MAN.1857

Prime Computer Microcoders Handbook

PRIME COMPUTER, INC., 145 PENNSYLVANIA AVENUE, FRAMINGHAM, MA. 01701

MICROCODERS

HANDBOOK

September 1974

PRIME
COMPUTER, INC.

[145 Pennsylvania Ave., Framingham, Mass. 01701]

Equipment characteristics, performance specifications, and operating procedures are subject to change without notice.

CONTENTS

Page

SECTION 1 INTRODUCTION

SECTION 2 MICROCODE CONTROL OF THE CPU

SECTION 3 TIMING

CONTENTS (Cont)

CONTENTS (Cont)

SECTION 1

INTRODUCTION

1.1 SCOPE OF DOCUMENT

The intent of this document is to present enough information to permit a person with a solid assembly language and machine language programming background to prepare complete microcode programs. This includes a study of what is feasible through the use of microcode; the algorithms that are appropriate; and a discussion of good practices for coding, assembling, debugging, and executing microcode. The largest body of material to be covered is a detailed description of the way that the microcode controls the individual hardware components in the machine. The document begins with the simplest description of microcode control and slowly builds on that description until the complete capability of the 64-bit microcode word is demonstrated.

1.2 WHY MICROCODE

The fundamental reason for implementing an algorithm or functionality in writable control store rather than in standard users' software is to gain an increase in performance. Execution speeds of microcode algorithms versus software algorithms can be expected to be at least twice and perhaps as many as ten times as fast. Secondly, there are capabilities that exist in microcoding that simply are not available to the standard user-level software. Examples of these capabilities include: the ability to build custom high-speed I/O disciplines, the ability to implement different memory management techniques other than the virtual memory which is available standardly, and the ability to do byte operations directly on memory. Examples when microcoding may be justifiable include (1) customer algorithms, (2) emulating a different machine architecture, (3) adding new instructions, and (4) building high level language run-time interpreters.

1.2.1 Custom Algorithms

A custom algorithm that might be put in microcode should be an easily and completely definable set of operations to translate or reduce data from one form to another. Additionally, this one algorithm or operation should be the major function of the machine so that the improved speed of the execution of this algorithm can be directly translated into improved system performance. Examples of algorithms that might be appropriate include a fast Fourier Transform or any of the standard statistical calculation equations (means, deviations, significance calculations and so forth).

Whether a function can be profitably put into microcode is primarily a question of economics that depends very much on the situation. However, some functions are technically more likely to profit from being microcoded than others. In particular, table reductions involving many additions and subtractions and perhaps multiplications and division, temporary storage, and different paths of operations depending on the particular value of the in-process calculation could be more profitably microcoded than another algorithm that is fundamentally a simple translation of one table to another. The reason for this relative advantage is that, if there are many calculations and decisions and relatively few memory references, a programmable machine with a typical memory referencing instruction of slightly over one and a half microseconds is substituted for a programmable machine with an arithmetic or decision time of 200 nanoseconds.

Unfortunately, the main memory still runs at the same speed it always did, so if the microcode algorithm is primarily high-speed memory bound, relatively smaller improvements in execution time can be expected.

Another factor to consider when deciding if a custom algorithm is worth microcoding is to determine if the algorithm is already microcode limited when it is coded in assembly language. This could be true if the assembly language code consists primarily of multiplies, divides and long shifts where the majority of the execution time is determined by the speed of the microcode processor.

1.2.2 Emulating Machines

In emulating, the intent is to substitute the standard Prime Computer Instruction set for that of some different machine, probably with a different architecture. The primary motive for this activity is normally to take advantage of existing software and some other assembly language. In this case, there are two sides to the feasibility question. It's obvious that Prime itself was able to build an instruction set in microcode so that, of course, the task is in some sense useful. However, it is also true that a moderate amount of special hardware help specific to the Prime instruction set is available in the standard Prime Computer that would not be available when emulating a different machine.

1.2.3 Adding Instructions

Adding instructions to the Prime instruction complement is feasible and useful in a number of ways. Examples of potential candidates for instructions to be added would of course include both user-specific instructions and, potentially, other general-purpose instructions that are not part of the Prime repertoire. Testing, setting and resetting particular bits in memory is an example of an instruction that could be added to the standard Prime instruction set using microcoding techniques.

1.2.4 High Level Language Run-Time Interpreters

An example of a high level language run-time interpreter is a set of microcode that executes using a table generated by a Basic compiler, at least as far as computational tasks are concerned. In general, a run-time interpreter differs from the emulation of a complete machine since I/O device handling; and perhaps file handling may be done by calls to standard systems software or assembly language I/O file handling routines.

1.3 WHY NOT MICROCODE?

Aside from the restrictions placed on the suitability for using microcode in the previous four paragraphs, there are other factors that must be considered before embarking on a microcoding project. The first of these is the question of development time. Because microcode is much closer to the hardware logic of the machine, it is much further from the aids that can be given to the programmer such as higher level languages and on-line debug. Second, many attributes of the Prime Computer that are normally assumed to be part of the hardware specifications of that computer are, in fact, microcode routines. Such features as DMX latency; in fact, all of the I/O functionality; the operation of the control panel; and the basic instruction fetch can be affected in non-obvious ways by errors in the new or added microcode routine. Of course, the fact that these features are microcoded can be considered as an almost undreamed of flexibility to be exploited in specific application-oriented sets of microcode.

Another factor that must be considered before serious attempts at microcoding can be considered is that of microcode space available for the function. The writable control store capability is limited to 256, 52-bit microcode words. Examples of the amount of functionality that can be put into that space are: the complete Prime Instruction Set including DMA occupies only 256 words and the single precision floating point package occupies 185 words of microcode. Of course, the total microcode address space is much larger than 256 words. However, standard Prime hardware for the processors offered to date does not implement this entire address space.

1.4 IMPLEMENTATION AIDS

1.4.1 Writable Control Store (WCS)

The logic for Writable Control Store is contained on a printed circuit board that connects to the Prime 300 central processor printed circuit board by cables on the back of the processor board and by the I/O bus backplane. This board implements 256, 52-bit words of very high speed RAM. This memory may be written into using PIO instructions and then may be entered by executing any of the enter and execute class of instructions implemented on the Prime 300. For more details, refer to Section 8.5, "Writable Control Store Board".

1.4.2 Microcode Assembler

The microcode assembler is a macro package inserted into a file of microcode source language. This macro package is used by the standard Prime macro assembler (PMA) to assemble the microcode source statements. As a result of this two-step approach to microcode assembly; and also because of the complexity of the microcode word, microcode assembles at the rate of about one line of object code per second. The minimum configuration on which a microcode assembler package can be expected to work is a 32K DOS/VM system. Features of this assembler are described in Section 8.3.

1.4.3 Microcode Debug Package

To be specified.

SECTION 2

MICROCODE CONTROL OF THE CPU

2.1 DESCRIPTION OF THE MACHINE

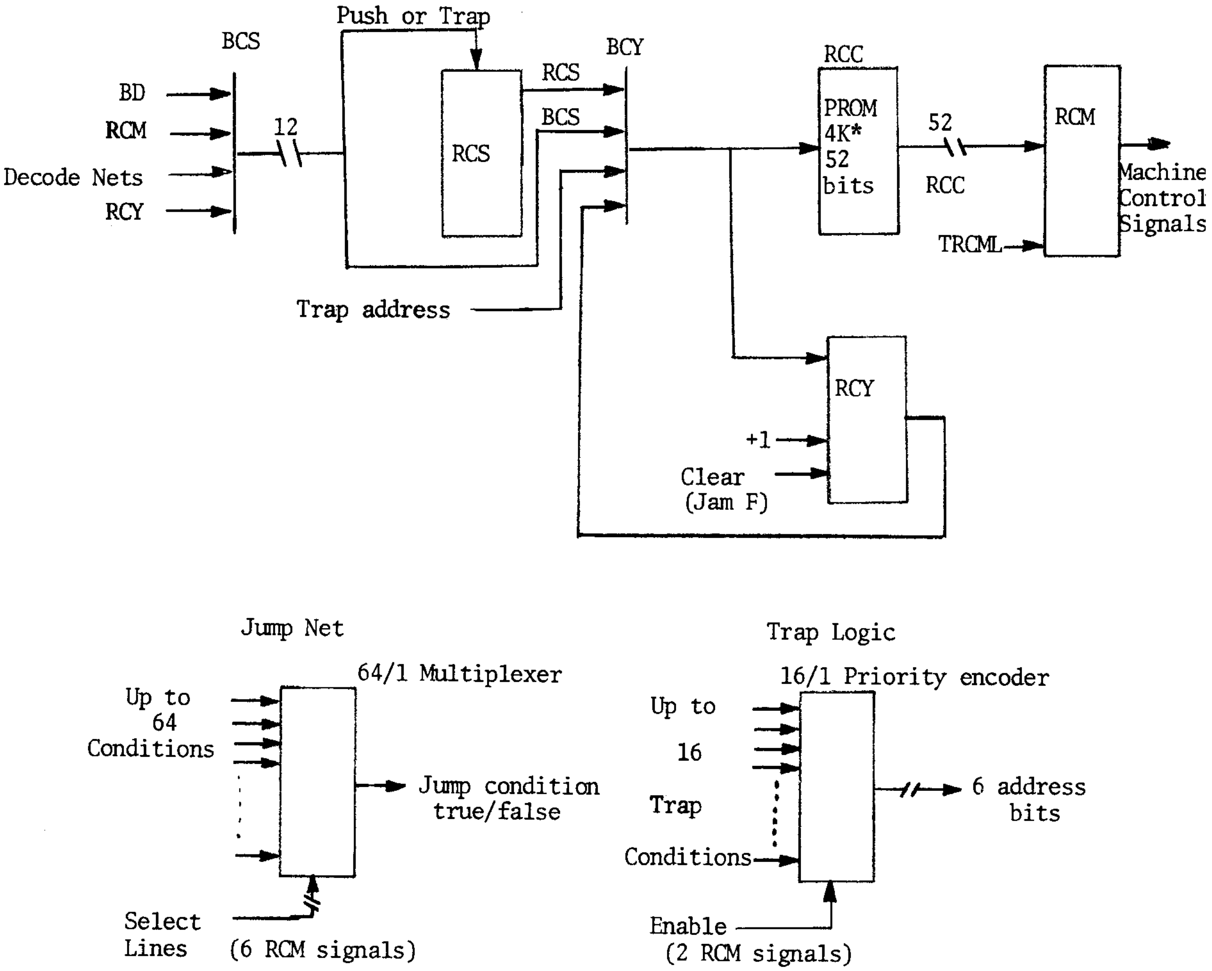
For the purposes of discussion, the hardware processor board may be divided logically into three major sections. The first section is the control logic, the second section is the controlled logic, and the third section consists of three major subsystems. The subsystems are memory, I/O and parity. Each of the three major sections of a machine is directly controlled by a field or combinations of fields in the microcode. For a diagram of these fields, refer to Appendix A, Figure 2. In addition, each major section has features and capabilities that are only controlled by the microcode and functionalities that are completely and totally implemented in hardware.

2.1.1 Control Logic

The control logic is illustrated in Figure 2-1. The functionality implemented in the control logic includes sequencing of the microcode to the next instruction or returning to the fetch cycle. A microcode trap capability is available that uses a three-deep push-pop stack and can interrupt normal microcode sequence on any one of 16 potential events. Finally, the control unit can be used for subroutine linkage via the same three-deep stack that is used by the microcode trapping capability. The primary purpose of the control logic is to obtain the control word to be executed in the next read-only memory cycle, and have it read into the register control memory (RCM) at the end of the execution of the current cycle. The signal TRCML is used to change the state of the control memory. The microcode control of the control unit is primarily through fields 2, 11 and 12.

Figure 2-1

Control Logic



Glossary:	BCS	Stack source Bus
	BCY	Control Memory Address Bus
	BD	D Bus
	RCM	Control Memory (current instruction)
	RCS	Three deep push/pop stack
	RCY	Control Memory address register
	Decode Net	User instruction Microcode entry point mapping logic
	Trap address	Micro-trap entry point
	TRCML	"Load Control Memory" signal
	RCC	Control Memory Store

2.1.2 Controlled Logic

The controlled logic forms the main body of busses, registers and logic operators to be used in performing the data manipulation desired. Figure 2-2 illustrates the major components of the controlled logic. The basic structure consists of two busses, bus B and bus D with registers and arithmetic and logic unit (ALU) suspended in between. All data paths are 16 bits wide with parity per byte for a total of 18 bits, unless otherwise indicated on Figure 2-2. The microcoder has available in a given micro-instruction three registers for holding in process data: RY, RM and the selected register file register. The register file can be read and be written into the same micro-step. However, only one register within the file can be operated on in the same micro-step. Shifting is accomplished by selecting the appropriate path from the ALU to the D bus. The arithmetic and logic unit is the standard 72-181 TTL device. This means it can add and subtract the full 16-bit numbers in parallel, it can increment the A input or the B input, and it can perform 16 logical functions on the A and B inputs. The microcode fields that control this logic are fields 1, 2, 4, 5, 6, 7 and 8 with a little bit of help from the others.

2.1.3 Major Subsystems

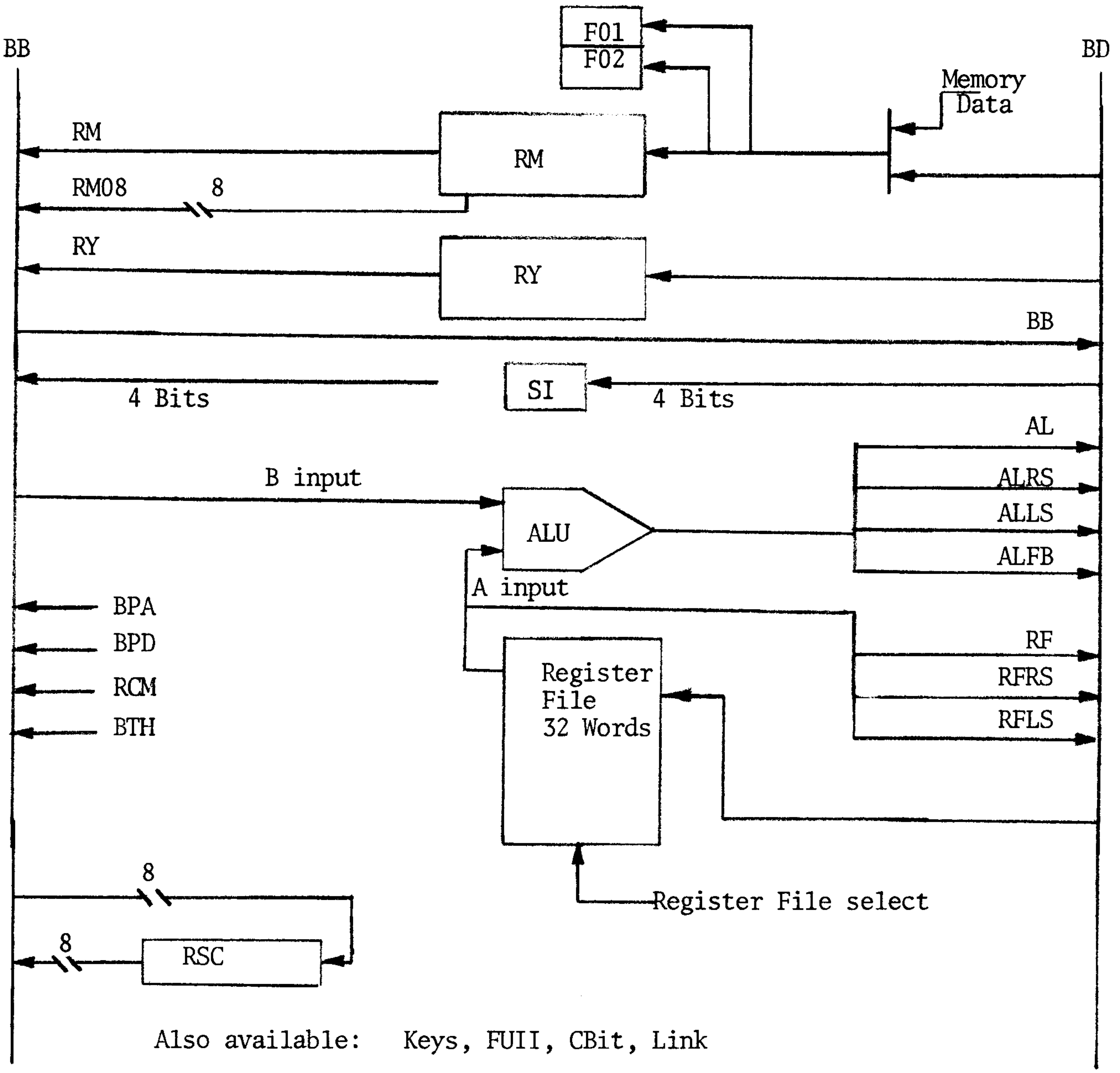
The memory subsystem adds two busses to the control logic in Figure 2-2. These are the BMD or Memory Data Bus and BMA or Memory Address Bus. Figure 2-3 illustrates the components of the memory subsystem. Note that the paging hardware shown requires microcode support in order to implement a virtual memory system. Microcode field 9, primarily, controls the memory subsystem. The clock field (#8) controls the time when data is taken from the memories and put into RM. The basic memory control timing signals, and the complete activity of memory referesh, is controlled by hardware in the box labeled Memory Timing in Figure 2-3.

The input/output subsystem illustrated in Figure 2-4 includes the two bidirectional buses, Bus Peripheral Data (BPD), and Bus Peripheral Address (BPA) in addition to a hardware box that generates the various input/output control signals under control of microcode fields 6 and 7.

The parity subsystem is entirely implemented in hardware, except it can be enabled and disabled using RCM bit 7 and/or PARIM. This is evident from Figure 2-5. Logic consists of a number of independent parity error detection circuits with their outputs ORed together to determine if any of them has discovered a parity error.

