APPLICATION OF SOFTWARE TECHNOLOGY TO A FUTURE SPACECRAFT COMPUTER DESIGN

Robert J. LaBaugh
Martin Marietta Corporation
Denver, Colorado

An Independent Research and Development task* at Martin Marietta has been investigating advanced spacecraft computer systems for the past couple of years. The task objectives are to demonstrate how major improvements in spacecraft computer systems can be obtained from recent advances in hardware** and software technology. This presentation covers the major software topics which have been addressed during the task.

Investigations into integrated circuit technology performed at the beginning of the task indicated that the CMOS/SOS chip set being developed for the Air Force Avionics Laboratory at Wright Patterson had the best potential for improving the performance of spaceborne computer systems. An integral part of the chip set is the bit slice arithmetic and logic unit (ALU). The flexibility allowed by microprogramming, combined with the software investigations described below, led to the specification of a baseline architecture and instruction set.

---

*This work was conducted by the Denver Division of Martin Marietta Corporation under Independent Research and Development Project Authorization D-80D.

**Related paper, "Application of Advanced Electronics to a Future Spacecraft Computer Design", in Microprocessor Hardware Technology Session.

One of the goals was to design an instruction set similar to modern minicomputer instruction sets, with multiple user registers. Another goal was to provide the throughput and precision required for flight applications, and at the same time provide an instruction set which would ease the programming of such applications. Several assembly language application programs, along with the necessary microcode, were written to help define the features to be included in the processor design. One of the areas this was used for was determining the number of registers in the system. An increase in the number of floating point registers from four to eight provided around a 10% improvement in execution time for one of the application programs. The number of registers was limited, however, by a desire to store them in the 16 physical registers internal to the ALUs. This would avoid delays in accessing the registers and help keep the parts count down. The need for scratch registers by some of the microprograms was another limiting factor. As a compromise between having as many registers as possible and the limits imposed by the ALUs, it was decided there should be seven floating point registers and eight general purpose registers. Partly to accommodate all the user registers desired, the system was designed to have two arithmetic processing units: one to handle floating point operations and store the floating point registers; and the other to handle general purpose registers, and system registers such as stack pointers and the program counter. Only one of the two processors is active at any given time. Which processor is active during a cycle is determined by means of a bit in the microword. This was done so that the processors could share the 26 bits in the microword needed for ALU control.

The precision and format of floating point operands was derived in part from the results of a study on high precision attitude computations. The main goal of the study was to characterize the drift rate of the integrator as a function of operand precision and rate sampling interval. One of the conclusions was that 32 bit floating point operands, with 24 bit mantissas, were adequate for currently envisioned projects. The study was based on data using operands with binary normalization. To remain consistent with this, we decided to use binary rather than hex normalization which can result in only 21 bits of significance in a 24 bit mantissa.

Because of the characteristics of the chip set and a desire for fairly high performance, a horizontal, rather than vertical, microword was used. There are 34 fields in the microword, which is 80 bits wide. As wide as the microword is, a fair number of fields still have to be decoded. Encoded fields are primarily used for things like selection of operand source and destination, the source for register specifications, and condition code selection. A wide microword, however, puts constraints on the number of words of control store because of the high non-recurring cost of ROMs. In an effort to keep the size of the control store within limits, and to assure adequate system performance, the microcode was developed concurrently with the circuit design. This allowed us to make various hardware versus software trades at a time when changes to the hardware design could be accomplished without too much difficulty.

