(KL+ND,J)-X(KL-ND,J))*DZ2
246300              YL = (Y(KL+ND,J)-Y(KL-ND,J))*DZ2
246400              ZL = (Z(KL+ND,J)-Z(KL-ND,J))*DZ2
246500              XX(J,1) = (YK*ZL-ZK*YL)*RJ
246600              XX(J,2) = (ZK*XL-XK*ZL)*RJ
246700              XX(J,3) = (XK*YL-YK*XL)*RJ
246800              XX(J,4) = -OMEGA*(Z(KL,J)*XX(J,2)-Y(KL,J)*XX(J,3))
246900          10 CONTINUE
247000              RETURN
247100              END
```

Figure 3-22. SAM Extended FORTRAN for Subroutine XXM1

```
1    R 243200       PROCEDURE XXM(M,LA,J1A,J2A)
2    R 243300       GLOBAL/BASE/NMAX,JMAX,KMAX,LMAX,JM,KM,LM,GAMMA,GAMI,SMU,FSMACH
3    R 243500      2,RM,CNBR,PI,INVISC,LAMIN,NP
4    R 243600       GLOBAL/GEO/NB1,NB2,RFRONT,RMAX,XR,XMAX,DRAD,DXC
5    R 243700       GLOBAL/READ/IREAD,IWRIT,NGRI
6    R 243800       GLOBAL/VIS/RE,PR,RMUE,RK
7    R 243900       EXTENDED/VARS/Q(720,30,6)
8    R 244000       EXTENDED/VARO/S(720,30,5)
9    R 244100       EXTENDED/VAR1/X(720,30),Y(720,30),Z(720,30)
10   R 244200       LOCAL/VAR3/P(120,30),XX(60,4),YY(60,4),ZZ(60,4)
11   R 244400       CONTROL/COUNT/NC,NC1,DT
12   R 244500       GLOBAL/FLSH/DX2,DY2,DZ2
```

Figure 3-23. Comparison of Modified FORTRAN and SAM Extended FORTRAN for Subroutine XXM1

3.2.4 Subroutine STEP (Loop DO 30 & DO 40)

The arrays Q and S which have been declared to exist in Extended Memory have the following extents

```
Q(720,30,6)
S(720,30,5)
```

A partitioning in effect of the first extent of 720 into 2 parts occurs at run time with the variable ND. The first index then has an extent ND and the second index has an extent equal to LMAX. This means that if ND*LMAX 720 certain memory locations are not utilized. This causes some degradation in performance for the SAM in all three access modes.

Each of the three types of accesses of the Q & S arrays which are required by the DO 20, DO 30 and DO 40 loops in SUBROUTINE STEP will be discussed. Because of a complex first order linear recurrence the index J in the DO 20 loop must be done serially while the K&L indices are parallel (see example below). Similarly for the DO 30 loop K is the serial index while J&L are the parallel ones. For DO 40 L is the serial index and K&K the parallel ones.

An example of the structure of the program is given below.

```
      DO 20 L=2,LM
      DO 20 K=2,KM
      DO 18 J=1, JMAX
       KL = (L-1)*ND+K
      RR = 1.0/Q(KL,J,6)
        (plus many other statements including a complex first order
recurrence in J)

   18 CONTINUE
   20 CONTINUE
```

This is a Case I access as described in Appendix A. The ISKIP=ND. For ease in handling this generality of splitting the first extent it is assumed that 720/ND is integral with value ND. The number of cycles necessary to access the L's and J's is equal to

No. of Cycles = (LD*30+512-1)/512

For the specific case given in the benchmark where ND is equal to 15 then LD is equal to 48 and the No. of cycles equal to 3.

On cycle 1 one is accessing all L's from 1 to LD and J's from 1 to 10 and for the 11th J one is accessing L's from 1 to 32. This is done for each K from 1 to ND. Figure 3-26 maps this accessing of indices from Extended Memory into the processors.

The last loop, the DO 40 Loop has the L index as the serial index for the recurrence relation and the K&J indices as the parallel ones. The structure is

```
      DO 40  J=2, JM
      DO 40  K-2, KM
      DO 38  L1, LMAX
      LK = (L-1)*ND+K
      RR = 1.0/Q(KL,J,6)
      (plus many other statements including a first order linear
recurrence in L)
   38 CONTINUE
   40 CONTINUE
```

This can be considered to be a Case II or Case V accessing pattern as discussed in Appendix A. Since the accessing of Q & S is identical a "semi smart" compiler can chose which of the two cases it wishes to consider this. I.e., Q(KL,J,6) can really be represented as Q(K,L,J,6) with K varying from 1 to ND, L from 1 to LMAX and with J varying from 1 to 30. Since both J&K are totally parallel and all access to Q&S are in the same sense of K,L,J the "semi smart" compiler can pick which way to do it. In this case because ND is unknown at run time it would pick Case II.

The memory layout is shown in Figure 3-24. The accessing pattern is described in Appendix A as being of Type 3. This means that the SAM will access 512 elements of the Q array at one time for J=1, then 512 for J=2 etc., until J=30. This would mean all K's would be accessed from 1 to ND up to an L value L(last) such that 512 values are accessed.

For example if ND=10 then 52L values would be accessed each for K values 1 to 10 except for L=52 which would only access K=1 & K=2.

On the next complete cycle those remaining K and L values would be accessed up to a maximum of 720. Figure 3-25 shows thus.

As can be seen this could be inefficient if ND*LMAX < 512 and these parameters were set at run time. A more efficient procedure could be worked out which would have the same flexibility, either by recompiling with compile time parameters or else with more efficient coding to permit compaction of the Q array (see Appendix C).

The next loop DO 30 has the K index as the serial index for the recurrence relation. Its structure is

```
     DO 30 J-2,JM
     DO 30 L=2,LM
     DO 28 Kl,KMAX
     KL-(L-1) AND +K
     RR = 1.0/Q(KL,J,6)
     (plus many other statements including a first order
recurrence relation on K)
  28 CONTINUE
  30 CONTINUE
```

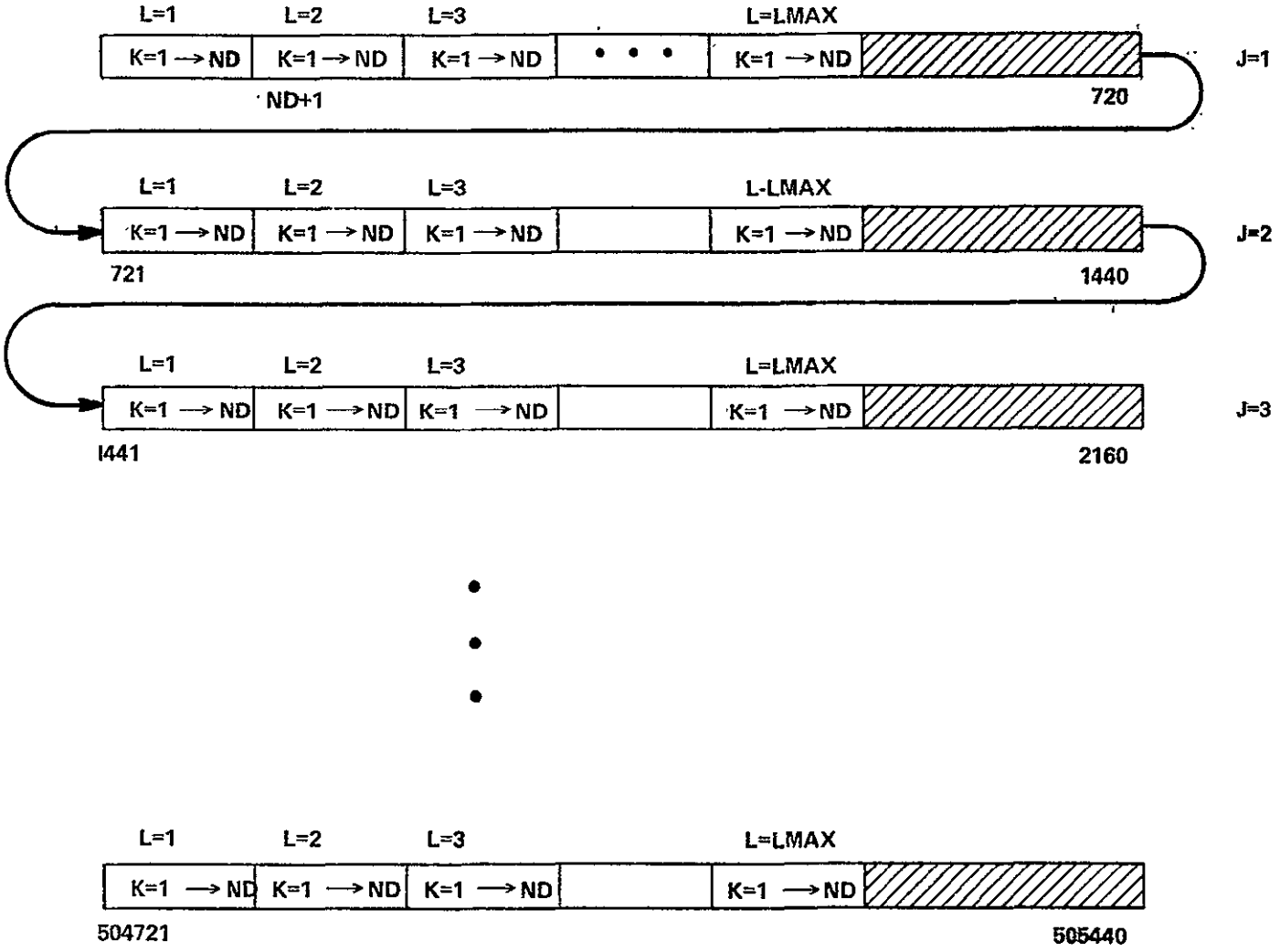


Figure 3-24. Memory Layout for Q Array

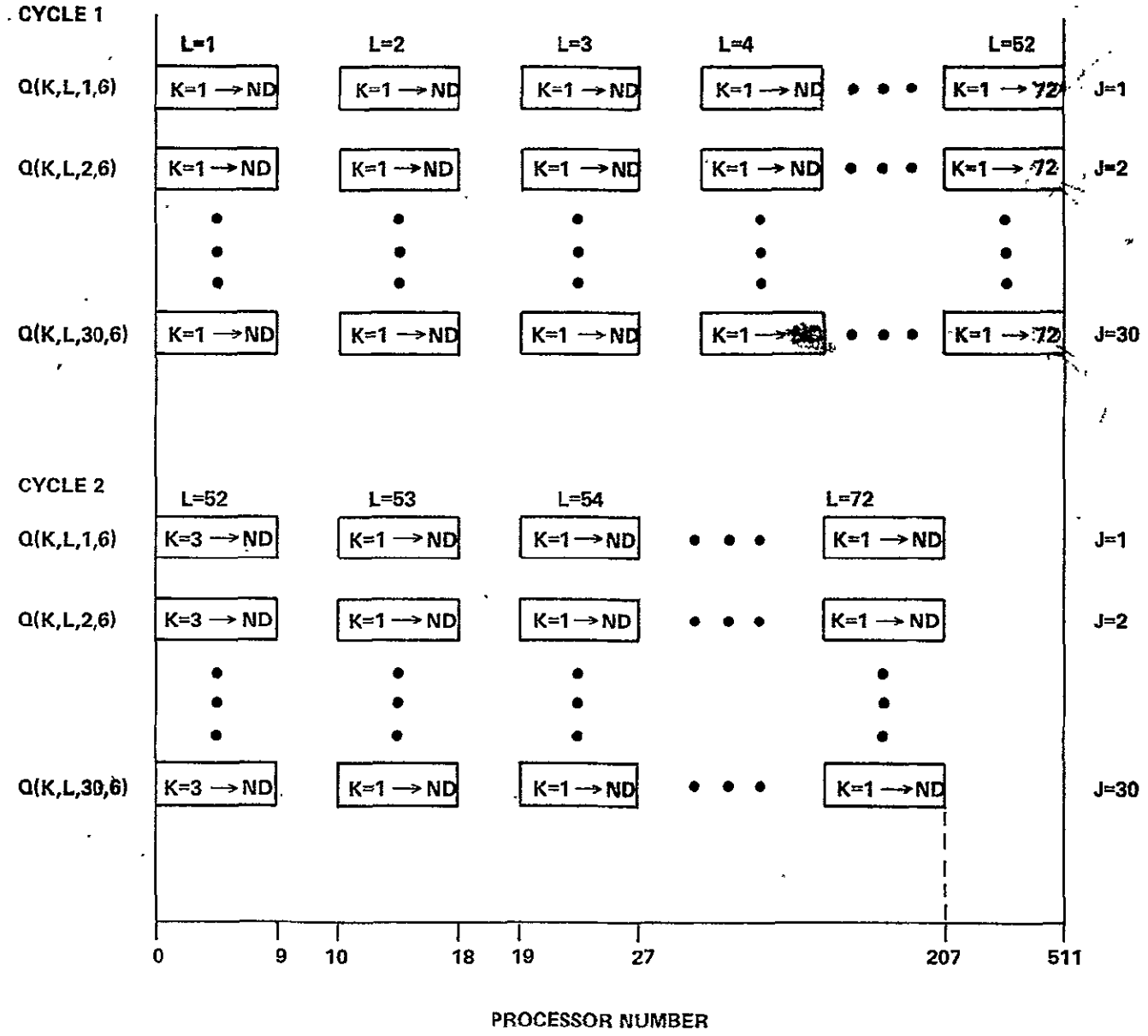


Figure 3-25. Processor Index Values as a Function of Cycle DO 20 Loop Subroutine STEP (ND=10)

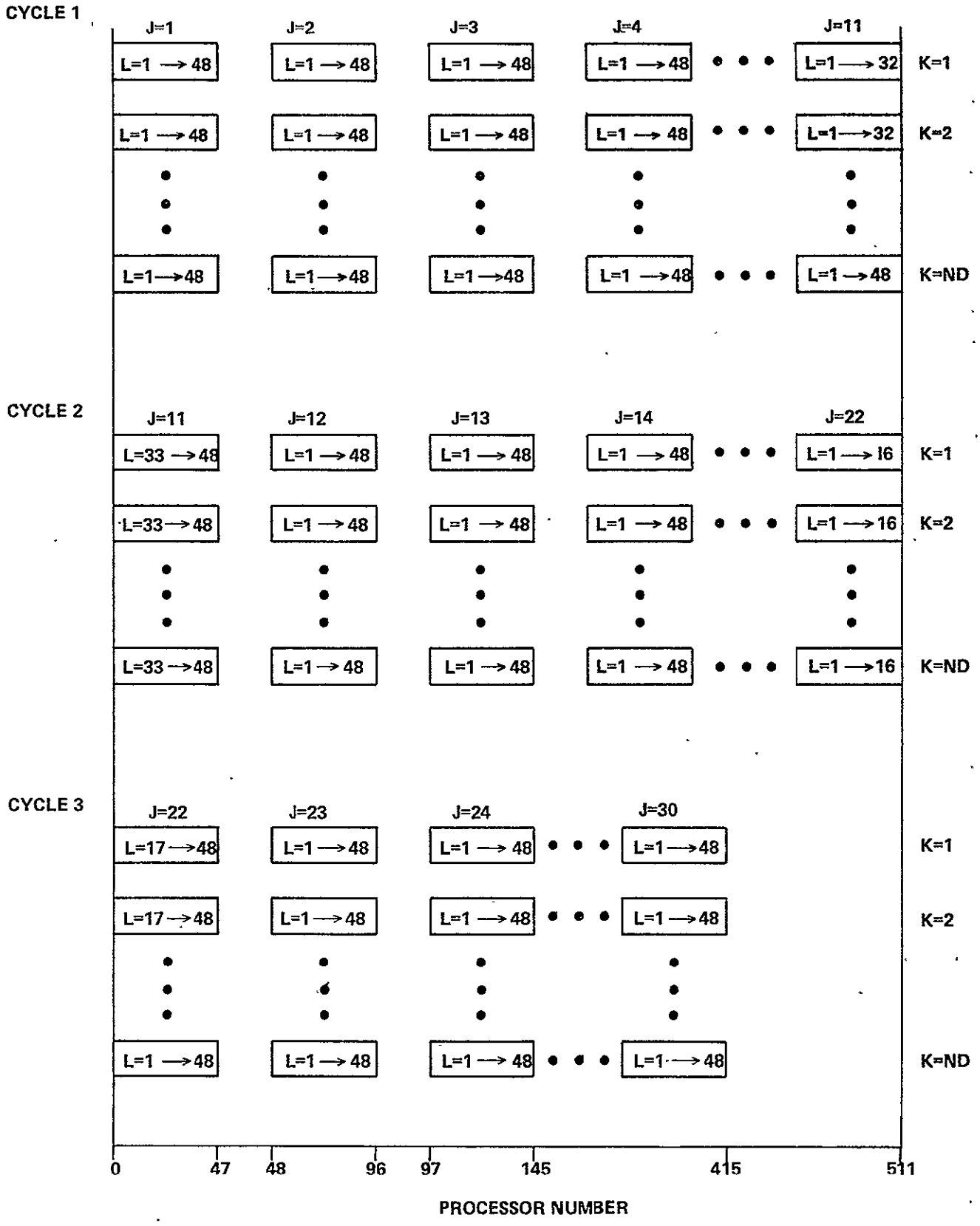


Figure 3-26. Processor Index Values as a Function of Cycle DO30 Loop Subroutine STEP (ND=15)

Figure 3-27 shows how the indices will appear in the various processors. This case requires subiterations of the cycles as on page A-10. The number of cycles is equal to (ND*30+512-1)/512 which for an ND of 10 means only one cycle. ISKIP=720.

3.2.5 Functions and Macros)

Functions on the FMP will include not only the mathematical intrinsics, such as ARCTAN, LN, EXT, and SQRT which are expected of any compiler, but also a family of functions that are brought about because of the parallel nature of the FMP.

Math Intrinsics

Math intrinsics (ARTAN, LN, EXP, SQRT) are well understood. Some will be in-line code, some are subroutine calls. All execute locally to the processor. Since there is nothing new or different for the FMP, we need not digress to discuss them at this point.

Global Intrinsics

A form of intrinsic function seen in a parallel language, for which there is no analog in a serial machine, is that function which operates across the declared parallelsim. A global sum is the sum of all the elements specified by all the instances of the index set of the DOALL. A global maximum is the largest element across the entire DOALL.

To reduce compiler complexity, and to eliminate user programmers' doubts as to whether parallel operation has been achieved as a result of compiler analysis, global intrinsics will be supplied.

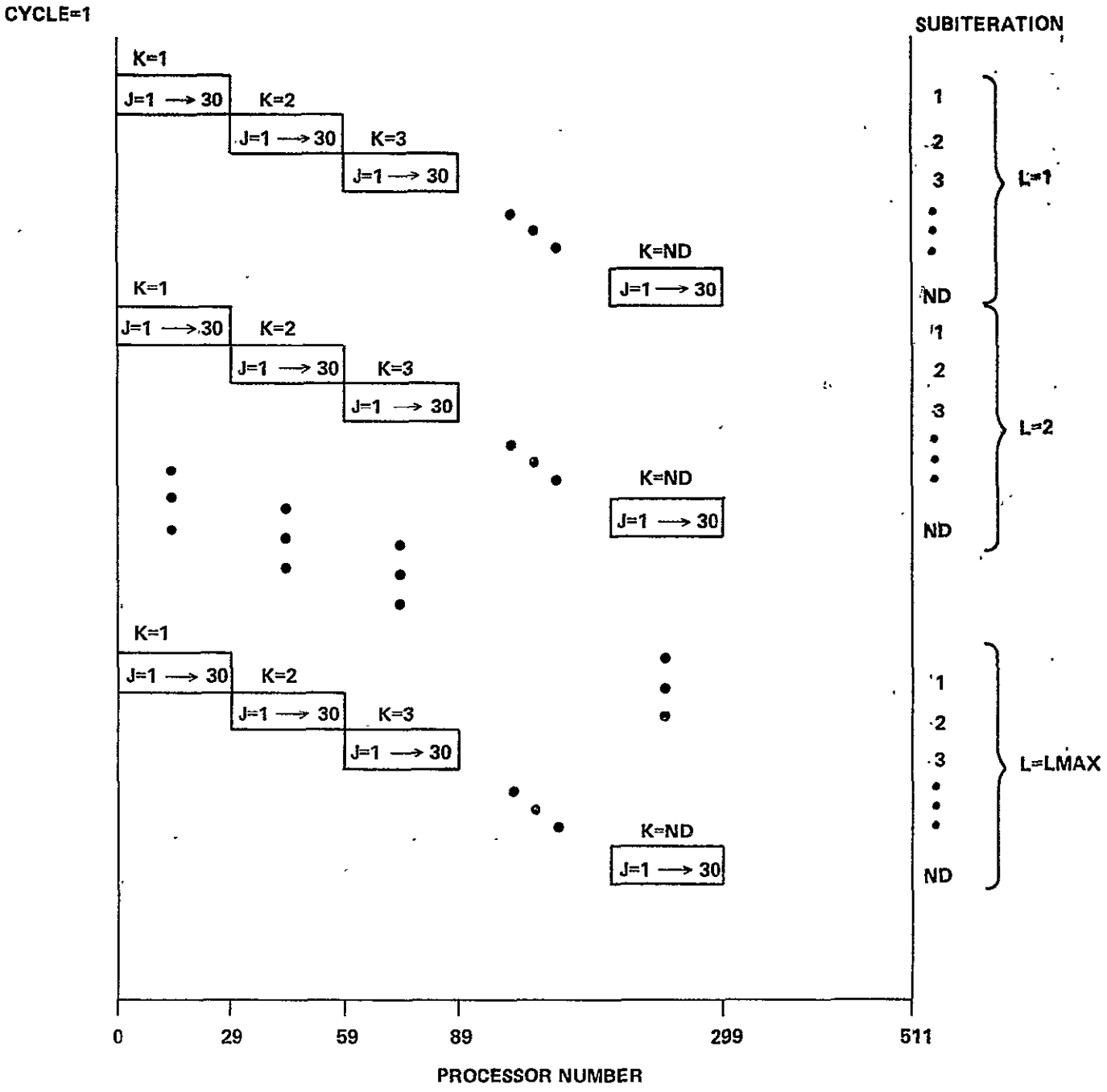


Figure 3-27. Processor Index Values as a Function of Cycle DO 40 Loop Subroutine STEP (ND=10)

To replace the following serial FORTRAN

```
      A = 0.0
      DO 1 J = 1,1000
      A = A + B(J)
   1  CONTINUE
```

the language will allow:


```
     DOALL, J=1,100
     A = GLOBALSUM(B(J))
     ENDDO
```

The global operations will presumably include all of the following. Assume that we are inside a DOALL loop expressed as DOALL, J=JSTART,JEND.

Function	Definition
GOLBALSUM(A(J))	$\sum_{J=JSTART}^{JEND} A(J)$
GLOBALPRODUCT(A(J))	$\prod_{J=JSTART}^{JEND} A(J)$
GLOBALMAX(A(J))	Largest of A(JSTART, A(JSTART+1), ... A(JEND)
GLOBALMIN(A(J))	Smallest of all A(J) JSTART $\leq$ J $\leq$ JEND

Global functions are logarithmic in efficiency, that is, it takes nine steps to produce the 512-way sum across the 512 processors in one cycle. When the result (such as "A"), is a LOCAL variable, it is produced across the entire extent of the DOALL.

An extension of the global operation is the formation of a parallel linear recurrence in nine (= $\log_2 512$) steps as demonstrated by Shyh-Ching Chen in his doctor's thesis at the U. of Ill. In Fortran, consider

```
      DO 1 J=1,1000
      A(J+1) = B(J)*A(J) + C(J)
1     CONTINUE
```

This takes 1000 steps, each with one multiply, and one add. A parallel algorithm exists that produces the same result in 10 steps. The parallel algorithm can easily be implemented on the FMP.

With the inclusion of the parallel linear recurrence as a function in the language, the programmer has two ways of writing his linear recurrences. For example, given the serial FORTRAN

```
      DO 1 J=1,1000
      DO 1 K=1,1000
      A(J,K+1) = A(J,K) * B(J,K) + C(J,K)
1     CONTINUE
```

there are two ways to write it in FMP FORTRAN given that the order of nesting the loops is irrelevant otherwise. Namely:

Method I:

```
      DOALL, J=1,1000
      DO 1 K=1,1000
      A(J,K+1) = A(J,K) *B(J,K) + C(J,K)
   1  CONTINUE
      ENDDO
```

Method II:

```
      DOALL, K=1,1000
      DO 1, J=1,1000
      A(J,K+1) = RECURRENCE( A(J,K) * B(J,K) + C(J,K))
   1  CONTINUE
      ENDDO
```

Method I, which executes the recurrence serially in an inner loop, runs about nine times as fast as method II, which executes each one of the recurrences in parallel across each value of J in turn. That is, method I is 512 times as fast as a serial machine, while method II is 57 times faster than a single serial processor. The RECURRENCE function is included only for those cases where method I is not an available option.

CHAPTER 4

SIMULATION

4.1 SIMULATION GOALS

The simulation effort during this extension of the feasibility study has two distinct goals. The first is the requirement of the statement-of-work for this extension that a simulation of the FMP be prepared, and at least one simulation run. The second, is to get a head start on those simulations needed for phase II, and described in Chapter 6 as the mechanism for settling various trade-offs. The statement of work also calls for the selection of "metrics", that is, selected portions of the benchmark programs to be used as inputs to the simulations to measure the performance of the projected FMP.

Detailed instruction by instruction timing of code execution in CU and EU is necessary to ensure that the required throughput can be achieved. The design of major system components must be specified in sufficient detail to provide structure, logic, and timing parameters for system simulation. This information is in Chapter 2.

Compiler functioning, including FORTRAN extensions for the FMP, are also needed and are found in Chapter 3. Hand compilation methods must be specified. In the case of the current extension, a single metric, subroutine TURBDA, has been selected and hand compiled for this purpose. Further definition of hand compilation is needed for phase II. In particular, how much compiler sophistication

will be achieved in the first version affects hand compilation, and this is still a subject for discussion. At this time it is best to make conservative assumptions, again in order to reduce the element of risk in the simulated system performance predictions.

The design details and design choices outlined above have been made definite though at this time for the first of the detailed simulations which are required to establish confidence in the feasibility and throughput capability of the SAM architecture. Any or all of the details may be changed as a result of further study or the availability of more advanced components. Of course, all such changes would be supported by simulation studies to maintain or increase confidence in the correctness of the system design.

4.2 SELECTION OF METRICS

It is Burroughs understanding that the final selection of metrics will be the Government's. Metric selection is a function of the architecture that is to be measured. For example, in a conventional serial uni-processor, the distinction between "serial" and "parallel" streams of code is irrelevant, and should have no bearing. With parallel processors such as the two designs being proposed for the FMP (NAS2-9456 and NAS2-9457 final reports) the arrangement of data in memory affects the efficiency of parallelism, and metrics should be selected such that all "directions" of access of that data are represented. What is important is that the metrics selected be "representative", both with

respect to the operations being performed by the target architecture, and the codes that will be run on the FMP. Some "representative" of every kind of code that the FMP will run is wanted, but the results should be weighted according to the expected frequency of each "kind." "Kind" refers to the sort of interaction with the architecture that is represented, whether parallelism is two dimensional or one-dimensional, the direction of accessing, presence or absence of branches in inner loops, and so on; all the things that may have an affect on the way the selected architectures behaves.

The metric that has been selected as the one that shall be used in the single simulation that will be run during the extension of the contract is SUBROUTINE TURBDA. Like most of both the implicit and explicit codes, it exhibits a great deal of parallelism, but with some operations conditional on subscript, so that different things are being done at different subscripts. It thus tests the architecture's ability to do different things at different grid points. It includes fetches from, and stores to, the program's data base (in extended memory), exercising the data transfer paths from the program data base to the processing resource proper. It contains sufficient arithmetic manipulation to exercise that aspect of the FMP (although probably less than a "typical" subroutine). It contains significant amounts of index computation both on loop controls and on subscripts. For the FMP design of Reference 1, it exercises the synchronization, which is an essential feature of that design.

4.3 SIMULATION MODELS

The NASF system simulation modeling will be done at three levels of detail, with results from a detailed model being used to determine parameter values for the next higher level model.

The most detailed level of modeling is the instruction timing model for CU and processors. For example, the model for the processor has as resources the PDM, PPM, instruction registers and decoding, multipliers, adders, data and index registers, etc., corresponding to the detailed processor design. A metric for this model is a sample code sequence generated by hand compilation of a FORTRAN section typical of the Navier-Stokes codes. Each instruction is modeled by a sequence of tasks, each requiring one or more of the resources, and executing for the specified number of clocks. Instruction fetch and decode is such a task sequence and the extent of overlap with instruction execution is modeled. Similarly, the extent to which instructions can overlap is modeled by the use of queueing for resources, or by logic tests, in exact correspondence with the processor design. The output reports from running this model can be used to determine parameters for the next level model. An important performance factor to be determined is the extent to which the address calculations for EM accesses can be interlaced with, and overlapped by the floating point calculations. The next level of simulation will be the flow model processor, including the CU, processor, EM, and DBM. The interactions to be measured are the CU and processor code execution times (previously determined), and

data transfers between EM and DBM. The metric will be a sequence of code executions and data transfers approximating the main body of computation in a Navier-Stokes code. The results will show the throughput performance of the FMP, together with the utilizations of EM and CU, which interface with DBM and the rest of the system.

When we wrote the simulation model, we found that the instruction level model needed to include the interaction between CU and EU, combining the first and second levels. The lowest level simulation model therefore is detailed to the instruction level, but includes CU, a number of processors, and access and data transmission timing of the Extended Memory and Transposition Network. Simulation of a number of selected code sections on this model will provide the parameters required to model the execution of complete jobs and sequences of jobs through the Facility.

The overall system model will include the host, File Memory, Data Base Memory and their interfaces with each other and CU and EM. The metrics will be presumed scenarios of user requests for NSS jobs. The sequence of scheduling, initialization, NSS operation, and output will be modeled. Important functions to be modeled are data base and program transfers from File Memory to DBM to EM, CUM, and PDM, allocation of DBM space, the sequence of FMP operations, including data and program input, computation, snapshot and data outputs, and changeover to the next job. Only the FMP

scheduling and control load on the host will be modeled; the amount of host capacity available for other necessary work can be measured, or the host can be loaded to any desired level by undefined "background" jobs and the effect on NASF throughput measured.

The overall simulation effort will have two functions: first to support the validity of the SAM architecture by modeling all essential system functions and interfaces in sufficient detail and demonstrating proper function of the model, and second to show the throughput capability of the system for aerodynamic simulation jobs by tracing the throughput step-by-step from the instruction level to the user interface.

Simulations will be written in Burroughs Operational System Simulator (BOSS) a discrete-events simulator whose input language is the flow-graph of the process being simulated. The instruction level simulation of Section 4.5 is written in BOSS, the second and third level simulations of Phase II will be written in BOSS. In Phase II, the instruction-level simulator may be rewritten in ALGOL, since substantial improvement in simulation execution time is expected.

4.4 BOSS SIMULATOR

The BOSS simulator was used for the simulations because of the relative ease of modeling with BOSS and the short time available. Special timing simulator programs for EU and CU code execution probably could have been completed in three months. Simulations at different levels of detail will be used to get performance predictions ranging from the EU instruction execution to the user interface level.

A discrete events simulator, such as BOSS, models the activity of a system as a definite sequence of states. The model changes state only at discrete points, called events, which occur at definite instants of time. Every event can be predicted at the occurrence of some prior event, and the new state of the system model resulting from each event can be completely determined from that event and the prior state. In practice the event prediction and state change calculations are often probabalistic, because the real system is too complex to be modeled in full detail. The state variables of the model are mostly binary logic variables such as busy/not busy or happened/not happened, and processing of an event involves the accessing of state tables and evaluation of binary decision functions. Arithmetic operations rarely occur except in the evaluation of continuous probability functions where they are used in the binary decisions or in predicting the times of future events.

The BOSS simulator program runs on a B 6700 or B 7700 Burroughs computer. It is a general purpose discrete events simulator, with emphasis on ease of modeling and efficiency in execution, in exchange for some restrictions on the size and generality of models. BOSS has been used by the Federal and Special Systems Group at Paoli mainly for simulating the hardware and software functions of data processing systems, and improvements and enhancements over several years have made it especially useful for this purpose.

In a BOSS simulation the element of model activity is a TASK. A task is characterized by its requirement for resources and by the algorithm specified for predicting its execution time. A task is initiated upon completion of its predecessor requirement, which is usually a logical combination (AND or OR) of one or more prior task endings. The task may wait in queue until the required resources are available; the selected resource units are then made busy for the execution time. At the task ending event, resources are released, queues are served, and the predecessor requirements of any successor tasks are updated. Several kinds of test-and-branch constructs are available to cause conditional selection of one out of two or more successor tasks.

The direct interaction of tasks is restricted to structures of tasks grouped together and called PROCESSES. When a process is initiated, one or more "starting tasks" within it are initiated without predecessors, and the activity within it passes from tasks to task until such time as there is no further task activity, when that active version of the process ends. Except for competition for resources, and certain special constructs, there is no interaction between the active tasks in separate active processes.

The static structure of a BOSS model is described by the structures of the tasks and their interactions within processes and by the numbers and kinds of resources available. The dynamic state of activity is described by the states of activity of processes and tasks. Every task is a member of some process, and there is no activity in the system model until some process is initiated.

Initiation of processes at specified times corresponds to external loads causing activity in the system. Processes can also be initiated as subroutines, or by task endings in other processes. Many processes can be active concurrently, including multiple but distinct and independent versions of the same process. Similarly, within a process, many tasks may be active in parallel, including multiple independent versions of the same task. Thus, it is easy to model a highly parallel system with many concurrent activities, including cases where many of the parallel activities are very similar in structure.

The basic BOSS structure described above is sometimes inadequate or inconvenient for modeling some parts of the system. Therefore, there is available a superposed structure of local and global variables upon which arithmetic operations can be performed at task endings. These variables can be addressed directly or indirectly, and their values can be used to control branching at task endings or to modify the resource requirement or execution time of specified tasks. This extension permits a certain amount of programming of capabilities not available in the basic BOSS structure. In this way, for example, the activity in one process can be influenced by actions occurring in another process.

Figure 4-1 shows graphically the process of implementing and debugging simulations in BOSS, showing the various steps that the simulation programmer and the BOSS simulator go through in achieving the final result.

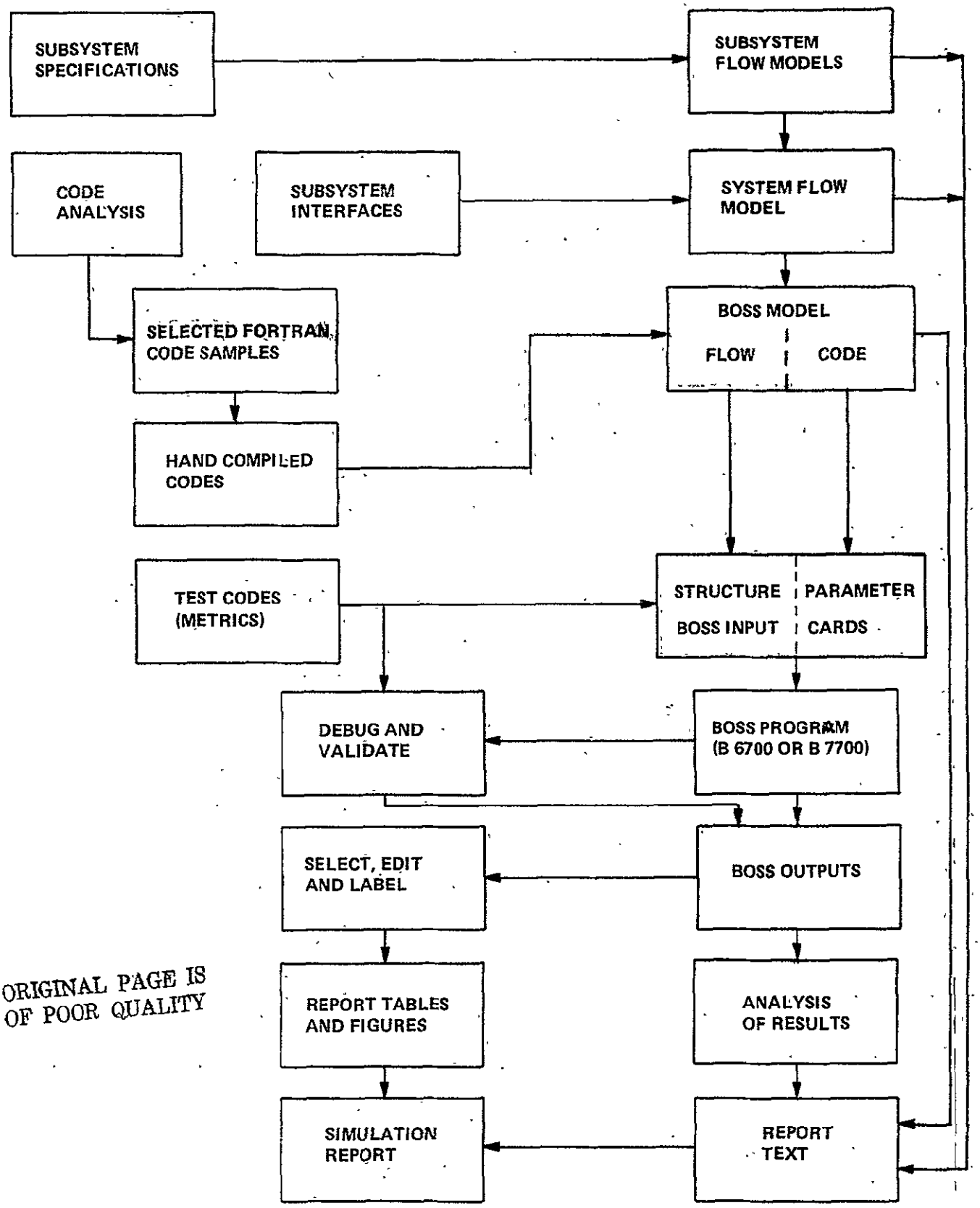


Figure 4-1. Flow of Simulation

4.5 SIMULATION MODEL FOR THE CURRENT STUDY

The overall structure of the model is shown in Figure 4-2. The Control Unit and Processor models are driven by code files prepared by hand compilation of a selected FORTRAN code segment. All the operators of CU and EU are modeled in detail so that any code may be simulated. Additional operators may be easily added if needed. Conditional branching cannot be modeled in complete detail since the model is a timing model, and does not simulate the processing of data. Such branches are therefore modeled by specifying the number of times one path is taken for each time the other is taken. The count can be specified probalistically. For most branches this will do well enough. The cases where branching depends on the Processor Number, will be handled by a later extension.

The Control Unit model includes its processor, a single memory (CUM), and seven of the control functions interacting with the processor EU's, as shown. Any desired number of processors can be modeled, but the number actually used will be small (4 to 10) to avoid excessive machine time to run the simulations. Details of instruction overlap in the CU are not modeled; instruction execution times are not allowed to overlap, but CUM data fetches or stores can overlap this execution time of prior or following instructions. A data fetch of one instruction must come after a data store (if any) of the preceding instruction. In case of contention for CUM by program fetches, the data accesses have priority, but do not abort program fetches already in progress. The program look-ahead stack has a capacity of four code segments, which is two memory words for opcode formats using 24-bit segments.

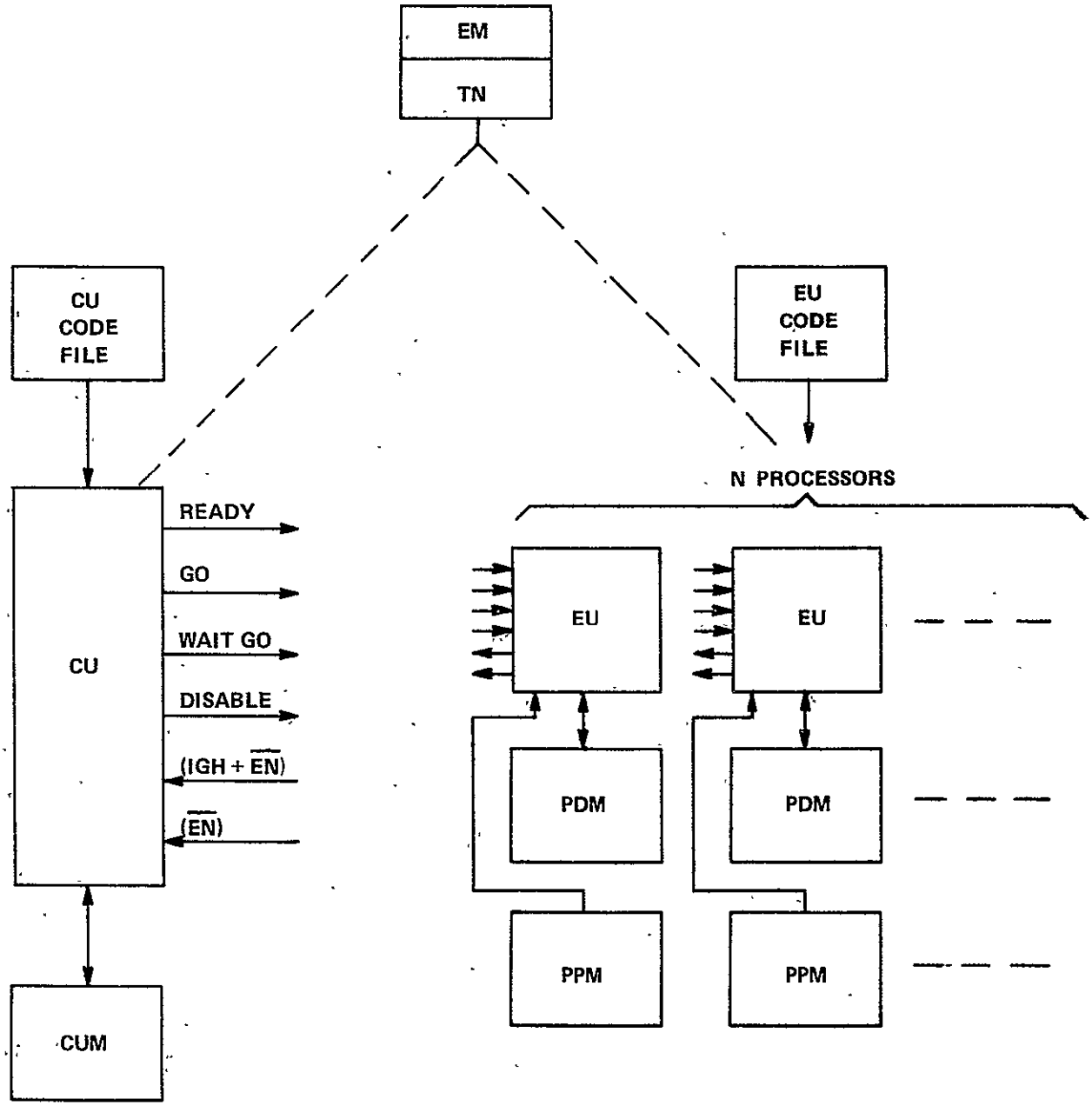


Figure 4-2. Structure of Model

Each Processor consists of an Execution Unit (EU) and separate program and data memories (PPM and PDM). The EU is modeled in some detail in order to properly simulate instruction overlap, as shown in Figure 4-3. The operation is as follows:

4.5.1 Program Fetch. The Program Counter (PCR) addresses the next instruction, which is available at PPM three clocks after the address is available. As soon as a full word of program stack is empty, the next code word is read to the stack from PPM, and PCR is incremented. When a branch occurs, the program stack is emptied and the new code word is available three clocks after the new PCR is set.

4.5.2 Scoreboard. Each instruction records in the scoreboard the times at which it will release each resource that it will use. The next instruction must wait in stack until all resources that is will need will be available when needed. The Scoreboard and Decoding are modeled logically, but not as resources for which there could be queueing.

4.5.3 Holding Registers. If any resource is required at a time later than instruction start, that instruction must wait in the corresponding Holding Register. If that Holding Register is tied up by the previous instruction, then the current instruction must wait, even though it could otherwise start.

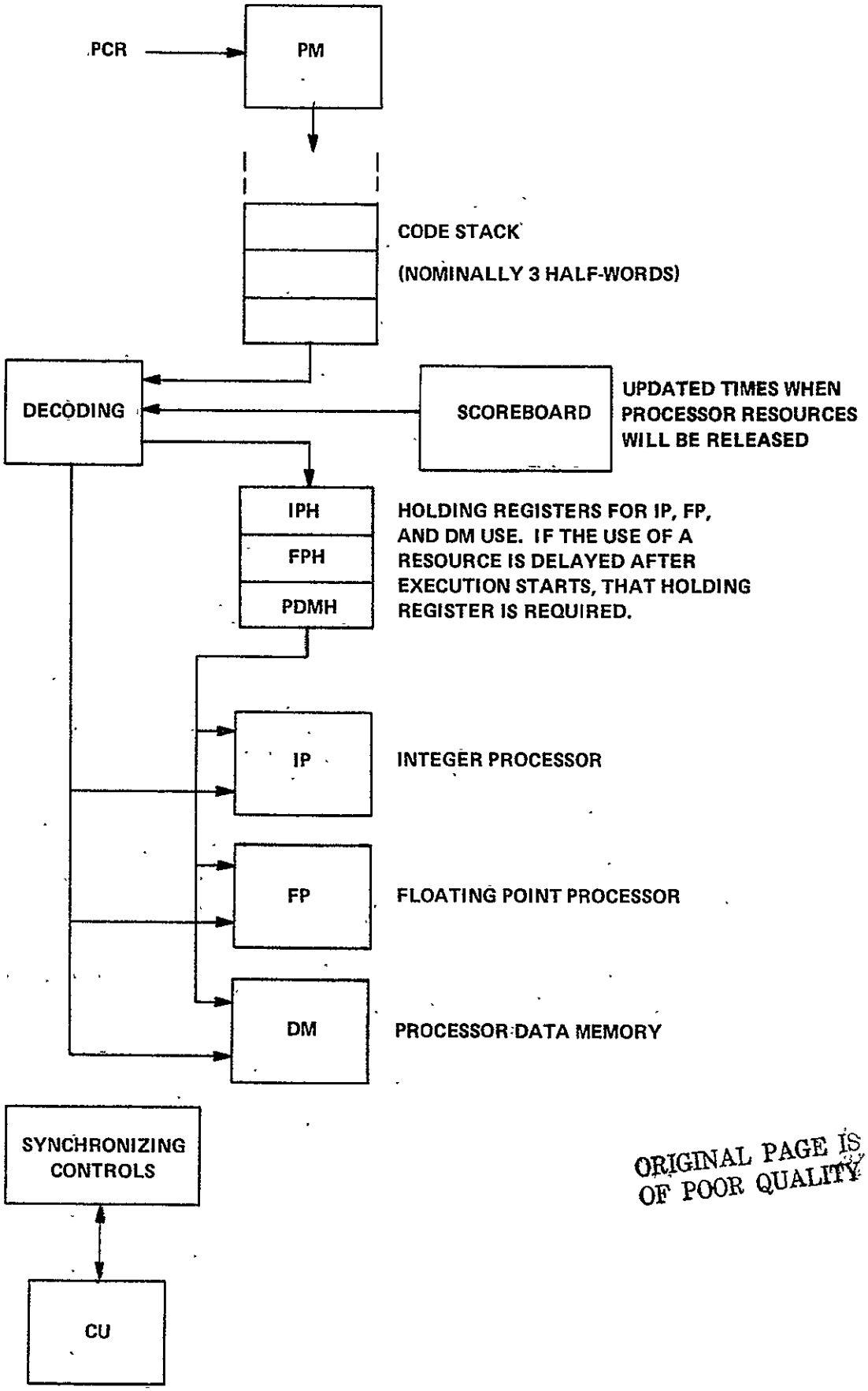


Figure 4-3. Execution Unit Model

4.5.4 Integer Processing, Floating Point Processing, PDM (IP, FP, DM). These are modeled as resources, although the Scoreboard should assure that there will be no queueing for them. The utilization of these resources will give information about the efficiency of overlap and the fraction of elapsed time that the FP is in use.

4.5.5 Synchronizing Controls. The timing of synchronizing controls is assumed to take 3 clocks for a round trip from CU to EU and back to CU. This is modeled as no delay from CU to EU since the control signal arrives at the same time as the corresponding clock pulse from the central clock. The 3 clocks delay is then all in the return path from EU to CU. The actions of the Synchronizing Controls are as follows:

4.5.5.1 READY. The CU raises the ready at the proper time in synchronized instructions where the EU's must wait for CU action before proceeding (LOADEM, STOREM). Any EU which reaches such an instruction before CU waits for the READY level. CU will wait for (IGH and $\overline{EN}$) and then turn off the READY level.

4.5.5.2 (IGH + $\overline{EN}$). This is level equivalent to a logic function generated as follows: When an enabled (EN) EU comes to the proper point in a synchronized instruction it raises the output line corresponding to I Got Here (IGH). This same level is raised all the time an EU is disabled ($\overline{EN}$), hence (IGH + $\overline{EN}$). IGH is turned off by GO from CU. The (IGH + $\overline{EN}$) lines for all EU's are ANDed at the CU to procude its (IGH + $\overline{EN}$) input. In the model this logic function is performed by maintaining separate counts of the number of EU's enabled (#EN) and in the I Got Here state (#IGH). (IGH + $\overline{EN}$) is true when #EN = #IGH.

4.5.5.3 $\overline{EN}$. $\overline{EN}$ is ture when #EN=0 (no EU's are enabled).

4.5.5.4 GO. When (IGH + $\overline{EN}$) becomes true at the CU, it raises the GO level for one clock. All enabled CU's, on receipt of this signal, turn off the IGH level and continue the instruction in which they were waiting.

4.5.5.5 Wait GO. When CU sends this signal (one clock), all enabled EU's enter the IGH state (waiting for GO) in place of the next instruction start. The current instruction is or will be finished.

4.5.5.6 Disable. When CU sends this signal (one clock), all enabled EU's enter the disabled ($\overline{EN}$) state in place of the next instruction start. The current instruction is or will be finished.

4.5.6 Extended Memory and Transposition Network. The EM and TN are not modeled as resources that may be busy; thus it is assumed that during execution of CU-EU code, the EM is never in use for DBM transfers. The EM access time and data transmission time through TN are properly modeled in the execution time of the LOADEM and STOREM instructions.

4.5.7 Code Simulated. The hand-compiled TURBDA assembly codes are given in Table 4-1 and 4-2, together with an assembly coded SQRT, which is a simplified version omitting the tests and branches for negative argument and for negative exponent.

4.5.7.1 Processor Code. The large amount of integer computation at the beginning of each pass through the TURBDA loop would give a low utilization of the Floating Point unit, were not for the large block of FP calculation in

Table 4-1. TURBDA Processor Code Simulated by Model

	(ICALL not simulated)	
	FL	
	FDIVM	
	IL	
	IL	
L3	ITIX, L4 (Drop through 2 times, then exit to L4)	
	ISHL	
	IPNO	
	IADD	
	IDIVL	
	ISTORE	
	IMULL	
	IFETCH	
	IL	
L14	ITIX, L1 (Drop through 20 times, then exit to L1)	
	IADDL	
	ID521	
	LOADEM	
	IADDL	
	ID521	
	IL	
	IFETCH	
	IEQL (No Branch)	
	IL	
L100	LOADEM	
	IADDL	
	ID521	
	IL	
	IFETCH	
	IEQL (No Branch)	
	IL	
L200	LOADEM	
	IFETCH	
	IGT (No Branch)	
	IFETCH	
	IFETCH	
	IGT (No Branch)	
	IEQL (No Branch)	
	IEQL, L20	
	(Jump to L20)	
	FFETCH	Never Executed
	FABS	
	FMUL	
	FSTORE	
	IJUMP, L40	
L20	FFETCH	
	FFETCH	
	FADD	
	FABS	
	FMULL	
	FMUL	
	FSTORE	
	JUMP, L40	
	(Jump to L40)	
L30	FFETCH	Never Executed
	FFETCH	
	FADD	
	FABS	
	FMULL	
	FMUL	
	FSTORE	
L40	FL	
	FFETCH	
	FMUL	
	ENTER SQRT	
	FMUL	
	FL	
	FADD	
	FDIV	
	IADDM	
	ID521	
	STOREM	
	JUMP L14	
	(Jump L14)	
L1	JUMP L3	
	(Jump to L3)	
L4	STOP	
SQRT	IUPK3	
	IADDL	
	IANDL	
	ISHL	
	ISUB	
	IADD1	
	IANDL	
	ISUB	
	ISHL	
	IADD	
	IADDL	
	IPAK3	
	FADD	
	FNEG	
	FL	
	FMUL	
	FMAD	
	FMUL	
	FMUL	
	FMAD	
	FMUL	
	FMUL	
	FMAD	
	FMUL	
	FMUL	
	FNEG	
	FMUL	
	IRETURN	

the SQRT routine which is called once per loop. ICALL and IRETURN are both estimated at 23 clocks, which may be pessimistie and considerably reduces the FP utilization of SQRT. In an inner loop such as this, SWRT should probably be written in-line, since it will occupy no more than 20-30 words, and about 50 clocks are saved.

Note that the outer loop, starting at L3, is executed twice, and each time the inner loop, starting at L14, is executed 20 times. This is a sufficiently large sample of code execution to give valid statistics. Within the inner loop, in the actual code, each EU will execute one of three branches, depending on the index states. In the simulation, only the branch starting at L20 (the longest of the three) is executed. The other two are never executed, as indicated.

In the actual code, two of the LOADEM's are conditional (LOADEMC). However, only the EM address and EM data input are conditional, the timing being the same, so the simulator makes no distinction.

4. 5. 7. 2 Control Unit Code. The Control Unit code of Table 4-2 begins with LOOP, because the model starts with all EU's waiting for GO. When (IG + $\overline{EN}$) is true, LOOP causes both CU and EU's to branch to specified addresses by the LOOP instruction, and this is a convenient way to get the simulator to jump to the desired addresses in the simulated code files.

Table 4-2. TURBDA Control Unit Code Simulated