Figure 2-2

Controlled Logic

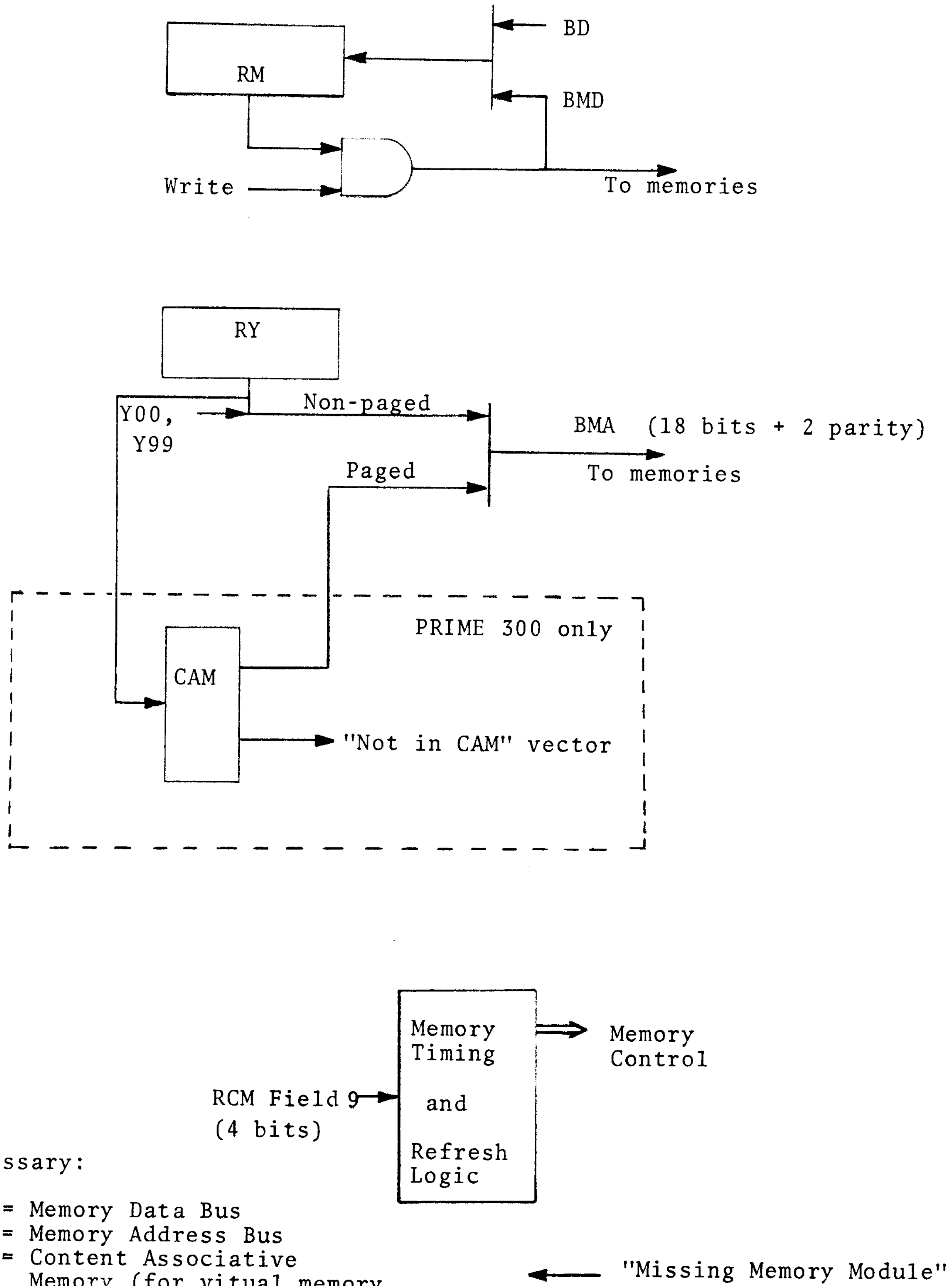


Also available: Keys, FUII, CBit, Link

Glossary:		
	BB	B Bus
	BPA	Peripheral Address Bus
	BPD	Peripheral Data Bus
	XXLS	XX Left Shift
	XXRS	XX Right Shift
	ALFB	ALU byte interchange
	RSC	Shift counter
	BTH	Top hat bus (not now used)

Figure 2-3.

Memory Subsystem



Glossary:

BMD= Memory Data Bus
BMA= Memory Address Bus
:AM= Content Associative
Memory (for vitual memory
Logical physical mapping.)

Figure 2-4

I/O Subsystem

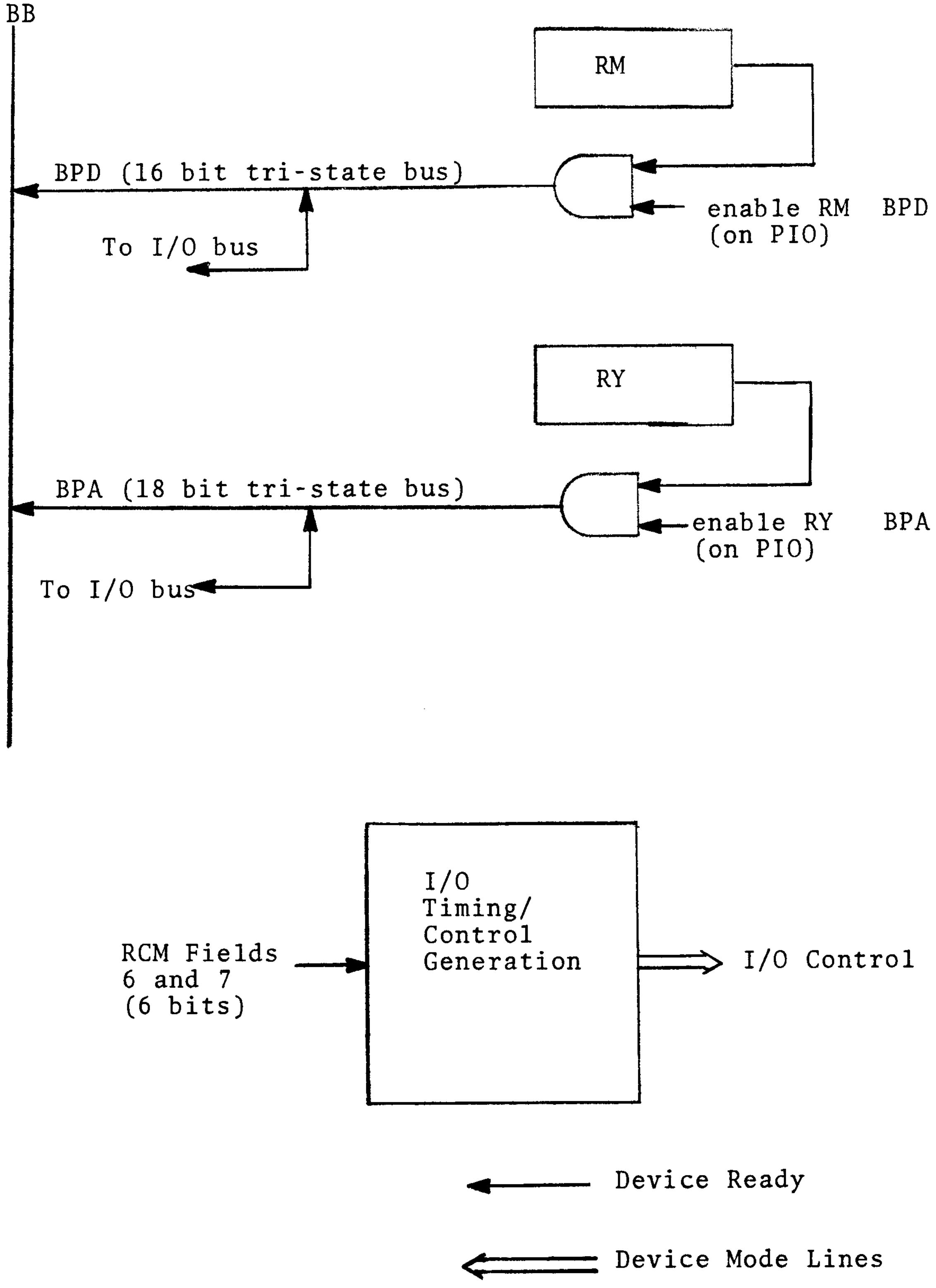


Figure 2-5

Parity Subsystem

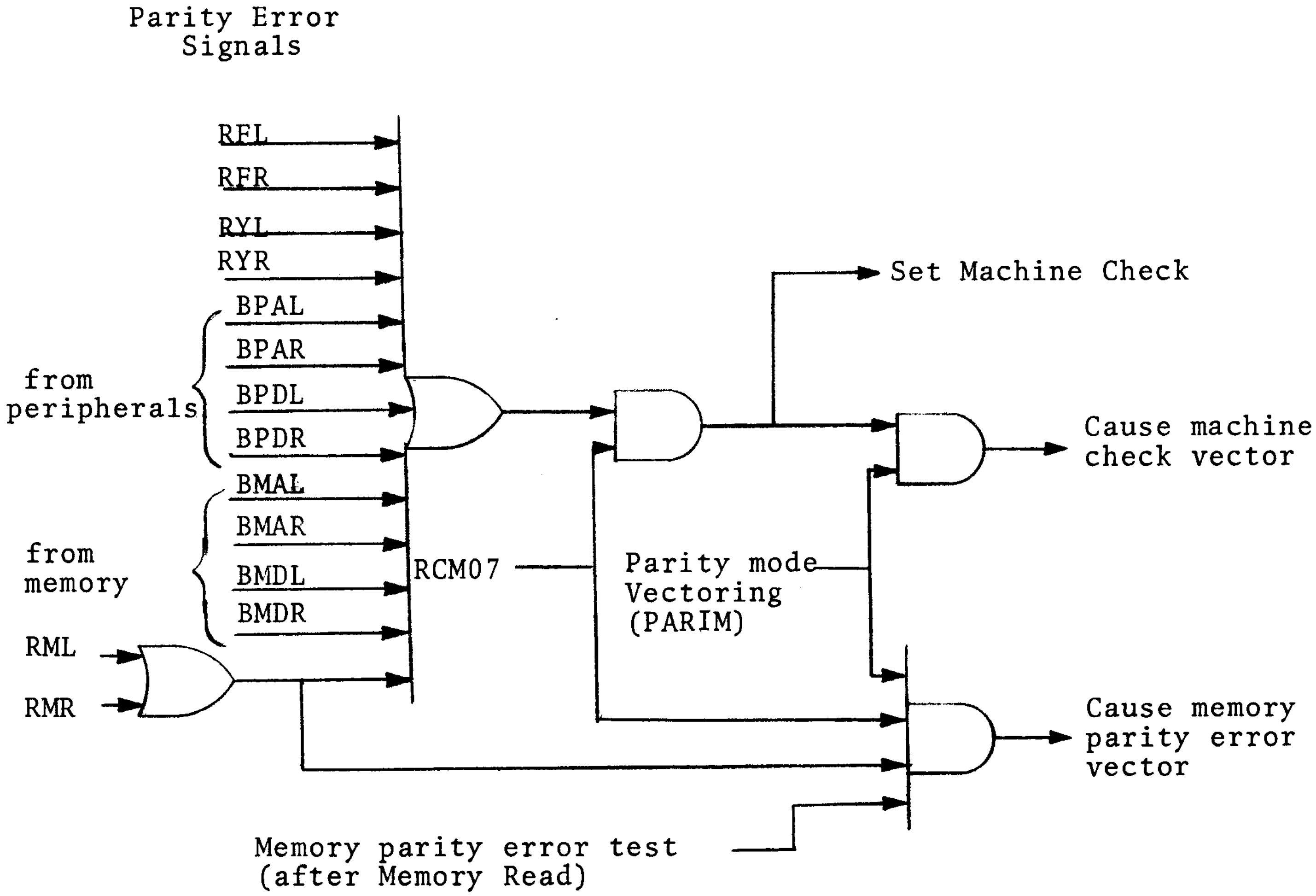
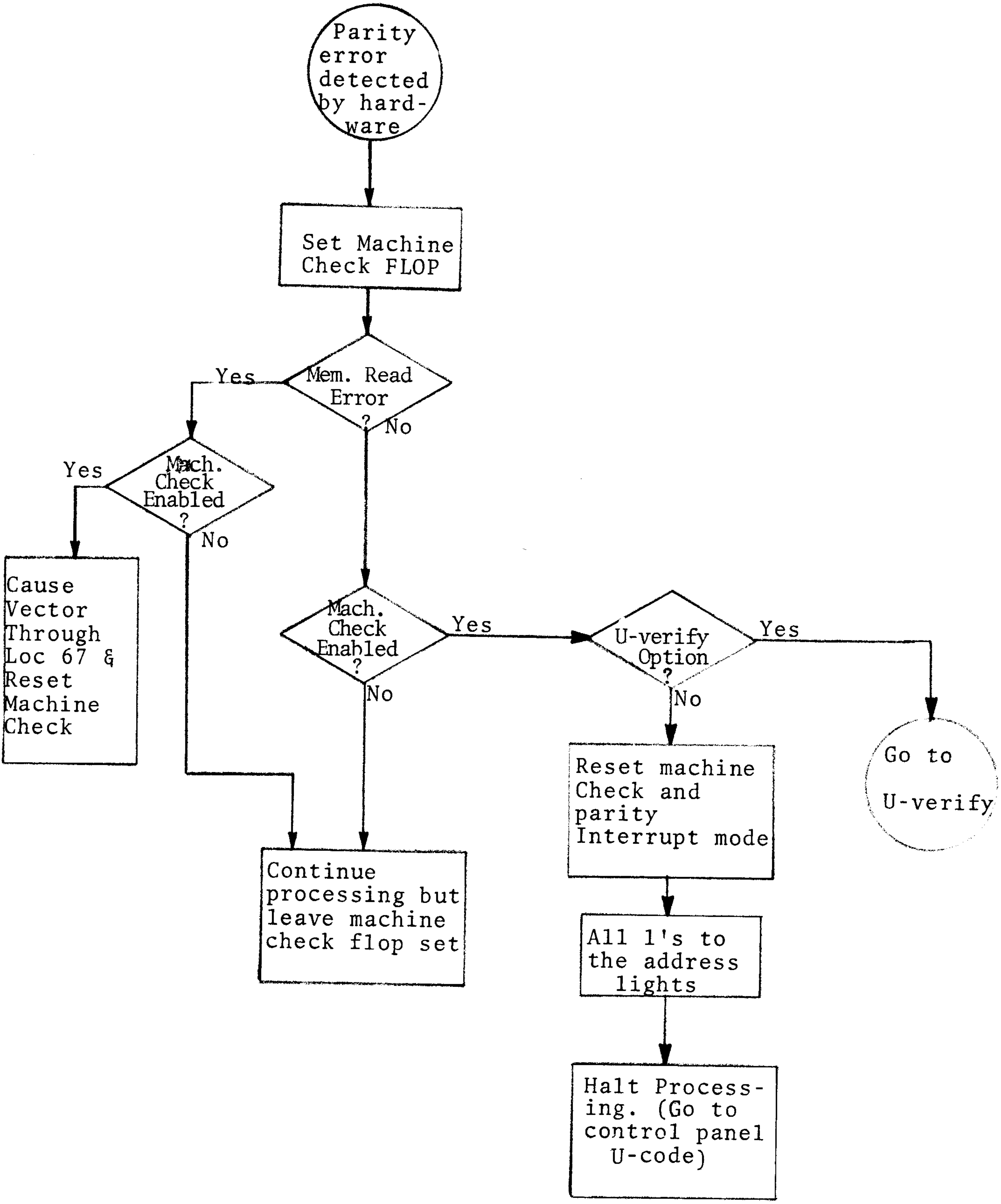


Figure 2-5.1

PARITY DETECTION LOGIC DIAGRAM


Parity error detected by hard-ware
Set Machine Check FLOP
Mem. Read Error ?
Yes
No
Mach. Check Enabled ?
Yes
No
Cause Vector Through Loc 67 & Reset Machine Check
Mach. Check Enabled ?
Yes
No
U-verify Option ?
Yes
No
Go to U-verify
Reset machine Check and parity Interrupt mode
Continue processing but leave machine check flop set
All 1's to the address lights
Halt Process-ing. (Go to control panel U-code)

2.2 INTRODUCTION TO MICROCODE

2.2.1 Register-to-Register Transfers

The simplest microcode operation is transferring information from one register to another register. The first example is the transfer of a register in the register file to RM. In order to do this, the appropriate register must be selected; the register file must be allowed on the Bus D; and, finally, the results must be put into RM. In this example, it is assumed that RA was the register to be selected. Section 2.2.2, Example 1 shows the microcode fields that must be encoded to perform the desired operation: transfer of register A to register M. The unfilled fields have no effect on this operation. Also, in Example 1, the abbreviated form that indicates the same operation using the µ-code (microcode) assembler is shown below the listing of each field.

Figure 2-6, the short dashed lines show the path that the data takes on this transfer.

For the next example, it is desirable to take information in RM and move it back down to RA in the register file. To do this, RM is selected as the source of bus D; and, finally, the register file must be used as the destination of the information on bus D. As Section 2.2.2 Example 2 shows fields 1, 2, 6, 7 and 8 in the microcode must be used to select this particular data transfer. Again, none of the other fields are used. The short-hand or RR form of expressing this transfer is also illustrated. The RM to register file (RA) transfer is illustrated using dotted lines on Figure 2-6.