Fairly early in the task an absolute assembler and an instruction set simulator were developed. These allowed the software development to proceed while the hardware design was being completed, and the hardware was being built. Langley Research Center has recently provided Pascal and HAL compiler frontends. The Pascal system includes a compiler which produces P-code, and a P-code interpreter. The HAL system consists of a compiler which produces HALMAT, a program which translates HALMAT into H-code, and an H-code interpreter. H-code is P-code with a few extra instructions and an expanded run time library. A program to translate from P-code/H-code to assembly language has been developed and Pascal and HAL routines have been executed on the instruction set simulator. The translator has undergone several refinements to improve the code generated. The initial version mimicked P-code fairly closely by keeping the expression evaluation stack in memory. By redefining register usage so that the top of the stack was kept in registers wherever possible, a 50% improvement in memory usage and execution time was achieved. Floating point push and pop instructions were also added. An investigation of a one instruction lookahead in the translation process indicated a further 12 to 14 percent improvement in memory usage and 7 to 30 percent improvement in execution time was possible. Future plans include continued investigation of P-code instruction lookahead in the translation process and further examination of the impact of high order languages on the instruction set.

BACKGROUND

OBJECTIVES:*

QUANTITATIVELY DETERMINE HOW RECENT ADVANCEMENTS IN HARDWARE** AND SOFTWARE TECHNOLOGY CAN BE USED TO OBTAIN IMPROVEMENTS IN SPACECRAFT COMPUTER CAPABILITIES.

CMOS/SOS INTEGRATED CIRCUITS
SEMI-CUSTOM LSI DEVICES
LEADLESS CARRIER PACKAGING
MICROPROGRAMMING
PASCAL, HAL, ADA, HIGHER ORDER LANGUAGE

*THIS WORK WAS CONDUCTED BY THE DENVER DIVISION UNDER INDEPENDENT RESEARCH AND DEVELOPMENT PROJECT AUTHORIZATION D-80D

**RELATED PRESENTATION, "APPLICATION OF ADVANCED ELECTRONICS TO A FUTURE SPACECRAFT COMPUTER DESIGN", IN SESSION IV: MICROPROCESSOR HARDWARE TECHNOLOGY

APPROACH

PRELIMINARY REQUIREMENTS AND IMPACT

S/W TO ASSIST IN ARCHITECTURE DESIGN

- ABSOLUTE ASSEMBLER
- INSTRUCTION SET SIMULATOR

MICROPROGRAM DESIGN

S/W DEVELOPMENT TOOLS

FEATURES REQUIRED IN PROCESSOR

MINICOMPUTER LIKE INSTRUCTION SET

MULTIPLE FLOATING POINT AND GENERAL PURPOSE REGISTERS

FLIGHT APPLICATIONS

- SUFFICIENT THROUGHPUT
- SUFFICIENT PRECISION
- EASE OF PROGRAMMING

REGISTER CONSIDERAIONS

TYPES:

- GENERAL FOR USER
- FLOATING POINT FOR USER
- SYSTEM FOR USER
- SYSTEM FOR MICROPROGRAMS

CONSTRAINTS:

- 16 PHYSICAL REGISTERS INTERNAL TO ARITHMETIC AND LOGIC UNIT DEVICES
- SEPERATE ALU DEVICES FOR FLOATING POINT

TRADES:

- PERFORMANCE SENSITIVITY TO SIZE OF REGISTER FILE
- EXTERNAL REGISTER FILE - DEGRADES PERFORMANCE

REGISTER CONSIDERATIONS

OPERAND PRECISION:

- LARGER OPERANDS REQUIRE MORE PARTS, LONGER CYCLE TIMES
- MICROCODE VS HARDWARE - SLOWER, LARGER CONTROL STORE NEEDED
- FLOATING POINT PRECISION - 32 BITS WITH BINARY NORMALIZATION SUPPORTED BY SPECIALIZED HARDWARE
- INTEGER PRECISION - 16 BIT WITH 32 BIT PERFORMED BY MICROCODE

FUNCTIONAL REGISTER UTILIZATION

16 BITS
PROGRAM COUNTER
USER STACK POINTER
PRIV STACK POINTER

GEN REG 0
GEN REG 1
GEN REG 2
GEN REG 3
GEN REG 4
GEN REG 5
GEN REG 6
GEN REG 7

32 BITS
FLOATING POINT REGISTER 1
FLOATING POINT REGISTER 2
FLOATING POINT REGISTER 3
FLOATING POINT REGISTER 4
FLOATING POINT REGISTER 5
FLOATING POINT REGISTER 6
FLOATING POINT REGISTER 7