```
        LOOP
        CL
        CL
L3      CTIX, L4 (Drop through 2 times, then Branch L4)
        CSHFN
        CMULL
        CFETCH
        CL
L14     CTIX, L1 (Drop through 20 times, then Branch L1)
        CADDL
        CADD
        CMD521
        CL
        LOADEM
        CADDL
        CADD
        CMD521
        LOADEM
        CADDL
        CADD
        CMD521
        LOADEM
        CADDL
        CADD
        CMD521
        STOREM
        CJUMP, L14 (Jump to L14)
L1      CJUMP, L3 (Jump to L3)
L4      CRETURN
        END SIMULATION
```

The only synchronization instructions in this code sample (aside from LOOP) are the three LOADEM's and the STOREM.

The CU and its synchrnoizing action are simulated in some detail to determine two things:

(1) How much do processors wait at sync points for other processors to catch up?

(2) Do processors ever wait at sync points for CU to catch up, and if so, how much?

4.6 SIMULATION RESULTS

The simulation runs were made with a model having the Control Unit and four processors. The code driving the model was the TURBDA code shown in Tables 4-1 and 4-2, except that the outer loop was reduced to one iteration and the inner loop to 10, in order to reduce machine time for these first trial runs. Under these conditions the simulation indicates that the abbreviated TURBDA runs 4600 clocks on 184 microseconds assuming a 25-megahertz clock. The full size TURBDA with two iterations in the outer loop and 31 in the inner loop would run about six times as long, or 1100 microseconds (27,600 clocks). The parallelism is 31x31 = 961, compared with 1024 possible in two iterations; so, the efficiency of array use is 93.8 percent in this case.

In the simulated TURBDA run, each processor performs 281 floating point operations lasting a total of 2407 clocks, for an average of 8.6 clocks per FLOP. The elapsed time of 4600 clocks yields an effective throughput of 1.53 MFLOPS

per processor. The array throughput would then be 782 MFLOPS, or 733 at 93.8 percent array efficiency for the 31x31x31 problem. As expected for TURBDA, these rates are considerably lower than 1000 MFLOPS. This reduced throughput has three causes:

(1) There are 40 EM accesses with the 281 floating point ops, or a ratio of only 7 to 1. The EM accesses themselves do not cause appreciable delay, but the integer operations required to calculate the EM addresses do cause delay.

(2) The floating point operations of TURBDA contain more than the normal proportion of multiplies and divides, raising the average duration from the nominal 7.3 clocks to 8.6 clocks per floating point operation.

(3) The function SQRT was simulated as a subroutine, with entry and return operators. It is likely that the compiler will put simple functions like SQRT in-line. If so, the total time would be only nine tenths that shown, for an 11 percent increase in measured throughput.

Some other conclusions of interest are:

(1) Control Unit processing causes essentially no delay (less than 0.5 percent of the total time)

(2) Extended memory accesses occupy 11.5 percent of the time, including all synchronizing delays.

(3) Program fetches cause little or no delay. The model does not measure such delays exactly, and should be modified to do so. Program memory is in use 42 percent.

(4) The utilization of the integer unit is 47 percent, data memory 10 percent and floating point unit 58 percent, for a total of 115 percent, indicating the approximate degree of overlap.

(5) The inner loop takes 450 clocks, of which 197 are in the SQRT routine. Two thirds of the floating point operations are in the SQRT routine.

Figure 4-4 is an example of one of the output tables of one of the simulation runs. The unit types represent various system resources as indicated by the row headings typed in on the left. In some cases the resource is used for internal control purposes in the model and does not represent a real system component, so is unlabelled. Some of the resources represent logic levels and signals such as $\overline{\text{READY}}$, $\overline{\text{GO}}$, IGH+$\overline{\text{EN}}$, EN=0. A processor or CU waiting for such a level or signal is modeled as queueing for the resource, which is created to represent the presence of the level or signal.

UNIT UTILIZATION STATISTICS

UNIT TYPE	UNIT ID NUMB	TOTAL TIMES USED	.....PERCENT OF ACTIVE TIME..... ...IN-USE... DELTA	IN-USE TOTAL	.DEPENDENT.. DELTA	DEPENDENT TOTAL	FREE TOTAL
CUM 24	3	173	11.27	11.27	0.00	0.00	88.73
CUPROC 25	4	233	98.89	98.89	0.00	0.00	1.11
CPSTAK 20	5	171	11.86	11.86	0.00	0.00	88.14
" 20	6	173	11.40	11.40	0.00	0.00	88.60
" 20	7	171	11.92	11.92	0.00	0.00	88.08
19	8	170	11.62	11.62	0.00	0.00	88.38
--- 23*	9	1	0.07	0.07	0.00	0.00	99.93
--- 28	10	186	0.91	0.91	0.00	0.00	99.09
READY 15	11	240	0.87	0.87	0.00	0.00	99.13
GO 17	12	246	0.00	0.00	0.00	0.00	100.00
(IGH+EN) 21	13	82	0.00	0.00	0.00	0.00	100.00
#EN=0 26	14	1	0.00	0.00	0.00	0.00	100.00
IU 3	15	506	47.01	47.01	0.00	0.00	52.99
FPU 4	16	364	57.74	57.74	0.00	0.00	42.26
PDM 5	17	154	9.97	9.97	0.00	0.00	90.03
PPM 7	18	1348	42.37	42.37	0.00	0.00	57.63
HOLD 8	19	1	0.00	0.00	0.00	0.00	100.00
REGIS- 9	20	53	9.60	9.60	0.00	0.00	90.40
TERS 10	21	52	10.18	10.18	0.00	0.00	89.82
QUEUED 13	22	558	77.55	77.55	0.00	0.00	22.45
PPSTAK 12	23	862	35.50	35.50	0.00	0.00	64.50
" 12	24	862	35.53	35.53	0.00	0.00	64.47
" 12	25	860	40.69	40.69	0.00	0.00	59.31

Figure 4-4. Sample of Simulation Output

CHAPTER FIVE
RELIABILITY

5.1 INTRODUCTION

This chapter presents two major aspects of the NASF reliability and trustworthiness; (1) an availability prediction of the FMP and (2) further development of the error detection and correction techniques to the various FMP elements. These topics are covered in sections 2 and 3 of this chapter, respectively.

The system availability design goal for the B7800 host system and the Flow Model Processor (FMP) is 90 percent or better. Also, it is desired that the probability of success for completing runs of ten minutes and one hour be equal to or greater than 98 percent and 90 percent, respectively, The following is the conventional formula for computing availability

$$A = \frac{MUT}{MUT + MDT}.$$

where,

A = Availability
MUT = Mean Up Time
MDT = Mean Down Time.

Up time is the duration during which the system is continuously up. Down time is the interval between up times. It can be seen that a system MUT = 9 hours or longer combined with a system MDT = 1 hour or less satisfies the availability goal. These values also satisfy the desired reliability, or probability of success, as evidenced by the following formula

$$R(t) = e^{-t/SMUT}$$

where,

R(t) = The probability of successfully completing a run as a function of t

t = Duration of the run (hours)

SMUT = System Mean Up Time (hours)

5.2 AVAILABILITY PREDICTION

The following methods were employed in preparing the FMP availability predictions discussed below.

- Standard component part failure rates were predicted using the reliability stress analysis prediction method of MIL-HDBK-217B.

- Potential improvements in reliability through the use of Single Bit Error Detection and Correction and Double Bit Error Detection (SECDED) in the FMP memories, fanout tree, and transposition network were analyzed using a mathematical model developed specifically for the proposed design of these elements.

- System Reliability, Availability, and Maintainability (RAM) characteristics were analyzed using Program DESIGN, which was developed by the Burroughs Corporation to aid in designing fault-tolerant computer systems.

MIL-HDBK-217B is used extensively throughout the electronics industry to predict the failure rates of electronic component parts. Since the prediction methods of MIL-HDBK-217B are quite detailed and documentation describing these methods is readily available, only the general aspects of component part failure rate predictions are discussed in this report.

Appendix B contains a description of the SECDED mathematical model, including the underlying assumptions associated with the development of this technique. A similar description of the mathematical model employed in Program DESIGN is in preparation.

5.2.1 OVERVIEW

The proposed Flow Model Processor (FMP) design will be implemented using state-of-the-art technology of today and currently proposed state-of-the-art technology for the time frame during which manufacturing of the FMP will be initiated. Obviously, accurate reliability projections for some of the LSI component parts required to implement the proposed machine are difficult at this point in time. Likewise, projections with respect to gains in reliability through the use of techniques such as Single Bit Error Detection and Correction and Double Bit Error Detection (SECDED) can only be hypothesized based on assumed failure modes until the design is completed, built, and tested.

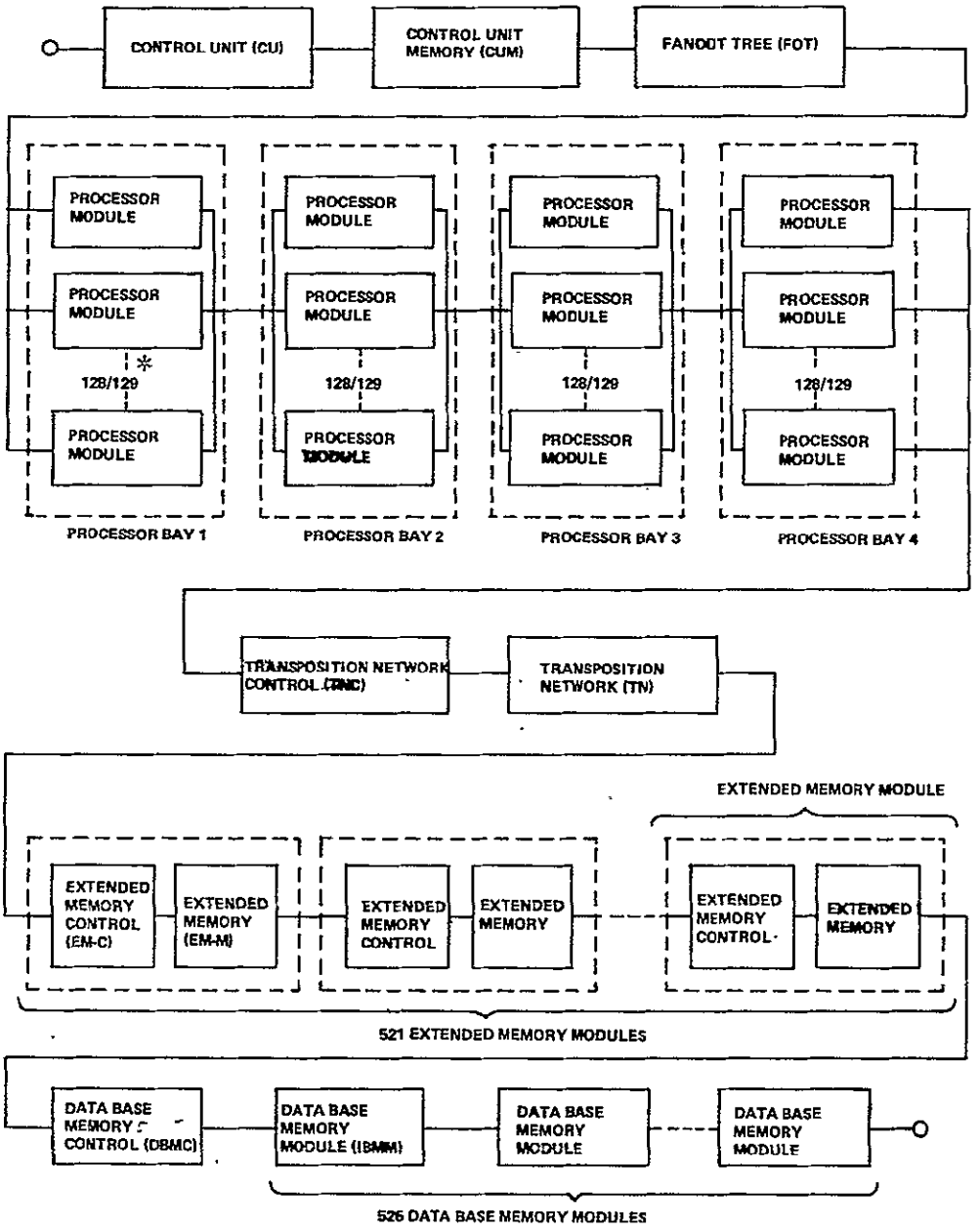
Recognizing that the above and additional considerations must be seriously addressed to ensure meeting the specified system availability requirements of 90 percent, an analysis has been conducted to bound the potential availability of the current FMP design. Both optimistic and conservative points of view have been considered for those conditions which can not be accurately projected at this point in time. In addition, sensitivity analyses have been conducted within the upper and lower projected availability bounds to determine where design attention must be concentrated in order to achieve the stated availability requirement and reap the greatest reliability and availability gains for the effort expended.

The results of this preliminary availability analysis serves two purposes. First, the analysis shows specific failure, recovery, and repair time reliability and maintainability estimates at the subsystem, module, and component part levels that are consistent with overall system availability of 90 percent and MTBF of 9 hours or better. Second, the analysis numerically bounds achievable Mean-Up-Time (MUT), Mean-Down-Time (MDT) and Availability estimates within the broad range of reasonably optimistic and pessimistic assumptions.

The following paragraph summarizes the results of this preliminary availability and the rationale for the assumptions made. As the FMP design progresses, the availability analysis will be iterated to further refine specific reliability and maintainability estimates to narrow the bounds of uncertainty associated with these preliminary projections.

5.2.2 Summary of Results

The first step in this analysis was to develop an overall Availability block diagram of the FMP (Figure 5-1). The estimated parts counts for all major elements, considering the types of component parts currently envisioned, were then prepared. For standard component parts, failure rates were predicted using the reliability stress analysis prediction method of MIL-HDBK-217B. Consideration was then given to the failure rates of large memory packages (16K, 64K, 256K) of the future. It was hypothesized that the best that could be expected in terms of reliability is achieving failure rates equivalent to those achievable today for 4K memory packages (approximately 0.1 Failures Per Million Hours (FPMH)). The worst reliability that one could expect to encounter was judged to be equivalent to the series failure rate build up for the number of 4K parts required to make up the larger memory packages; i.e. for 16K: 0.4 FPMH, for 64K: 1.6 FPMH, and for 256K: 6.4 FPMH. Using these component part failure rates for each of the major elements provided the upper and lower bounds with respect to projected device reliability.



*The notation M/N means that out of the N identical elements in the system, M must be operating for the system as a whole to be operating.

Figure 5-1. Availability Block Diagram of the FMP

Next, a mathematical model was developed to study the potential improvements from SECDED. Using this model, it was found that gains could vary from a lower bound factor of 2 to upper bound factors of 164 for 16K, 327 for 64K, and 653 for 256K memory packages.

Finally, redundancy was considered. In this case, the ability to automatically detect, isolate, and decommit failed elements without noticeable interruption was investigated. As an upper bound on reliability, perfect recovery was considered. The lower bound was established for a situation where no recovery without interruption could be achieved. In this portion of the analysis, both permanent type failures which require a repair action and intermittent type failures which only require a recovery action were factored into the computations.

Using the previously discussed upper and lower bound values, it was determined that the design potential availability for the currently proposed FMP is:

* Upper Bound: $A_{FMP} = 0.9995$ (see Figure 5-2)
* Lower Bound: $A_{FMP} = 0.9554$ (see Figure 5-3)

Both these optimistic and conservative forecasts indicate a high degree of confidence in the ability of the proposed design to meet the overall system availability requirement of 90 percent. Using the above upper and lower bound availabilities for the FMP, it can be shown that the required availability of the B7800 host system to meet the 90 percent system availability is:

* $A_{B7800} = .9004$ for the Upper Bound FMP Requirement
* $A_{B7800} = .9420$ for the Lower Bound FMP Requirement

The above required availability values for the B7800 host system are currently being exceded by Burroughs B7700 systems operating in the field today. Since the B7800 system is expected to be even more reliable and maintainable than currently available B7700 systems, the overall system availability requirement for the FMP and the B7800 host system appears to be reasonable and achievable.

The data used to obtain these results are presented and discussed in the following sections.

5.2.3 THE BOUNDS OF FMP AVAILABILITY

This section shows the bounds of the failure rates of all packages and subsystems. The bounds of MUT (Mean-Up-Time), MDT (Mean-Down-Time) and availability of the FMP are the highlights. The failure rate of the system is significantly reduced with judicious design and the following factors:

1. A ground-based benign environment, where there is nearly zero environmental stress with optimum engineering operation and maintenance
2. Use of high quality parts, MIL-M-38510, class B level commercial parts being strongly suggested
3. On-line processor spares
4. Error correction techniques, including SECDED.
5. Adequate maintainability, as reflected in time to repair.

5.2.3.1 PACKAGE FAILURE RATES

The circuit packages are the basic elements in the FMP and accompanying the reliability of the FMP is a function of the failure rates of these packages. As mentioned in the previous section, the failure rates of digital circuit packages are predicted with the guidelines of MIL-HDBK-217B. Table 5-1 shows the predicted failure rates and the operating environmental conditions of the

control or logic packages used in the FMP. For the memory packages, the lower bound of those failure rates is 0.1 FPMH. The assumed upper bound of the failure rate of an m-bit memory package ($m > 4{,}000$), denoted as λ_m, may be computed with the following formula; representing the failure rate of the same memory built of 4k-bit parts.

$$\lambda_m = m \times \frac{\text{UPPERBOUND F.R. FOR 4K MEMORY}}{\text{4K BIT}} \text{ FPMH}$$

$$\lambda_m = m \times \frac{1}{4{,}000} \text{ FPMH} = M \times 2.5 \times 10^{-5} \text{ FMPH}$$

Table 5.2 shows the upper bounds of the failure rates of a variety of memory packages.

5.2.3.2 THE FAILURE RATES AND MTBF OF SUBSYSTEMS

A subsystem contains the packages listed in Tables 5-1 and 5-2. The failure rates of the subsystems of the FMP are predicted by parts count method. The memory subsystems failure rates are modified by the SECDED reilability improvement factor which is defined as the ratio of the subsystem MTBF with SECDED to that without SECDED. The factor is discussed in detail in appendix B. It can vary from two to six hundred and more, depending on the size of the memory package. Table 5-3 presents the list of the packages, the failure rates and MTBF of the control or data processing subsystems. Table 5-4 and 5-5 show the bounds of the failure rate and MTBF's of the memory subsystems. The upper (lower) bounds of the failure rates (MTBF's) are predicted with the SECDED reliability improvement factor of two and the failure rates of the memory packages at their upper bounds. The lower (upper) bounds of the failure rates (MTBF's) are generated when the SECDED improvement factors are at their upper limits and the failure rates of the memory packages are on their lower bounds.

Table 5-1. The Predicted Failure Rates of the Control or Logic Packages

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*
1000 0001	ECL CONTROL-SSI-I	DIG	4	16	45	GE	B	1	0.00622
1000 0002	ECL CONTROL-SSI-II	DIG	6	16	45	GE	B	1	0.00778
1000 0003	ECL CONTROL-SSI-III	DIG	15	16	45	GB	B	1	0.01307
1000 0004	ECL CONTROL-SSI-IV	DIG	22	16	45	GE	B	1	0.01633
1000 0005	ECL CONTROL-MSI	DIG	40	16	60	GE	B	1	0.06082
1000 0006	ECL CONTROL-LSI	DIG	130	16	60	GE	B	1	0.13000

Table 5-2. The Upper Bounds of the Failure Rates of Memory Packages

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*
2000 0001	MOS 16K RAM	RAM	16000	22	60	GB	B	1	0.40000
2000 0002	MOS 64K RAM	RAM	64000	22	60	GB	B	1	1.60000
2000 0003	MOS 256K RAM	RAM	256000	22	60	GB	B	1	6.40000

Table 5-3. The Predicted Failure Rates and MTBFs of the Control Subsystems

LEVEL 1 DESIGNATION: PE

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
1000 0006	ECL CONTROL-LSI	DIG	130	16	60	GB	B	100	0.13000	12.99982
MTBF=	76924.16	12.9998 FAILURES PER MILLION HOURS								

LEVEL 1 DESIGNATION: CU

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
1000 0001	ECL CONTROL-SSI-I	DIG	4	16	45	GB	B	2000	0.00622	12.44043
1000 0005	ECL CONTROL-MSI	DIG	40	16	60	GB	B	1000	0.06082	60.82423
MTBF=	13549.15	73.7647 FAILURES PER MILLION HOURS								

LEVEL 1 DESIGNATION: FOT

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
1000 0002	ECL CONTROL-SSI-II	DI	6	16	45	GB	B	2000	0.00778	15.55517
MTBF=	64287.32	15.5552 FAILURES PER MILLION HOURS								

LEVEL 1 DESIGNATION: IN

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
1000 0004	ECL CONTROL-SSI-IV	DIG	22	16	45	GB	B	10480	0.01633	171.12779
MTBF=	5843.59	171.1278 FAILURES PER MILLION HOURS								

LEVEL 1 DESIGNATION: TNC

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
1000 0001	ECL CONTROL-SSI-I	DIG	4	16	45	GB	B	500	0.00622	3.11011
MTBF=	321532.40	3.111 FAILURES PER MILLION HOURS								

LEVEL 1 DESIGNATION: FM-C

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
1000 0003	ECL CONTROL-SSI-III	DIG	15	16	45	GB	B	30	0.01307	0.39214
MTBF=	2550138.28	0.3921 FAILURES PER MILLION HOURS								

LEVEL 1 DESIGNATION: CBM-C

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
1000 0003	ECL CONTROL-SSI-III	DIG	15	16	45	GB	B	1000	0.01307	13.07119
MTBF=	76504.15	13.0712 FAILURES PER MILLION HOURS								

Table 5-4. The Lower (Upper) Bounds of the Failure Rates (MTBF) of the Memory Subsystems

LEVEL 1 DESIGNATION: PEM

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
1000 0001	ECL CONTROL-SSI-I	DIG	4	16	45	GB	B	15	0.00622	0.09330
2000 0001	MOS 16K RAM	RAM	16000	22	60	GB	B	55	0.00061	0.03353

MTBF= 7884153.75 0.1268 FAILURES PER MILLION HOURS

LEVEL 1 DESIGNATION: PEPM

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
2000 0001	MOS 16K RAM	RAM	16000	22	60	GB	B	28	0.00061	0.01707
1000 0001	ECL CONTROL-SSI-I	DIG	4	16	45	GB	B	15	0.00622	0.09330

MTBF= 9060039.53 0.1104 FAILURES PER MILLION HOURS

LEVEL 1 DESIGNATION: CUM

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
2000 0001	MOS 16K RAM	RAM	16000	22	60	GB	B	55	0.00061	0.03353

MTBF= 29820925.34 0.0335 FAILURES PER MILLION HOURS

LEVEL 1 DESIGNATION: EM-M

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
2000 0002	MOS 64K RAM	RAM	64000	22	60	GB	B	55	0.00030	0.01676

MTBF= 59651634.45 0.0168 FAILURES PER MILLION HOURS

LEVEL 1 DESIGNATION: DBM-M

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
2000 0003	MOS 256K RAM	RAM	256000	22	60	GB	B	55	0.00015	0.00842

MTBF=118757793.48 0.0084 FAILURES PER MILLION HOURS

Table 5-5. The Upper (Lower) Bounds of the Failure Rates (MTBF) of the Memory Subsystems

LEVEL 1 DESIGNATION: PEM

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
1000 0001	ECL CONTROL-SSI-I	DIG	4	16	45	GB	B	15	0.00622	0.09330
2000 0001	MOS 16K RAM	RAM	16000	22	0	GB	B	55	0.20000	11.00000

MTBF= 90144.48 11.0933 FAILURES PER MILLION HOURS

LEVEL 1 DESIGNATION: PEPM

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
2000 0001	MOS 16K RAM	RAM	16000	22	60			28	0.20000	5.60000
1000 0001	ECL CONTROL-SSI-I	DIG	4	16	45	GB	B	15	0.00622	0.09330

MTBF= 175644.96 5.6933 FAILURES PER MILLION HOURS

LEVEL 1 DESIGNATION: CUM

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
2000 0001	MOS 16K RAM	RAM	16000	22	60		B	55	0.20000	11.00000

MTBF= 90909.09 11.0000 FAILURES PER MILLION HOURS

LEVEL 1 DESIGNATION: EM-M

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
2000 0002	MOS 64K RAM	RAM	64000	22	60		B	55	0.80000	44.00000

MTBF= 22727.27 44.0000 FAILURES PER MILLION HOURS

LEVEL 1 DESIGNATION: DBM-M

PART NUMBER	*PART DESCRIPTION	*TYPE	*G/T/B	*PINS	*TEMP	*ENV	*QUAL	*QUANT	*INDIVIDUAL FR*	TOTAL FR*
2000 0003	MOS 256K RAM	RAM	256000	22	60		B	55	3.20000	176.00000

MTBF= 5681.82 176.0000 FAILURES PER MILLION HOURS

The legends of these and following tables are defined as:

TYPE - Integrated circuit type
G/T/B - Number of gates, or of transistors, or of bits
TEMP - Junction temperature predicted with MIL-HDBK-217B
ENV - Environment (GB - ground-based benign or standard office environment)
QUAL - quality/screening level (B-MIL-M-38510, class B)
QUANT - not listed in table 5-1 or 5-2
INDIVIDUAL FR - individual faiure rate (per million hours)

Some of the other terminology in these and following tables and figures is as follows. Mnemonics representing elements of the FMP are the same as those shown in Figure 5-1, such as "FOT" for "fanout tree" or "TNC" for the "control portion of the transposition network". "MRT" has been used for "mean down time"; the programmer was thinking that all down time was repair time. "RE" recovery efficiency is the fraction of the time that a retry is successful. For example, for a single bit failure in memory covered by SECDED, RE is 1.000. For a catastrophic "single point" failure, RE is 0.000. "Single point" identifies those portions of the system where a failure at a single point disables the system.

5.2.3.3 AVAILABILITY OF THE FMP

The major task of this section is to assess the bounds of MUT, and availability using the program DESIGN. Using the program we can thoroughly investigate critical factors pertinent to the failure, repair, and recovery processes. As required, the following determinants of system interruption and downtime have been included:

* Permanent and Intermittent Hardware Failure and Repair Rates
* System Automatic Recovery Features
* System Manual Recovery Rates

Sufficient data have been collected for design new systems successfully. With these data and all informations from the previous sections, the program provides an output with all salient input data and analytical results. The computer printouts used designations matching those on the block diagram of Figure 5-1. Corresponding to Table 5-4, Figure 5-2 shows a print-output which points out the upper bounds of MUT, and availability of the FMP are 1,032 hours, 0.43 hours, and .9995, respectively, as the MTBF of the hard failure is the same as the MTBF of the intermittent failure. Similarly corresponding to Table 5-4, Figure 5-3 presents an output which shows the lower bounds of MUT and availability are 3.5 hours and .9554 respectively, when the MTBF of the hard failure is ten times of the intermittent failure.

5.2.3.4 SENSITIVITY ANALYSIS

Since some factors shown in the previous sections are uncertain, and the failure rates of the memory packages are unknown, a sensitivity analysis has been made to study how those factors affect MUT, MDT, and availability of the FMP. Here we perform an experiment with respect to all the factors. In the experiment, some wide range varieties are considered, as in the following:

1. Two levels of the failure rates of the memory packages, namely the upper bounds and the lower bounds as shown in Section 2.1

NAME	R	N	MTBF(P)	MTBF(I)	SPFM	DRT	SRT	RE(P)	RE(I)	DMRT	MUT	MRT	AVAIL
CU	1	1	13649	13649	0.000	1.00	0.00	0.000	0.000	0.10	6824.5	0.550	0.999919
CUM	2	2	9000000	—	0.000	0.25	0.00	0.000	0.000	0.10	NO EFFECT ON PERFORMANCE		
TNC	1	1	321532	321532	0.000	0.50	0.00	0.000	0.000	0.10	160766.0	0.300	0.999998
FDT	1	1	64287	—	0.000	0.25	0.00	0.000	0.000	0.10	64287.0	0.250	0.999996
TN	1	1	5843	—	0.000	0.25	0.00	0.000	0.000	0.10	5843.0	0.250	0.999957
EM-C	521521		2550138	2550138	0.000	1.00	0.00	0.000	0.000	0.10	2447.3	0.550	0.999775
EM-M	521521		9000000	—	0.000	0.25	0.00	0.000	0.000	0.10	17274.5	0.250	0.999986
DBMC	1	1	76504	76504	0.000	1.00	0.00	0.000	0.000	0.10	38252.0	0.550	0.999986
DBM	512512		9000000	—	0.000	0.25	0.00	0.000	0.000	0.10	17578.1	0.250	0.999986
PROC-1	128129		75545	75545	0.005	1.00	0.25	1.000	1.000	0.10	50162.4	0.222	0.999995
PROC-2	128129		75545	75545	0.005	1.00	0.25	1.000	1.000	0.10	50162.4	0.222	0.999996
PROC-3	128129		75545	75545	0.005	1.00	0.25	1.000	1.000	0.10	50162.4	0.222	0.999996
PROC-4	128129		75545	75545	0.005	1.00	0.25	1.000	1.000	0.10	50162.4	0.222	0.999996

LEGEND

R	:	Number of Devices Required to be Operating for Success
N	:	Number of Devices Available
MTBF(P)	:	Mean Time Between Failures – Permanent
MTBF (I)		Mean Time Between Failures – Intermittent
SPFM		Percentage of Failures that are Single Point Failures
DRT		Device Repair Time – Permanent Failures
SRT		Single Point Failure Repair Time – Permanent Failures
RE (P)	:	Recovery Efficiency – Permanent Failures
RE (I)		Recovery Efficiency – Intermittent Failures
DMRT	:	Device Manual Recovery Time

FMP TOTAL= 1032.1 0.43 0.99958549

Figure 5-2. Print Output of the Upper Bounds of MUT, MRT and Availability of the FMP

NAME	R	N	MTBF(P)	MTBF(I)	SPFM	DRT	SRT	RE(P)	RE(I)	DMRT	MUT	MRT	AVAIL
CU	1	1	13649	1365	0.000	1.00	0.00	0.000	0.000	0.10	1240.9	0.182	0.999857
CUM	2	2	90909	--	0.000	0.25	0.00	0.000	0.000	0.10	45454.5	0.250	0.999995
FDT	1	1	64287	--	0.000	0.25	0.00	0.000	0.000	0.10	6428.7.0	0.250	0.999996
TNC	1	1	321532	32153	0.000	0.50	0.00	0.000	0.000	0.10	29230.0	0.136	0.999995
TN	1	1	5843	--	0.000	0.25	0.00	0.000	0.000	0.10	5843.0	0.250	0.999957
EM-C	521521		2550138	255014	0.000	1.00	0.00	0.000	0.000	0.10	445.0	0.182	0.999592
EM-M	521521		22727	--	0.000	0.25	0.00	0.000	0.000	0.10	43.6	0.250	0.994302
DBMC	1	1	76504	7651	0.000	1.00	0.00	0.000	0.000	0.10	6955.4	0.182	0.999974
DBM	512512		5682	--	0.000	0.25	0.00	0.000	0.000	0.10	11.1	0.250	0.977969
PROC-1	128129		33572	3357	0.005	1.00	0.25	0.000	0.000	0.10	23.7	0.100	0.995787
PROC-2	128129		33572	3357	0.005	1.00	0.25	0.000	0.000	0.10	23.7	0.100	0.995787
PROC-3	128129		33572	3357	0.005	1.00	0.25	0.000	0.000	0.10	23.7	0.100	0.995787
PROC-4	128129		33572	3357	0.005	1.00	0.25	0.000	0.000	0.10	23.7	0.100	0.995787

LEGEND

R	:	Number of Devices Required to be Operating for Success
N	:	Number of Devices Available
MTBF(P)	:	Mean Time Between Failures – Permanent
MTBF (I)		Mean Time Between Failures – Intermittent
SPFM		Percentage of Failures that are Single Point Failures
DRT		Device Repair Time – Permanent Failures
SRT		Single Point Failure Repair Time – Permanent Failures
RE (P)		Recovery Efficiency – Permanent Failures
RE (I)		Recovery Efficiency – Intermittent Failures
DMRT	:	Device Manual Recovery Time

FMP TOTAL = 3.5 0.16 0.955484975

Figure 5-3. Printout Output of the Lower Bounds of MUT and Availability of the FMP

2. Two levels of SECDED improvement factors, taking "two" as the lower bound level while the upper level corresponding to the upper limit of different memory packages stated in Section 2.2

3. The ratio between the MTBT of intermittent failure to the MTBT of permanent failure are 1, 5 and 10.

4. The recovery efficiencies are chosen from 70% to 100% with 10% increment.

The results are summarized in Table 5-6. From the results we learn the availability changing only from 96.13 to 99.96% is not significantly affected by those factors. If the memory packages are of a low reliability level and SECDED improvement factors are low, MUT and MDT are affected slightly by them. On the other hand, if the memory packages are highly reliable and SECDED improvement factor is large, the MUT is increased by 200% to 300% and the MDT is decreased by 25% to 30% as the ratio between the MTBF for permanent failures (MTBF(P)) and the MTPF for intermittent failures (MTBF(I)) changes from 1 to 5. Under the same conditions the MUT increases very rapidly as the recovery efficiency is close to 100%. Finally it can be pointed out that the MUT is significantly affected by the reliability quality of the memory packages as expected.

5.3 ERROR DETECTION AND CORRECTION

5.3.1 Error Control Coverage

In the baseline system there are a number of mechanisms for error detection and correction. These include error detection and correction on all memories, with sufficiently powerful codes to guarantee uncorrected error rates lower than a specified requirement, and undetected error rates below an even lower required rate.

Table 5-6. Sensitivity Analysis of the MUT, MRT and Availability of the FMP

RUN NO.	PACKAGE FAILURE RATE	Reliability Improvement Factor	MTBF (P) / MTBF (I)	RECOVERY EFFICIENCY (%)	MUT	MDT	AVAIL-ABILITY
1	.1 f/Mh	*	1	70	194.3	.16	.9992
2	"	"	1	80	263.9	.18	.9993
3	"	"	1	90	411.4	.23	.9994
4	"	"	1	100	1032.1	.40	.9995
5	"	"	5	70	68.5	.12	.9982
6	"	"	5	80	95.0	.13	.9986
7	"	"	5	90	155.1	.15	.9990
8	"	"	5	100	421.5	.23	.9994
9	"	"	10	70	37.8	.11	.9971
10	"	"	10	80	52.7	.12	.9974
11	"	"	10	90	87.1	.13	.9985
12	"	"	10	100	249.2	.18	.9993
13	"	2	1	70	109.1	.18	.9984
14	"	2	1	80	135.3	.20	.9985
15	"	2	1	90	178.2	.23	.9987
16	"	2	1	100	260.9	.29	.9989
17	"	2	5	70	52.1	.14	.9974
18	"	2	5	80	68.9	.15	.9978
19	"	2	5	90	101.7	.17	.9983
20	"	2	5	100	194.0	.24	.9988
21	"	2	10	70	27.9	.12	.9957
22	"	2	10	80	37.8	.13	.9966
23	"	2	10	90	60.0	.14	.9976
24	"	2	10	100	145.4	.21	.9986

Note * : 16K RAM-- 164 ** : 16K RAM-- .4 f/Mh
64K RAM-- 327 64K RAM-- 1.6 f/Mh
256K RAM-- 653 256K RAM-- 6.4 f/Mh

The mechanisms fall into three classes. First, there are errors such that immediate correction is done, even if there is a single hard error in the machine. Error correction in memory is such. Second, there are errors that are detected immediately when they occur. Third, there is a repertoire of checks which is intended to detect as many as possible of those errors not detected immediately. For example, memory words are initialized to "invalid". As long as a substantial amount of memory is in the "invalid" state, there is a substantial chance of detecting a memory addressing error because of the "invalid" word fetched in response.

Table 5-7 shows the pecentage of the total chips in the FMP that are covered by each made of error correction. There are approximately ninety-eight thousand chips (49% of the machine) that have error correction capabilities applied to them in the baseline system. These are the memory chips. In addition there are about twelve thousand additional chips that are involved in data transfer paths of sufficient parallelism that the addition of error-correcting check bits in parallel would represent a modest (20% to 40%) increase in parts count. There are one hundred eighteen thousand chips in the baseline system that have immediate error detection. This includes all the memory chips plus the transposition network which has the EM error detection code on all data passed through it and parity on microcode ROMs. We could add about nine thousand chips to this total by putting a modulo-3 check digit on all arithmetic units and adding parity or SECDED to the parallel path from CU to processors. Additional chips would be required by such additional error detection.

Table 5-7. Error Control Methods and Applicability

Table 5-7. Error Control Methods and Applicability

UNIT	Error Control Methods Available at Reasonable Redundancy			Error Control Methods Obscure	
	No. Chips	Error Detection	Error Correction	No. Chips	Comments
PE	7k arith	mod-3 check digit for arith. parity on microcode.	Retry on error(?)	34k non-arith.	(Note 2)
PDM/ PPM	38k mem.	yes (Note 1). SECDED will work.	yes (Note 1). SECDED will work.	14k control	Many errors will be address errors, also
TN	10k	EM's SECDED catches hard errors(Note 1)	Under investigation	---	---
EM	31k mem	SECDED or better if needed. Note 1	SECDED or better if needed. Note 1	16k control	Note 2
Fanout	2K in parallel paths	Can add parity	Can add SECDED at 25%	1k single signal	Note 2
CU	½ mem.	Same as PDM	Same as PDM	3k	Random logic Note 2
DC		———	———	1k	DC not used during user program
DBM	29k mem.	SECDED or stronger. Note 1.	SECDED or stronger code. Scrubbing of errors. Note 1.	2k control	Note 2
TOTAL possible	127k	127k chips have error detectible at same clock that error occurs	120k chips have error correctible even if hard failure exists	71k	Dominated by PE logic, and memory controls. 41% of NSS.
TOTAL as per baseline	118k	118k chips have error detectible at same clock that error occurs	108k chips have error correctible even if hard failure exists	80k	Dominated by PE logic, and memory controls. 45% of NSS.

Note 1. This error detection/correction is included in the baseline system as described in the final report.

Note 2. Consistency checks, initialization to "invalid", confidence tests, etc. are designed to forestall any error from going undetected for too long. Undetected transient failures are the primary concern.

5.3.2 Improvements over Reference 1

Reference 1 lists a large number of reasonableness checks that attempt to monitor the errors in that 40% to 44% of the FMP for which direct error correction and error detection cannot be implemented simply. These include tests for "invalid", the code to which memory is initialized. These include a check for illegal opcodes, or memory addresses out of bounds, including bounds checks on index calculations. Unnormalized numbers should never be fetched for a floating point operation. The list goes on. All of these are helpful. None, obviously, gives absolute protection.

Three items should be added to the design of reference 1 in the area of error detection and correction. These follow.

5.3.2.1. On-line Processor Spares. An on-line spare processor is extremely effective in eliminating repair time, or postponing actual repair until convenient. Appendix C describes the implementation in detail. One spare per cabinet is provided.

5.3.2.2. Error Detection, Error Correction in PDM, PPM, and CUM. These memories, whose memory chips account for 19% of all the circuit packages in the FMP, are to be provided with error correction. The final report seems to have obscured this requirement by laying stress on an error correction method which quite possibly may not work. Likewise, error detection for uncorrectible errors is to be provided. SECDED is being provided in the baseline system, as of this report.

5.3.2.3. Error Correction in the Transposition Network. The error correction code of the EM provides error detection against hard failures in the transposition network and error correction against single transient failures. This is included already in the baseline system design, even though reference 1 failed to emphasize it. It is possible to provide a TN design which corrects for single hard errors in the TN, just as SECDED corrects for single hard errors in memory. The best code for this purpose has yet to be determined. One design adds three signals to the already nine-wide TN path. Four Hamming check bits are applied to the eight data bits in each byte. The OR of all twelve bits can serve instead of the strobe, since all parities are odd. The byte-correcting code is in effect concatenated with the SECDED code used in EM, so no overall parity is needed for error detection; the SECDED takes care of that.

5.3.3 Duplexed Computation

For an almost 100% check on the computation, one can repeat the user program, using a different set of 512 processors for the second run. Using the processor switching of Appendix C, one can run the problem first with the spare at the right end, and then second with the spare at the left end. If the answers agree, the answer is presumably free of any hardware error. Note that this method is simpler, from a hardware implementation point of view, than operating the processors in pairs which shadow each other, but, like having pairs of processors do the same computation, it also cuts the throughput in half.

5.3.4 Hard Error Tolerance

The habitual use of confidence and diagnostic checks, together with all the above error detection, assures that a hard failure cannot remain undetected for long in the FMP. Repair time is essentially zero for failures in that 82% of the chips in the FMP, where either error correction allows the FMP to continue to run in spite of the error, or processor switching switches in a spare processor while the bad processor is removed and replaced at leisure. For the remaining 18% of the components, repair is needed before the FMP can continue to run. Thus, detection of hard failure is more than adequately done and availability is aided by having 82% of the failures associated with "zero" repair time, or postponable repair.

5.3.5 Transients

60% of the packages, if involved in some transient error, will produce effects that are immediately detected and usually corrected, leaving 40% not covered. Obviously, it is better to include tests that have some chance of detecting error than not to have such tests. However, it is difficult to guarantee that all transient errors will be caught before the run ends for 99.9% of the runs. Even if we add mod-3 check digits in arithmetic, and parity in the CU-to-processor fanout tree, 36% of the packages remain in this category. The part of the machine where detection of transient error is less than perfect consists of the memory control and proecessor logic, primarily not the arithmetic portion of the processor, but instruction decoding, register addressing, shifting, and miscellaneous logic.

The main defense against transient error is, and always has been, proper electrical and logic design. Wiring rules, noise budgets, crosstalk calculations, maximum delay calculations, and so on, are all part of the design.

CHAPTER 6
TRADEOFFS DELINEATED

6.1 INTRODUCTION

The design of the FMP will result from tradeoffs among a number of factors

* Performance
* Reliability
* Availability
* Programmability
* Spectrum of Applications
* Cost
* Schedule
* Risk

The first four factors are explicitly mentioned in the statement of work for the extension to this study contract. The fifth, the spectrum of applications for which the FMP is to be designed, is mentioned here as it has a direct bearing on the results of some of the tradeoffs. For example, a scalar processor would probably not be included if the applications were strictly limited to aerodynamic flow and meterological problems. Yet the scalar processor will be necessary for some other applications and will interfere only slightly with the other desiderata.

Programmability covers two distinct aspects. First, is the system one with which the compiler writer can successfully contend? Second, is the system presented to the user, including its FORTRAN, an easy one?

Following are short discussions of specific issues where the result is a trade between factors. In many cases, simulation using test cases taken from the intended spectrum of applications is the appropriate tool to resolve the tradeoffs.

6.2 LANGUAGE DEFINITION

A part of the language definition in the extended FORTRAN to be used for the FMP in an exercise of trading off throughput vs. programmability. Proper language design finds some point where almost the maximum throughput of the machine can be applied to the desired spectrum of applications with little difficulity from language restrictions or awkard constructs. That is, the language restrictions necessary to ensure throughput do not interfere much with one's ability to write programs for the selected set of applications.

However, we note that programmability for all applications will interfere greatly with throughput, and that absolute maximum throughput for all applications is likely to require a depth of analysis beyond that feasible in the compiler.

6.3 MATCHING THE COMPILER AND THE INSTRUCTION SET

Hardware capabilities that are unused by the compiler are a waste of money and represent a flaw in the design. Capabilities in the language, that would be commonly and frequently used, for which the hardware provides no convenient way for the compiler to implement, result in awkward and inefficient code, and are also a flaw. However, the hardware, once specified, is not likely to

have its instruction set expanded much during the life of the machine, while the compiler presumably will continue to evolve during that same period. Therefore, it is the capabilities of that eventual hoped-for compiler, not the simplicity of the first one, against which the instruction set is to be judged. An example is the loading of PPM conditional on the "enable" bit. Our first compiler has no use for such a conditional capability. However, the capability costs almost nothing, since loading memory must be conditional on "enable" anyway, while the capability allows a type of concurrency between processors which we expect to be useful in the long run.

6.4 WORD FORMAT

In reference 1, a word format of 1 bit sign, 8 bits exponent, and 39 bits fraction part is suggested as ideal for the FMP. The BSP uses 1 bit sign, 11 bits exponent, and 36 bits fraction. The format with 7 bits exponent was determined as adequate for the Navier-Stokes application. The BSP format was arrived at after judging the precision and range requirments of a wide variety of applications. Thus, the BSP word format is more likely to be suitable for a wider variety of applications, some of which will require the additional range on the exponent, while the requirement of 10 decimal digits precision for the Navier-Stokes equations will be satisfied with either format.

Therefore, for the purpose of being adaptable to a wider range of applications, and not incidentally, for the additional purpose of being format-compatible with an existing commercial product, it is proposed to standarize on a word format containing 1 bit sign, 11 bits exponent, and 36 bits fraction part.

6.5 INSTRUCTION FORMATS

There is a well-known tradeoff between code file size and ease of decoding the individual instruction. For example, a full-length address field in the instruction allows the use of absolute addresses where appropriate, whereas if the instruction has a short address field, it must always be with respect to some base address held in the hardware.

In the present instance, a variation which we wish to test by simulation, during phase II, is the use of 32-bit and 16-bit instructions. The 16-bit instruction has room for only two register addresses; the 24-bit instruction contains three. Therefore the use of 16-bit formats will speed up instruction fetching while interfering with the optimization of the use of registers in the processor. According to one example tested, the instruction fetching is already faster than arithmetic execution, and 24-bit instructions will be preferred.

6.6 SECDED

Rigid requirements were set up for main memory in the FMP, consisting of PDM, PDP, and CUM. Less than one bit in 10^{16} is to be in error uncorrected, and less than one bit in 10^{18} is to be undetected. To satisfy these requirements, a single-error-correction, double-error-detection code is proposed. However, at this writing the actual error rates and failure mechanisms of the memory chips to be used are unknown. When these error rates and failure mechanisms become known, the SECDED should be reevaluated to make sure that it is neither too weak to cope with the error rates actually occurring, nor an overkill causing unnecessary cost. Since SECDED may permit the scheduling of repair while the system continues to run in degraded mode, it produces savings in maintenance cost while improving availability. The memory chips would have to be unbelievably reliable before SECDED did not pay for itself.

6.7 TRUSTWORTHINESS VS. THROUGHPUT

In considering error correction and detection, we credit the FMP, not with the total number of right answers it produces, but with the amount of answers that a rational user can use with confidence. One approach to trading off error correction and detection against raw throughput is to maximize this effective throughput. With no error correction at all, it is determined that most answers are probably wrong, and the effective throughput is practically zero, even though reams of so-called answers might be coming off the printer. With triple redundancy and voting on every element in the system, the throughput would be a fraction of the raw throughput with no error correction, but the answers would be very trustworthy. Somewhere between these extremes is an optimum. As explained in the last part of section five, the existing baseline system design has sufficient error detection that there is little chance for a hard error to go undetected for long. A more severe problem for the FMP is the defense against transient errors.

In the baseline system design described in reference 1, 54% of the packages in the system have single error correction, so that any single error produced in these packages is corrected during the run, which continues to produce correct answers. 11% of the packages have immediate detection of any errors in them, so the run terminates immediately if errors occur in them. The other 35% of the packages are covered by a variety of error checks, which are intended to eventually detect any errors. However, the detection is indirect and not immediate, and some transient errors will remain undetected.

C.3

If we apply additional error checks, throughput is reduced, but trustworthiness of the results is improved. Figure 6-1 is an oversimplified graphical representation of the effect. At some reasonable amount of error control circuitry, the effective throughput is maximized. Using f to represent the fraction of the total hardware devoted to error control (assuming total hardware remains constant), we can plot T_0, the "raw" throughput, equal to the number of inches in the pile of printout per hour, and T, the effective throughput which is the amount of useful answers produced. T_0 decreases with f. In fact, T_0 decreases faster than linearly with f, since (1-f) of the hardware is devoted to producing useful output, and the fraction f that checks for errors can only interfere. We can write:

$$T=(T_0 \times (1-f))/G(f)$$

The function G(f) can only increase with f, for any rational design.

Finding the form of the funtion G(f) is probably not feasible. What can be done, however, is to estimate the effect on the detected and undetected error rates for any particular proposed error detection/correction technique, together with its effect on parts count or raw throughput. Each proposed error control mechanism costs a certain percentage of the equipment, has a certain throughput reduction associated with it, and catches some percentage of otherwise uncaught errors.

As an example, consider the addition of a modulo 3 check digit to arithmetic computation. Generating the check digit takes almost as much additional logic as is already in the adders being checked. Thus, adding 7% to the chip count of the machine catches almost all errors occurring in what is now about 7% of the machine. In addition, the 7% new packages create errors of their own, which will usually be detected as arithmetic errors, so they do not add to the undetected error rate, but do create false alarms.

Is a 7% false alarm rate added to the rate of detected error, a 7% increase in parts count and power, plus the throughput reduction due to the extra clocks used for checking, a fair price to pay for the X% decrease in the rate of undetected error? When the actual percentages are determined, perhaps the question can be answered.

6.8 Parity within Processors

Data transfers within the processor have been designed on the expectation that the reliability and accuracy of digital operations in logic circuits can be made as perfect as desired at the design stage, using worst-case design. Whatever the error requirements, careful design can ensure that the performance exceeds them.

Parity checks on inter-register transfers could be implemented, including transfer to the memory address registers. Such parity checks will add about five chips to the processor logic for each parity check required. Four parity checkers, or twenty chips, may be needed. In addition, one clock, for the parity checking, will be added to many operations, including most of the operations that are now one clock long. Although no careful study of the situation has yet been done, it is apparent that parity checking internal to the processor will add 20% to the component count of the PE, will add errors of its own, and will degrade raw throughput significantly, while failing to check any of the processor logic operations, only the transfers.

6.9 INSTRUCTION FETCHING MECHANISM

In section two, the equipment description, a particular scheme for overlapping the execution of noninterfering instructions, and for doing some anticipatory instruction fetching was described. This scheme has not been validated in simulation to see how well it

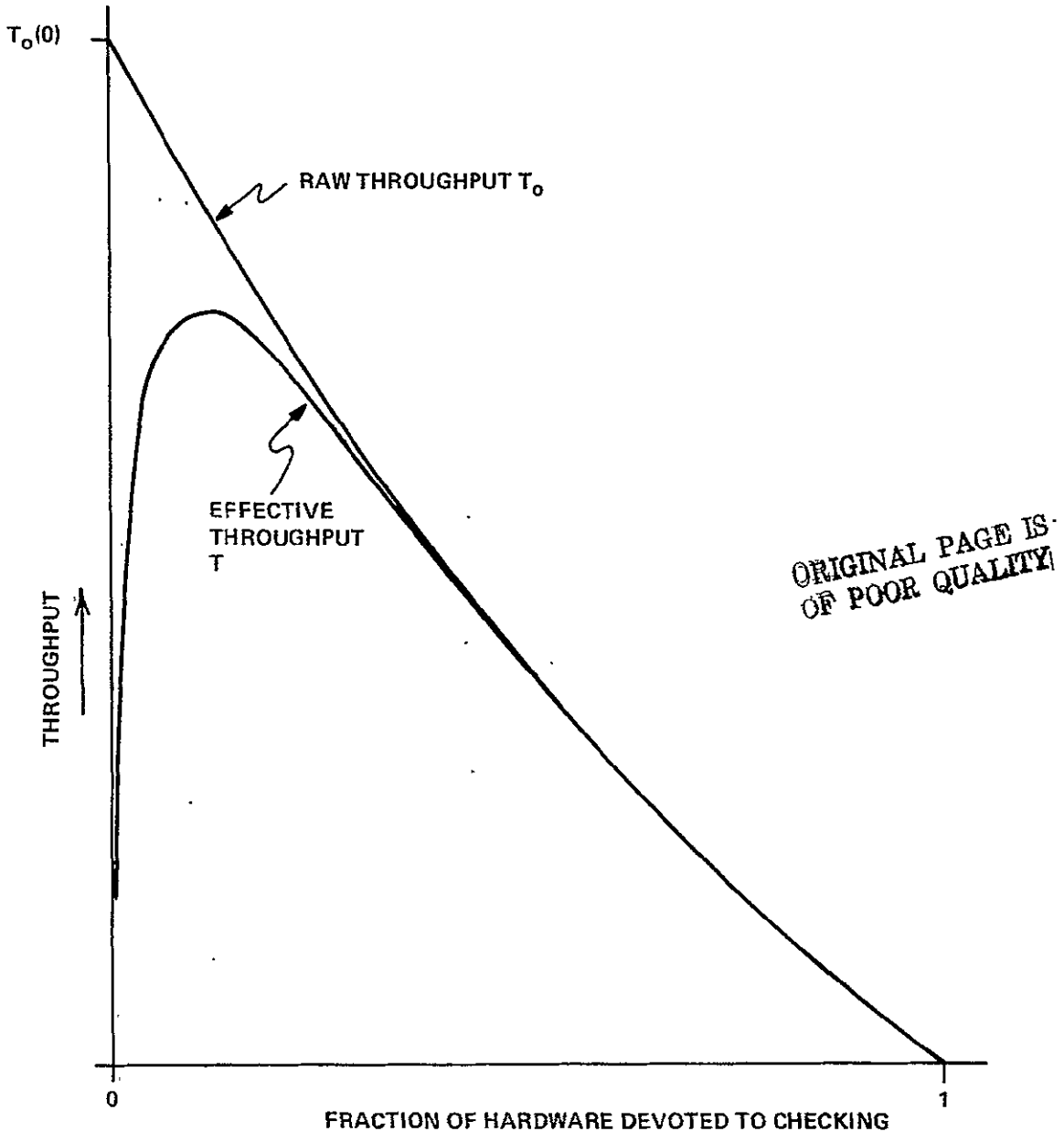


Figure 6-1. Throughput vs. Error Detection

works in real program streams as emitted by the compiler. Simulation studies to determine how simple an instruction fetching and overlap mechanism we can have and still maintain throughput would be desirable. Fortunately, most of the processor design details are independent of these decisions.

6.10 LOADEM AND STOREM BLOCK FETCHING

The baseline system as described in Chapter Two of this report omits from the LOADEM and STOREM instructions the ability to stream N words out of each EM module in parallel for a total of 512N words per instruction. Initial work on handcompiling from FORTRAN source for the NSS indicates that almost all fetching from EM is with N=1. (Example: SUBROUTINE TURBDA, See Ch. 3) If this turns out to be true in general, the block fetching capability is not worth the complexities it costs. Simulation, using test cases taken from real code, with multiple-word fetches allowed and disallowed, can be used to evaluate the effect on throughput. If N greater than 1 is necessary, the following changes to the baseline system of Chapter Two are seen:

* Rearrangement of data on DBM-EM transfers is required, as described in the final report, so that, for N>1, data in EM along the index in which streaming is taking place are all found in the same EM module. Rearrangement is neither needed or desirable when N=1.

* The requirement for rearrangement of data disallows most equivalencing on EM arrays, a restriction on normal FORTRAN that need not be imposed if N=1.

* EM module design becomes more complicated. To keep up with the TN streaming rate, the EM module is divided into two

submodules, as a side effect making the SECDED code less effective. A need to increment the EM address per word while streaming also adds complexity, especially since the increment is a large integer, not unity.

* There is additional compiler complexity.

Enforcing the restriction that N must be 1 thus enhances reliability and availability, while simplifying compiler and operating system, and having an undetermined effect on throughput.

6.11 OVERLAPPABLE EM ACCESS

A fourth instruction execution station could be added to the processor which would handle the EM access independently of the integer and floating point units at the expense of requiring two units contending for PDM, namely this EM unit, and the previously identified memory control. Having issued an EM fetch to this unit, no fetches from PDM would be allowed.

The amount of increased overlap obtainable is dependent on the compiler's being able to insert the EM fetches ahead of the place where the data is required. In some of the loops in the benchmark programs, this requires the insertion of the EM accesses for the next iteration inside the current interation. The question to be answered by a tradeoff study is whether the increased compiler complexity required to exploit such an addition to the design produces enough increased throughput to be worth the difference.

6.12 SINGLE PROCESSOR MEMORY

Processor memory is separated into two separate memories for the sake of increased throughput. Data fetching and instruction fetching go on in parallel. Furthermore, no conflict resolution between fetching program and data need be implemented. The traditional way of getting interlace between two memory modules in a single memory system is to make module number the least significant bit of the address. This particular method would not work in the processor, since data is fairly random, and program steps, although sequential, are interspersed with data fetches and stores. Thus, the two-memory design of the baseline system achieves better interlacing than the traditional scheme. However, it has the drawback that program and data memory is not interchangeable; a program just over 8192 words cannot overflow into data memory, and similarly for data.

An alternate design for the processor memory is as follows. Two modules of 16384 words each are used to form a single homogeneous address space. Module number is the most significant bit. The compiler assigns all program addresses to the upper module and all data addresses to the lower module, except that, if either module is full, the other module can be used.

The alternate design achieves just as good interlace of memory accesses as does the baseline system. When memory sizes are exceeded by either data or program but not by both together, the penalty is a slight slowdown, not an inability to run. Memory controls are slightly more complex, since program and data accesses will interfere whenever either overflows its normal half of the memory.

6.13 PROCESSOR PROGRAM MEMORY SIZE, CONTROL UNIT MEMORY SIZE

The processor program memory (8k words) was chosen to adequately hold the aerodynamic flow model programs. Overlay of code from CUM is easy and quick, and allows PPM to be smaller than the entire code file. However, PPM should be large enough so that overlay is not so frequent as to interfere with throughput.

An overlay capability can be provided so that program can overlay into CUM from DMB, via a buffer area in EM. Since such overlay is not needed for the flow model, it was not proposed as part of the initial capabilities of the operating system.

For a different spectrum of applications, larger code files and different sequences of execution may be encountered. Hence, the code storage capabilities of the FMP may have to be reevaluated if there is a change in the spectrum of applications.

6.14 EXTENDED MEMORY SPEED, TRANSPOSITION NETWORK SPEED

The baseline system extended memory is constructed of 64k-bit RAM chips, operated at the fastest reasonable cycle time available at the time the FMP is constructed. It was projected for the baseline system that the cycle time would be on the order of 200 to 250 ns for the chip, and that therefore a cycle time for the EM module of 280 ns was appropriate.

If the 64k-bit chip is in fact significantly faster than that, EM would be designed faster to match the chips. But, to go faster than allowed by the 64k- bit chips will require the use of 16k-bit RAM chips, a four-fold increase in memory chip count from 28,655 chips to 114,620, a 43% increase in the chip count in the FMP and a distinctly adverse effect on availability and cost.

The point to be determined by the tradeoff is whether to increase in throughput from using 16k-bit chips is worth the extra cost, additional failures, and extra power of using 16k-bit chips in the EM modules.

The results of this tradeoff will be a function of how much computation is accomplished per fetch from extended memory, which is very dependent on the specified spectrum of applications. It was clear that for the aerodynamic flow problems, and almost certainly for the meterological problems also, that the 64k-bit chips will have more speed than needed. It also appears (according to the Electronic Times of November 7), that actual 64k-bit chips will be faster than those postulated for the baseline system. Simulation, using inputs that represent the entire spread of intended applications, is the appropriate tool for investigating this tradeoff.

The TN speed and design will have to be adjusted to match the EM speed. Thus, the revision in TN design will also have to be factored into the tradeoff. An EM made faster by using 16k-bit chips is partially self-defeating, since the wire lengths from EM to processor, now about 40 feet, will get significantly longer when the EM quadruples in physical size.

6.15 CONTROL UNIT SPEED

The speed of the control unit, including the implementation of specific instructions such as DIV 521, DIV 512, and MOD 521 that are needed for specific CU actions (in this case, calculating EM address and TN settings), is best determined by simulation using test cases that cover the entire spectrum of applications. A very fast MOD 521 instruction has been described by C. R.Vora in U.S. patent 3,980,874. Since there is only one control unit in the

entire array, the optimum CU design is clearly that one that almost never interferes with throughput. On the other hand, a too fast and hence unnecessairly complex CU will have adverse effects on reliability and availability, and possibly will also make the compiler design more complex if some of the complexities require cooperation from the compiler to be effective. This optimum CU design is a function of the spectrum of applications.

6.16 SCALAR PROCESSOR

6.16.1 Dependency on Spectrum of Applications

The FMP has been described as an array of 512 processors and a control unit. The control unit concerns itself with synchronization, some address calculation, and loop control. All floating point arithmetic is done in the array. Aerodynamic flow models are well calculated on this machine. However, there are other applications, which do not have sufficient parallelism almost everywhere in the algorithm to be efficiently computed on this machine. If it is desired to broaden the spectrum of applications of the FMP, it is desirable, for some applications, to furnish a scalar processor to take over those portions of the floating-point calculation where most of the processors are idle waiting for a few to complete calculations. The term "scalar Processor", as used here, refers strictly to floating point scalar computations. Loop control and other program execution control where a single decision controls the processing of the entire array has been accomplished, on other architectures, by the "scalar processor" portion of the equipment. These functions are included as an essential part of the control unit, and in so far as they are scalar, the control unit is a scalar processor, whether or not specific equipment for handling floating point scalars is supplied.

An evaluation of which applications are going to require the addition of a scalar processor for efficient mapping onto the FMP has not been made. It is suspected that the meteorology applications are like the aero flow models and will not require a scalar processor. Whether a scalar processor is desirable, and which of the several options mentioned below for including a scalar processor in the design, is a function of the intended set of applications, and can therefore be defined properly only when NASA defines the amount and kind of extensibility of scope that is desired for the FMP. The baseline system as described includes the third of the three design options below.

6.16.2 Simple Scalar Processor The simplest recipe for providing a scalar processor capability in the FMP is simply to provide a faster, more powerful processor for processor number 0. The first processor is the one that will be assigned to vectors of length one; and which will be executing processor code when the compiler can find no parallelism. Thus, without doing anything special to the compiler, we gain some scalar capability by simply making the first processor a faster one. During parallel swatches of code, this processor cooperates with the others, and the program does not know that it is different. Those swatches of code where 512 processors are idle take much less time because the first processor has been made faster. When short swatches of scalar and vector code alternate, overlapping of scalar and vector operations occurs.

6.16.3 Added Processor The simple system does not give the scalar processor any particular speedup for accessing EM. It does not give the scalar processor any faster way of handling those actions that require cooperation with the control unit. At the expense of complicating the compiler, we can add scalar processor hardware that is separately programmed, and which can subsume some of the control unit functions for scalar processing.

Suppose we provide a separate, and different processor, which has its own access to extended memory, and which is designed to execute a more nearly independent code stream than that of the 512 processors in the array. Figure 6-2 shows a block diagram of the FMP with such a scalar processor represented. Langauge extensions and programming methods for using such a capability will have to be defined.

Extended memory is "core" for the FMP. The amount of accessing into extended memory by the scalar processor may be such that extended memory speed will be a bottleneck for those applications that make extensive use of the scalar processor capability. Hence, for some range of applications, a faster extended memory (and hence one with fewer bits per chip), must be provided. Using 16k-bit chips instead of 64k-bit chips, for more EM speed, increases from 29,176 memory chips to 116,704 memory chips, an increase of 44% of the package count of the entire NSS.

The added processor has LOADEM and STOREM instructions in its instruction stream which do not require the cooperation of the CU, merely contend with it for access to the extended memory. The synchronization between the added processor and the CU is thereby reduced, while requiring the compiler to determine when synchronization is required for correct execution of the program. Scalar processing and vector processor on the same data must be done in the correct order.

6.16.4 Enhanced Control Unit It has been suggested that scalar processor capability can be achieved by adding floating point instructions to the control unit. This also may imply that the control unit be speeded up from its no-scalar-processor design so it has the free time to perform as a scalar processor. The discussions about accessing EM apply to this option as well as they apply to the previous one.

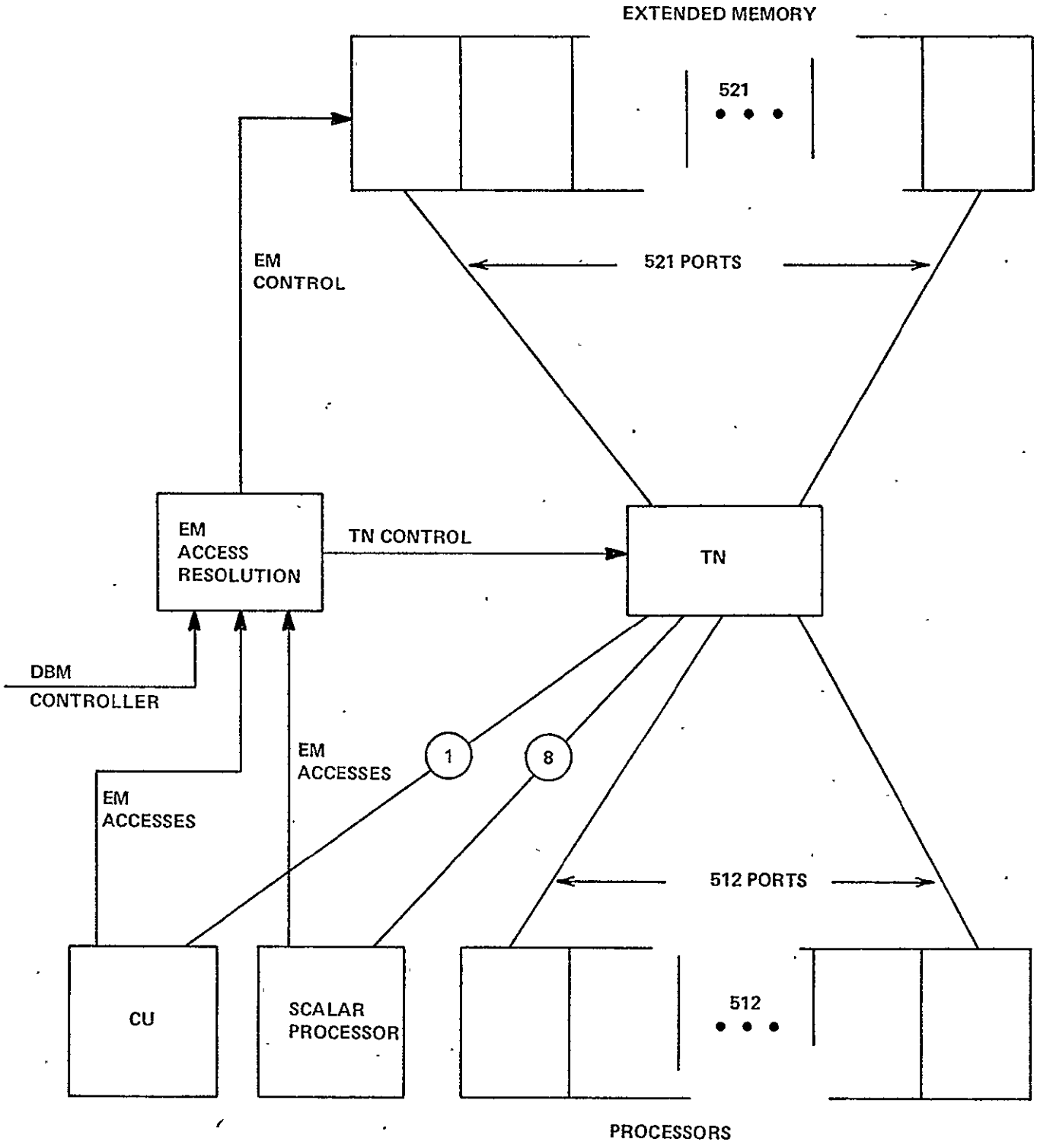


Figure 6-2. Added Scalar Processor

6.16.5 Recommendation Simulation of various programs across the entire spectrum of applications is recommended as a means of determining which of the several recipes for providing a scalar processor is to be adopted, if any. The budget for compiler writing is also to be consulted, since the separate processor requires additional decisions on the compiler's part, as well as additional language extensions perhaps.

6.17 MARGINAL CHECKING

A strategy for weeding out incipient failures in electronic equipment is to vary some parameter up and down from its nominal value, measure the margins, and determine when those margins are deteriorating, and what the faulure mode is at which they fail. The parameter being varied can be supply voltage, clock frequency, temperature, or anything else that appears to affect operation. It has been determined that marginal checking is useless for worst-case designed digital circuits. However, as noted in the final report, LSI cannot be worst-case designed in the conventional sense, and marginal checking may be valuable for weeding out those low-margin LSI packages that have a higher than normal transient error rate.

6.18 COMPONENT TECHNOLOGY

The speed of any given system architecture is ultimately limited by the performance of the circuit from which it is assembled. The final component choice for the FMP will weigh carefully the trade off of speed (and power) consideration against the risk and cost. The inital procurement cost of a more advanced technology providing more desirable performance is easily measured. It is

usually shown that the initial cost of more advanced circuit are easily justified in overall system performance improvements. (Thus reducing the cost per operation.) However, the risk in selecting a more advanced and higher performance circuit invariably may be considerable, with potential for affecting the production of system being built in a number of ways:

* The delivery may be slow due to low yields.
* Failure rates may be higher than anticipated.
* The performance characteristics of devices made in production may be degraded from the original developmental samples and design goals.
* Low usage may discourage development of second sources, and result in continued elevated prices.
* Unforeseen application problems discovered only during system checkout could require redesign or retrofit.

It would be very desirable from a system performance point of view to be able to use the fastest circuits possible. However, the possible risks that accompany this choice make it imperative that a very careful tradeoff analysis be conducted given the choice of a mature, slow (but adequate) speed technology and an advanced faster speed technology.

6.19 EXPANSABILITY

By expansibility we mean generalizability and expandability. The NASF design has many features allowing an upward compatible second copy, as well as features allowing the upgrading of the NASF itself. This section lists some of the areas in which expansibility is found.

6.19.1 Address Sizes The address sizes are uniformly larger than the memories they address, allowing the memories to be replaced by larger ones.

Data Base Memory holds 134 million words (2^{27} and is addressed by the control unit whose register size is 32 bits.

Extended memory holds 34 million words (just over 2^{25}) and is addressed by processor (32-bit integers) and control unit (32 bits).

Control unit memory holds 32k words (2^{15}) and is addressed by the control unit whose integers are 32 bits long. Care should be exercised not to insert 16-bit address register that cannot be expanded.

Processor data memory holds 16k words (2^{14}) and has a 16-bit address. A four-times expansion of PDM is thus permitted.

Processor program memory holds 8k words (2^{13}) and has a 16-bit address.

Upgrades by replacing the memories with larger ones are therefore very feasible.

6.19.2 Transfer Rates There are a number of options for increasing the transfer rates between portions of the FMP. Many of these are discussed in other paragraphs in this section, and clearly, new transfer rates could be chosen for any new design, depending on the results of tradeoff studies. As a retrofit, the easiest area to increase transfer rates is in the DBM-EM transfers. This is fortunate , since if some virtual memory scheme is implemented,this is the area of the baseline design that may have to be improved. Each EM module has a one-word buffer, so no EM changes at all are required for increased transfer rates, just increased parallelism is the accessing of these buffers. The DBM would have to be reconfigured for increased parallelism, assuming that current projections about CCD shift rates are correct.

6.19.3 Memory Size The address space allows increased memory size. The need for increased memory size could arise from a number of causes. CUM is required to hold enough program (both CU and array processor program) to keep the array busy for a reasonable amount of the time between program overlays from DBM. Thus, complex programs may require increased CUM size.

PDM size is the result of the requirement for temporary variables, and sometimes, for buffering data fetched from EM. The required PDM size is therefore applications-dependent. We believe that the aerodynamic flow problem requires a larger-than-typical PDM, and that larger PDM's are unlikely. However, the expansion opportunity is there.

PPM, on the other hand, must hold enough program to keep the processors busy for a reasonable time between overlays from CUM. For problems, like the aerodynamic model, where there is an inner loop, this implies that at least the inner loop be contained within the PPM. Overlay from CUM is fast, and this will allow reasonable efficiency even when this is not true.

DBM, the window in the computational envelope, must be large enough to hold results from the last job, space for the current job, and the objects being assembled for the next job. If job sizes are to grow, expandability of the DBM is a requirement.

6.19.4 Upgrades via Software Upgrading capability, by adding features to the software, can be accomplished without any hardware changes. The initial software is configured around the areodynamic flow model requirements. A number of features, not required by the aerodynamic flow models, can be added to handle a broader range of requirements, including:

* Windowing of data for executing jobs whose files exceed the size of EM.
* Language extensions, including such things as subscripted subscripts, linear recurrences on the parallel subscript, and so on.
* Vectorizer, to analyze nonparallel FORTRAN and produce FMP FORTRAN for operation on the parallel machine.
* Multiprogramming capability on the FMP. Proper implementation of multiprogramming may require hardware additions as well.

APPENDIX A

Preliminary Compiler Algorithms for Setting the Transposition Network

Definition of the FORTRAN extensions and restrictions for the NASF requires rigorous definition of the algorithms for setting the SKIP and OFFSET of the transposition network and matching them closely to the FORTRAN constructs.

The issues to be addressed in this memo are:

1. Matching of FORTRAN DOPARALLEL to EM accessing.
2. Requirements for multiple accessing within a DOPARALLEL construct.
3. Optimization of accessing for single access types.

As a preliminary step in addressing these issues a more complete definition of the DOPARALLEL statement needs to be formulated. The DOPARALLEL statement cannot be nested for this results in possible programmer error. Rather the DOPARALLEL statement is defined to have multiple increment sets.

i.e. DOPARALLEL J=J1,J2,J3; K=K1,K2,K3 ...

where J1 = initial value most rapidly varying index
J2 = final value most rapidly varying index
J3 = skip distance most rapidly varying index
K1 = initial value next most rapidly varying index
K2 = final value next most rapidly varying index
K3 = skip distance next most rapidly varying index
(...) ellipses indicates further increment sets

ENDDO;ENDDO

1. Matching Fortran DOPARALLEL to Extended Memory Accessing

Since the entire set of multidimensional DOPARALLEL statements is difficult to discuss, the specific example of three dimensional accessing with a 2 dimensional DOPARALLEL and a single dimensional inner loop will be described in detail. For this three dimensional case there are 6 possible access patterns for any given array corresponding to the possible permutation of the indices.

A(I,J,K)	Case I
A(K,I,J)	Case II
A(J,K,I)	Case III
A(I,K,J)	Case IV
A(J,I,K)	Case V
A(K,J,I)	Case VI

It is necessary for the compiler to determine the SKIP distance and the OFFSET of the transposition network for any of these accesses for the given DOPARALLEL construct. i.e.,