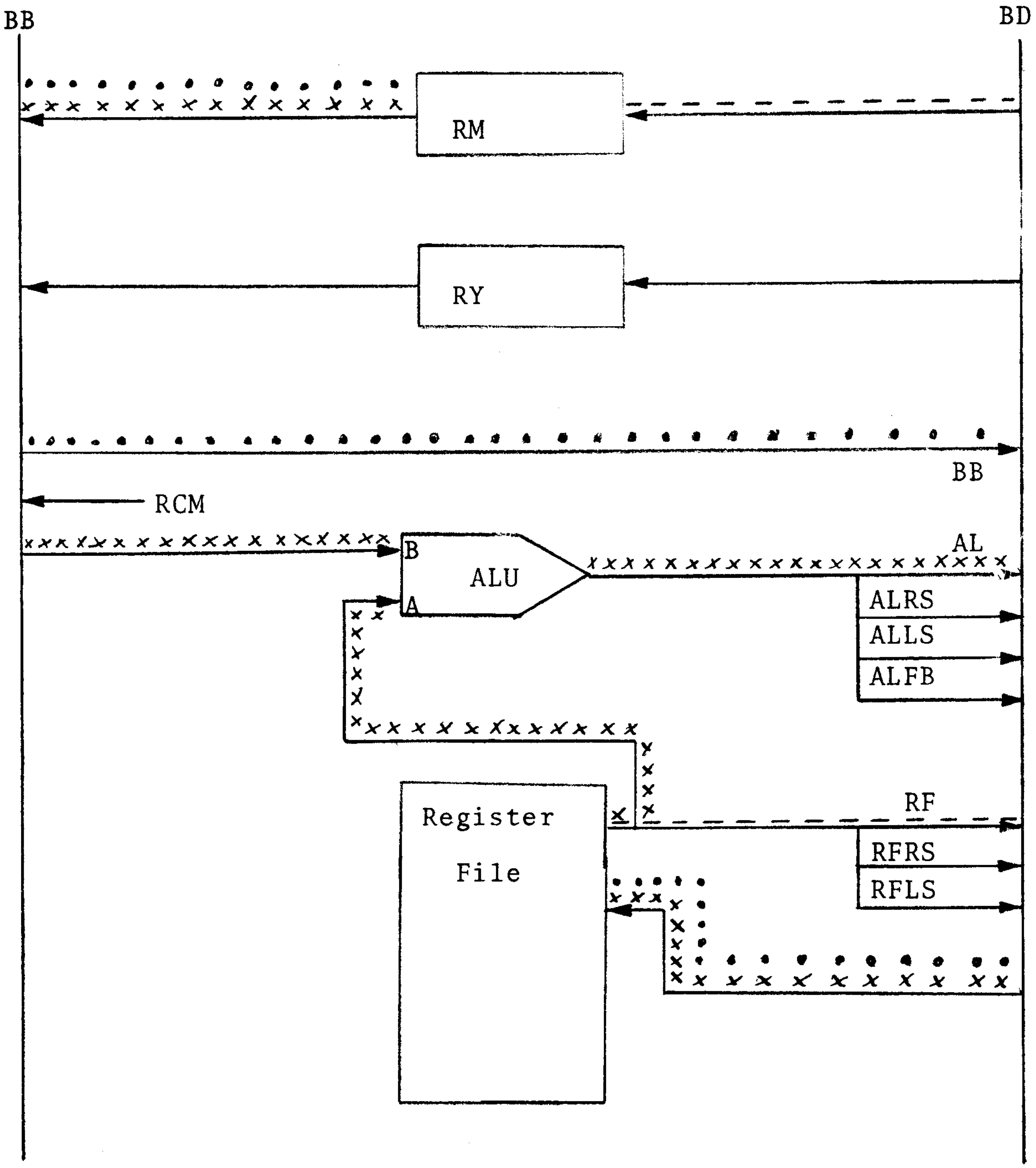
It should be clear from looking at the fields used in these two examples that microcode field 1 selects the source on the D bus, field 2 selects the source on to the B bus, fields 6 and 7 combined select a particular register in the register file, and field 8 determines the destination register from the D bus.

With this much information, it is possible to construct the entire set of register-to-register operations. Section 2.2.2 Example 3 shows the microcode sequence that interchanges the information in register A and register B. These two registers are assumed to be two of the 32 registers in the register file. The execution of code sequence accomplishes the functionality of the Prime instruction IAB.

For a final example of register-to-register transfer, consider putting the value 1 into register A. Numbers, masks and general data that are known by the micro program can be emitted

Figure 2-6

Controlled Logic
Data Paths



on to bus B by using the RCM source on bus B. If the microcode field corresponding to the bus B data is field 12, field 11 must be set to a zero to signal the control unit to interpret field 12 as data. Section 2.2.2 Example 4 shows that microcode fields 1, 2, 6, 7, 8, 11 and 12 are used in order to transmit a 1 from the control memory to register A. This operation is used to execute the Prime instruction LT or Logicize True.

2.2.2 Examples for Section 2.2.1

Example 1:

Field

1	2	3	4	5	6	7	8	9	10	11	12
RF	--	--	--	--	M	RA	RM	--	--	--	--

(Short Form)

RR RA ⇒ RM

Example 2:

BB	RM	--	--	--	M	RA	RF	--	--	--	--

(Short Form)

RR RM ⇒ RA

Example 3:

1	2	3	4	5	6	7	8	9	10	11	12
RF	--	--	--	--	M	RB	RM	--	--	--	--
RF	--	--	--	--	M	RA	RY	--	--	--	--
BB	RY	--	--	--	M	RB	RF	--	--	--	--
BB	RM	--	--	--	M	RA	RF	--	--	--	--

(Short Form)

RR	RB => RM
RR	RA => RY
RR	RY => RB
RR	RM => RA

Example 4:

1	2	3	4	5	6	7	8	9	10	11	12
BB	RCM	--	--	--	M	RA	RF	--	--	DATA	1

(Short Form)

RR	RCM = 1 => RA

2.2.3 Transfers Using the Arithmetic and Logic Unit (ALU)

Suppose that it is desired to add two numbers together, the first of these numbers is in RM and the second of these numbers is in RA. The results are to be put in RA. The first step is to get the correct data at the inputs of the ALU. This is done by selecting RM as the source for the B bus, and RA as the register file register. Next, the ALU must be conditioned to add, which is done by selecting the add function in the combined fields 4 and 5. Finally, the results have to be put back into the register file (RA) by selecting AL as the source of bus D and the register file as the destination of the operation. Section 2.2.4, Example 1, shows the microcode fields that are used to perform this activity. At this point, half of the capability of one microcode instruction is being used. Note the different format available for shorthand indication to the microcode assembler of this addition operation. In general, operations that use the arithmetic and logic unit (ALU) can be requested using the ALU macro of the microcode assembler. The line of X's on Figure 2-6 shows the active data paths for Example 1.

As is obvious from the Section 2.2.4 Example 1, fields 4 and 5 combine to control the Arithmetic and Logic Unit operation that is performed on the incoming data. The complete set of ALU operations are found in Appendix B.

Associated with the Arithmetic and Logic Unit are three flip-flops. The first to be discussed is the carry bit, also called the C bit, or FCBIT. This is a one bit storage register that can be loaded using field 9 of the microcode. The C bit can be loaded with any one of six different signals upon request: arithmetic overflow (AOVFL), shift overflow (SOVFL), but D bit 1 (BD01), RF01, 16-bit ALU carry (COUT), and the link source, (or divide overflow (DIVER)). The other two flip-flops are called the condition code and always contain the high order bit of the results of the ALU in one flip-flop and an indication if the results were equal to zero in the other. Whenever field 10 of the microcode says SET CC, the condition code can then be used in conjunction with the conditional branching logic to make a decision about the relative size of the number; equal to 0, not equal to 0, greater than 0, less than 0, etc.

The division of functions between fields 4 and 5 of the micro-code in controlling the arithmetic and logic Unit may now be inspected more closely. Field 5 is used to specify use of arithmetic or logic mode for the adder. If arithmetic mode is used, field 5 is used for carry-in: 0, 1, or the C bit must be considered. Field 4 directly encodes one of 16 arithmetic or logical functions available in the ALU integrated circuit.

Section 2.2.4, Example 2 is an attempt to put together what has been learned about ALU and RR type microcode operation. This double precision add, assumes that registers A and B are to be treated as a standard double-word number, and the new second double-word number to be added to this register is now stored with the high order half in register 13 and the low order half in the M register (RM). As example 2 shows, it takes four microcode steps to complete a double precision add. In the second step, the high order bit of RB must be masked out. This bit would have been set if there had been overflow on the first add step. In the fourth step, it is possible to use the C bit that was established in the first step; and at the end of the operation, reload the C bit with the new overflow information.

It is also possible to use the ALU to perform single input operations and to generate constants of either minus 1 or 0.

2.2.4 Examples for Section 2.2.3

Example 1:

Field

1	2	3	4	5	6	7	8	9	10	11	12
AL	RM	--	ADD	0	M	RA	RF	--	--	--	--

(Short Form)

ALU RA PLUS RM => RA

Example 2:

1	2	3	4	5	6	7	8	9	10	11	12
AL	RM	--	ADD	0	M	RB	RF	AOVFL	--	--	--
AL	RCM	--	AND	L	M	RB	RF	--	--	DATA	$7FFF
RF	--	--	--	--	M	13	RM	--	--	--	--
AL	RM	--	ADD	CBIT	M	RA	RF	AOVFL	--	--	--

(Short Form)

```
ALU     RB    PLUS    RM => RB     C=AOVFL
ALU     RB     AND   RCM   = $7FFF =>RB
RR      13  => RM
ALU     RA     PLUS   RM + CBIT  => RA
C = AOVFL
```

2.2.5 Shifting

Shifting is accomplished in microcode using the bus D source. RFRS or RFLS are the two inputs most frequently used for shifting. RFRS, for instance, means that each output of the register file is shifted right one place before it is brought to the output of the D bus. The difficulty of shifts is that the information to be shifted in one end can be different coming out, depending on the functionality that is wanted with the shift. In addition, there exists a one bit link that can be loaded with a variety of signals for use in multiple precision shifting operations.

The loading of the link and the various end conditions that are to be used are combined and are called the shift and end conditions. Microcode fields 2, 4 and, to some extent 9, are used to determine the appropriate shift/end conditions. Microcode fields 2 and 4 are available because, in a typical shift, bus B and the ALU are not used.

There is no logical reason for the choice of the bit patterns in fields 2 and 4 and for the end conditions that are generated. As a result, shift and end conditions must be chosen from a table (Appendix A Field 1). There is no choice but to look up the appropriate bits to put into fields 2 and 4 in order to select the particular shift and end conditions for the direction that one wishes to shift. In Example 1, Section 2.2.6, the objective is to do a right rotate of one place on RA and put the results back into RA. From the shift and end condition for bus D select, it is determined that a rotate end condition requires the selection of the 2, in field 2, and a 0 in field 4. Fields 1, 6, 7 and 8 are filled in as required to get the correct bus activity. For shift operations, there is no equivalent to the ALU and RR short cut macros, and the entire microcode word must be written out. These are called CPU macros. Fields not used in the CPU macro must have the default microcode selection put in their place. Fields 3 and 10 are the only ones that have default values other than 0. Default for field 3 is a 2 and the default for field 10 is a 4. Section 2.2.6, Example 2, illustrates a double word long logical left shift (LLL) of one place. In this case, it takes two instructions to accomplish the shift because only 16 bits can be shifted at one time. However, it is important that the bit that is shifted out of the low order half (in this case, RB) be saved for shifting into the high order half on the second half of the shift. To make this easy to accomplish, there is a single bit register called the Link that is loaded from a source selected by the same shift and end condition fields that have already been used (fields 2 and 4). However, the Link is loaded only if field 9 is being used to select one of the sources for the carry bit or if the selection is LINK. (Numerically, codes 1 through 7 enable the Link to be loaded.) In the example, the carry bit is loaded with the high order bit of RA that would otherwise be lost in a single left shift. The proper end condition for the RB shift should put a 0 into bit 16, shift bit 2 into bit 1, and load the Link with bit 1. From the table of shift and end conditions (Appendix A Field 1) the first shift and end condition shown (encoding a 7 in both field 2 and field 4) provides the proper results. For the high order shift, the Link is loaded into bit 16, and RF02 is put into RF01. A shift and end condition that accomplishes this is a 3 and a 7 in fields 2 and 4, respectively. Note that a 3 and 6, or a 3 and 9 shift combination would have worked as well, because it does not matter what value goes into the Link bit.

To count the number of shifts made, a general purpose 8-bit scratch shift counter register (RSC), was created outside of the register file. This counter can be loaded from bus B and can be read in as the low order half of bus B. Field 10 of the microcode (the Independent Action Code field (IAC)) is used to control the loading and counting of the shift counter. There is an IAC called load shift count (LOADRSC) and another for incrementing the shift count (INCRSC). The shift counter incrementation takes place at the end of the ROM cycle.

Thus, if incrementation and testing are done in the same ROM cycle, the test reflects the count before incrementation. In Section 2.2.11, Example 1 shows how the shift counter can be used in conjunction with counting the number of times to loop on a shift.

2.2.6 Examples for Section 2.2.5

Example 1: Right Rotate

1	2	3	4	5	6	7	8	9	10	11	12
RFRS	2	--	0	--	M	RA	RF	--	--	--	--

(Short Form)

```
CPU    RFRS   2   2   0   0   M   RA ;
       RF     NOP NOP
```

Example 2: Long Left Shift

1	2	3	4	5	6	7	8	9	10	11	12
RFLS	7	--	7	--	M	RB	RF	LINK	--	--	--
RFLS	3	--	7	--	M	RA	RF	RF01	--	--	--

(Short Form)

```
CPU    RFLS    7    2    7    0    M    RB ;
       RF     LINK
CPU    RFLS    3    2    7    0    M    RA ;
       RF     RF01
```

2.2.7 Branching and Subroutining

One of the primary advantages of the Prime micro-processor design is the capability to test the results of a previous operation and conditionally jump on those results while at the same time performing a new and different operation using the controlled logic.

There are a total of 64 selectable jump conditions. Of this set, about 50 are implemented. They are listed in the microcode field description under Appendix A, Field 11. Several have already been discussed; they include the two condition codes, the carry bit, and the shift counter.

In Section 2.2.8, Example 1, RA is loaded with a 0 if RA<RM, a "1" if RA = RM, and a "2" if RA>RM, through the use of conditional branches. It is necessary to test for arithmetic overflow on the original subtract, and no destination clocking is selected. The symbol used in the destination clocking field (field 8) for this is clock (CLC).

The calling of subroutines in the microcode is accomplished by using the three-deep push down stack. Two instructions are required to enter a microcode subroutine, and one to return. The two entry instructions are: first, to load the microcode stack; and second, to branch to the subroutine. The return is made using the special operator in field 11. When this special operator is used, indicated by an S, all of the conditional branches are still available, and with the branch condition false, the next sequential instruction is executed. However, if the branch condition is true, one of a number of special actions is taken. Example 2 of Section 2.2.8 shows the standard subroutine linking using the microcode stack. Another IAC is used to force the loading of the microcode stack.

Two other capabilities that exist, using the special operator in field 11, are the 16-way branch and the N-way branch. The 16-way branch is similar to a regular jump except that the low order four bits of the jump address are taken directly from bus D. This means that any of 16 different locations is entered starting from the address specified taken modulo 16. As an example: if the location to be branched to is specified as T3 that is assigned to location hexadecimal 112, then; for hexadecimal number 0 through F on the low order four bits of the bus D, the next microcode instruction to be executed is taken from location 110 through 11F hexadecimal. Section 2.2.8, Example 3, shows an example of the use of a 16-way branch.

The multi-way branch permits any form of decode to be done across an arbitrarily large microcode address space. In this case, the entire D-bus serves as the address for the next microcode instruction to be executed. Depending on the value of bus D, any one of the 4K addressable microcode words can be accessed. It is expected that this capability be used, in general, by building a microcode decode word. This can be done by masking out the low order four or five or six bits and then ORing in the low order decode bits. Section 2.2.8, Example 4, shows how a 64-way branch on the low order bits of RA can be implemented, giving a start of execution after the branch beginning at T4.

2.2.8 Examples for 2.2.7

Example 1:

*Compare RM vs RA If RA > RM, 2 => RA

If RA = RM, 1 => RA

If RA < RM, 0 => RA

1	2	3	4	5	6	7	8	9	10	11	12	
--	RM	--	SUB	1	M	RA	CLC	AOVFL	SETCC	--	--	
--	--	--	--	--	--	--	CLC	--	--	JUMP	FCBIT	T1
AL	--	--	ZERO	L	M	RA	RF	--	--	JUMP	LT	EXIT
AL	--	--	INC	1	M	RA	RF	--	--	JUMP	EQ	EXIT
AL	--	--	INC	1	M	RA	RF	--	--	JUMP	TRUE	EXIT
AL	--	--	INC	0	M	RA	CLC	--	SETCC	JUMP	TRUE	T2

(Short Form)

```
                    ALU     RA   MINUS    RM  => NULL    C=  AOVFL ;
                            SETCC

                    RR      NOP => NOP    NOP   JUMP ON FCBIT TO T1
          T2        ALU     CON    0  => RA    NOP   JUMP ON LT TO EXIT
                    ALU     INC   RA  => RA    NOP  JUMP ON EQ TO EXIT
                    ALU     INC   RA  => RA    NOP  GO TO EXIT
          T1        ALU     INC    RA + 0  => NULL   SETCC GO TO T2
```

Note: Step T1 is necessary because overflow occurred on the first step. The only way overflow can occur on a subtract is if the signs are unlike. Therefore, the sign of RA alone is sufficient to determine the results of the comparison.

Example 2: Subroutine two's compliments RA and returns

1	2	3	4	5	6	7	8	9	10	11	12
BB	RCM	0	--	--	--	--	CLC	--	PUSHBD	Data	RTNA
--	--	--	--	--	--	--	CLC	--	--	Jump TO TCA	

RTNA is the return address
continue code

TCA

AL	--	--	NOT	L	M	RA	RF	--	--	--	--
AL	--	--	INC	1	M	RA	RF	--	--	S	TRUE POP

(Short Version)

```
                         RR     RCM  =   RTNA => NULL     PUSHBD
                         RR     NOP  =>  NOP     NOP    GO TO TCA
RTNA  (Return Address)
TCA                      ALU    NOT       RA   =>  RA
                         ALU    INC       RA     => RA  NOP  S   ON TRUE, POP
```

2.2.8 Examples for 2.2.7 (Cont)

Example 3:

1	2	3	4	5	6	7	8	9	10	11	12
RF	--	0	--	--	M	RA	--	--	--	S TRUE	16WAYS T3

Example 4: 64 Way branch starting at T4

AL	RCM	--	AND	L	M	RA	RM	--	--	DATA	$3F
BB	RM	--	--	--	M	13	RF	--	--	--	--
AL	RCM	--	ADD	0	M	13	RF	--	--	DATA	T4
RF	--	0	--	--	M	13	--	--	--	S	TRUE, BD

ort Version)

```
ALU   RA         AND    RCM   = $3F  => RM
RR    RM    =>   13
ALU   13         PLUS   RCM   =  T4  => 13
RR    13    =>   NULL   NOP   S       TRUE, BD
```

2.2.9 Traps

Field 3 controls the enabling of the trap logic in the Prime microcode. Of the 16 total possible microcode traps available, 15 are placed in one class and the other trap (DMX) is placed in another class. Each of the two bits in microcode field 3 controls the trap associated with it. If the bit is set, the traps are enabled for that ROM cycle; and if the bit is reset, the traps are disabled. The mnemonics for controlling these are shown in Appendix A, Field 3.