NOTES:

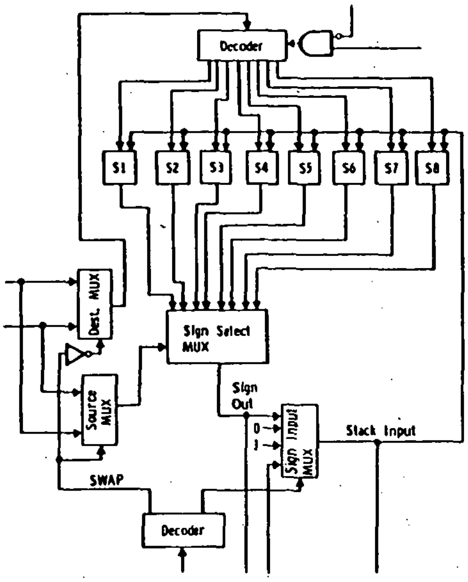
- FLOATING POINT SIGN BIT KEPT IN UNIQUE REGISTER FILE
- 5 OTHER 16-BIT REGISTERS RESERVED FOR MICROPROGRAMMER
- 1 OTHER 32-BIT REGISTER RESERVED FOR MICROPROGRAMMER
- GENERAL REGISTERS 1 - 7 CAN BE USED AS INDEX REGISTERS

EXAMPLE OF PHYSICAL REGISTER UTILIZATION

FLOATING POINT PROCESSOR

0	SCRATCH
1	FLT PT REG 1 MANTISSA
2	FLT PT REG 2 MANTISSA
3	FLT PT REG 3 MANTISSA
4	FLT PT REG 4 MANTISSA
5	FLT PT REG 5 MANTISSA
6	FLT PT REG 6 MANTISSA
7	FLT PT REG 7 MANTISSA
8	SCRATCH
9	FLT PT REG 1 EXPONENT
A	FLT PT REG 2 EXPONENT
B	FLT PT REG 3 EXPONENT
C	FLT PT REG 4 EXPONENT
D	FLT PT REG 5 EXPONENT
E	FLT PT REG 6 EXPONENT
F	FLT PT REG 7 EXPONENT

24 BITS



THE FLOATING POINT SIGN BIT FILE IS EMBEDDED IN CUSTOMIZED LOGIC

MACRO LEVEL INSTRUCTION SET

106 INSTRUCTIONS

8 CATAGORIES

- FIXED POINT
- INDEX/COUNTER REGISTER
- FLOATING POINT
- LOGICAL
- BRANCH
- STACK AND REGISTER SAVE AND RESTORE
- EXECUTIVE FUNCTIONS
- MISCELLANEOUS

10 FORMATS

- REGISTER-REGISTER
- REGISTER
- REGISTER-ADDRESS
- REGISTER-IMMEDIATE
- INDEX-REGISTER
- INDEX EXTENDED
- ADDRESS
- INDEX-ADDRESS
- SPECIAL
- SPECIAL EXTENDED