```
      EMARRAY A(IFIRST, ISECOND, ITHIRD)
      DOPARALLEL J=1, JLIM; K=1, KLIM
      DO 1 I=1 ILIM
        S(i) = Access Case (i)
    1 Continue
      ENDDO; ENDDO
```

The equations for setting the Transposition Network (SKIP and OFFSET) are given in Tables 1A through 1C. Table 1D provides a table for determining index parameters. It is assumed, of course, that the array has been laid out in memory in the FORTRAN sense.

To clarify these equations a complete example is worked out in detail in Figures 1-7. The chosen array; A(5,3,7) has extents less than the number of memory modules (11) and processing elements (10) in a manner similar to that of the NASF problems.

Equations for

Transition Network OFFSET Calculations

Given Quantities

N = Number of processors

M = Number of memory modules

IAØ = Base address of array having index parameters IØ, JØ, KØ

IFIRST = extent of first parameter in array

ISECOND = extent of second parameter in array

ITHIRD = extent of third parameter in array

Determined Quantities from Figure 1

ICLIM = Total number of cycles

IDEL = Skip distance associated with I parameter

JDEL = Skip distance associated with J parameter

KDEL = Skip distance associated with K parameter

ILIM = Array extent assciated with I parameter

JLIM = Array extent associated with J parameter

KLIM = Array extent associated with K parameter

Defined quantities

IC = cycle number

NN = subiteration number

K1 = (N*(IC-1))/(JLIM) + KØ = least rapidly varying index*

J1 = (N*(IC-1) - (K-KØ) * JLIM + JØ = most rapidly varying index*

IAØØ = IAØ + (J-JØ)*JDEL + (K-KØ)*KDEL

Transposition Setting SKIP distance = JDEL

*J1, K1 values for processing element Ø

1st subiteration

Table 1A

-OFFSET Calculation for Transposition Network (Subiteration = 1)
for given I value

IADD(IC,1) = IAØØ + (I-IØ) * IDEL (address of first element to be fetched)

OFFSET (IC,1) = (IADD(IC,1)) MOD(M)

-OFFSET Calculation for Transposition Network (all other subiterations*)
for given I value

IADD(IC,NN) = IAØ + (I-IØ)*IDEL + (K1-KØ + NN-1) *KDEL
(address of first element to be fetched on this iteration)

IP (IC,NN) = (NN-1)*JLIM -J1+JØ
(processor that needs to obtain this first element on this iteration)

OFFSET (IC,NN) = (IADD(IC,NN) - IP(IC,NN)*JDEL) MOD(M)

*Subiterations $2 < NN \leq NX$

where $NX = \frac{2N+1+(JLIM-J1)}{N} + 1$ $\qquad N\lambda = i + \frac{N+J1-1}{JLIM}$

If (NN.EQ.NX). AND (K(NN).EQ.KLIM) further subiterations do not need to be performed. K(NN) is the K index value of the 1st element of the NNth subiteration.

Table 1B

Parameter Assignments for Arbitrary Array Extents
and Number of Processors

CASE	ILIM	JLIM	KLIM	IDEL	JDEL	KDEL	ICLIM
1 (I,J,K)	JFIRST	ISECOND	ITHIRD	1	IFIRST	IFIRST*ISECOND	(ISECOND*ITHIRD +N-1)/N
2 (K,I,J)	ISECOND	ITHIRD	~~ISECOND~~ I FIRST	IFIRST	IFIRST*ISECOND	1	(IFIRST*ITHIRD) +N-1)/N
3 (J,K,I)	ITHIRD	IFIRST	ISECOND	IFIRST*ISECOND	1	IFIRST	(IFIRST*ISECOND +N-1)/N
4 (I,K,J)	IFIRST	ITHIRD	ISECOND	1	IFIRST*ISECOND	ISECOND	(ISECOND*ITHIRD +N-1)/N
5 (J,I,K)	ISECOND	IFIRST	ITHIRD	IFIRST	1	IFIRST*ISECOND	(IFIRST*ITHIRD) +N-1)/N
6 (K,J,I)	ITHIRD	ISECOND	ITHIRD	IFIRST*ISECOND	IFIRST	1	(IFIRST*ISECOND +N-1)/N

EM ARRAY A(IFIRST, ISECOND, ITHIRD)
Number of Processors = N
Table 1C

Index Value Determination

TEMP = IADD(IC,NN) - IAØ) - (I-1)*JDEL

Case	TEMP	J	K	IVAL	JVAL	KVAL
1	NO	J	K	I	J	K
2	YES	TEMP/JDEL+1	(TEMP-(J-1)*JDEL) /KDEL+1	K	I	J
3	NO	J	K	J	K	I
4	YES	TEMP/JDEL+1	(TEMP-(J-1)*JDEL) /KDEL+1	I	K	J
5	YES	TEMP-(K-1)*KDEL) /JDEL+1	TEMP/KDEL+1	J	I	K
6	YES	TEMP/JDEL+1	(TEMP-(J-1)*JDEL) /KDEL+1	K	J	I

Table 1D

Figure 1 details the memory layout, assuming an arbitrary starting point for the first element. The remaining Figures show the six possible cases.

Utilizing the equations of Table 1 one can determine all the parameters and the SKIP and OFFSET for any case. For example taking CASE II (since it is more complex with access A(K,I,J)) the parameters are:

Given Quantities (Table 1A)

N=10
M=11
IAØ=19
IFIRST=5
ISECOND=3
ITHIRD=7

Determined quantitites (Table 1C)

ICLIM = (IFIRST*ITHIRD+N-1)/N (5*7+10-1)/10 =4
IDEL=5
JDEL=15
KDEL=1
ILIM=5
JLIM=7
KLIM=3
IØ, JØ, KØ=1

Assume that one wishes to determine the SKIP and OFFSET and subsequently the IVAL, JVAL & KVAL of the indices for the second cycle, second subiteration, inner loop index number 3 - i.e. transposition setting #12

Defined Quantities (Table 1A)

IC=2
NN=2
I=3

Memory Layout for Array A(5,3,7)

Address within Memory	0	1	2	3	4	5	6	7	8	9	10
11	337	347	537								
10	217	317	417	517	127	227	327	427	527	137	237
9	126	226	326	426	526	136	236	336	436	536	117
8	525	135	235	335	435	535	116	216	316	416	516
7	434	534	115	215	315	415	515	125	225	325	425
6	314	414	514	124	224	324	424	524	134	234	334
5	223	323	423	523	133	233	333	433	533	144	214
4	132	232	332	432	532	113	213	313	413	513	123
3	531	112	212	312	412	512	122	222	322	422	522
2	411	511	121	221	321	421	521	131	231	331	431
1	x	x	x	x	x	x	x	x	111	211	311
0	x	x	x	x	x	x	x	x	x	x	x

Memory Modules

No. Memory Modules = 11

No. Processing Elements = 10

Absolute address AØ = 19

Memory Module No. M# = 8 = (19) MOD 11

Address in MOdule A# = 1 = (19) DIV 11

Address of any element AE# = Address A(L1, L2, L3)

$$A\emptyset + (L1-1) + 5x(L2-1) + 5 \times 3(L3-1)$$

Figure 1

Case I