The Appendix D describes each of the traps and, in general, the interrupts they generate. However, there are a few rules that the microcoder must keep in mind with regard to the trap.

Rule #1: in general, all traps but DMX are always enabled. This is automatically selected by the RR and ALU macros unless specifically requested otherwise.

Rule #2: DMX traps must be permitted no less frequently than once every 1.5 microseconds or system DMX latency is compromised.

Rule #3: traps must be disabled if bad parity can be generated as a result of a particular operation, and that parity must be corrected by running the offending data through the adder in any of the adder's modes before traps can be re-enabled. Bad parity can be caused by the following:

1. Willful generation of bad parity using the RCM EMIT pseudo-op instead of the DATA pseudo-op in the CPU macro.

2. Inputting data from device address 20.

3. Inputting serial interface.

4. Reading from a non-existent memory location, or a location already containing bad parity.

Rule #4: When the microcode stack is explicitly pushed in a microcode instruction, the trap logic must be disabled, because the control logic can only push one item into the stack in one micro-step. The trap logic must save the microcode return point in the stack so the PUSHBD independent action code and the DECODE step must both have traps disabled. Similarly, if a multi-way or 16-way branch is executed, no returnable trap can be permitted (DMX, Page, or Address).

Trap codes:

0	None
1	DMX (only)
2	NX (all but DMX)
3	ALL

for RR and ALU macros, use TR= if traps different from the explicit traps are to be specified.

Rule #5: if a 160 nanosecond clock is selected, traps must be disabled. The reason for this is: the control unit cannot successfully determine if there is going to be a trap in 160 nanoseconds. Section 3, microcode timing in greater detail.

The Restricted Execution trap feature is available using field 10 codes RXM and RXMF. Use of these IAC's forces a trap if the µ-code step is executed while the machine is running in Restricted Execution or virtual mode.

Restricted Execution traps occur after the execution of the instruction following the one that had a field 10 of RXM or RXMF. If traps are disabled in the next instruction executed, the RXM trap is missed. Careful attention must be paid to all possible sequences (Address traps, DMX breaks, etc.) to ensure that the RXM trap unequivocally occurs.

Section 2.2.10 Multiply and Divide

Special hardware in the Prime Computer enables the more rapid execution of multiply and divide than would otherwise be possible. The special multiply logic permits the automatic selection within a microcode step of either the arithmetic unit output or the register file output, depending on whether the link contains a 1 or a 0 at the beginning of the cycle. This special hardware permits two microcode steps executed 15 times to perform a full 16-bit multiply leaving a 32-bit result. The algorithm employed is always shift, adding only if the bit of the multiplicand is a 1. Section 2.2.11, Example 2, shows the set up and operation of the standard muliply loop.

A new feature of the microcode must be introduced in order to be able to activate the special multiply logic. This feature is called the emit-action code (EAC). It is analogous to field 10, except the emit field must be dedicated in order to use the emit-action code. If field 11 is coded as EAC or a 2, then field 12 can contain an emit-action.

Division is not so straightforward. However, there is also some special hardware that permits a non-performing style of divide to be implemented more efficiently. This hardware is similar to the special hardware used by the multiply. It is turned on from the microcode using an emit action code, and it also causes an automatic selection of the arithmetic unit or the register file on the D bus. The difference, however, is that the divide logic monitors the high order output bit of the ALU and compares it to the previous high order ALU bit stored as part of the condition code. If the two bits are the same, then the ALU output is selected. If the bits are different, the register file output is selected. The correct shift and end conditions are selected if either an add or a subtract operation is being requested of the ALU.

Divide is more complicated than multiply because division by unlike signed numbers does not map as nicely as multiplication by signed numbers. A complete divide, including a properly signed remainder, involves a fair amount of clean up or end condition fix up at the completion of the basic divide loop. For details of what is required, see Appendix C "Microcode Listings" for the divide. However, it is possible to demonstrate the basic inner divide loop. Section 2.2.11, Example 2, illustrates this loop for divide.

A non-performing divide was selected because the decision to perform the arithmetic operation or the shift on a bit-by-bit basis is faster than a standard restoring divide and requires much less microcode than a standard non-restoring divide. Non-performing divide means that the sign bit of the attempted arithmetic operation is monitored. If the sign of the remainder being accumulated would be changed by allowing the operation to be completed, the operation is not performed. Instead, the partial remainder is shifted one place to the left. For a more thorough explanation of various ways to divide using two's complement arithmetic, refer to Flores[1].

The divide logic information, whether to perform the subtract, is also significant for more than switching the source of the D-bus. If a quotient bit of one is obtained and the subtract is performed on the first iteration (assuming like signed numbers), then the division has had an overflow. Because the information on whether the first arithmetic operation was performed indicates divide overflow, it is available as one of the conditions selected to be loaded into the C bit. Finally, the perform-or-not information is, in effect, the quotient bit for that cycle of the divide algorithm. This is available to be loaded into the link if the standard divide microcode step is used. The link can be emptied into a separate register (or back into the low order register as in the standard divide).

I. Flores, Ivan: Logic of Digital Arithmetic
Prentiss Hall, Inc., 1963

2.2.11 Examples

Example 1 MPY RB * RA:

```
        RR    RA => RM

* Load shift counter = - 15 and zero out RA

        CPU    AL   RCM   NX   ZERO L M RA;
               RF   NOP   LOADRSC;
               DATA   - 15

* Now Load link with LSB of RB

        CPU    RFRS   5   NX   6   0   M   RB;
               RF   LINK

* Main MPY Loop - first step uses special MPY aid

* Switching between ALRS and RFRS depending on the
* state of the link

        CPU    ALRS   RM   NX   ADD 0   M   RA;
               RF   LINK   NOP
               EAC   MPYLOGIC

        CPU    RFRS   0   ALL   6   0   M   RB;
               RF   LINK INCRSC;
               JUMP   ON   RSCNEM1   to   * -1


* Multiply Loop is finished except for final Subtract
* (with no shift).  The same special logic can be used
* again.

        ALU    RA Minus RM => RA   c = AOVFL;
               NOP   EAC   MPYLOGIC
```

Example 2:

```
     Divide  RA|RB  (positive)/RM (positive)

             Quotient  => RB
             Remainder => RA

* Set condition code = positive, Divide overflow
* is detected if first subtract yields a positive
* result because 16 magnitude bits (or more) are required
* to hold correct results and we only have 15.

     CPU    - RCM  NX  ZERO  L  0  0;
            CLC  NOP  SETCC;
            DATA  -15

* Test for error (overflow)

     CPU    ALLS  RM  NX   SUB 1  M  RA;
            CLC  DIVER  NOP;
            EAC  DIVLOGIC

* Pre-load link - exit on error

     CPU    RFLS  3  ALL  3  0  M  RB;
            RF  LINK  NOP;
            JUMP ON FCBIT TO DIVER

* Main loop - 15 iterations, 1 quotient bit/cycle
* first must = 0.

     CPU    ALLS  RM  NX  SUB 1 M  RA:
            RF  LINK  NOP;
            EAC  DIVLOGIC

     CPU    RFLS  3  ALL  3  0  M  RB;
            RF  LINK  INCRSC;
            JUMP  ON  RSCNEM1  to  * - 1

* final iteration must not shift remainder.  First ins.
* sets up link.

     CPU    ALLS  RM  NX  SUB 1  M  RA;
            200  LINK  NOP;
            EAC  DIVLOGIC

     CPU    AL  RM  NX  SUB 1  M  RA;
            RF  NOP  NOP;
            EAC  DIVLOGIC

     CPU    RFLS  3  ALL  6  0  M  RB;
            RF
```

SECTION 3

TIMING

3.1 INTRODUCTION TO TIMING

One of the relatively unique features on the Prime microcode is the ability for every microcode step to specify the time needed for that step to execute. The clock control microcode field (field 5) permits any 16 combinations of destination registers and time. Timing is specific to a particular microcode processor and also specific to the microcode address space from which code is being executed. As a result, the timing for a particular instruction is processor dependent. Because the previous sections have been general, no clock or timing information has been included. On both a Prime 200 and Prime 300 Central Processor, the various clocks available range from 160 ns up to 280 ns. If the microcoder uses the CPU Macro, a decision must be made as to which of the clocks is appropriate to the activities commanded by the other fields in that microcode instruction. In addition, if an operation must take longer than 280 ns, it is possible to have two identical microcode instructions differing only because the first has a 200 or 280 ns microcode clock without changing any of the registers. This can permit clocks of 440 or 480 or 400 ns to be a construction of two successive microcode steps.

The clocks also control the destination registers so that a typical clock would be RM200 to specify that register M should be updated 200 ns after the start of the cycle. For the full list of clocks for a Prime 200 and a Prime 300, see Appendix A, Field 8. The Prime 100 clocks control the same destination registers as those for the Prime 200 but the times for all clocks are 360 ns for a single step.

Another functionality within the clock field is automatic coordination with memory. It is possible to select a clock into RM on MRDY. This forces the memory data bus to be loaded into RM after the memory in access is complete. Similarly, it is possible to load RY conditional on Y busy. This delays the changing of state of RY until the memory is finished with that register. However, there are minimum times associated with these clocks. Even if the memory cycle was started several micro steps in the past, there is still a minimum time associated with the particular ROM cycle.

3.2 TIMING CONSTRAINTS

Different models of the Prime family have different basic timing constraints. The basic timing constraints for different functions are as follows:

> For the control unit on a Prime 200, if traps are disabled (TR = 0 or None), and there is no conditional branch specified, the minimum time to perform an activity is 160 ns. If traps

are enabled or if a conditional branch is permitted, the minimum time lengthens to 200 ns. Finally, in the control unit, if an N-way or 16-way branch is attempted, the minimum time becomes 280 ns.

For the controlled unit, times are associated with register-to-register, ALU, multiply/divide, and shift operations; each of which could take a different length of time. For the Prime 200, those times are, respectively: 160, 280, 400 and 240 ns.

For the Prime 300, the control unit has the same restrictions as for the Prime 200. However, the controlled unit has been speeded up so that ALU operations and shift operations take 200 ns; multiply, 240; and divide, 280 ns.

RY destinations on a Prime 300 are loaded 40 ns early, permitting memory access times to be improved by 40 ns in many cases. Unfortunately, this loading also means that if RY is both the source and destination of a single micro-step, the results of the adder or BB cannot be guaranteed to be stable at the end of the micro-step. This means that the C-bit, condition code, the shift counter, or the register file could be loaded with bad information.

Examples:

```
       GOOD                                 BAD

   ALU  INC   RY => RY               ALU  INC   RY   => RY SETCC

   ALU  RA PLUS  =5  => (RY,RF)      ALU  RA PLUS RY  =>  (RY,RF)
   ALU  RA PLUS  RY  => RY           ALU  RA PLUS RY  =>  RY C = AOVFL
   ALU  RA PLUS  RY  => RA SETCC
```

For extended microcode, whether it is in WCS or fast or slow P-ROM, the control unit remains the same. However, the control unit remains the same with the fast P-ROM, and 80 ns slower in the slow P-ROM and WCS portions of the extended control store. These extensions cause some of the longer controlled clocks to become over 280 ns, the maximum specifiable in one cycle. When the control unit must go to multiple cycles, a first cycle becomes the minimum CLC and the second cycle must be a 280 ns clock. Table 3-1 summarizes these items.

The Prime 100 is very slow and the times shown may not be minimum, but they are the only times available.

Some microcode activities have special timing constraints associated with them, this includes the PUSHBD independent action code which requires a minimum 280 ns clock. JAMF and any sort of POP off the stack require the same clock as is used for the conditional branch. The decode step requires a minimum 280 ns clock, except for the Prime 100 that requires a 360 ns clock. If traps are enabled, a minimum clock of 200 ns must be chosen.

Table 3-1

Required Minimum Micro Instruction Times

WCS Micro Instruction Times

1. The following operations require 240 ns clock times.

   A. Register to Register Operation
   B. ALU Operations
   C. Multiply Operations
   D. Shift Operations

2. These operations require 280 ns clock times.

   A. Microcode Branch Operations
   B. JAMF Operation
   C. Divide Operation

3. The 16-way branch and NOTRF16 branch require 480 nsec.

	CONTROLLED UNIT					CONTROL UNIT				
Processor/ Extension	RR	ALU	MPY	DIV	Shift	No Branch	*2 Branch	16-Way N-Way*2 Branch	JAMF	Not RF16 Branch
100*1	360	360	360	720	360	360	360	360	360	NA
200	160	280	280	400	240	160	200	280	200	280
FPROM	200	280	280	400	240	200	240	440	240	280
300	160*3	200*3	240	280	200*3	160	200	280	200	280
FPROM	240	240	240	280	240	240	280	480	280	480
WCS	240	240	240	280	240	240	280	480	280	480

*1 All single instruction clocks are 360 ns despite requests for other clocks.

*2 Some branches off-board can be faster.

*3 RY destinations are 40 ns longer.

When working with the microcode assembler, if the CPU macro is used, the clock chosen must be explicitly specified. However, if the RR or ALU macro is used, the fastest clock possible always is chosen by the particular macro. For example, for a Prime 200, the RR macro chooses a 160 ns clock and disabled traps, if there is a destination clock of 160 ns available. If, however, a conditional branch or a "go to" statement is specified, then the RR macro automatically chooses a 200 ns clock and enables traps (TR = NX). If traps are specified equal to something other than NONE or 0, then a 200 ns clock is selected and the traps are put as they are specified.

The ALU macro operates in a similar fashion. On the Prime 200, a 280 ns clock is selected and traps are set to NX. In the Prime 300, a 200 ns clock is selected for the ALU macro, unless the destination includes RY, in which case a 240 ns clock is chosen.

To aid in the selection of the correct clocks for the ALU and RR and CPU macros, microcode intended to execute in different portions of P-ROM must define the label P300 equal to a value of 0, 1, 2, 3, etc., depending on where the microcode is to be executed. For more details, refer to Section 8.1.3 on the microcode assembler and preparing microcode for assembly.

With the information discussed so far, it is now possible to precisely calculate the time it takes to execute any microcode algorithm (except for the times spent waiting for high speed memory to finish). Memory timing is covered in the next section. There are two other factors that must be considered. The first is trap extension time, and the second is memory refresh time. Again, the memory refresh is covered in the next section. However, the trap extension time is important for any algorithm that might use address or page traps or DMX operations. If the trap logic requests a break in the microcode sequence, the cycle being executed is automatically extended to 360 ns in order to allow the trap address to be propagated to the address lines and the return address to be saved on the stack. This is why it is a good microcode practice to permit DMX break on instructions that do a memory read and, therefore, already take longer than the 360 ns cycle that occurs if a DMX trap is required.

SECTION 4

OPERATING WITH MEMORY

The Prime CPU has been designed to allow easy microcode synchronization with high-speed memory. Memory cycles are requested using microcode field 9 which is also used to control the carry bit and the link bit. It is possible to request a 16-bit (word) read or a write on either of the left or the right byte or on both bytes. These memory operations can be either mapped or absolute. The memory cycle begins at the beginning of the microcode step that requested it. Time for a memory cycle is determined by the processor and memory configuration of a particular system.

Table 4-1

Memory Cycles

Mem. Op.	P100	P200	P300	
			Fast	Slow
Read acc.	680	600	440	600
cyc.	800	720	640	720
Write	920	840	800	840
Refresh	840	840	800	840
Paged Read Acc	-	-	520	680
Cyc.	-	-	720	800
Paged Write	-	-	880	920
Overlapped Write Pos?	No	No	Yes	Yes

Mapping slows down memory when paging is enabled.

With overlapped write, the memory may be started in the same cycle that RM is loaded.

Access time is defined to be the time from the start of a μ-code command of a memory read to the time the data is clocked into RM from memory.

Cycle time is the time memory takes to do a full cycle and be able to be started in another cycle.

Table 4-1 shows the timing relations for the different memories.

Before requesting a memory cycle, it is necessary to load RY with the desired memory address. Loading RY also ensures that the memory is free for use because the μ-code clock delays loading RY until the current cycle is completed.

On a memory write cycle, it is also necessary to load RM. On a Prime 200, this must be done before the memory cycle is requested. Both RM and RY must not be altered until the memory cycle is finished. This can be ensured by loading RY and by the RM280 clock for RM because all such clocks wait for the completion of the memory cycle.

For the Prime 300, a memory write cycle can be started in the same μ-code in which RM is loaded. The RM200 clock must not be used for this because it, like the RM280 clock for the P200, waits for the memory to finish the current cycle. The ability to start a memory cycle before RM is loaded saves, typically. one μ-step per memory write.

Normal memory operation of a read cycle starts a memory read and puts the information into RM during the same μ-code step. This makes that step equal in time to the memory access. RM is loaded from memory on any clock that is conditional on MRDY (memory ready). The single step read operation is the only way that guarantees that address traps the registers from 0 to '37 as memory and guarantees the paging mechanisms will work. If neither of these capabilities is required, it is possible to overlap memory and μ-code more efficiently.

This overlap can be used to start a memory read cycle and then use RM as a scratch register for 1 to 3 μ-code steps before loading RM from memory. The DMA and DMT μ-code illustrates one use of this capability. However, a 200 ns clock is the minimum that can be used to start a memory cycle, if traps are disabled; a 280 ns clock is the minimum if traps are permitted.

Read-modify-write cycles (like IRS or IMA) on a single memory location can be done, but the first memory cycle must be finished before the next can begin. This can be accomplished as follows:

1. Take RY to RM before starting the next cycle.
2. Load RM using a clock that waits for memory (RM200 for P300, RM280 for P200).