MICROPROGRAM DESIGN

HORIZONTAL RATHER THAN VERTICAL

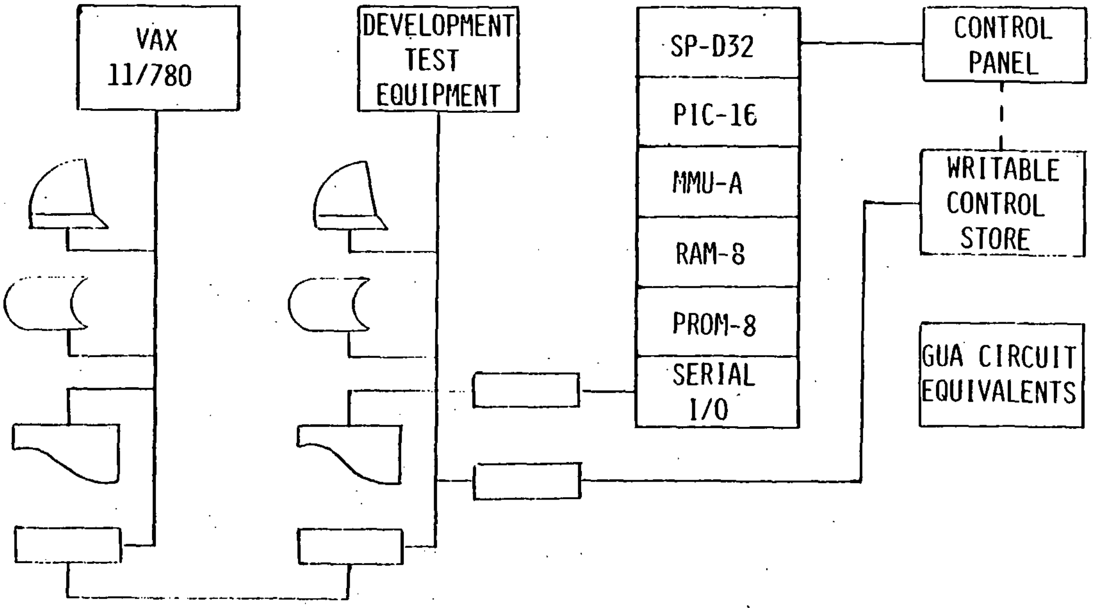
- DECODING FIELDS DEGRADES PERFORMANCE
- FORCE LOGIC TO QUIESCENT STATE WHEN NOT BEING USED
- BECAUSE OF WIDE MICROWORD, NEED TO KEEP NUMBER OF WORDS OF CONTROL STORE AS SMALL AS POSSIBLE

MICROPROGRAMS DEVELOPED CONCURRENTLY WITH HARDWARE DESIGN

- ASSURE ADEQUATE PERFORMANCE
- CONTROL STORE LIMITED BECAUSE OF HIGH NON-RECURRING COST

PRIMARY SOFTWARE MODULES & STATUS		REQMTS	DESIGN	IMPLEMENTATION
ASMA-D32:	ABSOLUTE ASSEMBLER	COMPLETE	COMPLETE	COMPLETE
ASMR-D32:	RELOCATABLE ASSEMBLER	COMPLETE	COMPLETE	COMPLETE
LNK-D32:	LINK EDITOR	COMPLETE	COMPLETE	COMPLETE
SIM-D32:	INSTRUCTION SET SIMULATOR	COMPLETE	COMPLETE	COMPLETE
PAS-LC1:	PASCAL COMPILER	COMPLETE	COMPLETE	COMPLETE
HAL-LC1:	HAL COMPILER	COMPLETE	COMPLETE	COMPLETE
RTEX:	REAL TIME EXECUTIVE	IN PROGRESS	1981	1981
SSP-D32:	SCIENTIFIC SUBROUTINE PACKAGE	IN PROGRESS	IN PROGRESS	IN PROGRESS
STD-2:	SELF TEST/DIAGNOSTIC ROUTINES	IN PROGRESS	IN PROGRESS	1981