```
EMARRAY  A(5,37)
DOPARALLEL J=1,3; K=1,7
DO 1 I = 1,5
S1 = A(I,J,K)
1 CONTINUE
  ENDDO
  ENDDO
```

SKIP = JDEL = 5

Setting Number	Cycle	Sub Iteration	OFFSET	PE NUMBER 0	1	2	3	4	5	6	7	8	9	ADD
1	1	1	8	111	121	131	112	122	132	113	123	133	114	19
2	1	1	9	211	221	231	212	222	232	213	223	233	214	20
3	1	1	10	311	321	331	312	322	332	313	323	333	314	21
4	1	1	0	411	421	431	412	422	432	413	423	433	414	22
5	1	1	1	511	521	531	512	522	532	513	523	533	514	23
6	2	1	3	124	134	115	125	135	116	126	136	117	127	69
7	2	1	4	224	234	215	225	235	216	226	236	217	227	70
8	2	1	5	324	334	315	325	335	316	326	336	317	327	71
9	2	1	6	424	434	415	425	435	416	426	436	417	427	72
10	2	1	7	524	534	515	525	535	516	526	536	517	527	73
11	3	1	9	137										119
12	3	1	10	237										120
13	3	1	0	337										121
14	3	1	1	437										122
15	3	1	2	537										123

Figure 2

Case II

```
EM ARRAY A(5,3,7)
DOPARALLEL J=1,7; K=1,5
DO 1 I = 1,3
S2 = A(K,I,J)
1. CONTINUE
 ENDDO
 ENDDO
 :
```

SKIP = JDEL = 15

Setting Number	Cycle	Sub Iter	OFFSET	PEM Number 0	1	2	3	4	5	6	7	8	9	Assigned ADD	PE#
1	1	1	8	[111]	112	113	114	115	116	117				19	0
2	1	2	3								[211]	212	213	20	7
3	1	1	2	[121]	122	123	124	125	216	217				25	0
4	1	2	8								[221]	222	223	26	7
5	1	1	7	[131]	132	133	134	135	136	137				30	0
6	1	2	2								[231]	232	233	31	7
7	2	1	10	[214]	215	216	217							65	0
8	2	2	5					[311]	312	313	314	315	316	31	7
9	2	1	4	[224]	225	226	227							70	0
10	2	2	10					[321]	322	323	324	325	326	26	4
11	2	1	9	[234]	235	236	237							75	0
12	2	2	4					[331]	332	333	334	335	336	27	4
13	3	1	1	[317]										111	0
14	3	2	7		[411]	412	413	414	415	416	417			22	1
15	3	3	2									[511]	512	23	8
16	3	1	6	[327]										116	0
17	3	2	1		[421]	422	423	424	425	426	427			27	1
18	3	3	7									[521]	522	28	8
19	3	1	0	[337]										121	0
20	3	2	6		[431]	432	433	434	435	436	437			32	1
21	3	3	1									[531]	532	33	
22	4	1	9	[513]	514	515	516	517						53	0
23	4	1	3	[532]	524	525	526	527						58	0
24	4	1	8	[533]	534	535	536	537						63	0

Figure 3

Case III

```
EM ARRAY A(5,3,7)
DOPARALLEL J=1,5; K=1,3
DO 1 I = 1,7
S3 = A(J,K,I)
1 CONTINUE
  ENDDO
  ENDDO
```

SKIP = JDEL = 1

Setting Number	Cycle	Sub Iter	OFFSET	PEM Number 0	1	2	3	4	5	6	7	8	9	Assigned ADD	PE#
1	1	1	8	111	211	311	411	511	121	221	321	421	521	19	0
2	1	1	1	112	212	312	412	512	122	222	322	422	522	34	0
3	1	1	5	113	213	313	413	513	123	223	323	423	523	49	0
4	1	1	9	114	214	314	414	514	124	224	324	424	524	64	0
5	1	1	2	115	215	315	415	515	125	225	325	425	525	79	0
6	1	1	6	116	216	316	416	516	126	226	326	426	526	94	0
7	1	1	10	117	217	317	417	517	127	227	327	427	527	109	0
8	2	1	7	131	231	331	431	531						29	0
9	2	1	0	132	232	332	432	532						44	0
10	2	1	4	133	233	333	433	533						59	0
11	2	1	8	134	234	334	434	534						74	0
12	2	1	1	135	235	335	435	535						89	0
13	2	1	5	136	236	336	436	536						104	0
14	2	1	9	137	237	337	437	537						115	0

Figure 4

Case IV

```
EM ARRAY A(5,3,7)
DOPARALLEL J=1, 7; K=1, 3
DO 1 I = 1,5
S4 = A(I,K,J)
1 CONTINUE
 ENDDO
 ENDDO
```

SKIP = JDEL = 15

Setting Number	Cycle	Sub Iter	OFFSET	PEM Number 0	1	2	3	4	5	6	7	8	9	Assigned ADD	Assigned PE#
1	1	1		[111]	112	113	114	115	116	117				19	0
2	1	1	7								[121]	122	123	24	7
3	1	1	9	[211]	212	213	214	215	216	217				20	0
4	1	2	8								[221]	222	223	25	7
5	1	1	10	[311]	312	313	314	315	316	317				21	0
6	1	2	9								[321]	322	323	26	7
7	1	1	0	[411]	412	413	414	415	416	417				22	0
8	1	2	10								[421]	422	423	27	7
9	1	1	1	[511]	512	513	514	515	516	517				23	0
10	1	2	0								[521]	522	523	28	7
11	2	1	3	[124]	125	126	127							69	0
12	2	2	2					[131]	132	133	134	135	136	29	4
13	2	1	4	[224]	225	226	227							70	0
14	2	2	3					[231]	232	233	234	235	236	30	4
15	2	1	5	[324]	325	326	327							71	0
16	2	2	4					[331]	332	333	334	335	336	31	4
17	21	1	6	[424]	425	426	427							72	0
18	2	2	5					[431]	432	433	434	435	436	32	4
19	2	1	7	[524]	525	526	527							73	0
20	2	2	6					[531]	532	533	534	535	536	33	4
21	3	1	9	[137]										119	0
22	3	1	10	[237]										120	0
23	3	1	0	[337]										121	0
24	3	1	1	[437]										122	0
25	3	1	2	[537]										123	0

Figure 5

Case V

```
EM ARRAY A(5,3,7)
DOPARALLEL J=1, 5; K=7
DO 1 I = 1,5
S5. = A(J,I,K)(illegible)
1 CONTINUE
  ENDDO
  ENDDO
```

SKIP = JDEL = 1

Setting Number	Cycle	Sub Iter	OFFSET	PE# Number 0	1	2	3	4	5	6	7	8	9	ADD	Assigned PE#
1	1	1	8	111	211	311	411	511						19	0
2	1	2	7						112	212	312	412	512	34	5
3	1	1	2	121	221	321	421	521						24	0
4	1	2	1						122	222	322	422	522	39	5
5	1	1	7	131	231	331	431	531						44	0
6	1	2	6						132	232	332	432	532	59	5
7	2	1	5	113	213	313	413	513						49	0
8	2	2	4						114	214	314	414	514	69	5
9	2	1	10	123	223	323	423	523						54	0
10	2	2	9						124	224	324	424	524	69	5
11	2	1	4	133	233	333	433	533						59	0
12	2	2	3						134	234	334	434	534	69	5
13	3	1	2	115	215	315	415	515						79	0
14	3	2	1						116	216	316	416	516	94	5
15	3	1	7	125	225	325	425	525						84	0
16	3	2	6						126	226	326	426	526	99	5
17	3	1	1	135	235	335	435	535						89	0
18	3	2	0						136	236	336	436	536	104	5
19	4	1	10	117	217	317	417	517						109	0
20	4	1	4	127	227	327	427	527						114	0
21	4	1	9	137	237	337	437	537						119	0

Figure 6

Case VI

```
EM ARRAY A(5,3,7)
DOPARALLEL J=1, 3; K=1, 5
DO 1 I = 1,7
S6 = A(K,J,I)
1 CONTINUE
  ENDDO
  ENDDO
```

SKIP = JDEL = 5

Setting Number	Cycle	Sub Iter	OFFSET	PEM Number 0	1	2	3	4	5	6	7	8	9	ADD	Assigned PE#
1	1	1	8	[111]	121	131								19	0
2	1	2	5				[211]	221	231					20	3
3	1	3	2							[311]	321	331		21	6
4	1	4	10										[411]	22	9
5	1	1	1	[112]	122	132								34	0
6	1	2	9				[212]	222	232					35	3
7	1	3	6							[312]	322	332		36	6
8	1	4	3										[412]	37	9
9	1	1	5	[113]	123	133								49	0
10	1	2	2				[213]	223	233					50	3
11	1	3	10							[313]	323	333		51	6
12	1	4	7										[413]	52	9
13	1	1	9	[114]	124	134								64	0
14	1	2	6				[214]	224	234					65	3
15	1	3	3							[314]	324	334		66	6
16	1	4	0										[414]	67	9
17	1	1	2	[115]	125	135								79	0
18	1	2	10				[215]	225	235					80	3
19	1	3	7							[315]	325	335		81	6
20	1	4	4										[415]	82	9
21	1	1	6	[116]	126	136								94	0
22	1	2	3				[216]	226	236					95	3
23	1	3	0							[316]	326	336		96	6
24	1	4	8										[416]	97	9
25	1	1	10	[117]	127	137								109	0
26	1	2	7				[217]	227	237					110	3
27	1	3	4							[317]	327	337		111	6
28	1	4	1										[417]	112	9
29	2	1	5	[421]	431									27	0
30	2	2	2			[511]	521	531						23	2
31	2	1	9	[422]	432									42	0
32	2	2	6			[512]	522	532						38	2
33	2	1	2	[423]	433									57	0
34	2	2	10			[513]	523	533						53	2
35	2	1	6	[424]	434									72	0
36	2	2	3			[514]	524	532						68	2
37	2	1	10	[425]	435									87	0
38	2	2	7			[515]	525	535						83	2
39	2	1	3	[426]	436									102	0
40	2	2	0			[516]	526	536						98	2
41	2	1	7	[427]	437									117	0
42	2	2	4			[517]	527	537						113	2

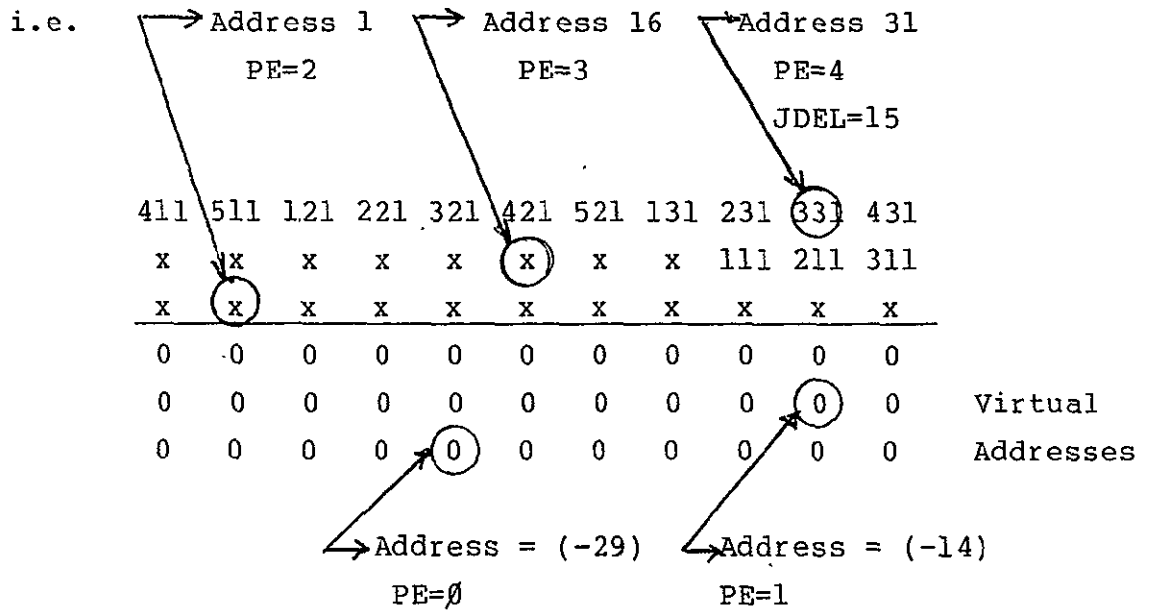
Figure 7

```
K1=(10+1)/7+1=2
J1=10-1*7+1=4
SKIP=JDEL=15
```

Using the OFFSET calculation equation for NN=2 in Table 1B one obtains

```
IADD(2,2)=19 + (3-1)*5 (2-1 + 2-1)*1
         = 19 + 10 + 2 = 31
IP (2,2) = (2-1) * 7-4+1 = 4
OFFSET (2,2) = (IADD(2,2) - IP(2,2)*15) MOD (11)
             = (31-4*15) MOD (11)
             = (-29) MOD (11) = 4
```

This OFFSET calculation may appear strange at first glance. SInce one wishes this element to be produced in processing element 4 one needs to determine what the "virtual" address of the array element would have been to put an element into processing element Ø.



Mode bits for PE's #0,1,2,3 will produce null fetches.

Having now determined the SKIP and the OFFSET one may wish to determine the specific indices of the element. This is done by means of Table 1D.