3. Wait long enough (200 ns after the access time for fast memories, 160 ns for slow).

Note, this sequence fails:

```
  (P300)
      CPU   00   NX   0 0 0 0   RMMRDY   MREAD

      CPU   AL RM NX INC 1 XM DISABLE RM280 MWRITE
```

This sequence works:

```
      CPU   00   NX 0 0 0 0    RMMRDY   MREAD

      ALU   INC   RM => RM

      CPU   0 0 NX 0 0 0 0   200   MWRITE
```

Refresh of memory normally takes care of itself. However, refresh is of importance in timing analysis. In particular, after requesting a write cycle, before RM or RY can be used again, worst case timing is a full memory write cycle plus a full refresh cycle. Of course, synchronization can always be established by using a clock that waits until memory is available for RM and RY.

SECTION 5

I/O OPERATION

The Prime μ-code is the basic timing generator for the signal sequences on the I/O bus. The interrelationship between the signal generation and worst case times for the I/O bus is complex at best. This section of the Microcoders Handbook is an attempt at outlining the constraints and techniques. Detailed timing analysis is beyond the scope of this document. The only way to guarantee results with the I/O bus μ-code interface is to allow at least as much time as is allowed in the appropriate I/O algorithm.

5.1 SIGNAL GENERATION

The various I/O signals are generated from μ-code fields 6 and 7. If field 6 = 2 or 3, RY and RSC, respectively, are used as the source for the register file address. This leaves field 7 available for uses other than register file specification. The I/O signals are generated from field 7. Individual bits control each I/O signal. If more than one bit is set, both signals are produced.

Signals can be pulses or levels, and can start and end at different nominal times. Pulses begin and end within one μ-code step. Levels are started in one cycle and can continue until turned-off.

Table 5-1 summarizes the nominal times and nature for each signal.

5.2 PROGRAMMED INPUT/OUTPUT (PIO)

The sequence of signals for PIO is:

1. Load RY with the instruction.

2. Start PIO (280 ns clock minimum).

3. Wait for a valid Ready response (200 ns minimum).

4. Test Ready (280 ns minimum).

If Output transfer:

5. Enable RM to BPD (RM is assumed to be loaded).

6. Generate Strobe (maintain RM => BPD).

7. Stop Strobe (maintain RM => BPD).

8. Done (stop RM =>BPD).

9. Stop PIO.

If Input transfer:

5. Read BPD

6. Generate Strobe

7. Stop Strobe

8. Stop PIO

5.3 DMX TRANSFERS

DMX sequences are more complicated. In general, the address phase is independent from the data phase. The address phase for a given DMX transfer is the time from the start of DMX Enable to the start of the next Enable.

During the address phase, the algorithm must determine the type of transfer, use the discipline associated with the algorithm to obtain the memory address, generate a Clear Priority Net (CPN) signal and also send out 'End of Range' information if it is called for by the algorithm.

The link between the address and data phase is that Strobe must begin before the address phase ends. The data phase is when the memory is read or written into and the transfer of information is made.

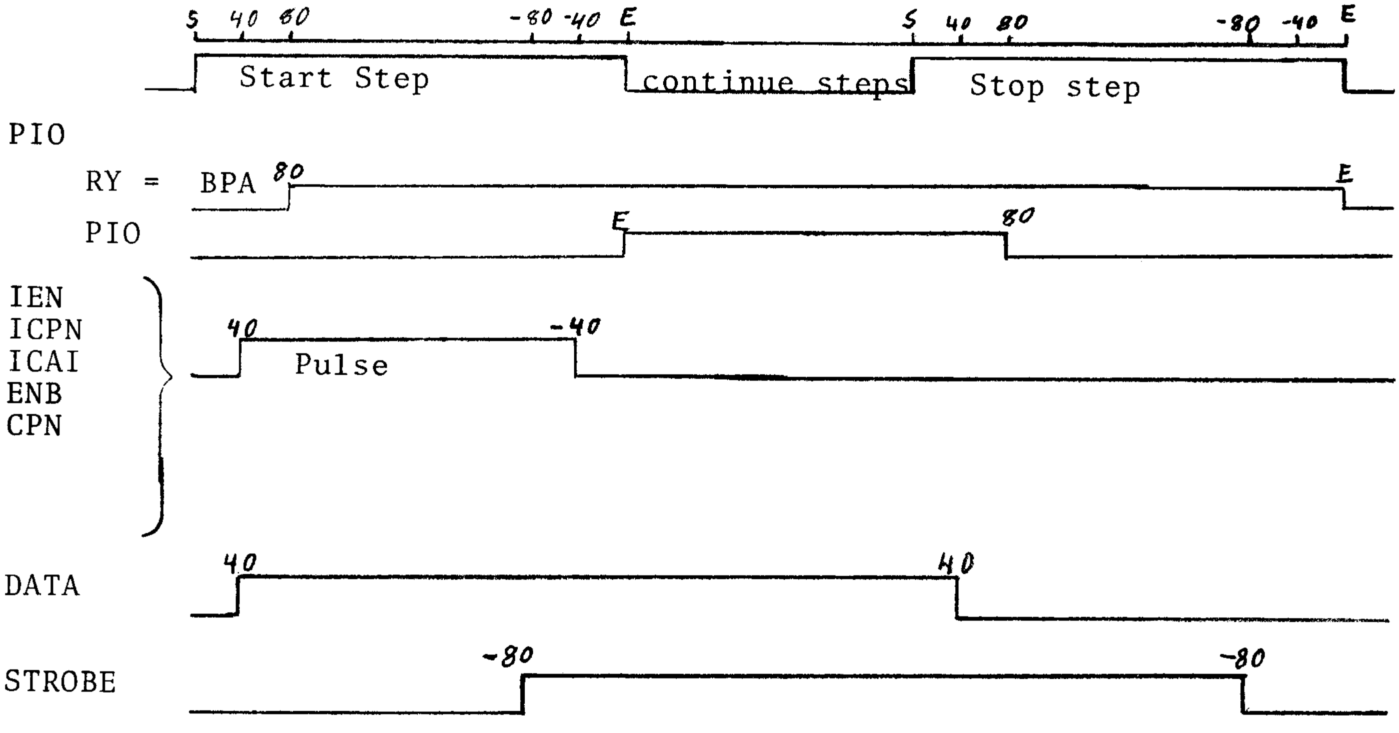
Overlapped transfers are possible in the sense that the data phase of cycle N can be running concurrently with the address phase of cycle N+1.

For a detailed description of the relationships required for I/O bus operation, see the GPIB manual.

(continued on page 5-4)

Table 5-1
Timing and Signals

Signal	Description	Generate/Turn Off	
PIO	Enables RY = BPA Generates PIO	Start: Continue: Stop:	RY, PIO any RY or RSC in field 6 any M or XM in field 6
IEN	Interrupt Enable	Start: Continue: Stop:	RY, IEN RY, IEN any other
ICPN	Interrupt Clear Priority Net	Pulse	RY, ICPN
ICAI	Interrupt Clear Active Interrupt	Pulse	RY, ICAI
ENB	DMX Enable	Pulse	RSC, ENB
DATA	Enables RM = BPD	Start: Continue: Stop:	RSC, DATA RSC, DATA any other
CPN	Clear DMX Priority Net	Pulse	RSC, CPN
STROBE	I/O Bus Strobe	Start: Continue: Stop:	RSC, STROBE RSC, STROBE any other



Times are in nano-seconds and are nominal only.

The relationships are:

a. Enable must be generated in a 280 ns cycle.

b. At least a 160 ns cycle must pass after an enable before the mode lines are tested.

c. CPN must be generated at least 200 ns before the next enable.

d. END of Block information can only be valid on the trail edge of a CPN in a 240 or 280 ns cycle.

e. Strobe must be generated in the step prior to the next Enable.

f. Data must be read in before Strobe is ended (and at least 280 ns after Strobe is started).

g. Data on output is only valid at the trailing edge of Strobe. If read from memory, and if the contents of RM are already enabled to BPD when RM is loaded from BMD, a 240 ns cycle in which Strobe ends is the minimum that allows the data to get to the controller.

5.4 INTERRUPT TRANSFERS

External interrupts operate similar to DMX with an Interrupt Enable starting the process. At the end of the Enable signal, the highest priority active device enables its address and mode lines onto the Bus. BPCOMD2 has the meaning of Memory Increment if true, and a memory increment operation is done.

For normal vectored interrupts, the address lines are read to obtain the interrupt vector location and an ICPN is issued to clear the priority net down to the active device. The rest of the net is cleared with an ICPCAI which is generated by the instruction CAI.

For non-vectored interrupts, the address lines are ignored, and the ICPCAI is issued implicitly before the ICPN.

Memory increment uses the address lines for the location to be incremented. An End of Range signal is sent back down the bus whenever the location is incremented to zero. The signal is valid only at the trailing edge of ICPN.

Timing constraints:

a. IENB must be 480 ns long (minimum).

b. Address lines are valid 280 ns after enable until ICPN.

c. Mode lines are valid for test 400 ns (minimum) after IENB.

d. ICPN and ICAI must occur in a 200 ns or longer cycle. If End of Range information is given a 280 ns minimum, ICPN is required.

The two signals cannot be generated in the same µ-step.

SECTION 6

INTERFACING TO STANDARD MICROCODE

There are only very minor differences between the basic microcode on the Prime 300 and Prime 200/100. These differences are in the Clock Field (field 8) and are primarily the addition of an RY240 clock and the loss of the RMRF280 clock.

Other differences have been explained in the previous sections.

Register assignments are common to both Prime 200 and Prime 300. These are:

Loc.	Name/Use		Mnemonic
0	X Register		RX
1	A Register		RA
2	B Register		RB
3	Stack/		RS
4	Floating PT Accumulator		FLTH
5	(double word)		FLTL
6	Visible Shift Counter		VSC
7	Program Counter		RP
10	Page Map Address Register		PMAR
11	Flex, UII	Floating	11
12	Effective Address Save	Point	12, EAS
13	M-scratch	Scratch	13
14	RY saved	DMX scratch	YSAVE
15	RM saved		MSAVE
16	RSC saved		RSCSAVE
17	DMC scratch		17
20 ↕ 37	Reserved for user (normally DMA channels)		

The rest of this section describes using some of the features built into the existing Prime μ-code.

6.1 USING EXISTING ALGORITHMS

There are various Prime Instructions that require many μ-steps for completion. Examples include PIO, MPY, DVD, and all the floating point instructions.

The control unit and current algorithms are not set up to permit easy use of existing instructions as subroutines because all of them return to the fetch cycle on completion. To use these instructions as subroutines, the following procedure must be used:

1. Save (RP) somewhere (probably in memory as all the scratch registers are used by floating point).

2. Load a three-memory location with an "Enter and Execute" instruction point to the desired place to enter in μ-code.

3. Load RP with a pointer to the "Enter and Execute".

4. Load RY with the memory location of the memory argument.

5. Jump to the μ-code for the instruction.

The above process is quite painful, but step 2 can be done ahead of time and the rest can easily be a μ-code subroutine.

There are several subroutines that exist inside of the floating point package as pure subroutines. These include single and double precision load, adjust (align) and normalize routines. The linkage to the first two routines is by pushing a return address onto the three deep μ-code stack (the routines themselves use another level of the stack) and then branching to the routine.

Unfortunately, the adjust routine does not always return. If the two numbers are too far apart to be aligned, an exit directly to the normalize routine can occur.

The normalize routine itself is not really a subroutine, as it always exits to the fetch cycle.

6.2 μ-CODE FOR VIRTUAL MODE EXECUTION

To produce μ-code that works on a system to be run in full virtual mode, it is necessary to:

a. Taking DMX latency into account.

b. Allow the external interrupts to work.

c. Let the mapping hardware, μ-code and software work.

DMX latency is easily provided for by enabling DMX traps at least every 1.4 micro-seconds in the code.

Making code interruptible is more difficult. It is possible to use the F1 branch condition to determine if an interrupt is pending. If the condition is true, either the control panel wants a halt or a power failure has been detected or an interrupt or memory increment is requested.

These various functions may be sorted out in one of two ways.

First, a branch to location zero or FHALT can be made. This permits the processing of the external request. However, the program counter will be used on returning, so the same procedure used for calling μ-code already written as a subroutine must be used. The second method is to perform the interrupt logic in the new μ-code. This method requires additional space but permits complete control of the machine to be maintained.

Writing μ-code that operates in a virtual memory environment is even more complicated. There are two basic ways to view the problem. In the first, the new μ-code looks like a new instruction executable by the virtual user. To do this, the μ-code sequence must either be interruptible or short enough so interrupts can be locked out. Memory references must be mapped and traps must be enabled for the step that starts memory. If the cycle is a read from memory, the cycle must be finished in the same step in which it is begun. A CAM update can occur automatically. Page fault vectors can also occur. Because the program counter is automatically backed up and the instruction is re-executed after the paging software has found the missing page, the P-counter must be pre-loaded with the address +1 (FUII cleared) or address +2 (FUII set) before the memory reference is attempted, if there is a chance that the page will not be in memory.

In the second method of writing virtualizable μ-code, the user "sees" an entirely different machine wholly created in new μ-code. In this case, a new fetch cycle is written and the μ-code must be made interruptible. The paging problem can be handled by changing the entry point into the μ-code and allowing the P-counter (register 7) to be decremented, and the EVMX found there to be re-executed. If this is done, the user visible machine must use some other register as a virtual program counter.

SECTION 7

MICROCODE TACTICS

This section briefly describes some techniques that have been found useful in inventing and encoding μ-code algorithms. Standard good programming practices are all that are generally required, but some specific tactics have been found useful.

The primary difference between coding for a normal assembly level language and coding for Prime μ-code, is the additional paralleled capability available in μ-code. Conventional flowcharting techniques have frequently been found to be inadequate guides in revealing where the inherent parallelism can be exploited.

One technique that has proved useful is to break the original algorithm into:

a. Decisions:

These generally require one use of a conditional branch per decision. This requires the condition code or C-Bit to be ready at least one step before and will use fields 11 and 12. A minimum μ-code algorithm requires at least as many steps as there are decisions.

b. Constants:

Except for 0, 1, and -1, these will have to come from the μ-code. (0, 1, and -1 can be generated in the ALU.) Often it is possible to store a constant somewhere more convenient for further use so that only each distinct constant need be counted.

Because constants, like conditional branches, must use fields 11 and 12, the total of decisions and conditional branches is equal to the minimum number of steps.

c. BD use:

It is genrally possible to count the number of times some result must be put somewhere. This requires a trip through BD.

Using the information on BD use, decisions, and constants, the next step is to try to generate μ-code sequence that uses the maximum number of fields per instruction. For example, in one operation, it is possible to test and conditionally branch on the condition code, perform a memory read, transfer RM to register '12, and to set the condition code to show a zero:

```
CPU   BB   RM  ALL   ZERO   L   M   12;
RMRFMRDY   MREAD   SETCC;
JUMP   ON   NE   TO   S12
```

The free use of parallelism must be tempered however by remembering that it is not possible to use one field to specify two things:

```
ALU   RA   PLUS   RCM   =   1 =>RM;
      NOP   GO   TO   S12
```

will not work because fields 11 and 12 are twice specified; once by the RCM = 1, and once by the GO TO.

The following, however, is legal:

```
ALU   INC   RA =>RM   NOP;
      GO   TO   S12
```

It accomplishes the same thing as the previous example, but only uses fields 11 and 12 once, for the GO TO.

Some other examples of exploiting parallelism include:

1. Increment RP RP and RY load the shift counter with a -15.

```
CPU   AL   RCM   NX   INC   1   M   RP;
      RYRF240   NOP   LOADRSC;
      DATA   -15
```

2. Shift RA the number of places in the shift counter, return to the fetch cycle when finished:
Load the C-bit with the bit shifted out of RF01.

```
CPU   RFLS   7   ALL   7   0   M   RA;
      RF240   RF01   INCRSCF;
      JUMP   ON   RSCNEM1   TO   *
```

3. Perform a subtract of RM and RA; to RA, Jump on a not equal condition.

```
ALU   RA   MINUS   RM   RA   SETCC;
      JUMP   ON   NE   TO   S12
```

Use of scratch registers is another area where considerable speed increase can often be realized by carefully assigning the use of the available registers. Within the register file, registers 11, 12, and 13 are always available. Registers 0 - 6 are also open if their function in normal μ-code can be ignored. For example, 4, 5, and 6 are available if the floating point accumulator can be destroyed.

The registers RM and RY are also open. These have some extra power. They are always available, and transfers from one to the other, or to any register in the file takes only one step. Transfer from one register in the file to another takes two steps (using RM or RY for intermediate storage). A number in RM or RY may be TWO's complemented in one step. However, if RY is both the source and destination for an operation, neither the C-bit nor the condition code is valid.

Finally, if a register is required for a very brief period, then registers YSAVE, MSAVE, RSCSAVE and 17 are available, if no DMX traps or page traps are permitted during the time the register is in use.

The use of high-speed registers, instead of memory, gains an immediate improvement of at least seven times in access time. In addition, a single micro-step can select a register, then read and modify it.

SECTION 8

MICROCODE WRITING AND IMPLEMENTING

8.1 GENERATION PATH

1. Generate μ-code algorithm.

2. Select Program Organization.

   This activity includes selection of program segments and subroutines, register assignment and other general tasks. Flow charting could be part of this step.

3. Generate Code.

   It is at this time that parallelism is exploited and the logic reduction effort is most intense.

4. Create Source.

   During this step, detailed knowledge of the system editor (ED) and μ-code assembler formats is required. Section 8.2 and 8.3 covers this information in detail.

5. Assemble Source.

   If step 4 has been properly done, inputting the DOS command: PMA Filename (<u>Filename</u>) is all that is required.

6. Print Listing.

7. Examine and correct assembler flagged errors.

Section 8.3.5 covers some of the more frequently encountered errors.

8. Repeat 5, 6, and 7 until no errors are detected by the assembler.

9. Prepare a load module for WCS. See Section 8.4.

10. Load the WCS board and begin debug. Section 8.5 contains some examples and details of this process.

11. Correct errors found in debug on the source and repeat steps 5-10 until program is judged to be working.

12. Finished.

8.2 SOURCE CREATION

Source code must be generated in a form compatible with the PMA assembler and also with the macro package called MA which acts as a microcode assembler. In the remainder of this section, it is assumed that the user has a detailed knowledge of ED and how to use it. However, a few notes should prove helpful.