COMPUTER DEMONSTRATION UNIT SET UP



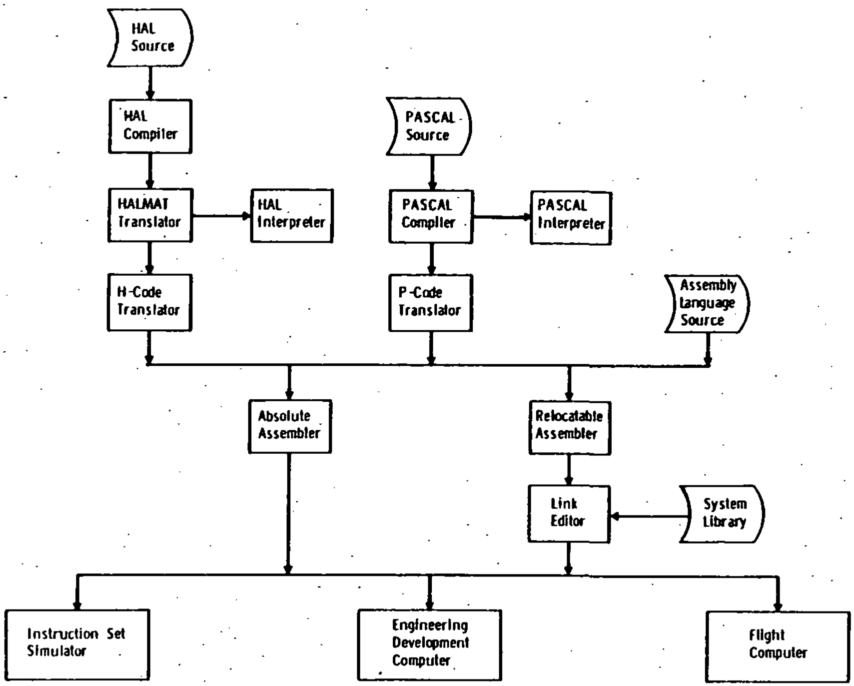
HIGH ORDER LANGUAGE CAPABILITIES

PASCAL AND HAL COMPILERS
- FROM LANGLEY RESEARCH CENTER
- WRITTEN IN PASCAL

PATH PASCAL COMPILATION PROCESS
- PRODUCES P-CODE
- INTERPRETER FOR P-CODE

HAL COMPILATION PROCESS
- PHASE 1 PRODUCES HALMAT
- HALMAT TO H-CODE
- INTERPRETER FOR H-CODE

TRANSLATOR FROM P-CODE/H-CODE TO ASSEMBLY LANGUAGE



TRANSLATOR REFINEMENTS

SAMPLE PROGRAMS

- CALCULATE PI TO SIX DIGITS
- BINARY SEARCH

PRELIMINARY DESIGN:

- STACK IN MEMORY

FIRST REVISION:

- TOP OF STACK KEPT IN REGISTERS
- FLOATING POINT PUSH AND POP ADDED

SECOND REVISION:

- LOOK AT TWO P-CODE INSTRUCTIONS BEFORE GENERATING CODE

PROCESSOR SOFTWARE COMPARISON

NUMERIC TEST PROGRAM - PI APPROXIMATION

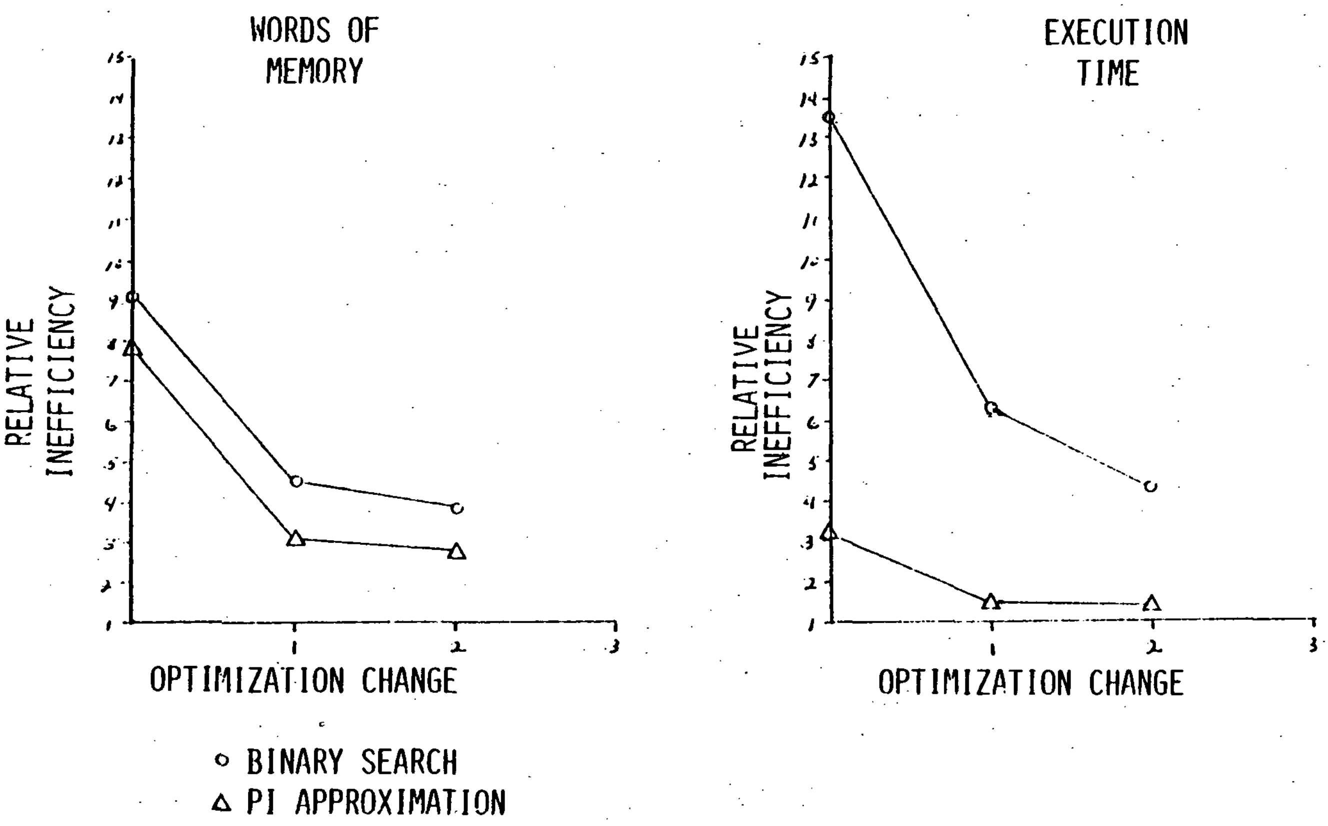
	MAC-16	PDP 11/34M	ATAC-16
ASSEMBLY LANGUAGE			
LINES OF CODE	40	37	50
WORDS OF MEMORY	74	68	79
EXECUTION TIME	11969	14909	12412
PASCAL LANGUAGE			
LINES OF CODE	28	28	-
WORDS OF MEMORY	243	347	-
EXECUTION TIME	16691	-	-
HAL LANGUAGE			
LINES OF CODE	28	-	28
WORDS OF MEMORY	254	-	157
EXECUTION TIME	16885	-	13722

PROCESSOR SOFTWARE COMPARISON

NON-NUMERIC TEST PROGRAM - BINARY SEARCH

	MAC-16	PDP 11/34M	ATAC-16
ASSEMBLY LANGUAGE			
LINES OF CODE	25	26	24
WORDS OF MEMORY	31	31	24
EXECUTION TIME	143	170	127
PASCAL LANGUAGE			
LINES OF CODE	17	17	-
WORDS OF MEMORY	138	107	-
EXECUTION TIME	886		-
HAL LANGUAGE			
LINES OF CODE	17	-	17
WORDS OF MEMORY	142	-	65
EXECUTION TIME	937	-	665

CODE GENERATOR IMPROVEMENT HISTORY



FUTURE PLANS

TRANSLATOR:

- MULTI P-CODE INSTRUCTION LOOKAHEAD
- MULTI PASS - OPTIMIZE REGISTER USAGE

DIRECT HALMAT TO ASSEMBLY LANGUAGE CONVERSION
BASED ON HALMAT TO H-CODE PROGRAM

CONTINUE EXAMINING HOL IMPACT ON INSTRUCTION SET