```
Temp = (31 - 19) - (3 - 1) *5
     = 12 - 10 = 2
J = 2/15 + 1 = 1
K = (2 - (1-1) *15)/1 + 1 = 3
A(IVAL, JVAL, KVAL) = A(K,I,J) = A(3,3,1)
```

In a similar fashion one can determine the SKIP and OFFSET for any setting number for any of the six possible cases. Additionally Table II gives a listing of a computer program which performs these computations.* Representative output is given in the appendix for the set of cases listed below.

IAØ	Mem Mod	#PEs	IFIRST	ISECOND	ITHIRD
19	11	10	5	3	7
19	11	10	9	5	6
19	11	10	6	2	8
27	13	11	6	2	8

2. Requirements for Multiple Accessing within DOPARALLEL Construct.

The compiler will recognize if a variety of access types occur within a given DOPARALLEL and will modify the basic access algorithm. For example given

*Note this is a very preliminary algorithm and should not be considered "proven" software in any sense.

Table II

```
 $SET LIST
 $RESET FREE
FILE  5=COMPILER/DATA,UNIT=DISK,RECORD=14,BLOCKING=30
FILE  6=FILE6,UNIT=PRINTER
C  ***************************************************************
C
C
C                BURROUGH'S CORPORATION
C                COMPILER ALGORITHM  FOR DETERMINING
C                TRANSPOSITION NETWORK SETTINGS
C                OFFSET AND SKIP DISTANCE FOR 3-D ARRAYS
C
C
C  ***************************************************************

      DIMENSION IADD(10,10),IP(10,10),ISET(10,10)
C*************************************************************
C
C                INPUT VARIABLES
C                ITYPE = ARRANGEMENT OF ARRAY INDICES
C                        ITYPE = 1  (I,J,K)
C                        ITYPE = 2  (K,I,J)
C                        ITYPE = 3  (J,K,I)
C                        ITYPE = 4  (I,K,J)
C                        ITYPE = 5  (J,I,K)
C                        ITYPE = 6  (K,J,I)
C                IAO    = BASE ADDRESS OF ARRAY A WITH INDICES IO,JO,KO
C                M      = NUMBER OF MEMORY MODULES
C                N      = NUMBER OF PROCESSING ELEMENTS
C                IFIRST= ARRAY EXTENT OF FIRST DIMENSION OF ARRAY
C                ISECND= ARRAY EXTENT OF SECOND DIMENSION OF ARRAY
C                ITHIRD= ARRAY EXTENT OF THIRD DIMENSION OF ARRAY
C                JO     = INDEX VALUE OF BASE ADDRESS
C                KO     = INDEX VALUE OF BASE ADDRESS
C
C  ***************************************************************
      READ(5,100) ITYPE,IAO,M,N,IFIRST,ISECND,ITHIRD,KO,JO
      WRITE(6,111) ITYPE,IAO,M,N,IFIRST,ISECND,ITHIRD,KO,JO
C  ***************************************************************
C
C                SET UP OF INITIAL PROBLEM PARAMETERS INDEPENDENT
C                       OF ORDERING:
C                       IDEL,JDEL,KDEL = SKIP DISTANCES
C                       ILIM,JLIM,KLIM = ARRAY EXTENTS
C                       ICLIM          = NUMBER OF CYCLES
C
C  ***************************************************************
      IF(ITYPE.EQ.1) GO TO 1
      IF(ITYPE.EQ.2) GO TO 2
      IF(ITYPE.EQ.3) GO TO 3
      IF(ITYPE.EQ.4) GO TO 4
      IF(ITYPE.EQ.5) GO TO 5
      IF(ITYPE.EQ.6) GO TO 6
      IF(ITYPE.LT.1) GO TO 7
      IF(ITYPE.GT.6) GO TO 7
 1    IDEL = 1
      JDEL = IFIRST
      KDEL = IFIRST*ISECND
      ICLIM= (ISECND*ITHIRD+N-1)/(N)
      JLIM = ISECND
      ILIM = IFIRST
      KLIM = ITHIRD
      GO TO 8
 2    IDEL = IFIRST
      JDEL = IFIRST*ISECND
      KDEL = 1
      ICLIM= (IFIRST*ITHIRD+N-1)/(N)
      JLIM = ITHIRD
      ILIM = ISECND
```

```
      KLIM = IFIRST
      GO TO 8
 3    ICEL = IFIRST*ISECND
      JDEL = 1
      KDEL = IFIRST
      ICLIM= (IFIRST*ISECND+N-1)/(N)
      JLIM = IFIRST
      ILIM = ITHIRD
      KLIM = ISECND
      GO TO 8
 4    IDEL = 1
      JDEL = IFIRST*ISECND
      KDEL = IFIRST
      ICLIM= (ISECND*ITHIRD+N-1)/(N)
      JLIM = ITHIRD
      ILIM = IFIRST
      KLIM = ISECND
      GO TO 8
 5    IDEL = IFIRST
      JDEL = 1
      KDEL = IFIRST*ISECND
      ICLIM= (IFIRST*ITHIRD+N-1)/(N)
      JLIM = IFIRST
      ILIM = ISECND
      KLIM = ITHIRD
      GO TO 8
 6    JDEL = IFIRST
      KDEL = 1
      IDEL = IFIRST*ISECND
      ICLIM= (IFIRST*ISECND+N-1)/(N)
      JLIM = ISECND
      ILIM = ITHIRD
      KLIM = IFIRST
      GO TO 8
 7    WRITE(6,101)
 01   FORMAT(2X,'YOU HAVE AN ERROR IN ITYPE')
      GO TO  80
 8    WRITE(6,112) IDEL,JDEL,KDEL,ICLIM,JLIM,ILIM
      WRITE(6,114)
C*****************************************************************
CCC
CCC         START OF CYCLE LOOP
CCC
CCC
C*****************************************************************
      DO 10 IC = 1,ICLIM
      IVV= N*(IC-1)
      K=(IVV)/(JLIM) + KO
      K1= K
      J=(IVV)-(K-1)*(JLIM) + JO
      J1=J
      IADD= IA0+(J-1)*JCEL+(K-1)*KDEL
C     WRITE(6,113) IC,IVV,J,K
C*****************************************************************
CCC
CCC       START OF INNERMOST LOOP INDEX ---I---
CCC
C*****************************************************************
      DO 20 I = 1,ILIM
      IADD(IC,1) = IADD +(I-1)*IDEL
      IP(IC,1) = 0
C*****************************************************************
CCC
CCC       SUBITERATION LOOPS
CCC
C*****************************************************************
      DO 30 NN= 1,N
      N2=NN-1
      IF (NN.EQ.1) GO TO 9
      IADD(IC,NN) = IA0+(I-1)*ICEL + (K1-1+N2)*KDEL
      IP(IC,NN) = N2*JLIM-J1+1
 9    CONTINUE
C     WRITE(6,100) IC,NN,IADD(IC,NN),IP(IC,NN)
      IF(IP(IC,NN).GT.N-1) GO TO 20
      ISET(IC,NN)= (IADD(IC,NN)-IP(IC,NN)*JCEL)
```

```
C*****************************************************************
C
C
C        ADJUSTING ISET TO POSITIVE NUMBER
C
C
C*****************************************************************
         DO 40 KAP= 1,100
         IF(ISET(IC,NN).GE.0) GO TO 50
         ISET(IC,NN) = ISET(IC,NN) + M
   40    CONTINUE
C        WRITE(6,100) IC,NN,ISET(IC,NN)
   50    ISET(IC,NN) = MOD(ISET(IC,NN),M)
C*****************************************************************
C
C
C        DETERMINATION OF INDEX VALUES -- FIRST ELEMENT OF SET--
C        NOT REQUIRED FOR OFFSET AND SKIP DETERMINATIONS
C
C
C  ***************************************************************
         IF(ITYPE.EQ.1) GO TO 201
         IF(ITYPE.EQ.2) GO TO 202
         IF(ITYPE.EQ.3) GO TO 203
         IF(ITYPE.EQ.5) GO TO 205
         IF(ITYPE.EQ.6) GO TO 206
  201    IVAL=I
         JVAL=J
         KVAL=K
         GO TO 207
   02    TEMP=(IADD(IC,NN)-IA0 - (I-1)*IDEL)
         J=TEMP/JDEL + 1
         K= (TEMP -(J-1)*JDEL)/KDEL + 1
         IVAL=K
         JVAL=I
         KVAL=J
         GO TO 207
  203    IVAL=J
         JVAL=K
         KVAL=I
         GO TO 207
  204    TEMP=(IADD(IC,NN)-IA0 - (I-1)*IDEL)
         J=TEMP/JDEL + 1
         K= (TEMP -(J-1)*JDEL)/KDEL + 1
         IVAL=I
         KVAL=J
         JVAL=K
         GO TO 207
  205    TEMP=(IADD(IC,NN)-IA0 - (I-1)*IDEL)
         K=TEMP/KDEL + 1
         J=(TEMP-(K-1)*KDEL)/JDEL + 1
         IVAL=J
         JVAL=I
         KVAL=K
         GO TO 207
  206    TEMP=(IADD(IC,NN)-IA0 - (I-1)*IDEL)
         J=TEMP/JDEL + 1
         K= (TEMP -(J-1)*JDEL)/KDEL + 1
         IVAL=K
C  ***************************************************************
C
C
C        END OF INDEX COMPUTATIONS
C
C
C  ***************************************************************
         JVAL=J
          KVAL= I
   207    NUM = NUM + 1
          IF(NN.EQ.1) GO TO 31
         IF((IADD(IC,NN)-IADD(IC,1)).EQ.JDEL*JLIM*N2) GO TO 20
    31    WRITE(6,115) NUM,IC,NN,ISET(IC,NN),IVAL,JVAL,KVAL
         IF((ITYPE.EQ.1).OR.(ITYPE.EQ.3)) GO TO 20
         IF((IC+1).EQ.ICLIM) GO TO 30
         IF(K.EQ.KLIM) GO TO 20
    30   CONTINUE
    20   CONTINUE
    10   CONTINUE
   100   FORMAT(2X,12I5)
   111   FORMAT(5X,' ITYPE',,'   IA0 ',,'MEMCD',,' #PES ',,'   IFIRST',,
     C'   ISECOND',,'   ITHIRD',,'   K0 ',,'   J0 ',,//5X,4I5,1I7,1I9,3I7//)
```

```
112   FORMAT(5X,'   IDEL',',   JDEL',',   KDEL',',  ICLIM',',   JLIM',',
     C'   ILIM'/5X,6I6/)
113   FORMAT(5X,'CYCLE',','SUBITER',',    J   ',',    K'/5X,4I6)
114   FORMAT(5X,'   NUM',',    CYCLE',',   SUBITER',',   OFFSET',',   IVAL',',
     C '   JVAL',',   KVAL')
115   FORMAT(5X,2I5,1I9,1I7,3X,3I6)
80      CONTINUE
        END
```