8.2.1 Ed Notes

a. The tabset normally used is 8, 16, 21, 24, 32, 39, 45, 51

b. Semi-colons are frequently required to indicate a μ-code line is continued on the next source line. The editor will not accept them. One strategm for inserting semi-colons is to substitute another character (& or #, etc.) for the semi-colon and then changing all of these characters to the semi-colon using a global Change command to the Editor.

8.2.2 Normal Source

In addition to the μ-code steps themselves, additional instructions are necessary or customary. The following shows a typical example of μ-code source:

```
*   SAMPLE MICRO-CODE                    /Line for printout
                                          header.


P300  XSET  3                            /Required - sets
                                          assembler for WCS.


$INSERT  MA                              /Load in μ-code
                                          macro masters.


  IDNT  (SAMPLE MICRO-CODE)              /Required - see 8.3.1


  ORG   $C00                             /WCS starting address


        (body of μ-code)


  END
```

8.3 MICROCODE ASSEMBLER

The microcode assembler consists of several macro calls. The detail of the mnemonics and format is summarized in Appendices A and B. There are also several utilities associated with the assembler.

The microcode assembler uses the idea of fields. Fields are given argument numbers by position starting with argument one (1) as the first value after the macro. By definition, argument 0 is the label. Each argument must be separated by a comma or by one or more spaces. Null arguments are assumed to take a default value. (Zero for all fields except 3 and 10 which have 2 and 4 as default.) Several examples are given to show how the assembler interprets spaces and commas.

```
    #1  CPU  RF,,,,,,RA...       /5 blank Fields
    #2  CPU  RFΔ,,,,,RA...       /5 blank Fields
    #3  CPU  RFΔ,Δ,,,,RA...      /5 blank Fields
    #4  CPU  RF,ΔΔ,,,,M,RA       /4 blank Fields
```

Normally, the position of an argument determines its interpretation. For example, the CPU macro typically contains 12 fields and, hence, 12 arguments. The RR Macro can contain as few as two and as many as six arguments and be meaningful.

Exceptionally, items that resemble arguments (i.e., they are separated by commas or spaces) can be interpreted as simply adding to the readability of the macro without being further interpreted by the macro. Examples of this type of word for the CPU ALU and RR macros are "ON" and "TO" and "=>". They are formally termed "noise words".

Also to be contended with, are words that serve to define the argument following it as an argument with a specific number regardless of position. Examples of these "positional noise words" are "C=", "TR=", "=", and "+". These words can be anywhere in the argument string and serve to identify the following argument as having a specific "place". Note that "+1" is interpreted far differntly than "Δ+Δ1". In the former case, a 1 is added to the value of the previous symbol. In the latter, field 5 is set to 1.

Another feature of microcode fields 6 and 12 is that arguments can be 'or'ed together within parenthesis. There is a limit of 32 characters that may be in the area enclosed by parenthesis.

8.3.1 Auxiliary Macros

```
(label)  IDNT    ARG1, ARG2 . . . ARG10
```

There can be up to 10 arguments of up to 30 characters each. (This serves as header for the tape listing.) This macro is required before other μ-code to initialize the microcode assembler.

```
Example:  IDNT  (PRIME 211 REV E MICRO-CODE),_;
                (PRE-RELEASE VERSION),_;
                (APRIL 9, 1973)
```

```
(label)  ORG    ARG1
```

ORG is part of the full symbolic statement label capability in this assembler. Labels are maintained and usable in exactly the same way as for standard assembler code. * can also be used as can *-1 and *+1, etc. ORG is useful for absolutely positioning a given statement (like a trap location).

8.3.2 CPU Macro

This macro simply encodes on a field by field basis the symbols defined for each microcode field as defined in Appendix A. It is normally used when the activities desired cannot be conveniently specified using the RR or ALU Macros. The limitation is that no self-checking is done so illegal clock speeds and nonsense transfers are as happily assembled as legitimate instructions. Let the user beware!

8.3.3 RR Macro

The Register to Register transfer Macro is intended to be used for non-arithmetic transfer operations of a normal sort. The chores of allowing proper clocks and traps is taken care of implicitly. Additionally, Register file selection is checked to find if the same register is used for both source and destination. If not, an error is logged.

Appendix B defines the format and symbols of the RR Macro. The RR Macro automatically selects the proper values for fields 1 to 12. However, argument <3> can be used to override the normally selected NOP and TR=, C=, and = override fields 3, 9, and 11 and 12, respectively. (Note that C=, TR= and = must have at least one space on either side.)

The fastest clock possible for the options selected is chosen.

For clocking RY or No destination traps are set = NX unless otherwise specified.

Note that IAC's do not change the chosen clock.

8.3.4 ALU Macro

The ALU Macro is intended to be used for arithmetic operations of a normal type. Like the RR macro, the fields are implicitly filled where possible. Appendix B shows the format and special symbols used in the ALU macro.

8.3.5 Microcode Error Messages

The Micro-code assembler has various cryptic methods of informing the user of his transgressions. Frequently, a major disaster is flagged with a little comment. On the other hand, a missing space or comma can generate a string of up to 14 errors.

Figure 8-1 shows a simple micro-code program with problems.

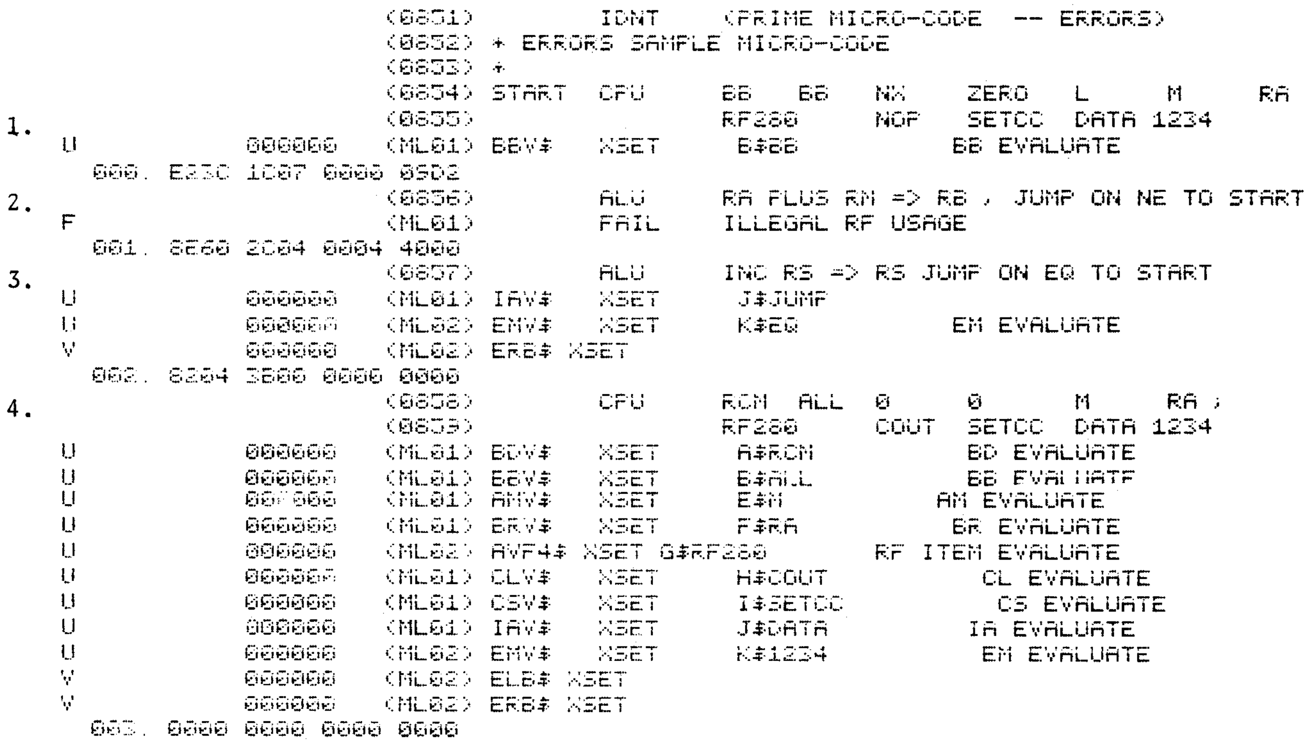
Example 1 shows the ordinary result of mistyping a field symbol. In this case, field 2 (BB select) was typed in as BB instead of RCM. The macro package appended a B$ to the label and found the results undefined. This problem is shown by the error message that includes the comments "BB evaluate".

Example 2 demonstrates the type of error statement which can be expected from the attempt to transfer information from one register in the file to another in one instruction. (In this case, both RA and RB are used.)

Example 3 shows the typical result of forgetting the comma required to define the missing field. The line should be:

```
          ALU  INC  RS ⇒ RS , JUMP ON EQ TO START
```

Example 4 shows the consequence of the omission of a single field (BB select) in a CPU macro. Each following field is misdefined, producing error statements for all of them. Note that after each XSET there is a "letter" $ followed by one of the source fields. The comments areas show the field that the assembler thought it was working on.



```
                       (0851)         IDNT     (PRIME MICRO-CODE  -- ERRORS)
                       (0852) * ERRORS SAMPLE MICRO-CODE
                       (0853) *
                       (0854) START  CPU      BB   BB   NX    ZERO   L     M     RA
1.                     (0855)                 RF280     NOP   SETCC  DATA 1234
  U          000000    (ML01) BBV#    XSET     B#BB          BB EVALUATE
   000. E230 1007 0000 05D2
2.                     (0856)         ALU     RA PLUS RM => RB , JUMP ON NE TO START
  F                    (ML01)         FAIL    ILLEGAL RF USAGE
   001. 8E60 2004 0004 4000
3.                     (0857)         ALU     INC RS => RS JUMP ON EQ TO START
  U          000000    (ML01) IAV#    XSET     J#JUMP
  U          000000    (ML02) EMV#    XSET     K#EQ          EM EVALUATE
  V          000000    (ML02) ERB# XSET
   002. 8204 3800 0000 0000
4.                     (0858)         CPU     RCM  ALL  0     0      M     RA ;
                       (0859)                 RF280     COUT  SETCC  DATA 1234
  U          000000    (ML01) BDV#    XSET     A#RCM          BD EVALUATE
  U          000000    (ML01) BBV#    XSET     B#ALL          BB EVALUATE
  U          000000    (ML01) AMV#    XSET     E#M          AM EVALUATE
  U          000000    (ML01) BRV#    XSET     F#RA          BR EVALUATE
  U          000000    (ML02) AVF4# XSET G#RF280      RF ITEM EVALUATE
  U          000000    (ML01) CLV#    XSET     H#COUT          CL EVALUATE
  U          000000    (ML01) CSV#    XSET     I#SETCC          CS EVALUATE
  U          000000    (ML01) IAV#    XSET     J#DATA          IA EVALUATE
  U          000000    (ML02) EMV#    XSET     K#1234          EM EVALUATE
  V          000000    (ML02) ELB# XSET
  V          000000    (ML02) ERB# XSET
   003. 0000 0000 0000 0000
```

Figure 8-1. Errors Sample Microcode

8.4 PREPARATION OF A LOAD MODULE

After a microcode assembly has been successfully completed, a procedure must be performed to generate a block of code suitable for loading WCS.

The assembly produces an object file of microcode that must be loaded using the loader.

A sequence to do this is:

```
FILMEM
LOAD
LO  B ← (filename)
MA
QU
```

This sequence generates a memory image of the microcode starting at '10,000+ 4*(starting microcode address). For the WCS starting address, this is '10000+ $C00*4 = '40000. A 256 word module uses 256 * 4 = 1024 = '2000 words of HSM. This module may be directly loaded into the WCS board, or it may be saved on the disk.

8.5 WRITABLE CONTROL STORE BOARD (WCS) OR EXTEND CONTROL BOARD (XCS)

The XCS option provides extended microprogramming capability to Prime central processors and microprogrammed options. XCS features include:

1. 512 words of PROM for extending the central processor microcode (FPROM).

2. 512 words of PROM for further extension of central processor microcode with reduced control unit speeds (SPROM).

3. 256 words of writable RAM (random access memory) designated as Writable Control Store (WCS) used for further extension of central processor or micro-programmed controller microcode. WCS can also be used for dynamic microprogram testing and execution from either central processor or microprogrammed controllers when used in the simulator mode. When operating in simulator mode, the simulated module is disabled and WCS data is provided whenver the simulated module is addressed.

4. Provisions are included for connecting an external PROM copier to allow PROM programming utilizing data from WCS.

5. A Field Engineering Panel Interface is included as an aid for troublshooting the central processor and the XCS. Switches and indicators are contained on the standard FEP hand control.

8.5.1 PROM COPY Interface

The PROM COPY Interface consists of a cable between XCS and a PROM copier. Addresses to this site are provided from the PROM copier and are sent to WCS. Data to this site is provided by bits 61 through 64 of WCS and is sent to the PROM copier. The procedure for copying PROM is as follows:

1. With power OFF, connect PROM copier to XCS.
2. Turn power ON.
3. Insert new PROM in copier.
4. Execute a CRA and an OTA '324.
5. LDA with data to be programmed into location 0 of PROM in bits 13 through 16 of A register.
6. Execute four OTA '124 (data is written into proper cell during 4th OTA).
7. LDA with data to be programmed into location of PROM into bits 13 through 16 of A register and continue as in E and F above until all 256 locations are loaded.
8. LDA = '040000
9. OTA '324
10. HLT
11. HLT (or JMP to suitable terminal I/O response routine).
12. Depress PROM copier button.
13. Verify copy.
14. Remove PROM and insert new PROM.
15. Start CP and jump to routine to load WCS with next data to be copied.
16. Go to 4 above.

8.5.1.1 Instructions for Use of PROM, μ-Code Program

For example programs, see Appendix E.

8.5.2 FPROM Interface

The FPROM Interface consists of a central processor BCY (control memory address bus) buffer and address decoder plus 13 or 26 PROMs.

8.5.3 SPROM Interface

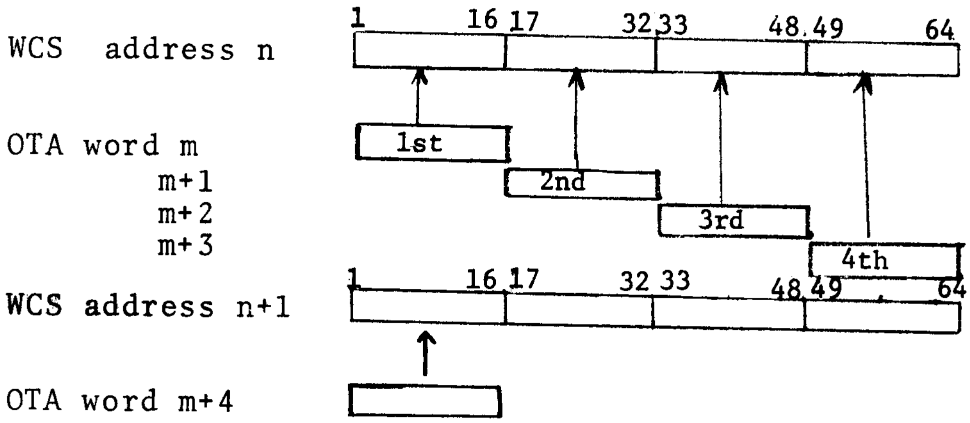
The SPROM Interface consists of the following:

1. A multiplexer for selecting central processor BCY.
2. Either 13 or 26 PROMs.
3. A data buffer for interfacing with the central processor.
4. Dip sockets wired to address and data for interfacing to external microprogrammed controllers.

8.5.4 WCS Interface

The WCS Interface consists of the following:

1. An incrementing register used for loading/fetching data into/from WCS. The WCS will pack four central processor words into one microprogrammed word, then increment the WCS address register (see below).



2. Module decoders for disabling the simulated module of the central processor, XCS, or external microprogrammed controller. These signals are sent to the controller as DSMODn where n equals the module to be disabled.

3. A WCS address multiplexer for selecting one of the following:

    1. PROM copier

    2. Central processor BCY

    3. WCS address (used when writing into or fetching data from WCS)

4. An address comparator used to enable WCS when addressing a simulated module.

5. 52 RAMs

8.5.4.1 WCS LOAD PROGRAM

```
SAMPLE SIMPLE WCS LOAD PROGRAM                                    PAGE 0001


                    (0001) * SAMPLE SIMPLE WCS LOAD PROGRAM
                    (0002) *
                    (0003) *   PROGRAM ASSUMES U-CODE STARTS AT '40000 (AS STANDARD
                    (0004) * WCS CODE DOES) AND LOADS ALL 256 WORDS OF WCS.
                    (0005) *
                    (0006) *  A NON-ZERO A REGISTER SETTING ON ENTRY WILL BE USED TO
                    (0007) * OVERRIDE THE DEFAULT '40,000 SETTING.
                    (0008) *
                    (0009) *  A NON-ZERO B REGISTER SETTING ON ENTRY WILL BE USED TO
                    (0010) * OVERRIDE THE STANDARD WCS MODE TO ENTER SIMULATE MODE
                    (0011) * WITH SIMULATION OF THE ADDRESS SPACE SPECIFIED IN B.
                    (0012) *
                    (0013) *
                    (0014) *   THE ROUTINE CAN ALSO BE USED AS A SUBROUTINE BY INSERTING
                    (0015) * AN ENTRY POINT BEFORE START, AND EXITING WITH A JMP*
                    (0016) *
                    (0017) *
                    (0018)        REL
                    (0019) *
000000:             (0020) START  ELM
000001:   031724    (0021)        OCP     '1724          INITIALIZE WCS
000002: 35.000026   (0022)        LDX     ='40000        DEFAULT START
000003:   100040    (0023)        SZE
000004: 35.000001A  (0024)        LDX     1              SPECIFIED START
000005: 02.000027   (0025)        LDA     =-(256*4)      # OF WORDS
000006: 04.000025   (0026)        STA     COUNT
000007: 22.000000A  (0027) LOOP   LDA     0,1
000010:   170124    (0028)        OTA     '124           LOAD 1/4 WSC WORD -- WCS BUMPS OWN POINTER.
000011: 01.000010   (0029)        JMP     *-1
000012:   140114    (0030)        IRX
000013: 12.000025   (0031)        IRS     COUNT
000014: 01.000007   (0032)        JMP     LOOP
000015:   031724    (0033)        OCP     '1724          RESET WCS BOARD
000016:   140204    (0034)        XCB
000017:   101040    (0035)        SNZ
000020:   000000    (0036)        HLT                    RETURN OR JMP TO PROGRAM.
000021:   170324    (0037)        OTA     '324           OUTPUT SIMULATE STARTING ADDRESS
000022: 01.000021   (0038)        JMP     *-1
000023:   030324    (0039)        OCP     '324           ENTER SIMULATE MODE.
000024:   000000    (0040)        HLT                    RETURN OR JMP TO PROGRAM.
                    (0041) *
000025: 00.000000A  (0042) COUNT  DAC     **
                    (0043) *
          000026    (0044)        END

000026: 00.040000A
000027: 00.176000A
```

8.5.5 Firmware Description

Table 8-1 describes the allocation of CP/XCS microprocessor address space.

Table 8-1

All addresses in hexadecimal.

		MODULE
000 - 0FF	CPU	0
100 - 0FF		1
800 - 8FF	SPROM	4
900 - 9FF		5
400 - 4FF	FPROM	2
500 - 5FF		3
C00 - CFF	RAM	6

Portions of XCS that are used with the central processor should follow the standard formats.

A. Field 8, Destination and Time of cycle. Times are specified (see Section 3 - Timing).

CP Simulator Mode - When RAM is utilized in the simulator mode, the microcode must be written with field 8 times equal to that required for the address space that is being simulated. WCS hardware delays are introduced when entering this mode to compensate for any restrictions that might otherwise be imposed by field 8.

8.5.6 Software Description

In addition to standard I/O instructions, the XCS utilizes four instructions that are restricted.

			OP CODE
EPMX	-	Entei Paging Mode and jump to XCS	'237
EVMX	-	Enter Virtual Mode and jump to XCS	'723
ERMX	-	Enter Restricted Mode and jump to XCS	'721
LPMX	-	Leave Paging Mode and jump to XCS	'235

Each of these instructions have the following format and require three HSM locations:

CP word	n	(instruction word)
	n+1	(pointer to address m)
	m	(address in XCS to jump to)

8.5.7 I/O Instructions

The XCS is always ready and always skips on its INAs and OTAs.

OCP '1724 - INITIALIZE

Places WCS in same status as following a MASTER CLEAR with the exception of the clock slow down, which is not disabled until initiation of the next instruction. WCS address register and comparator are set to a default address of $C00. ($ indicates hexadecimal notation.) Simulator mode is disabled. Copy address is disabled.

OCP '324 - SIMULATE MODE

1. No external microprogrammed option attached.

   a. Lengthens all micro instructions by 80 nanoseconds.

   b. Disables the simulated module associated with the central processor.

   c. Provides WCS data to the central processor whenever the disabled module is addressed.

2. External microprogrammed option attached.

   a. Disables the simulated module associated with the microprogrammed option.

   b. Provides WCS data to microprogrammed option whenever the disabled module is addressed.

NOTE: IF SIMULATING CENTRAL PROCESSOR MODULE Ø or 1 - CARE SHOULD BE TAKEN TO ENSURE COMPATIBILITY OF MICRO-STEPS FOR A SMOOTH TRANSITION FROM THE ACTIVE MODULE TO THE SIMULATED MODULE. OCP SIMULATE MODE TAKES EFFECT DURING THE OCP OR THE FOLLOWING INSTRUCTION. PROVISIONS HAVE ALSO BEEN INCLUDED TO ENTER SIMULATE MODE BY MEANS OF A SWITCH (SS3 ON FEPII).

OTA '324 - OTA ADDRESS

1. Output the starting address of the WCS microcode addresses to be loaded, fetched, or simulated. If loading for normal WCS mode into location $C00, no OTA '324 is necessary if no prior OTA '324 has been issued. If loading any other address or consecutive addresses, the starting address must be OTA'd. Only one module at a time can be simulated, therefore, care should be taken to avoid outputting words across the module boundary. Only entire 256 word modules can be simulated. Any individual word or consecutive words being simulated can be changed by disabling simulator mode (OCP initialize), OTA the individual address to be changed, OTA the data to be changed, OCP Simulate Mode.

2. Select PROM copier for address to and data from WCS if bit 2 of the A register is set.

3. Deselect PROM copier if bit 2 of A register is not set.

OTA '124 - OTA DATA

1. Loads data into WCS from the A register. When used with the central processor, use the following format:

Word 1	1	RCC	16
Word 2	17	RCC	32
Word 3	unused	45 RCC	48
Word 4	49	RCC	64

The above format is generated from a LOAD.

2. Increment WCS address register after four OTAs.

INA '1124 - ID NUMBER

The format of the data is as follows:

1 3	4 8	9 10	11 16
0 0 0	SLOT #	0 0	'24

PROGRAMMING NOTES

1. When using WCS in its normal mode, no OCPs or OTA address are necessary. Powering on of the central processor sets the XCS into the normal (addresses equal $C00 - $CFF) WCS mode. Simply OTA the data to load WCS. The data is now available for execution. Halting the processor and master clearing does not affect the data. WCS data is available(just as the standard central processor PROM data is available) simply by addressing it.

2. When using XCS in simulator mode, first OTA a starting address then OTA the data and finally, OCP Simulate Mode. To leave Simulate Mode, issue an OCP initialize or Master Clear.

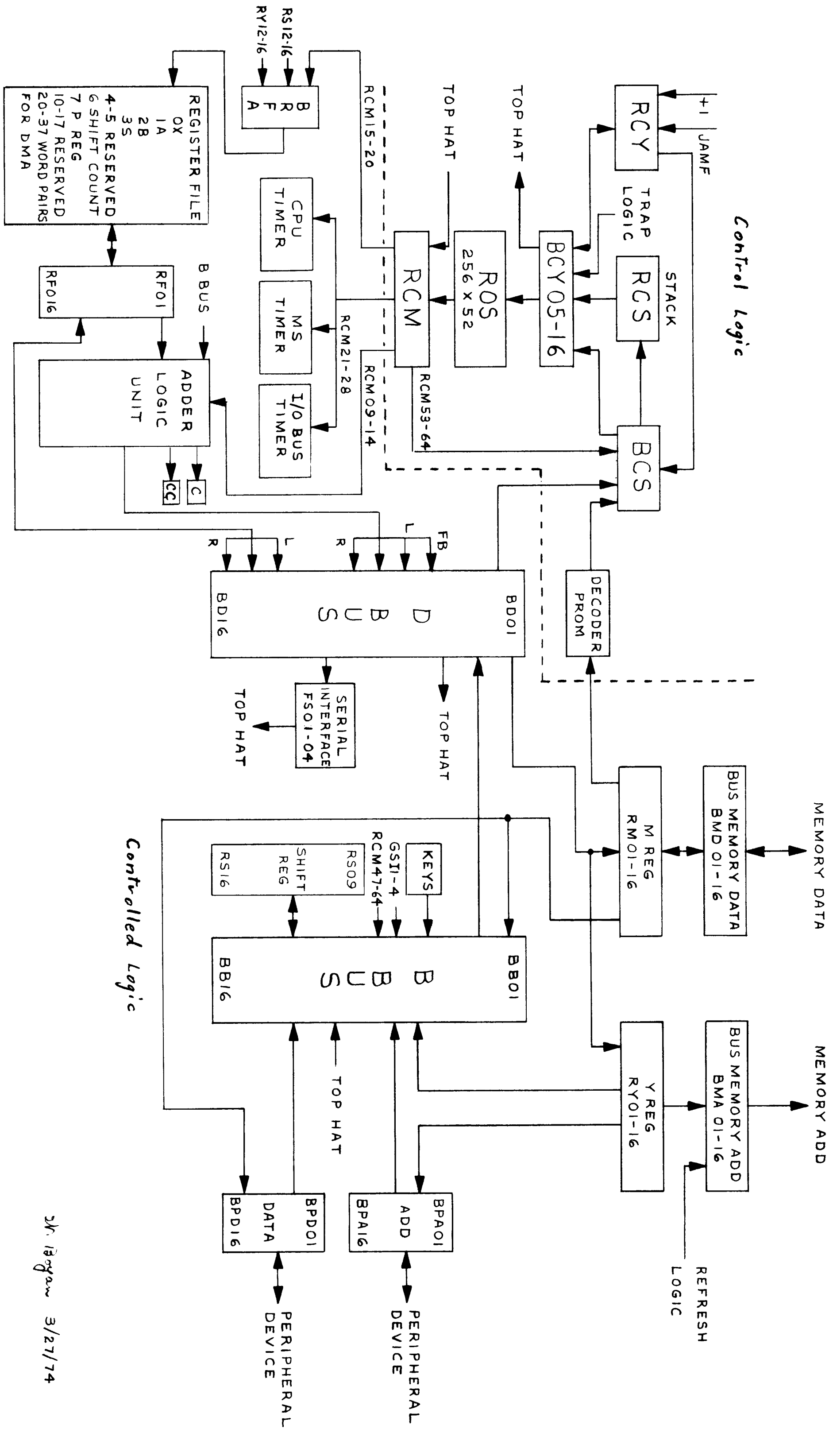
3. To add 80 nanoseconds to all microcode steps, issue an OCP simulate after an OCP initialize or master clear or use SS3 on REPII (maintenance troubleshooting feature).

Note: Do not address SPROM after OTAing a SPROM address without first initializing (OCP '1724) or entering simulate mode (OCP '324).

APPENDIX A

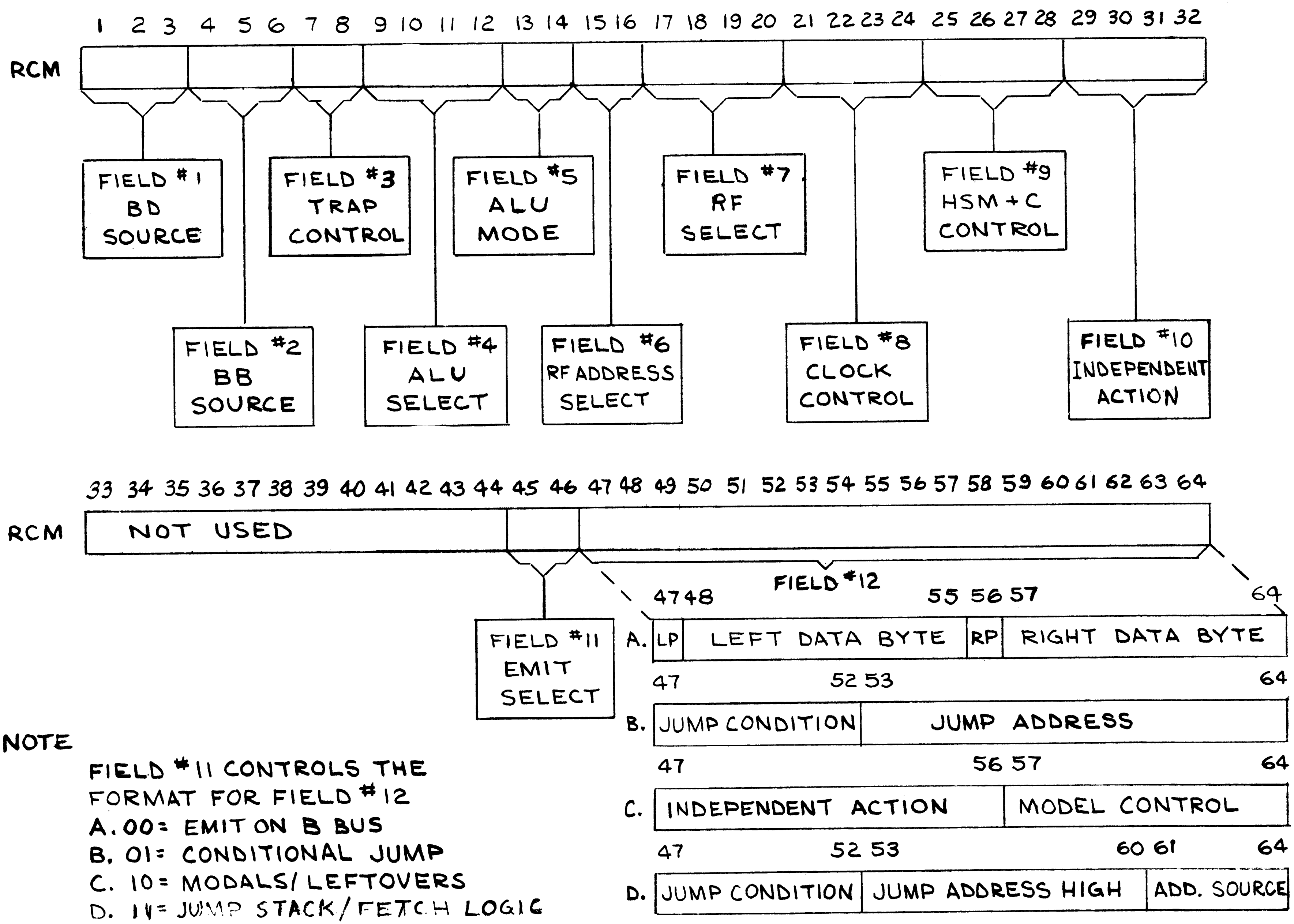
BLOCK DIAGRAM, MICROCODE FORMAT AND FIELD DESCRIPTIONS

CPU BLOCK DIAGRAM



CPU BLOCK DIAGRAM

W. Bogan 3/27/74



NOTE

FIELD #11 CONTROLS THE FORMAT FOR FIELD #12
A. 00= EMIT ON B BUS
B. 01= CONDITIONAL JUMP
C. 10= MODALS/LEFTOVERS
D. 11= JUMP STACK/FETCH LOGIC

MICROCODE FORMAT (CPU)

APPENDIX A (cont)

Basic M-code Field Description

Figure 1

FIELD DESCRIPTIONS			ROM BITS
Field #1	BD Source Select	3	RCM 01-03
Field #2	BB Source Select	3	RCM 04-06
Field #3	Trap Control	2	RCM 07-08
Field #4	ALU Select	4	RCM 09-12
Field #5	Arithmetic Carry/Mode Control	2	RCM 13-14
Field #6	RF Source Select (BRFA)	2	RCM 15-16
Field #7	Register File Source Select	4	RCM 17-20
Field #8	BD Destination and Clock Control	4	RCM 21-24
Field #9	Carry Source, Main Memory Operation	4	RCM 25-28
Field #10	Independent Actions	4	RCM 29-32
Field #11	Emit Select	2	RCM 45-46
Field #12	Emit Field	16	RCM 47-64
	Reserved for Future Use	12	RCM 33-44

Field #1: D-BUS SOURCE SELECT

		Destination = D-Bus (BDxx)	Shift End Conditions*				Modifying Fields	
Field #1	Source	01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16	BD16	BD02	BD01	Link**	Field #2	Field #4
0,RF,(blank)	RFxx	01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16	0		RF02	RF01	7, BTH	7
			RF01		RF02	don't care	1,KEYS,RSC	8
			Link		RF02	RF01	3,RM	7
1, RFLS	RFxx	03 04 05 06 07 08 09 10 11 12 13 14 15 16	0		RF01	RF02	7,BTH	3,ZERO,2A
			Link		RF02	DIVGQ1	3,RM	9,SUB
			Link		RF02	DIVGQ1	3,RM	6,XOR,ADD
			Link		RF01	RF02	3,RM	3,ZERO,2A
2, RFRS	RFxx	02 03 04 05 06 07 08 09 10 11 12 13 14 15		RF01	RF01	RF16	3,RM	6,ADD
				RF01	0	RF16	5,BPA	6,XOR,ADD
				RF01	RF16	RF16	2,RY	6,ADD
				Link	RF01	RF16	0,RM08,SI,	6,XOR,ADD
				RF01	Link	RF16	1,KEYS,RSC	6,XOR,ADD
				RF01	RF01	RF16	3,RM	6,XOR,ADD
3, ALFB	ALxx	09 10 11 12 13 14 15 16 01 02 03 04 05 06 07 08						
4, AL	ALxx	01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16						
5, ALLS	ALxx	02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 ←	Link			DIVGQ1	3,RM	6 or 9
6, ALRS	ALxx	01 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15				AL16	3,RM	6,XOR,ADD
7, BB	BBxx	01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16						

RF(xx)	Register File (bit xx)	ALLS	ALU left shifted one bit
RFLS	Register File left shifted one bit	ALRS	ALU right shifted one bit
RFRS	Register File right shifted one bit	BB(xx)	B Bus (bit xx)
AL(xx)	Arithmetic & Logic Unit (ALU) (bit xx)	BD(xx)	D Bus (bit xx)
ALFB	ALU with bytes swapped	DIVGQ1	Incremental divide quotient bit

* It is possible to create other shift end conditions, but these cover all the useful cases.

**if enabled by Field #9 = 1 to 7

Figure 2

Figure 3

Field #2 : B-BUS SOURCE SELECTION

Field #2	Source	LP	01	02	03	04	05	06	07	08	RP	09	10	11	12	13	14	15	16
	Destination = B-Bus (BB xx)																		
0,RM08,SI, (blank)	RMxx GSIx	—	08	08	08	08	08	08	08	08	*	--	--	--	--	1	2	3	4
1, KEYS , RSC	FDPM FCBIT FAM1 FAM2 RSxx	—	↓	↓	0	0	↑	↑	0	0	*	09	10	11	12	13	14	15	16
2,RY	RYxx	LP	01	02	03	04	05	06	07	08	RP	09	10	11	12	13	14	15	16
3, RM	RMxx	LP	01	02	03	04	05	06	07	08	RP	09	10	11	12	13	14	15	16
4, RCM	RCMxx	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
5, BPA	BPAxx	LP	01	02	03	04	05	06	07	08	RP	09	10	11	12	13	14	15	16
6, BPD	BPDxx	LP	01	02	03	04	05	06	07	08	RP	09	10	11	12	13	14	15	16
7, BTH	BTHxx	LP	01	02	03	04	05	06	07	08	RP	09	10	11	12	13	14	15	16

"—"=generated byte parity bit
"--"=undefined
"LP"=left byte parity bit
"RP"=right byte parity bit

*No Parity on BBRP for GSIX or RSXX. If these inputs are used, new parity must be generated (by going through the adder).