```
      DOPARALLEL J=1, JLIM; K=1, KLIM
      DOO 1 I=1, ILIM
      S1 = A(I,K,J) * A(K,I,J)
    1 Continue
      ENDDO: ENDDO
```

it is obviously required that for a given J,K pair that a specific processing element must receive both of them. If one considers the previous example and determines the assigned processing element for

Type I A(3,2,5) PE# 3
Type II A(2,3,5) PE#1

But this is wrong. Both of these accesses must go to the same processing element. The solution to this apparent dilema is to expand the array size at compile time by "squaring" it if one of these type accesses occurs, anywhere in the program, i.e. given

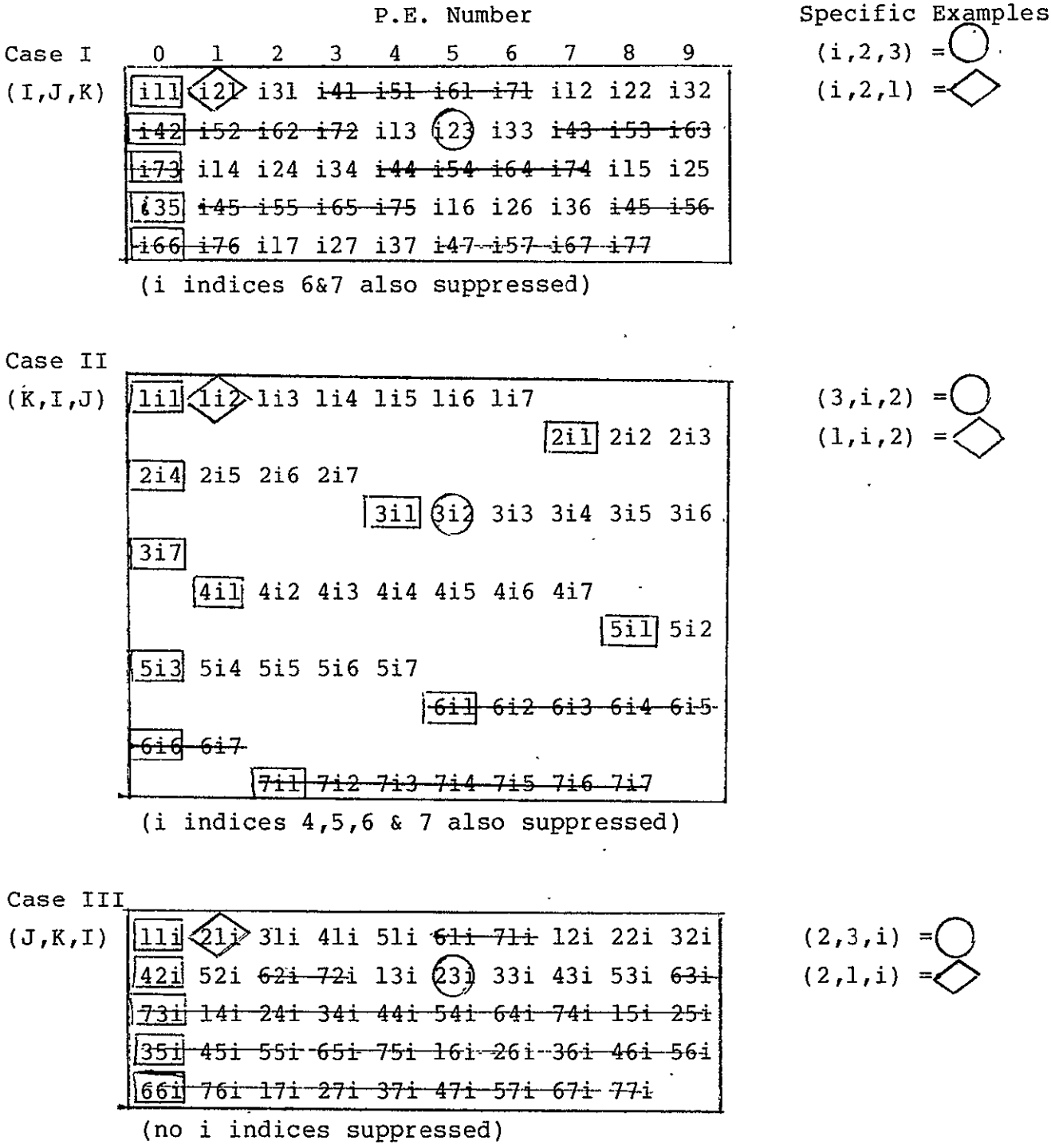
the array A(5,3,7) with extents 5,3,7

one expands it to square by increasing all extents to the largest one, i.e., 7 and accessing the array as though it were of size A (7,7,7).

This is demonstrated in detail in Figure 8A&B for all 6 accessing patterns. The I index, the innermost, is not iterated for each cycle. As is obvious one obtains the correct J,K pair in each processing element as is required. The appendix contains the examples listed below.

IAØ	Mem Mod	#PEs	IFIRST	ISECOND	ITHIRD
19	11	10	3	3	3
19	11	10	5	5	5
27	13	11	6	6	6
19	11	10	7	7	7

Extended Accessing
of Array A(5,3,7)*



*I index indicated by i, assuming iteration
deleted elements indicate null fetches

Figure 8A

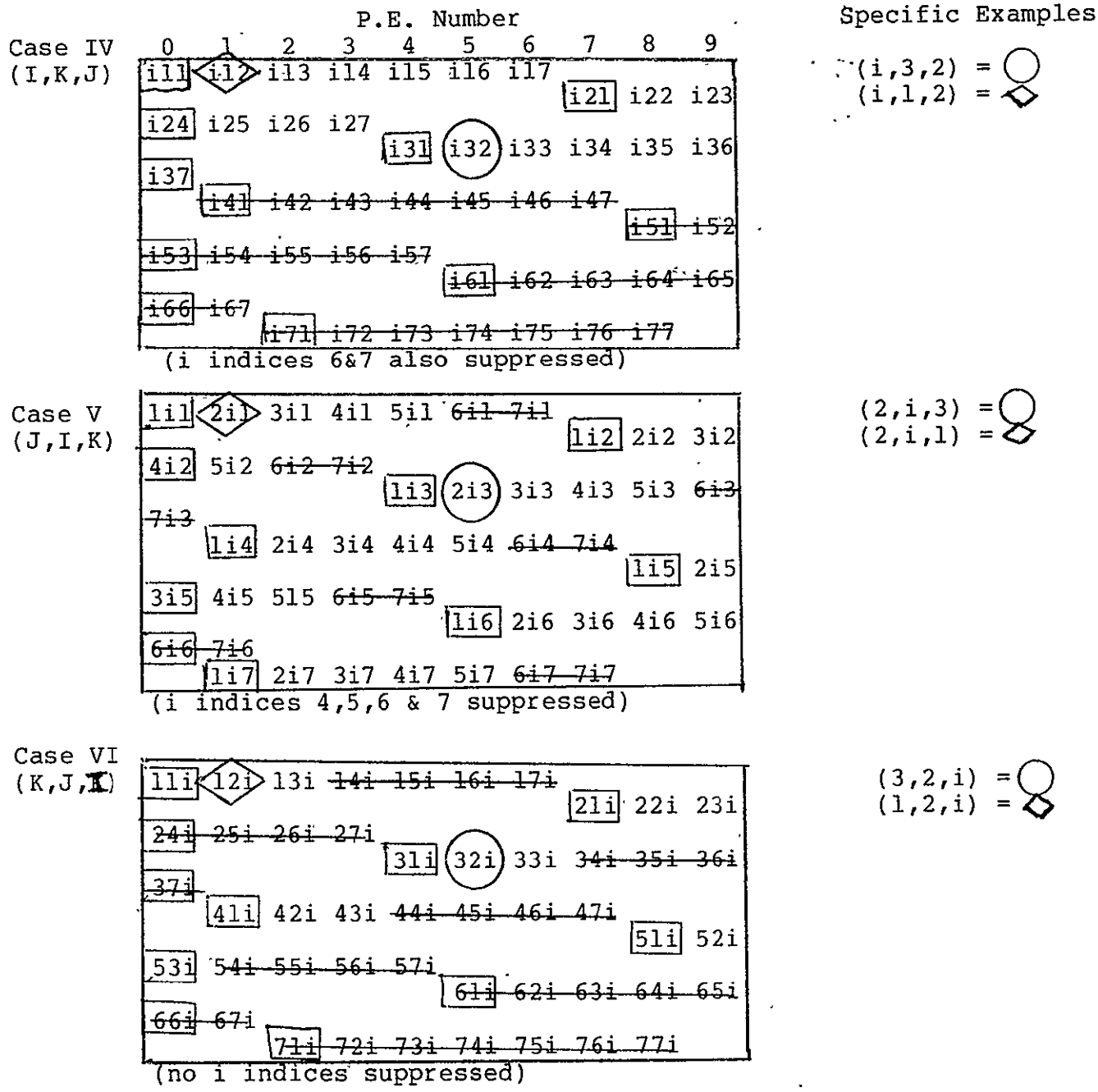


Figure 8B

3. Optimization of accessing for Single Access Type

If a single type of access occurs within a DOPARALLEL construct and is one of the less favorable ones then the compiler will reverse the order of the DOPARALLEL construct. Case I and III are already optional. Case IV and VI would be inverted, i.e., the construct would be DOPARALLEL K=1, KLIM; J=1, JLIM.

Cases III and V would reamin as written with a warning to the user.

Appendix A
Normal Accessing

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
1	19	11	10	5	3	7	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	5	15	3	3	

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	K AL
1	1	1	8	1	1	
2	1	1	9		1	1
3	1	1	10		1	1
4	1	1	0	4	1	
5	1	1	1	5	1	1
6	2	1	3	1		4
7	2	1	4	2	2	4
8	2	1	5	3	2	4
9	2	1	6	4	2	4
10	2	1	7	5	2	4
11	3	1	9	1	3	7
12	3	1	10	2	3	7
13	3	1	0	3	3	7
14	3	1	1	4	3	7
15	3	1	2	5	3	7

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
2	19	11	10	5	3	7	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
5	15	1	4	7	3

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	3	2	1	1
3	1	1	2	1	2	1
4	1	2	8	2	2	1
5	1	1	7	1	3	1
6	1	2	2	2	3	1
7	2	1	10	2	1	4
8	2	2	5	3	1	1
9	2	1	4	2	2	4
10	2	2	10	3	2	1
11	2	1	9	2	3	4
12	2	2	4	3	3	1
13	3	1	1	3	1	7
14	3	2	7	4	1	1
15	3	3	2	5	1	1
16	3	1	6	3	2	7
17	3	2	1	4	2	1
18	3	3	7	5	2	1
19	3	1	0	3	3	7
20	3	2	6	4	3	1
21	3	3	1	5	3	1
22	4	1	9	5	1	3
23	4	1	3	5	2	3
24	4	1	8	5	3	3

ORIGINAL PAGE IS
OF POOR QUALITY

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
3	19	11	10	5	3	7	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
15	1	5	2		7

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	K AL
1	1	1	8	1	1	
2	1	1	1	1	1	
3	1	1	5	1	1	
4	1	1	9	1	1	
5	1	1	2	1	1	
6	1	1	6	1	1	6
7	1	1	10	1	1	7
8	2	1	7	1	2	1
9	2	1	0	1	3	2
10	2	1	4	1	3	3
11	2	1	8	1	3	4
12	2	1	1	1	3	5
13	2	1	5	1	3	6
14	2	1	9	1	3	7

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
4	19	11	10	5	3	7	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	15	5	3	7	5

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	7	1	1	1
3	1	1	9	2	1	1
4	1	2	8	2	1	1
5	1	1	10	3	1	1
6	1	2	9	3	1	1
7	1	1	0	4	1	1
8	1	2	10	4	1	1
9	1	1	1	5	1	1
10	1	2	0	5	1	1
11	2	1	3	1	4	2
12	2	2	2	1	4	2
13	2	1	4	2	4	2
14	2	2	3	2	4	2
15	2	1	5	3	4	2
16	2	2	4	3	4	2
17	2	1	6	4	4	2
18	2	2	5	4	4	2
19	2	1	7	5	4	2
20	2	2	6	5	4	2
21	3	1	9	1	7	3
22	3	1	10	2	7	3
23	3	1	0	3	7	3
24	3	1	1	4	7	3
25	3	1	2	5	7	3

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
5	19	11	10	5	3	7	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
5	1	15	4	5	3

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	7	1	1	1
3	1	1	2	1	1	1
4	1	2	1	1	2	1
5	1	1	7	1	2	1
6	1	2	6	1	3	1
7	2	1	5	1	1	3
8	2	2	4	1	1	4
9	2	1	10	1	2	3
10	2	2	9	1	2	4
11	2	1	4	1	3	3
12	2	2	3	1	3	4
13	3	1	2	1	1	5
14	3	2	1	1	1	6
15	3	1	7	1	2	5
16	3	2	6	1	2	6
17	3	1	1	1	3	5
18	3	2	0	1	3	6
19	4	1	10	1	1	7
20	4	1	4	1	2	7
21	4	1	9	1	3	7

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
6	19	11	10	5	3	7	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
15	5	1	2	3	7

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	5	2	1	1
3	1	3	2	3	1	1
4	1	4	10	4	1	1
5	1	1	1	1	1	2
6	1	2	9	2	1	2
7	1	3	6	3	1	2
8	1	4	3	4	1	2
9	1	1	5	1	1	3
10	1	2	2	2	1	3
11	1	3	10	3	1	3
12	1	4	7	4	1	3
13	1	1	9	1	1	4
14	1	2	6	2	1	4
15	1	3	3	3	1	4
16	1	4	0	4	1	4
17	1	1	2	1	1	5
18	1	2	10	2	1	5
19	1	3	7	3	1	5
20	1	4	4	4	1	5
21	1	1	6	1	1	6
22	1	2	3	2	1	6
23	1	3	0	3	1	6
24	1	4	8	4	1	6
25	1	1	10	1	1	7
26	1	2	7	2	1	7
27	1	3	4	3	1	7
28	1	4	1	4	1	7
29	2	1	5	4	2	1
30	2	2	2	5	1	1
31	2	1	9	4	2	2
32	2	2	6	5	1	2
33	2	1	2	4	2	3
34	2	2	10	5	1	3
35	2	1	6	4	2	4
36	2	2	3	5	1	4
37	2	1	10	4	2	5
38	2	2	7	5	1	5
39	2	1	3	4	2	5
40	2	2	0	5	1	6
41	2	1	7	4	2	7
42	2	2	4	5	1	7

ITYPE	IAO	MEMCD	#PES	IFIRST	ISECCNC	ITHIRC	KO	JO
1	19	11	10	9	5	6	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	9	45	3	5	9

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	1	9	2	1	1
3	1	1	10	3	1	1
4	1	1	0	4	1	1
5	1	1	1	5	1	1
6	1	1	2	6	1	1
7	1	1	3	7	1	1
8	1	1	4	8	1	1
9	1	1	5	9	1	1
10	2	1	10	1	1	1
11	2	1	0	2	1	1
12	2	1	1	3	1	1
13	2	1	2	4	1	1
14	2	1	3	5	1	1
15	2	1	4	6	1	1
16	2	1	5	7	1	1
17	2	1	6	8	1	1
18	2	1	7	9	1	1
19	3	1	1	1	1	5
20	3	1	2	2	1	5
21	3	1	3	3	1	5
22	3	1	4	4	1	5
23	3	1	5	5	1	5
24	3	1	6	6	1	5
25	3	1	7	7	1	5
26	3	1	8	8	1	5
27	3	1	9	9	1	5

ITYPE	IAO	MEMCD	#PES	IFIRST	ISECOND	ITHIRD	KO	JO
2	19	11	10	9	5	6	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
9	45	1	6	6	5

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	3	2	1	1
3	1	1	6	1	2	1
4	1	2	1	2	2	1
5	1	1	4	1	3	1
6	1	2	10	2	3	1
7	1	1	2	1	4	1
8	1	2	8	2	4	1
9	1	1	0	1	5	1
10	1	2	6	2	5	1
11	2	1	2	2	1	1
12	2	2	8	2	1	1
13	2	1	3	1	1	1
14	2	2	0	2	2	1
15	2	1	6	3	2	1
16	2	2	1	4	2	1
17	2	1	9	2	3	1
18	2	2	4	3	3	1
19	2	1	10	4	3	1
20	2	2	7	2	4	1
21	2	1	2	3	4	1
22	2	2	8	4	4	1
23	2	1	5	2	5	1
24	2	2	0	3	5	1
25	2	1	6	4	5	1
26	3	1	2	4	1	1
27	3	2	8	5	1	1
28	3	1	0	4	2	1
29	3	2	6	5	2	1
30	3	1	9	4	3	1
31	3	2	4	5	3	1
32	3	1	7	4	4	1
33	3	2	2	5	4	1
34	3	1	5	4	5	1
35	3	2	0	5	5	1
36	4	1	2	6	1	1
37	4	2	8	7	1	1
38	4	1	0	6	2	1
39	4	2	6	7	2	1
40	4	1	9	6	3	1
41	4	2	4	7	3	1
42	4	1	7	6	4	1
43	4	2	2	7	4	1
44	4	1	5	6	5	1
45	4	2	0	7	5	1
46	5	1	7	7	1	1
47	5	2	2	8	1	1
48	5	1	8	9	1	1
49	5	2	5	7	2	1
50	5	1	0	8	2	1
51	5	2	6	9	2	1
52	5	1	3	7	3	1
53	5	2	9	8	3	1
54	5	1	4	9	3	1
55	5	2	1	7	4	1
56	5	1	7	8	4	1
57	5	2	2	9	4	1
58	5	1	10	7	5	1
59	5	2	5	8	5	1
60	5	1	0	9	5	1
61	6	1	7	9	1	2
62	6	1	5	9	2	2
63	6	1	3	9	3	2
64	6	1	1	9	4	2
65	6	1	10	9	5	2

ITYPE	IAO	MEMCD	#PES	IFIRST	ISECCND	ITHIRD	KO	JO
3	19	11	10	9	5	6	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
45	1	9	5	9	6

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	1	9	1	1	2
3	1	1	10	1	1	3
4	1	1	0	1	1	4
5	1	1	1	1	1	5
6	1	1	2	1	1	6
7	2	1	7	2	2	1
8	2	1	8	2	2	2
9	2	1	9	2	2	3
10	2	1	10	2	2	4
11	2	1	0	2	2	5
12	2	1	1	2	2	6
13	3	1	6	3	3	1
14	3	1	7	3	3	2
15	3	1	8	3	3	3
16	3	1	9	3	3	4
17	3	1	10	3	3	5
18	3	1	0	3	3	6
19	4	1	5	4	4	1
20	4	1	6	4	4	2
21	4	1	7	4	4	3
22	4	1	8	4	4	4
23	4	1	9	4	4	5
24	4	1	10	4	4	6
25	5	1	4	5	5	1
26	5	1	5	5	5	2
27	5	1	6	5	5	3
28	5	1	7	5	5	4
29	5	1	8	5	5	5
30	5	1	9	5	5	6

ITYPE	IAO	MEMCD	#PES	IFIRST	ISECCND	ITHIRD	KO	JO
4	19	11	10	9	5	6	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	45	9	3	6	9

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	0	1	1	1
3	1	1	9	2	1	1
4	1	2	1	2	1	1
5	1	1	10	3	1	1
6	1	2	2	3	1	1
7	1	1	0	4	1	1
8	1	2	3	4	1	1
9	1	1	1	5	1	1
10	1	2	4	5	1	1
11	1	1	2	6	1	1
12	1	2	5	6	1	1
13	1	1	3	7	1	1
14	1	2	6	7	1	1
15	1	1	4	8	1	1
16	1	2	7	8	1	1
17	1	1	5	9	1	1
18	1	2	8	9	1	1
19	2	1	10	1	5	2
20	2	2	2	1	5	2
21	2	1	5	1	5	2
22	2	2	0	2	5	2
23	2	1	3	2	5	2
24	2	2	6	2	5	2
25	2	1	1	3	5	2
26	2	2	4	3	5	2
27	2	1	7	3	5	2
28	2	2	2	4	5	2
29	2	1	5	4	5	2
30	2	2	8	4	5	2
31	2	1	3	5	5	2
32	2	2	6	5	5	2
33	2	1	9	5	5	2
34	2	2	4	6	5	2
35	2	1	7	6	5	2
36	2	2	10	6	5	2
37	2	1	5	7	5	2
38	2	2	8	7	5	2
39	2	1	6	8	5	2
40	2	2	9	8	5	2
41	2	1	1	8	5	2
42	2	2	7	9	5	2
43	2	1	10	9	5	2
44	2	2	2	9	5	2
45	3	1	4	1	3	4
46	3	2	7	1	3	4
47	3	1	5	2	3	4
48	3	2	8	2	3	4
49	3	1	6	3	3	4
50	3	2	9	3	3	4
51	3	1	7	4	3	4
52	3	2	10	4	3	4
53	3	1	8	5	3	4
54	3	2	0	5	3	4
55	3	1	9	6	3	4
56	3	2	1	6	3	4
57	3	1	10	7	3	4
58	3	2	2	7	3	4
59	3	1	0	8	3	4
60	3	2	3	8	3	4
61	3	1	1	9	3	4
62	3	2	4	9	3	4
63	3					

ORIGINAL PAGE IS
OF POOR QUALITY

ITYPE	IAD	MEMCD	#PES	IFIRST	ISECCNC	ITHIRD	KO	JO
5	19	11	10	9	5	6	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
9	1	45	6	9	5

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	0	1	1	2
3	1	1	6	1	2	1
4	1	2	9	1	2	2
5	1	1	4	1	3	1
6	1	2	7	1	3	2
7	1	1	2	1	4	1
8	1	2	5	1	4	2
9	1	1	0	1	5	1
10	1	2	3	1	5	2
11	2	1	10	2	1	1
12	2	2	2	1	1	2
13	2	1	8	2	2	1
14	2	2	0	1	2	2
15	2	1	6	2	3	1
16	2	2	9	1	3	2
17	2	1	4	2	4	1
18	2	2	7	1	4	2
19	2	1	2	2	5	1
20	2	2	5	1	5	2
21	3	1	1	3	1	1
22	3	2	4	1	1	2
23	3	1	10	3	2	1
24	3	2	2	1	2	2
25	3	1	8	2	3	1
26	3	2	0	1	3	2
27	3	1	6	2	4	1
28	3	2	9	1	4	2
29	3	1	4	2	5	1
30	3	2	7	1	5	2
31	4	1	3	4	1	1
32	4	2	6	1	1	2
33	4	1	1	4	2	1
34	4	2	4	1	2	2
35	4	1	10	4	3	1
36	4	2	2	1	3	2
37	4	1	8	4	4	1
38	4	2	0	1	4	2
39	4	1	6	4	5	1
40	4	2	9	1	5	2
41	5	1	5	5	1	1
42	5	2	8	1	1	2
43	5	1	3	5	2	1
44	5	2	6	1	2	2
45	5	1	1	5	3	1
46	5	2	4	1	3	2
47	5	1	10	5	4	1
48	5	2	2	1	4	2
49	5	1	8	5	5	1
50	5	2	0	1	5	2
51	6	1	7	6	1	1
52	6	1	5	6	2	1
53	6	1	3	6	3	1
54	6	1	1	6	4	1
55	6	1	10	6	5	1

ITYPE	IAO	MEMCO	#PES	IFIRST	ISECCNC	ITHIRC	KO	JO
6	19	11	10	9	5	6	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
45	9	1	5	5	6

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	8	2	1	1
3	1	1	9	1	1	2
4	1	2	9	2	1	2
5	1	1	10	1	1	3
6	1	2	10	2	1	3
7	1	1	0	1	1	4
8	1	2	0	2	1	4
9	1	1	1	1	1	5
10	1	2	1	2	1	5
11	1	1	2	1	1	6
12	1	2	2	2	1	6
13	2	1	10	3	1	1
14	2	2	10	4	1	1
15	2	1	0	3	1	2
16	2	2	0	4	1	2
17	2	1	1	3	1	3
18	2	2	1	4	1	3
19	2	1	2	3	1	4
20	2	2	2	4	1	4
21	2	1	3	3	1	5
22	2	2	3	4	1	5
23	2	1	4	3	1	6
24	2	2	4	4	1	6
25	3	1	1	5	1	1
26	3	2	1	6	1	1
27	3	1	2	5	1	2
28	3	2	2	6	1	2
29	3	1	3	5	1	3
30	3	2	3	6	1	3
31	3	1	4	5	1	4
32	3	2	4	6	1	4
33	3	1	5	5	1	5
34	3	2	5	6	1	5
35	3	1	6	5	1	6
36	3	2	6	6	1	6
37	4	1	3	7	1	1
38	4	2	3	8	1	1
39	4	1	4	7	1	2
40	4	2	4	8	1	2
41	4	1	5	7	1	3
42	4	2	5	8	1	3
43	4	1	6	7	1	4
44	4	2	6	8	1	4
45	4	1	7	7	1	5
46	4	2	7	8	1	5
47	4	1	8	7	1	6
48	4	2	8	8	1	6
49	5	1	5	9	1	1
50	5	1	6	9	1	2
51	5	1	7	9	1	3
52	5	1	8	9	1	4
53	5	1	9	9	1	5
54	5	1	10	9	1	6

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
1	19	11	10	6	2	8	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	6	12	2	2	6

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	K AL
1	1	1	8	1	1	1
2	1	1	9	2	1	1
3	1	1	10	3	1	1
4	1	1	0	4	1	1
5	1	1	1	5	1	1
6	1	1	2	6	1	1
7	2	1	2	1	1	5
8	2	1	3	2	1	6
9	2	1	4	3	1	6
10	2	1	5	4	1	6
11	2	1	6	5	1	6
12	2	1	7	6	1	6

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
2	19	11	10	6	2	8	1	1

ICEL	JDEL	KDEL	ICLIM	JLIM	ILIM
6	12	1	5	8	2

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	1	2	1	1
3	1	1	3	1	2	1
4	1	2	7	2	2	1
5	2	1	0	2	1	3
6	2	2	4	3	1	1
7	2	1	6	2	2	3
8	2	2	10	3	2	1
9	3	1	3	3	1	5
10	3	2	7	4	1	1
11	3	1	9	3	2	5
12	3	2	2	4	2	1
13	4	1	6	4	1	7
14	4	2	10	5	1	1
15	4	1	1	4	2	7
16	4	2	5	5	2	1
17	5	1	2	6	1	1
18	5	1	8	6	2	1

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
3	19	11	10	6	2	8	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
12	1	6	2	6	3

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	K AL
1	1	1	8	1	1	1
2	1	1	9	1	1	2
3	1	1	10	1	1	3
4	1	1	0	1	1	4
5	1	1	1	1	1	5
6	1	1	2	1	1	6
7	1	1	3	1	1	7
8	1	1	4	1	1	8
9	2	1	7	5	2	1
10	2	1	8	5	2	2
11	2	1	9	5	2	3
12	2	1	10	5	2	4
13	2	1	0	5	2	5
14	2	1	1	5	2	6
15	2	1	2	5	2	7
16	2	1	3	5	2	8

ORIGINAL PAGE IS
OF POOR QUALITY

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
4	19	11	1C	6	2	8	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	12	6	2	8	6

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	6	1	1	1
3	1	1	9	2	1	1
4	1	2	7	2	1	1
5	1	1	10	3	1	1
6	1	2	8	3	1	1
7	1	1	0	4	1	1
8	1	2	9	4	1	1
9	1	1	1	5	1	1
10	1	2	10	5	1	1
11	1	1	2	6	1	1
12	1	2	0	6	1	1
13	2	1	5	1	3	2
14	2	1	6	2	3	2
15	2	1	7	3	3	2
16	2	1	8	4	3	2
17	2	1	9	5	3	2
18	2	1	10	6	3	2

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
5	19	11	1C	6	2	8	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
6	1	12	5	6	2

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	3	1	1	2
3	1	1	3	1	2	1
4	1	2	9	1	2	2
5	2	1	2	5	1	2
6	2	2	8	1	1	3
7	2	3	3	1	1	4
8	2	1	8	5	2	2
9	2	2	3	1	2	3
10	2	3	9	1	2	4
11	3	1	2	3	1	4
12	3	2	8	1	1	5
13	3	1	8	3	2	4
14	3	2	3	1	2	5
15	4	1	2	1	1	6
16	4	2	8	1	1	7
17	4	1	8	1	2	6
18	4	2	3	1	2	7
19	5	1	7	5	1	7
20	5	2	2	1	1	8
21	5	1	2	5	2	7
22	5	2	8	1	2	8

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
6	19	11	1C	6	2	8	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
12	6	1	2	2	8

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	8	2	1	1
3	1	3	8	3	1	1
4	1	4	8	4	1	1
5	1	5	8	5	1	1
6	1	1	9	1	1	2
7	1	2	9	2	1	2
8	1	3	9	3	1	2
9	1	4	9	4	1	2
10	1	5	9	5	1	2
11	1	1	10	1	1	3
12	1	2	10	2	1	3
13	1	3	10	3	1	3
14	1	4	10	4	1	3
15	1	5	10	5	1	3
16	1	1	0	1	1	4
17	1	2	0	2	1	4
18	1	3	0	3	1	4
19	1	4	0	4	1	4
20	1	5	0	5	1	4
21	1	1	1	1	1	5
22	1	2	1	2	1	5
23	1	3	1	3	1	5
24	1	4	1	4	1	5
25	1	5	1	5	1	5
26	1	1	2	1	1	6
27	1	2	2	2	1	6
28	1	3	2	3	1	6
29	1	4	2	4	1	6
30	1	5	2	5	1	6
31	1	1	3	1	1	7
32	1	2	3	2	1	7
33	1	3	3	3	1	7
34	1	4	3	4	1	7
35	1	5	3	5	1	7
36	1	1	4	1	1	8
37	1	2	4	2	1	8
38	1	3	4	3	1	8
39	1	4	4	4	1	8
40	1	5	4	5	1	8
41	2	1	2	6	1	1
42	2	1	3	6	1	2
43	2	1	4	6	1	3
44	2	1	5	6	1	4
45	2	1	6	6	1	5
46	2	1	7	6	1	6
47	2	1	8	6	1	7
48	2	1	9	6	1	8

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
1	27	13	11	6	2	8	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	6	12	2	2	6

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	1	1	1	1
2	1	1	2	2	1	1
3	1	1	3	3	1	1
4	1	1	4	4	1	1
5	1	1	5	5	1	1
6	1	1	6	6	1	1
7	2	1	2	1	2	6
8	2	1	3	2	2	6
9	2	1	4	3	2	6
10	2	1	5	4	2	6
11	2	1	6	5	2	6
12	2	1	7	6	2	6

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
2	27	13	11	6	2	8	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
6	12	1	5	8	2

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	1	1	1	1
2	1	2	10	1	1	1
3	1	1	7	1	1	1
4	1	2	3	2	1	1
5	2	1	12	2	1	4
6	2	2	8	3	1	1
7	2	1	5	2	2	4
8	2	2	1	3	2	1
9	3	1	10	3	1	7
10	3	2	6	4	1	1
11	3	1	2	5	1	1
12	3	1	3	3	2	7
13	3	2	12	4	2	1
14	3	1	8	5	2	1
15	4	1	4	5	1	2
16	4	2	0	6	1	1
17	4	1	10	5	2	2
18	4	2	6	6	2	1
19	5	1	2	6	1	5
20	5	1	8	6	2	5

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
3	27	13	11	6	2	8	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
12	1	6	2	6	8

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	1	1	1	1
2	1	1	0	1	1	2
3	1	1	12	1	1	3
4	1	1	11	1	1	4
5	1	1	10	1	1	5
6	1	1	9	1	1	6
7	1	1	8	1	1	7
8	1	1	7	1	1	8
9	2	1	12	6	2	1
10	2	1	11	6	2	2
11	2	1	10	6	2	3
12	2	1	9	6	2	4
13	2	1	8	6	2	5
14	2	1	7	6	2	6
15	2	1	6	6	2	7
16	2	1	5	6	2	8

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JO
4	27	13	11	6	2	8	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	12	6	2	8	

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	1	1	1	1
2	1	2	2	1	1	2
3	1	1	3	2	1	1
4	1	2	3	2	1	1
5	1	1	3	2	1	1
6	1	2	4	3	1	1
7	1	1	4	4	1	1
8	1	2	5	4	1	1
9	1	1	5	5	1	1
10	1	2	6	5	1	1
11	1	1	6	6	1	1
12	1	2	7	6	1	2
13	2	1	4	1	4	2
14	2	1	5	2	4	2
15	2	1	6	3	4	2
16	2	1	7	4	4	2
17	2	1	8	5	4	2
18	2	1	9	6	4	2

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JO
5	27	13	11	6	2	8	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
6	1	12	5	6	2

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	1	1	1	1
2	1	2	7	1	1	2
3	1	1	7	1	2	1
4	1	2	8	1	2	2
5	2	1	5	6	1	2
6	2	2	11	1	1	3
7	2	2	4	1	1	4
8	2	1	11	6	2	2
9	2	2	4	1	2	3
10	2	2	10	1	2	4
11	3	1	2	5	1	4
12	3	2	8	1	1	5
13	3	2	1	1	1	5
14	3	1	8	5	2	4
15	3	2	1	1	2	5
16	3	2	7	1	2	6
17	4	1	12	4	1	6
18	4	2	5	1	1	7
19	4	2	11	1	1	8
20	4	1	5	4	2	5
21	4	2	11	1	2	7
22	4	2	4	1	2	8
23	5	1	9	3	1	8
24	5	1	2	3	2	8

Appendix B
Extended or Squared Accessing

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KC	JC
6	27	13	11	6	2	8	1	

IDEL	JDEL	KDEL	ICLIM	JLIM	ILI
12	6	1	2	2	

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	K	AL
1	1	1	1	1	1		
2	1	2	3	2	1		1
3	1	3	5	3	1		1
4	1	4	7	4	1		
5	1	5	9	5	1		
6	1	6	11	6	1		
7	1	1	0	1	1		
8	1	2	2	2	1		
9	1	3	4	3	1		
10	1	4	6	4	1		
11	1	5	8	5	1		
12	1	6	10	6	1		
13	1	1	12	1	1		
14	1	2	1	2	1		
15	1	3	3	3	1		
16	1	4	5	4	1		
17	1	5	7	5	1		
18	1	6	9	6	1		
19	1	1	11	1	1		
20	1	2	0	2	1		
21	1	3	2	3	1		
22	1	4	4	4	1		
23	1	5	6	5	1		
24	1	6	8	6	1		
25	1	1	10	1	1		
26	1	2	12	2	1		
27	1	3	1	3	1		
28	1	4	3	4	1		
29	1	5	5	5	1		
30	1	6	7	6	1		
31	1	1	9	1	1		
32	1	2	11	2	1		
33	1	3	0	3	1		
34	1	4	2	4	1		
35	1	5	4	5	1		
36	1	6	6	6	1		
37	1	1	8	1	1		
38	1	2	10	2	1		
39	1	3	12	3	1		
40	1	4	1	4	1		
41	1	5	3	5	1		
42	1	6	5	6	1		
43	1	1	7	1	1		
44	1	2	9	2	1		
45	1	3	11	3	1		
46	1	4	0	4	1		
47	1	5	2	5	1		
48	1	6	4	6	1		
49	2	1	12	6	1		1
50	2	1	11	6	2		
51	2	1	10	6	2		
52	2	1	9	6	2		
53	2	1	8	6	2		
54	2	1	7	6	2		
55	2	1	6	6	2		
56	2	1	5	6	2		

ORIGINAL PAGE IS
OF POOR QUALITY

ITYPE	IAO	MEMCD	#PES	IFIRST	ISECCNC	ITHIRD	KO	JO
1	19	11	10	3	3	3	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	3	9	1	3	3

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	1	9	2	1	1
3	1	1	10	3	1	1

ITYPE	IAO	MEMCD	#PES	IFIRST	ISECCNC	ITHIRD	KO	JO
2	19	11	10	3	3	3	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
3	9	1	1	3	3

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	4	2	1	1
3	1	3	0	3	1	1
4	1	1	0	1	2	1
5	1	2	7	2	2	1
6	1	3	3	3	2	1
7	1	1	3	1	3	1
8	1	2	10	2	3	1
9	1	3	6	3	3	1

ITYPE	IAO	MEMCD	#PES	IFIRST	ISECCNC	ITHIRD	KO	JO
3	19	11	10	3	3	3	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
9	1	3	1	3	3

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	1	6	1	1	2
3	1	1	4	1	1	3

ITYPE	IAO	MEMCD	#PES	IFIRST	ISECCNC	ITHIRD	KO	JO
4	19	11	10	3	3	3	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	9	3	1	3	3

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	6	1	1	1
3	1	3	4	1	1	1
4	1	4	2	1	1	1
5	1	1	9	2	1	1
6	1	2	7	2	1	1
7	1	3	5	2	1	1
8	1	4	3	2	1	1
9	1	1	10	3	1	1
10	1	2	8	3	1	1
11	1	3	6	3	1	1
12	1	4	4	3	1	1

ITYPE	IAO	MEMCD	#PES	IFIRST	ISECCND	ITHIRD	KO	JO
5	19	11	10	3	3	3	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
3	1	9	1	3	3

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	3	1	1	2
3	1	3	9	1	1	3
4	1	1	0	1	2	1
5	1	2	6	1	2	2
6	1	3	1	1	2	3
7	1	1	3	1	3	1
8	1	2	9	1	3	2
9	1	3	4	1	3	3

ITYPE	IAO	MEMCD	#PES	IFIRST	ISECCND	ITHIRD	KO	JO
6	19	11	10	3	3	3	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
9	3	1	1	3	3

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	0	2	1	1
3	1	3	3	3	1	1
4	1	1	6	1	1	2
5	1	2	9	2	1	2
6	1	3	1	3	1	2
7	1	1	4	1	1	3
8	1	2	7	2	1	3
9	1	3	10	3	1	3

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
1	19	11	10	5	5	5	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	5	25	3	5	5

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	1	9	2	1	1
3	1	1	10	3	1	1
4	1	1	0	4	1	1
5	1	1	1	5	1	1
6	2	1	3	1	1	3
7	2	1	4	2	1	3
8	2	1	5	3	1	3
9	2	1	6	4	1	3
10	2	1	7	5	1	3
11	3	1	9	1	1	5
12	3	1	10	2	1	5
13	3	1	0	3	1	5
14	3	1	1	4	1	5
15	3	1	2	5	1	5

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
2	19	11	10	5	5	5	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
5	25	1	3	5	5

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	5	2	1	1
3	1	1	2	1	2	1
4	1	2	10	2	2	1
5	1	1	7	1	2	1
6	1	2	4	2	3	1
7	1	1	1	1	4	1
8	1	2	9	2	4	1
9	1	1	6	1	5	1
10	1	2	3	2	5	1
11	2	1	10	3	1	1
12	2	2	7	4	1	1
13	2	1	4	3	2	1
14	2	2	1	4	2	1
15	2	1	9	3	3	1
16	2	2	6	4	3	1
17	2	1	3	3	4	1
18	2	2	0	4	4	1
19	2	1	8	3	5	1
20	2	2	5	4	5	1
21	3	1	1	5	1	1
22	3	1	6	5	2	1
23	3	1	0	5	3	1
24	3	1	5	5	4	1
25	3	1	10	5	5	1

ORIGINAL PAGE IS
OF POOR QUALITY

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
3	19	11	10	5	5	5	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
25	1	5	3	5	5

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	1	0	1	1	2
3	1	1	3	1	1	3
4	1	1	6	1	1	4
5	1	1	9	1	1	5
6	2	1	7	1	3	1
7	2	1	10	1	3	2
8	2	1	2	1	3	3
9	2	1	5	1	3	4
10	2	1	8	1	3	5
11	3	1	6	1	5	1
12	3	1	9	1	5	2
13	3	1	1	1	5	3
14	3	1	4	1	5	4
15	3	1	7	1	5	5

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
4	19	11	10	5	5	5	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	25	5	3	5	

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	9	1	1	2
3	1	1	9	2	1	1
4	1	2	10	2	1	1
5	1	1	10	3	1	1
6	1	2	0	3	1	1
7	1	1	0	4	1	1
8	1	2	1	4	1	1
9	1	1	1	5	1	1
10	1	2	2	5	1	2
11	2	1	7	1	1	2
12	2	2	8	1	1	3
13	2	1	8	2	1	3
14	2	2	9	2	1	3
15	2	1	9	3	1	3
16	2	2	10	3	1	3
17	2	1	10	4	1	3
18	2	2	0	4	1	3
19	2	1	0	5	1	3
20	2	2	1	5	1	3
21	3	1	6	1	1	5
22	3	1	7	2	1	5
23	3	1	8	3	1	5
24	3	1	9	4	1	5
25	3	1	10	5	1	5

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
5	19	11	1C	5	5	5	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
5	1	25	3	5	5

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	6	1	1	2
3	1	1	2	1	2	1
4	1	2	0	1	2	2
5	1	1	7	1	3	1
6	1	2	5	1	3	2
7	1	1	1	1	4	1
8	1	2	10	1	4	2
9	1	1	6	1	5	1
10	1	2	4	1	5	2
11	2	1	3	1	1	3
12	2	2	1	1	1	4
13	2	1	8	1	2	3
14	2	2	6	1	2	4
15	2	1	2	1	3	3
16	2	2	0	1	3	4
17	2	1	7	1	4	3
18	2	2	5	1	4	4
19	2	1	1	1	5	3
20	2	2	10	1	5	4
21	3	1	9	1	1	5
22	3	1	3	1	2	5
23	3	1	8	1	3	5
24	3	1	2	1	4	5
25	3	1	7	1	5	5

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
6	19	11	1C	5	5	5	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
25	5	1	3	5	5

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	6	1	1	1
3	1	1	0	1	1	2
4	1	2	9	2	1	2
5	1	1	3	1	1	3
6	1	2	1	2	1	3
7	1	1	6	1	1	4
8	1	2	4	2	1	4
9	1	1	9	1	1	5
10	1	2	7	2	1	5
11	2	1	10	3	1	1
12	2	2	8	4	1	1
13	2	1	2	3	1	2
14	2	2	0	4	1	2
15	2	1	5	3	1	3
16	2	2	3	4	1	3
17	2	1	2	3	1	4
18	2	2	6	4	1	4
19	2	1	0	3	1	5
20	2	2	9	4	1	5
21	3	1	1	5	1	1
22	3	2	4	5	1	2
23	3	1	7	5	1	3
24	3	1	10	5	1	4
25	3	1	2	5	1	5

ORIGINAL PAGE IS
OF POOR QUALITY

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
1	27	13	11	6	6	6	1	1

ICEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	6	36	4	6	6

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	1	1	1	1
2	1	1	2	2	1	1
3	1	1	3	3	1	1
4	1	1	4	4	1	1
5	1	1	5	5	1	1
6	1	1	6	6	1	1
7	2	1	2	1	6	2
8	2	1	3	2	6	2
9	2	1	4	3	6	2
10	2	1	5	4	6	2
11	2	1	6	5	6	2
12	2	1	7	6	6	2
13	3	1	3	1	5	4
14	3	1	4	2	5	4
15	3	1	5	3	5	4
16	3	1	6	4	5	4
17	3	1	7	5	5	4
18	3	1	8	6	5	4
19	4	1	4	1	4	6
20	4	1	5	2	4	6
21	4	1	6	3	4	6
22	4	1	7	4	4	6
23	4	1	8	5	4	6
24	4	1	9	6	4	6

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
2	27	13	11	6	6	6	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
6	36	1	4	6	

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	1	1	1	1
2	1	2	7		1	1
3	1	1	7	1	2	1
4	1	2	0	2	2	1
5	1	1	0	1	3	1
6	1	2	6	2	3	1
7	1	1	6	1	4	1
8	1	2	12	2	4	1
9	1	1	12	1	5	1
10	1	2	5	2	5	1
11	1	1	5	1	6	1
12	1	2	11	2	6	1
13	2	1	0	2	1	6
14	2	2	6	3	1	1
15	2	1	12	4	1	1
16	2	1	6	2	2	6
17	2	2	12	3	2	1
18	2	2	5	4	2	1
19	2	1	12	2	3	6
20	2	2	5	3	3	1
21	2	3	11	4	3	1
22	2	1	5	2	4	6
23	2	2	11	3	4	1
24	2	3	4	4	4	1
25	2	1	11	2	5	6
26	2	2	4	3	5	1
27	2	1	10	4	5	1
28	2	1	4	2	6	6
29	2	2	10	3	6	1
30	2	2	3	4	6	1
31	3	1	5	4	1	5
32	3	2	11	5	1	1
33	3	2	4	6	1	1
34	3	1	11	4	2	5
35	3	2	4	5	2	1
36	3	2	10	6	2	1
37	3	1	4	4	3	5
38	3	2	10	5	3	1
39	3	3	3	6	3	1
40	3	1	10	4	4	5
41	3	2	3	5	4	1
42	3	2	9	6	4	1
43	3	1	3	4	5	5
44	3	2	9	5	5	1
45	3	3	2	6	5	1
46	3	1	9	4	6	5
47	3	2	2	5	6	1
48	3	2	8	6	6	1
49	4	1	10	6	1	4
50	4	1	3	6	2	4
51	4	1	9	6	3	4
52	4	1	2	6	4	4
53	4	1	8	6	5	4
54	4	1	1	6	6	4

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
3	27	13	11	6	6	6	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
36	1	6	4	6	6

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	1	1	1	1
2	1	1	11	1	1	2
3	1	1	8	1	1	3
4	1	1	5	1	1	4
5	1	1	2	1	1	5
6	1	1	12	1	1	6
7	2	1	12	6	2	1
8	2	1	9	6	2	2
9	2	1	6	6	2	3
10	2	1	3	6	2	4
11	2	1	0	6	2	5
12	2	1	10	6	2	6
13	3	1	10	5	4	1
14	3	1	7	5	4	2
15	3	1	4	5	4	3
16	3	1	1	5	4	4
17	3	1	11	5	4	5
18	3	1	8	5	4	6
19	4	1	8	4	6	1
20	4	1	5	4	6	2
21	4	1	2	4	6	3
22	4	1	12	4	6	4
23	4	1	9	4	6	5
24	4	1	6	4	6	6

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JO
4	27	13	11	6	6	6	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	36	6	4	6	6

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	1	1	1	1
2	1	2	12	1	1	1
3	1	3	2	2	1	1
4	1	2	0	2	1	1
5	1	1	3	3	1	1
6	1	2	1	3	1	1
7	1	1	4	4	1	1
8	1	2	2	4	1	1
9	1	1	5	5	1	1
10	1	2	3	5	1	1
11	1	1	6	6	1	1
12	1	2	4	6	1	1
13	2	1	5	1	6	2
14	2	2	3	1	6	2
15	2	3	1	1	6	2
16	2	1	6	2	6	2
17	2	2	4	2	6	2
18	2	3	2	2	6	2
19	2	1	7	3	6	2
20	2	2	5	3	6	2
21	2	3	3	3	6	2
22	2	1	8	4	6	2
23	2	2	6	4	6	2
24	2	3	4	4	6	2
25	2	1	9	5	6	2
26	2	2	7	5	6	2
27	2	3	5	5	6	2
28	2	1	10	6	6	2
29	2	2	8	6	6	2
30	2	3	6	6	6	2
31	3	1	7	1	5	4
32	3	2	5	1	5	4
33	3	3	3	1	5	4
34	3	1	8	2	5	4
35	3	2	6	2	5	4
36	3	3	4	2	5	4
37	3	1	9	3	5	4
38	3	2	7	3	5	4
39	3	3	5	3	5	4
40	3	1	10	4	5	4
41	3	2	8	4	5	4
42	3	3	6	4	5	4
43	3	1	11	5	5	4
44	3	2	9	5	5	4
45	3	3	7	5	5	4
46	3	1	12	6	5	4
47	3	2	10	6	5	4
48	3	3	8	6	5	4
49	4	1	9	1	4	6
50	4	1	10	2	4	6
51	4	1	11	3	4	6
52	4	1	12	4	4	6
53	4	1	0	5	4	6
54	4	1	1	6	4	6

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
5	27	13	11	6	6	6	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
6	1	36	4	6	6

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	1	1	1	1
2	1	2	5	1	1	2
3	1	1	7	1	2	1
4	1	2	11	1	2	2
5	1	1	0	1	3	1
6	1	2	4	1	3	2
7	1	1	6	1	4	1
8	1	2	10	1	4	2
9	1	1	12	1	5	1
10	1	2	3	1	5	2
11	1	1	5	1	6	1
12	1	2	9	1	6	2
13	2	1	3	6	1	2
14	2	2	7	1	1	3
15	2	3	11	1	1	4
16	2	1	9	6	2	2
17	2	2	0	1	2	3
18	2	3	4	1	2	4
19	2	1	2	6	3	2
20	2	2	6	1	3	3
21	2	3	10	1	3	4
22	2	1	8	6	4	2
23	2	2	12	1	4	3
24	2	3	3	1	4	4
25	2	1	1	6	5	2
26	2	2	5	1	5	3
27	2	3	9	1	5	4
28	2	1	7	6	6	2
29	2	2	11	1	6	3
30	2	3	2	1	6	4
31	3	1	9	5	1	4
32	3	2	0	1	1	5
33	3	3	4	1	1	6
34	3	1	2	5	2	4
35	3	2	6	1	2	5
36	3	3	10	1	2	6
37	3	1	8	5	3	4
38	3	2	12	1	3	5
39	3	3	3	1	3	6
40	3	1	1	5	4	4
41	3	2	5	1	4	5
42	3	3	9	1	4	6
43	3	1	7	5	5	4
44	3	2	11	1	5	5
45	3	3	2	1	5	6
46	3	1	0	5	6	4
47	3	2	4	1	6	5
48	3	3	8	1	6	6
49	4	1	2	4	1	5
50	4	1	8	4	2	6
51	4	1	1	4	3	6
52	4	1	7	4	4	6
53	4	1	0	4	5	6
54	4	1	6	4	6	6

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
6	27	13	11	6	6	6	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
36	6	1	4	6	6

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	K AL
1	1	1	1	1	1	1
2	1	2	5	1	1	1
3	1	1	11	1	1	2
4	1	2	2	2	1	2
5	1	1	8	1	1	3
6	1	2	12	2	1	3
7	1	1	5	1	1	4
8	1	2	9	2	1	4
9	1	1	2	1	1	5
10	1	2	6	2	1	5
11	1	1	12	1	1	6
12	1	2	3	2	1	6
13	2	1	6	2	6	1
14	2	2	10	3	1	1
15	2	3	1	4	1	1
16	2	1	3	2	6	2
17	2	2	7	3	1	2
18	2	3	11	4	1	2
19	2	1	0	2	6	3
20	2	2	4	3	1	3
21	2	3	8	4	1	3
22	2	1	10	2	6	4
23	2	2	1	3	1	4
24	2	3	5	4	1	4
25	2	1	7	2	6	5
26	2	2	11	3	1	5
27	2	3	2	4	1	5
28	2	1	4	2	6	6
29	2	2	8	3	1	6
30	2	3	12	4	1	6
31	3	1	2	4	5	1
32	3	2	6	5	1	1
33	3	3	10	6	1	1
34	3	1	12	4	5	2
35	3	2	3	5	1	2
36	3	3	7	6	1	2
37	3	1	9	4	5	3
38	3	2	0	5	1	3
39	3	3	4	6	1	3
40	3	1	6	4	5	4
41	3	2	10	5	1	4
42	3	3	1	6	1	4
43	3	1	3	4	5	5
44	3	2	7	5	1	5
45	3	3	11	6	1	5
46	3	1	0	4	5	6
47	3	2	4	5	1	6
48	3	3	8	6	1	6
49	4	1	11	6	4	1
50	4	1	8	6	4	2
51	4	1	5	6	4	3
52	4	1	2	6	4	4
53	4	1	12	6	4	5
54	4	1	9	6	4	6

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
1	19	11	10	7	7	7	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	7	49	5	7	7

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	1	9	2	1	1
3	1	1	10	3	1	1
4	1	1	0	4	1	1
5	1	1	1	5	1	1
6	1	1	2	6	1	1
7	1	1	3	7	1	1
8	2	1	1	1	4	2
9	2	1	2	2	4	2
10	2	1	3	3	4	2
11	2	1	4	4	4	2
12	2	1	5	5	4	2
13	2	1	6	6	4	2
14	2	1	7	7	4	2
15	3	1	5	1	7	3
16	3	1	6	2	7	3
17	3	1	7	3	7	3
18	3	1	8	4	7	3
19	3	1	9	5	7	3
20	3	1	10	6	7	3
21	3	1	0	7	7	3
22	4	1	9	1	3	5
23	4	1	10	2	3	5
24	4	1	0	3	3	5
25	4	1	1	4	3	5
26	4	1	2	5	3	5
27	4	1	3	6	3	5
28	4	1	4	7	3	5
29	5	1	2	1	6	6
30	5	1	3	2	6	6
31	5	1	4	3	6	6
32	5	1	5	4	6	6
33	5	1	6	5	6	6
34	5	1	7	6	6	6
35	5	1	8	7	6	6

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
2	19	11	10	7	7	7	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
7	49	1	5	7	7

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	7	2	1	1
3	1	1	4	1	2	1
4	1	2	3	2	2	1
5	1	1	0	1	3	1
6	1	2	10	2	3	1
7	1	1	7	1	4	1
8	1	2	6	2	4	1
9	1	1	3	1	5	1
10	1	2	2	2	5	1
11	1	1	10	1	6	1
12	1	2	9	2	6	1
13	1	1	6	1	7	1
14	1	2	5	2	7	1
15	2	1	2	2	1	4
16	2	2	1	3	1	1
17	2	1	9	2	2	4
18	2	2	8	3	2	1
19	2	1	5	2	3	4
20	2	2	4	3	3	1
21	2	1	1	2	4	4
22	2	2	0	3	4	1
23	2	1	8	2	5	4
24	2	2	7	3	5	1
25	2	1	4	2	6	4
26	2	2	3	3	6	1

27	2	1	0	2	7	4
28	2	2	10	3	7	1
29	3	1	7	3	1	7
30	3	2	6	4	1	1
31	3	3	5	5	1	1
32	3	1	3	3	2	7
33	3	2	2	4	2	1
34	3	3	1	5	2	1
35	3	1	10	3	3	7
36	3	2	9	4	3	1
37	3	3	8	5	3	1
38	3	1	6	3	4	7
39	3	2	5	4	4	1
40	3	3	4	5	4	1
41	3	1	2	3	5	7
42	3	2	1	4	5	1
43	3	3	0	5	5	1
44	3	1	9	3	6	7
45	3	2	8	4	6	1
46	3	3	7	5	6	1
47	3	1	5	3	7	7
48	3	2	4	4	7	1
49	3	3	3	5	7	1
50	4	1	0	5	1	3
51	4	2	10	6	1	1
52	4	1	7	5	2	3
53	4	2	6	6	2	1
54	4	1	3	5	3	1
55	4	2	2	6	3	1
56	4	1	10	5	4	3
57	4	2	9	6	4	1
58	4	1	6	5	5	3
59	4	2	5	6	5	1
60	4	1	2	5	6	3
61	4	2	1	6	6	1
62	4	1	9	5	7	3
63	4	2	8	6	7	1
64	5	1	5	6	1	6
65	5	2	4	7	1	1
66	5	1	1	6	2	6
67	5	2	0	7	2	1
68	5	1	8	6	3	6
69	5	2	7	7	3	1
70	5	1	4	6	4	6
71	5	2	3	7	4	1
72	5	1	0	6	5	6
73	5	2	10	7	5	1
74	5	1	7	6	6	6
75	5	2	6	7	6	1
76	5	1	3	6	7	6
77	5	2	2	7	7	1

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
3	19	11	10	7	7	7	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
49	1	7	5	7	7

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	1	2	1	1	2
3	1	1	7	1	1	3
4	1	1	1	1	1	4
5	1	1	6	1	1	5
6	1	1	0	1	1	6
7	1	1	5	1	1	7
8	2	1	7	4	2	1
9	2	1	1	4	2	2
10	2	1	6	4	2	3
11	2	1	0	4	2	4
12	2	1	5	4	2	5
13	2	1	10	4	2	6
14	2	1	4	4	2	7
15	3	1	6	7	3	1
16	3	1	0	7	3	2
17	3	1	5	7	3	3
18	3	1	10	7	3	4
19	3	1	4	7	3	5
20	3	1	9	7	3	6
21	3	1	3	7	3	7

22	4	1	5	3	5	1
23	4	1	10	3	5	2
24	4	1	4	3	5	3
25	4	1	9	3	5	4
26	4	1	3	3	5	5
27	4	1	8	3	5	6
28	4	1	2	3	5	7
29	5	1	4	6	6	1
30	5	1	9	6	6	2
31	5	1	3	6	6	3
32	5	1	8	6	6	4
33	5	1	2	6	6	5
34	5	1	7	6	6	6
35	5	1	1	6	6	7

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
4	19	11	10	7	7	7	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
1	49	7	5	7	7

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	2	1	1	1
3	1	1	9	2	1	1
4	1	2	3	2	1	1
5	1	1	10	3	1	1
6	1	2	4	3	1	1
7	1	1	0	4	1	1
8	1	2	5	4	1	1
9	1	1	1	5	1	1
10	1	2	6	5	1	1
11	1	1	2	6	1	1
12	1	2	7	6	1	1
13	1	1	3	7	1	1
14	1	2	8	7	1	1
15	2	1	8	1	4	2
16	2	2	2	1	4	2
17	2	1	9	2	4	2
18	2	2	3	2	4	2
19	2	1	10	3	4	2
20	2	2	4	3	4	2
21	2	1	0	4	4	2
22	2	2	5	4	4	2
23	2	1	1	5	4	2
24	2	2	6	5	4	2
25	2	1	2	6	4	2
26	2	2	7	6	4	2
27	2	1	3	7	4	2
28	2	2	8	7	4	2
29	3	1	8	1	7	3
30	3	2	2	1	7	3
31	3	1	7	1	7	3
32	3	2	9	2	7	3
33	3	1	3	2	7	3
34	3	2	8	2	7	3
35	3	1	10	3	7	3
36	3	2	4	3	7	3
37	3	1	9	3	7	3
38	3	2	0	4	7	3
39	3	1	5	4	7	3
40	3	2	10	4	7	3
41	3	1	1	5	7	3
42	3	2	6	5	7	3
43	3	1	0	5	7	3
44	3	2	2	6	7	3
45	3	1	7	6	7	3
46	3	2	1	6	7	3
47	3	1	3	7	7	3
48	3	2	8	7	7	3
49	3	1	2	7	7	3
50	4	1	2	1	3	5
51	4	2	7	1	3	5
52	4	1	3	2	3	5
53	4	2	8	2	3	5
54	4	1	4	3	3	5
55	4	2	9	3	3	5
56	4	1	5	4	3	5
57	4	2	10	4	3	5
58	4	1	6	5	3	5
59	4	2	0	5	3	5

60	4	1	7	6	3	5
61	4	2	1	6	3	5
62	4	1	8	7	3	5
63	4	2	2	7	3	5
64	5	1	2	1	6	6
65	5	2	7	1	6	6
66	5	3	1	1	6	6
67	5	1	3	2	6	6
68	5	2	8	2	6	6
69	5	3	2	2	6	6
70	5	1	4	3	6	6
71	5	2	9	3	6	6
72	5	3	3	3	6	6
73	5	1	5	4	6	6
74	5	2	10	4	6	6
75	5	3	4	4	6	6
76	5	1	6	5	6	6
77	5	2	0	5	6	6
78	5	3	5	5	6	6
79	5	1	7	6	6	6
80	5	2	1	6	6	6
81	5	3	6	6	6	6
82	5	1	8	7	6	6
83	5	2	2	7	6	6
84	5	3	7	7	6	6

ITYPE	IAO	MEMOD	#PES	IFIRST	ISECOND	ITHIRD	KO	JC
5	19	11	10	7	7	7	1	1

IDEL	JDEL	KDEL	ICLIM	JLIM	ILIM
7	1	49	5	7	7

NUM	CYCLE	SUBITER	OFFSET	IVAL	JVAL	KVAL
1	1	1	8	1	1	1
2	1	2	6	1	1	2
3	1	1	4	1	2	1
4	1	2	2	1	2	2
5	1	1	0	1	3	1
6	1	2	9	1	3	2
7	1	1	7	1	4	1
8	1	2	5	1	4	2
9	1	1	3	1	5	1
10	1	2	1	1	5	2
11	1	1	10	1	6	1
12	1	2	8	1	6	2
13	1	1	6	1	7	1
14	1	2	4	1	7	2
15	2	1	5	1	2	2
16	2	2	3	1	1	3
17	2	1	1	1	2	3
18	2	2	10	1	2	3
19	2	1	8	1	3	3
20	2	2	6	1	3	3
21	2	1	4	1	4	2
22	2	2	2	1	4	3
23	2	1	0	1	5	2
24	2	2	9	1	5	3
25	2	1	7	1	6	2
26	2	2	5	1	6	3
27	2	1	3	1	7	2
28	2	2	1	1	7	3
29	3	1	2	1	1	3
30	3	2	0	1	1	4
31	3	3	9	1	1	5
32	3	1	9	1	2	3
33	3	2	7	1	2	4
34	3	3	5	1	2	5
35	3	1	5	1	3	3
36	3	2	3	1	3	4
37	3	3	1	1	3	5
38	3	1	1	1	4	3
39	3	2	10	1	4	4
40	3	3	8	1	4	5
41	3	1	8	1	5	3
42	3	2	6	1	5	4
43	3	3	4	1	5	5
44	3	1	4	1	6	3
45	3	2	2	1	6	4

46	3	3	0	1	6	5
47	3	1	0	7	7	3
48	3	2	9	1	7	4
49	3	3	7	1	1	5
50	4	1	8	3	1	5
51	4	2	6	1	1	5
52	4	2	4	3	2	6
53	4	1	2	1	3	5
54	4	2	0	3	3	6
55	4	1	9	1	4	5
56	4	2	7	3	4	6
57	4	1	5	1	5	5
58	4	2	3	3	5	6
59	4	1	1	1	6	5
60	4	2	1	3	6	6
61	4	1	0	1	7	5
62	4	2	8	3	7	6
63	4	1	6	1	1	6
64	5	2	4	6	1	7
65	5	1	5	1	2	6
66	5	2	3	6	2	7
67	5	1	1	1	3	6
68	5	2	0	6	3	7
69	5	1	8	1	4	6
70	5	2	6	6	4	7
71	5	1	4	1	5	6
72	5	2	2	6	5	7
73	5	1	0	1	6	6
74	5	2	9	6	6	7
75	5	1	7	1	7	6
76	5	2	5	6	7	7
77	5	1	3	1	8	6
		2	1		7	7

APPENDIX B

SECDED RELIABILITY IMPROVEMENT MODELS

B. 1 INTRODUCTION

The reliability of a computing system can be significantly improved by employing single bit error correction and double bit error detection (SECDED) technology, which is thus used by the FMP to increase its reliability.

The report presents a model of reliability improvement assessment of a module operated with SECDED. It can be easily embedded in the system reliability prediction model. The final result is shown in a mathematical expression. The bounds of the reliability and the improvement factor are studied. A computer program coded on FORTRAN is also developed and validated, with double precision computation.

B. 2 MODEL

There are n chips in a module; a chip has m bits. A word which consists of n bits can be stored in this module by addressing each bit to a different chip. Without SECDED a bit failure induces the chip failure and the module failure as well. Assume the time to failure of a bit is exponential distributed, then the time to failure of a chip and that of a module are also exponential distributed.

In some cases, a bit hard failure could cause a chip failure with probability (1-S). We call it a catastrophic bit failure. Otherwise a bit failure is called non-catastrophic bit failure with probability S. Assuming the MTBF of the chip as a time unit, we have that the MTBF of a bit is m time units and the bit failure rate is 1/m. The MTBF and the failure rate of a module are 1/n and n respectively. The expected time between (i-1)th and ith bit failure, the expected time to ith bit failure, the probability of no two-bit failure in one word and the probability of two-bit failure in one word are stated in Table B-1. The module fails at the ith bit failure only when there is neither catastrophic failure nor two-bit failure in the same word before the ith bit failure, but there is a catastrophic failure or two-bit failure in the same word when the ith bit failure occurs. Since the transient and catastrophic failures of a module at the ith bit failure are mutually exclusive, the MTBF of a module with SECDED is given by

$$MTBF_m = (1 - S)\left\{\frac{1}{n} + \frac{m}{mn-1}\right\}$$

$$+ \sum_{i=2}^{m}\left\{\left[\sum_{k=1}^{i}\frac{m}{mn-(k+1)}\right]\left[\prod_{k=1}^{i-1}\frac{n(m-(k-1))}{mn-(k-1)}\right] S^{(i-1)}\left[(1-S) + \frac{Sn(i-1)}{mn-(i-1)}\right]\right\}$$

$$+ S^m\left[\sum_{k=1}^{m}\frac{m}{mn-(k-1)}\right]\left[\prod_{k=1}^{m}\frac{n(m-(k-1))}{mn-1}\right]$$

Table B-1. Expected Times and Probabilities

ith bit failure	1	2	—	i-1	i	—	m	m+1
Expected time between the (i-1)th and ith bit failure	$\frac{m}{mn}$	$\frac{m}{mn-1}$		$\frac{m}{mn-(i-2)}$	$\frac{m}{mn-(i-1)}$		$\frac{m}{mn-(m-1)}$	$\frac{m}{mn-m}$
Expected time to the ith bit failure	$\frac{m}{mn}$	$\frac{m}{mn-1} + \frac{m}{mn}$		$\sum_{k=1}^{i-1} \frac{m}{mn-(k-1)}$	$\sum_{k=1}^{i} \frac{m}{mn-(k-1)}$		$\sum_{k=1}^{m} \frac{m}{mn-(k-1)}$	$\sum_{k=1}^{m+1} \frac{m}{mn-(k-1-}$
Prob. of the ith bit failure being non-catastrophic failure	S	S		S	S		S	S
Prob. of the ith bit failure being catastrophic failure	1-S	1-S		1-S	1-S		1-S	1-S
Prob. of no. two bit failure in one word	1	$\frac{n(m-1)}{mn-1}$		$\frac{n(m-(i-2))}{mn-(i-2)}$	$\frac{n(m-(i-1))}{mn-(i-1)}$		$\frac{n-1}{mn-(m-1)}$	0
Prob. ot two bit failure in one word	0	$\frac{n-1}{mn-1}$		$\frac{n-(i-2)}{mn-(i-2)}$	$\frac{n)i-1)}{mn-(i-1)}$		$\frac{n(m-1)}{mn-(m-1)}$	1

From the above expression, the reliability improvement factor can be shown as $n.\,MTBF_m$. When S=1, we have the upper bound of the factor and $MTBF_m$. As S=0, we have the lower bound of the factor and $MTBF_m$, if m is large enough the lower bound of the factor is 2.

If the expected time between the (i+1)th and ith failure is fixed as n time units the expected time to the ith bit failure is $i \cdot n$. The MTBF of a module with SECDED is given by:

$$MTBF_m = (1 - S)\ \frac{2}{n}$$

$$+ \sum_{i=2}^{m} \left\{ \frac{i}{n} \left[\prod_{k=1}^{i-1} \frac{n(m-(k-1))}{mn-(k-1)} \right] S^{(i-1)} \left[(1-S) + \frac{Sn(i-1)}{mn-(i-1)} \right] \right\}$$

$$+ S^m\ \frac{m}{n} \left[\sum_{k=1}^{m} \frac{m}{mn-(k-1)} \right]$$

Similarly as S=1 or 0, we have the upper bound or the lower bound of the factor and $MTBF_m$, respectively. When m is large enough the difference between the $MTBF_m$'s of the two models is negligible and so is that between the factors. The program for computing the reliability improvement factor are given in the Table B-2.

```
CARD #0001            SUBROUTINE SECFAC(M,N,S,FAC)
CARD #0002      C
CARD #0003      CC
CARD #0004      C     THE PROG. COMPUTES THE SECDED RELIABILITY IMPROVEMENT FACTOR.
CARD #0005      C     M       # OF BITS IN A CHIP.                              ----(INPUT)
CARD #0006      C     N       # OF CHIPS IN A MODULE, ALWAYS EQUAL TO # OF BITS IN A WORD
CARD #0007      C                                                               ----(INPUT)
CARD #0008      C     S       THE PROB. OF A NON-CATASTROPHIC BIT FAILURE.      ----(INPUT)
CARD #0009      C     FAC      SECDED RELIABILITY IMPROVEMENT FACTOR,           ----(OUTPUT)
CARD #0010      C               DEFINED AS MTBF OF THE MODULE WITH SECDED
CARD #0011      C               DEVIDED BY MTBF OF THE MODULE WITHOUT SECDED
CARD #0012      CC
CARD #0013      C
CARD #0014            IMPLICIT REAL*8(A-H,O-Z)
CARD #0015            FM=FLOAT(M)
CARD #0016            FN=FLOAT(N)
CARD #0017            FMTFN=FM*FN
CARD #0018            SEDM=1./FN
CARD #0019            I=1
CARD #0020            PI=1.
CARD #0021      C   COMPUTING EXPECTED MTBF GIVEN THE 1ST BIT FAILURE IS CATASTROPHIC ONE
CARD #0022            SUM=(1.-S)*(SEDM+1./(FN-1.))
CARD #0023            SIM1=1.
CARD #0024            S1=SEDM
CARD #0025            DO 100 I=2,M+1
CARD #0026            SIM1=SIM1*S
CARD #0027      C   STOP COMPUTING NEGLIGIBLE QUANTITY
CARD #0028            IF(SIM1.LE. 1.E-10) GO TO 200
CARD #0029            FI=FLOAT(I)
CARD #0030            FIM1=FLOAT(I-1)
CARD #0031      C   COMPUTING PROB. OF NO 2-BIT FAILUR IN ONE WORD UP TO THE ITH BIT FAILURE
CARD #0032            PI=PI*(FM-(FIM1-1.))*FN/(FMTFN-(FIM1-1.))
CARD #0033      C   COMPUTING EXPECTED TIME UP TO THE ITH BIT FAILURE
CARD #0034            S1=S1+FM/(FMTFN-FIM1)
CARD #0035      C   COMPUTING PROB. OF THE MODULE FAILURE AT THE ITH BIT FAILURE
CARD #0036            S3=1.-S+S*(FIM1*FN)/(FMTFN-FIM1)
CARD #0037            FPROB=SIM1*PI*S3
CARD #0038      C   STOP COMPUTING NEGLIGIBLE QUANTITY
CARD #0039            IF(FPROB .LE. 1.E-20) GO TO 200
CARD #0040      C   COMPUTING EXPECTED MTBF GIVEN THE MODULE FAILED BEFORE OR
CARD #0041      C   AT THE ITH BIT FAILUR
CARD #0042            SUM=SUM+S1*FPROB
CARD #0043      100   CONTINUE
CARD #0044      200   FAC=SUM*FN
CARD #0045            RETURN
CARD #0046            END
```

APPENDIX C

SPARE PROCESSOR

INTRODUCTION

In Chapter 5, the reliability and availability calculations make use of the switching of spare processors. This appendix presents the method of switching in more detail, to support the claims made in Chapter 5. First, a discussion of the hardware that needs to be added to support the switching, and second, the implications for processor number are given.

SWITCHING

Figure C-1 shows a switching network, which amounts to one additional level of logic at the processor side of the transposition network. This network therefore increases the depth of the transposition network from ten levels to eleven levels of logic. Switching is electronic, under software control. The spare processor can occur at any location from processor 0 to processor 128 in the cabinet. Figure C-1 shows the first cabinet; the others are similar.

No switching is needed in the connections to and from the control unit. All outputs from the control unit to processors are broadcast to all processors; the inputs from processors to CU are either ANDed together with a 512-way AND, or ORed together with a 512-way OR in the fanout boards. The fanout

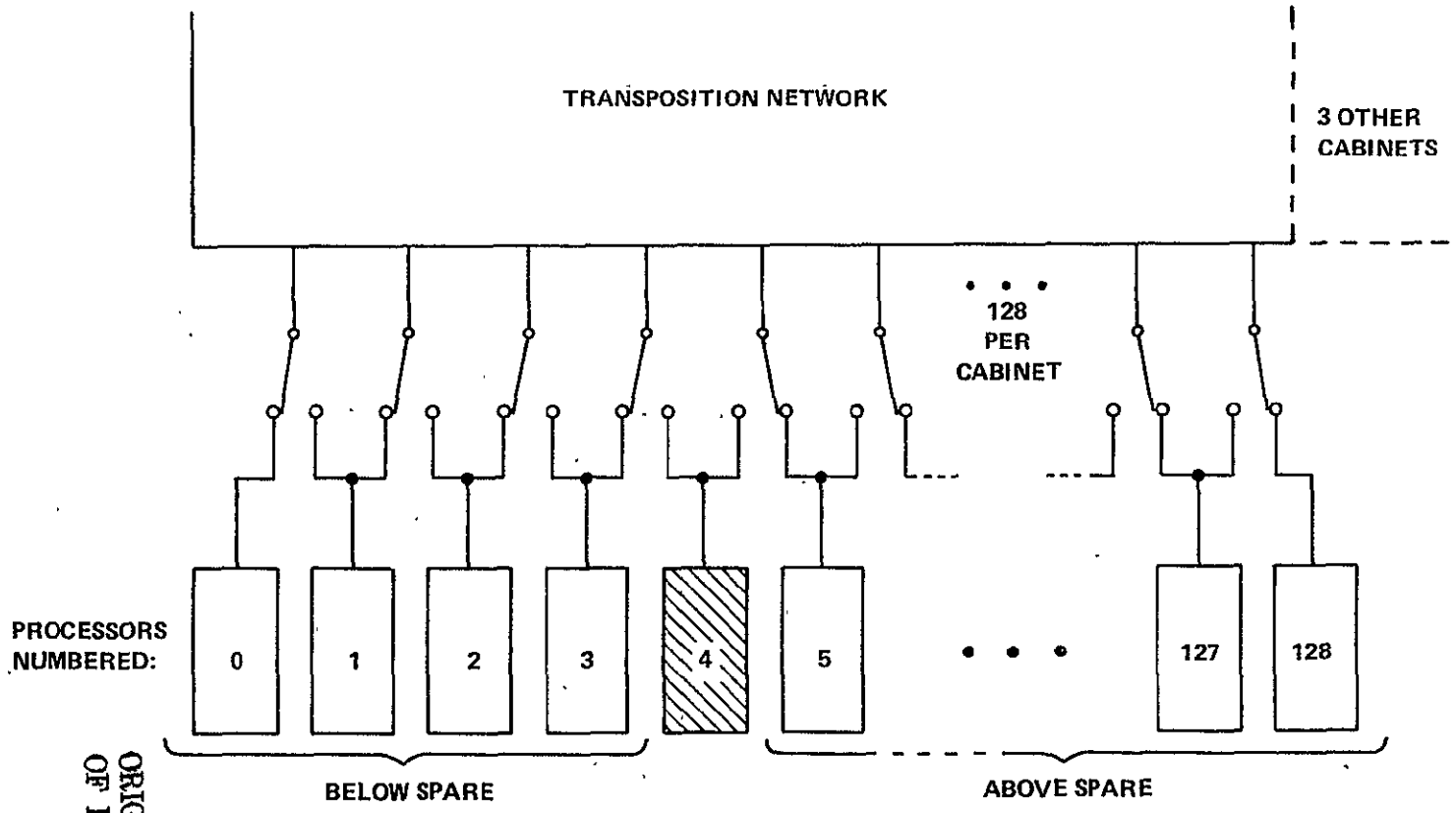


Figure C-1. Switching of Spare Processor in One Cabinet

board needs appropriate input from the spare processor to form the correct 512-way result. For example, in forming "all processors ready", or in forming "any processor enabled", the correct result will be achieved by having the spare processor's "enable" bit in the FALSE state.

PROCESSOR NUMBER

The PNO instruction produces processor numbers from 0 through 511 in the 512 processors that are switched into the system, independently of which ones are spare. Each processor in the cabinet has wired into its backplane a number from 0 through 128. Each cabinet has a number (0, 1, 2, or 3) set by a switch at the cabinet fanout board. If it were not for the spare processor, the cabinet number would be concatenated with the hard-wired number in the backplane to form the processor number. As it is, processors above the spare processor subtract 1 from their hard-wired number before concatenating it with the cabinet number to form the programmatic processor number as part of the PNO instruction. Thus, there are ten poles on each switch shown in Figure C-1. The eight data lines plus one strobe make nine poles for transposition network use, plus this bit for the PNO instruction to use in calculating the processor number.

SETTING THE SPARE PROCESSOR SWITCH

The setting of the spare processor switch is done only at a time when the array has halted. Switching is controlled from the diagnostic controller in response to commands from the host. Hence, the FMP programs are never aware of

which processor is spare, and as explained above, the FMP programs will always have an FMP of 512 processors, numbered from 0 through 511, on which to run.