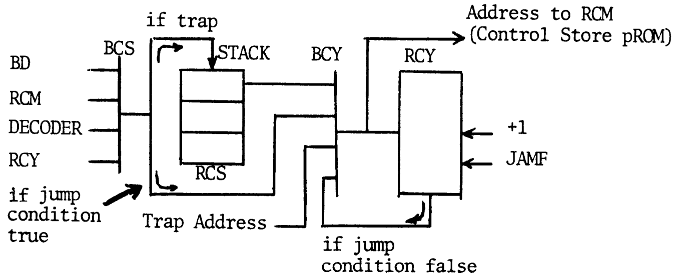
RM (xx)	M Register (bit xx)	RS (xx)	Shift Register (bit xx)
SI	Serial Interface	RY (xx)	Y Register (bit xx)
GSI (x)	Serial Interface (bit x)	BB (xx)	B-Bus (bit xx)
RSC (xx)	Shift Counter Register (bit xx)	BPA (xx)	Bus Peripheral Address (bit xx)
FDPM	Double Precision Mode Flag	BPD (xx)	Bus Peripheral Data (bit xx)
FCBIT	Carry Bit	BTH (xx)	Bus Top Hat (bit xx)
FAM 1	Address Mode Flag, bit 1		
FAM 2	Address Mode Flag, bit 2		

Figure 4

Field #3: TRAP CONTROL

Field #3 =	Control
0, NONE	No Traps
1, TDMX	DMX Only
2, NX, (blank)	All but DMX
3, ALL	All Traps

Control Store Address Diagram



EXPLANATION OF MICROCODE TRAPS:

Priority Highest to Lowest	Hardwired Trap Address (Hexidec.)	Description
1	6E	Missing Memory Module (generates interrupt)
2	6C	Memory Parity (generates interrupt)
3	6A	Central processor parity (generates machine check interrupt)
4	68	Restricted execution mode (generates interrupt)
5	66	Fetch read address trap (put RF into RM as appropriate instead of main memory to RM, load F01 and F02)
6	64	Read address trap (put RF into RM as appropriate instead of main memory to RM)
7	62	Write address trap (put RM into appropriate RF instead of main memory).
8	60	Page fetch read address trap (the CAM must be filled with the new page pointer if available; if not, generage a page fault interrupt; load F01 and F02)
9	7C	Page trap (same as 60 without loading of F01, F02)
11	72	Page writer violation (generates interrupt)
12	70	DMX (performs a DMX transfer without changing user execution flow)

Field #4: ALU FUNCTION SELECT ; Field #5: ARITHMETIC/CARRY MODE CONTROL

	The ALU output (AL) as a function of its inputs (RF, BB and FCBIT)			
Field #4	Field #5 = 0, (blank)	Field #5= 1, PLUS1	Field #5 = 2, CBIT	Field #5 = 3, L
0, RF, INC.	RF	RF + 1	RF + FCBIT	RF
1 , AND	(RF∧ BB--) + RF	(RF∧ BB --) + RF + 1	(RF ∧ BB --) + RF +FCBIT	RF∧BB
2	RF ∧ BB + RF	RF ∧ BB + RF + 1	RF ∧ BB + RF + FCBIT	RF∧ BB --
3, ZERO, 2A	RF + RF	RF + RF + 1	RF +RF + FCBIT	0
4, OR	RF ∨ BB	RF∨ BB + 1	RF ∨ BB + FCBIT	RF ⊻ BB
5, B, BB	(RF∨BB --) + (RF ∨ BB)	RF∨BB--) + (RF∨BB) + 1	(RF∨BB--) + (RF∨BB) + FCBIT	BB
6, XOR, ADD	RF + BB	RF + BB + 1	RF + BB + FCBIT	RF⊻BB
7	RF + (RF∨BB)	RF + (RF∨BB) + 1	RF + (RF∨BB) + FCBIT	RF --∧ BB
8	RF∨BB --	RF∨BB -- + 1	RF∨BB -- + FCBIT	RF ∨ BB --
9, SUB	RF - BB - 1	RF - BB	RF - BB -1 + FCBIT	(RF ⊻ BB) --
10, BBN	(RF∧BB) + (RF∨BB --)	(RF∧BB) + (RF∨BB --) +1	(RF∧BB) +(RF∨BB--)+ FCBIT	BB --
11, NOR	RF + (RF∨BB --)	RF + (RF∨BB --) + 1	RF + (RF∨BB--) + FCBIT	(RF∨BB) --
12, MINUS1	-1	0	-1 + FCBIT	1 (Logical)
13	(RF∧BB --) -1	RF ∧ BB --	(RF ∧ BB --)-1 + FCBIT	RF -- ∨BB
14	RF∧ BB-1	RF∧BB	RF∧BB-1 + FCBIT	(RF∧BB) --
15, RFNOT, ANOT,DEC .	RF-1	RF	RF-1 + FCBIT	RF --

ALU	Arithmetic and logic Unit
AL	ALU output
RF	Register File
BB	B Bus
FCBIT	Carry bit

+	Add (ArithmeticADD)
-	Subtract (arithmetic Minus)
∧	AND
∨	Inclusive OR
⊻	exclusive OR
--	NOT

Figure 5

Figure 6

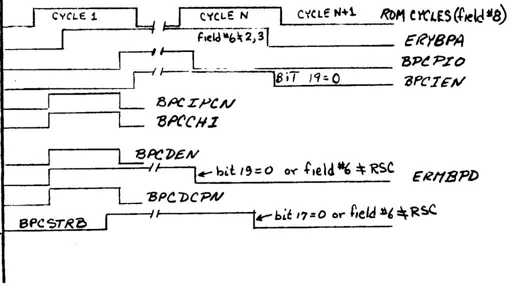
Fields #6, #7: REGISTER FILE SELECTION

Field #6 =	BFRA= Field #7 =	Register Selected	Description/Comment (Std. Macro-Instruction Use)
0,M, (blank)	0,RX, (blank)	0, RX	X Register: Index Register
	1, RA	1, RA	A Register: Arith, Shift, I/O
	2, RB	2, RB	B Register: Ext Arith, Shift
	3, RS	3, RS	Stack Pointer
	4, FLTH	4, FLTH	Floating Point High
	5, FLTL	5, FLTL	Floating Point Low
	6, VSC	6, VSC	Visible shift counter Accumulate
	7, RP	7. RP	P Register: Program Counter
	10,PMAR	10,PMAR	Page Map Address Register
	11,	11,	Scratch: μ-code Scratch Location
	12,EAS	12,EAS	Effective Address Save for ILL, UII,
	13	13	Scratch: μ-code Scratch Location
	14,YSAVE	14,YSAVE	Scratch: RY Save for Control Panel & DMA
	15,MSAVE	15,MSAVE	Scratch :RM Save for Control Panel & DMA
	16,RSC SAVE	16,RSC SAVE	Scratch÷RSC Save Location
0,M, (blank)	17	17	Scratch: μ-code Scratch Location
1,XM	0, (blank)	0,RX	Select Index Register
1,XM	1,DISABLE	-	Disable Register File
2,RY	0, (blank)	RY	RY12-16=BFRA
3,RSC	0 (blank)	RSC	RSC12-16=BFRA
		Function Generated	
2,RY	1, PIO	ERYBPA	Enable RY on to BPA
		BPCPIO	BPA stable for PIO transfers
2,RY	2, IEN	BPCIEN	Staticize Priority Interrupts
2,RY	4, ICPN	BPCICPN	Reset Interrupt Priority Net
2,RY	10, ICAI	BPCCHI	Clear Highest Active Priority Interrupt
3,RSC	1, ENB	BPCDEN	Staticize DMX Requests
3,RSC	2, DATA	ERMBPD	Enable RM on to BPD
3,RSC	4, CPN	BPCDCPN	Reset DMX Priority Net
3,RSC	10, STROBE	BPCSTRB	Strobe DMX Transfer

Expansion of Field #7 for Field #6 = 1-3

Field #6 = 1, XM	Field #6 = 2, RY	Field #6 = 3,RSC
Field #7 =	Field #7 =	Field #7 =
0,(Blank) 1, Disable All even numbers operate same as 0: odd as 1. future Prime processors may use 2-17 for additional capabilities	0, (Blank) 1, PIO 2, IEN 3=1∧2 4, ICPN 5=1∧4 6=2∧4 7=1∧2∧4 10,ICAI 11=1∧10 12=2∧10 13=1∧2∧10 14=4∧10 15=1∧4∧10 16=2∧4∧10 17=1∧2∧4∧10	0, (Blank) 1, ENB 2, DATA 3=1∧2 4, CPN 5=1∧4 * 6=2∧4 7=1∧2∧4 10,STROBE 11=1∧10 12=2∧10 13=1∧2∧10 14=4∧10 15=1∧4∧10 * 16=2∧4∧10 17=1∧2∧4 ∧10

*resulting action is questionable



Field #8: CLOCK CONTROL PRIME 200

Clock Control Field # 8 =	BUS to Register Transfer at end of cycle		Nominal Cycle Time (ns)	Cycle Extended Until
	Source	Destination		
0,200, (blank)	-	-	200	-
1, RM160	BD	RM	160	-
2, RM200	BD	RM	200	-
3, RMMRDY	BMD	RM	240	MRDY
4, RM280	BD	RM	280	YBSY--
5, RMRFMRDY	BMD BD	RM RF	280	MRDY
6, 280	-	-	280	-
7, RY280	BD	RY	280	YBSY--
8, RY200	BD	RY	200	YBSY--
9, RF160	BD	RF	160	-
10,RF200	BD	RF	200	-
11, RF240	BD	RF	240	-
12, RF280	BD	RF	280	-
13, RYRF280	BD BD	RY RF	280	YBSY--
14,CLMCLFCLI	BMD BD BRM BRM	RM RF F01 F02	280	MRDY
15, RMRF280	BD BD	RM RF	280	-

RM = Memory Data Register

BRM = RM Bus
Note: BRM = BD if RCM25 = 0
= BMD if RCM25 = 1

BMD = Memory Data Bus

MRDY = Memory Access Completed

YBSY-- = Memory Cycle Completed

-- = "NOT"

BD = D=Bus

F01 = Indirect Address Mode

F02 = Indexed Address Mode

Figure 7

Field #8: CLOCK CONTROL PRIME 300

Clock Control Field # 8 =	Bus to Register Transfer at end of cycle		Nominal Cycle Time (ns)	Cycle Extended Until
	Source	Destination		
0, 200	-	-	200	-
1, RM160	BD	RM	160	-
2, RM200	BD	RM	200	YBSY-
3, RY240	BD	RY	240	YBSY-
4, RM280	BD	RM	280	-
5, RMRFMRDY	BMD BD	RM RF	240	MRDY
6. 280	-	-	280	
7, RMMRDY	BMD	RM	200	MRDY
8, RY200	BD	RY	200	YBSY-
9, RF160	BD	RF	160	
10, RF200	BD	RF	200	
11, RF240	BD	RF	240	
12, RF280	BD	RF	280	
13, RYRF240	BD BD	RY RF	240	YBSY-
14, RY280	BD	RY	280	YBSY-
15, CLMCLFCLI	BMD BD BRM BRM	RM RF F01 F02	200	MRDY

RM = Memory Data Register

BRM = RM Bus
Note: BRM = BD if RCM25 = 0
= BMD if RCM25 = 1

BMD = Memory Data Bus

MRDY = Memory Access Completed

YBSY-- = Memory Cycle Completed

-- = "NOT"

BD = D=Bus

F01 = Indirect Address Mode

F02 = Indexed Address Mode

Figure 7A

Figure 8

Field #9: CARRY SOURCE, HSM OPERATION

Field #9 =	Signal Enabled on to Carry	Field #9 =	Memory Action Initialized
		Mapped Reference, if in Page Mode; if not, Absolute Refer.	
0, NOP, CNOP, (blank)	None (Link disabled)	8, MREAD	Read Memory (HSM-01-16→ BMD 01-16)
1, DIVER,LINKS	Divide overflow, or Link source	9, MRBW	Write Right Byte (RM01-16 → BMD 09-16 → HSM 09-16; HSM 01-08 unchanged)
2, COUT	True 16 Bit carry	10, MLBW	Write Left Byte (RM01-16 → BMD 01-08 →HSM 01-08; HSM 09-16 unchanged)
3, BD01	D-Bus bit 01 (BD01)	11, MWRITE	Write Memory (RM01-16 → BMD 01-16 →HSM 01-16)
		Absolute Reference, independent of Page Mode	
4, LINK	None (No change but LINK enabled)	12, AREAD	Read Memory (HSM01-16 → BMD 01-16)
5, RF01	Register File bit 01 (RF01)	13, ARBW	Write Right Byte (HSM 01-08 unchanged; RM01-16 → BMD09-16 → HSM09-16)
6, SOVFL	Shift Overflow (ALFHFT)	14, ALBW	Write Left Byte (HSM09-16 unchanged RM01-16→BMD 01-08 → HSM01-08)
7, AOVFL	Arithmetic Overflow (ALOVFL)	15, AWRITE	Write Memory (RM01-16 →BMD01-16 → HSM01-16)

HSM xx = Main Memory Word bit xx
BMD xx = Memory Data Bus bit xx

Figure 9

Field #10: INDEPENDENT ACTION

Field #10 =	Independent Action
0, JAMF	0→RCY05-16 (Go to Fetch)
1, INCRSCF	RSC + 1→RSC; 0→RCY 05-16 (Increment Shift Register (RSC); Go to Fetch)
2, RESTJAMF	Cause a Restricted Execution Trap if Restricted Execution Mode is set; Go to Fetch
3	0→RCY05-16 (Go to Fetch) Future processor models may expand definition
4, NOP, (blank)	No Operation
5, INCRSC	RSC + 1→RSC (Increment Shift Register (RSC))
6, REST	Cause a Restricted Execution Trap if Restricted Execution Mode is set
7, SETCC	Set Condition Codes: Staticize "AL01--", "AL01-16 = 0" at end of cycle until next SETCC
8, LOAD256K	RY15,16 → RY99,00 if RCM02=0. BPA99,00 → RY99,00 if RCM02=1 (set hiah order Memory Address bits for DMA, DMT)
9, FORCERD	Force absolute memory reads without changing memory request status flops (MWLB, MWRB). Used to read memory map after a paqe fault. Will override a concurrent memory write request.
10,HSMRESUME	Resume interrupted memory operation; ie. start another memory cycle using previously set but interrupted, memory request status flops.
11, DISABLERHBB	0→ BB09-16, (Force right hand Byte of B Bus to zero)
12, PUSHBD	BD→BCS→RCS and Push stack: Field #3 must be = 0: (D Bus to μ-code stack)
13, EAF	In 16K Address mode: 0 → BD01, 02; In 32K Mode: 0→BD01 (truncate D Bus per address mode for effective address formation)
14, LOADRSC	BB09-16→RSC09-16 (Load Shift Counter from B-Bus)
15, CLEARFUII	Clear FUII status flop (FUII can be set (Field #11= 2,#12 = 10), reset (Field #10 = 15) and tested (field #11 = 1,#12 = 21);normally used to staticize unimplemented instruction traps)

RCY (xx)	Control store address register (bit xx)
AL (xx)	Arithmetic and logic unit output (bit xx)
RY (xx)	Y Register - memory / peripheral address regiser (bit xx)
BB (xx)	B Bus (bit xx)
BD (xx)	D Bus (bit xx)
BCS	Control store bus

Summary of Field #11 and Field #12

Field #11	Field #12	
0, EMIT, (blank),	RCM47→BBLP RCM48 - 55→BB01-08 RCM56→BB RP RCM57 - 64 →BB09-16	if field #2 selects RCM
DATA	RCM 48-55 → BB01-08 RCM 57-64→ BB09-16 BBLP and BBRP generated	if field #2 selects RCM
1, J, JUMP	Jump condition: If the conditions decoded from bits RCM47-52 and listed below are true, then RCM53-64 → BCS05-16→BCY05-16→RCY05-16.	
2, M, EAC, MODAL	Emit Action is selected by RCM47-50; RCM54 Triggers nonvisable keys; RCM56 Triggers visible keys; RCM 57-64 contain the keys	
3,S	Jump Stack Condition & Fetch Logic Control; if the condition decoded from bits RCM 47-52 are true, create jump address (RCY05-16) as specified by RCM61-64 and check indexing conditions as enabled by RCM56-60; if false go to RCY + 1, but still check indexing conditions.	

Figure 10

Field #12: JUMP FIELD (for field 11 = 1)

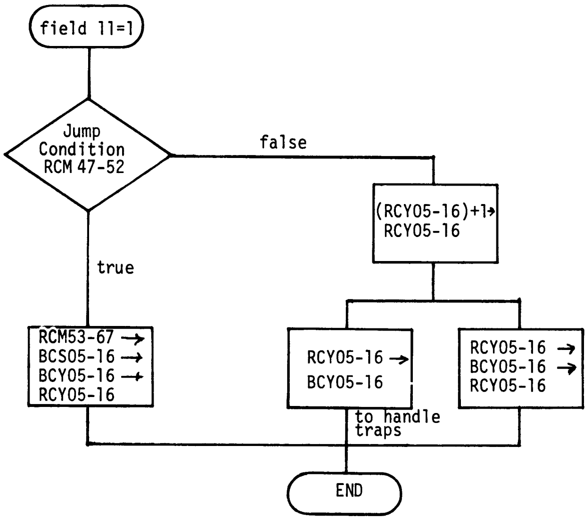


Figure 11A

Figure 11B

Field #12: JUMP FIELD (for field #11 = 1)

Field #12 =	Signal Tested	Description
0,T,True,(blank)	-	Unconditional branch
1, RM01	RM01 +	Sign or Indirect bit
2, RMGEM240	GDL240-	RM08-16≥ - 240; Relative mode addressing
3, BPSP1	BPCMOD2+	I/O Mode line; Interrupt: increment memory or external interrupt; INA: clear or OR
4, NE	FALZERO-	Test condition codes for not equal to zero
5, INPUT	BPCINMD+	DMX in input mode (set by external controller)
6, GE	FALMINUS-	Test condition codes for not minus;ie., greater than or equal to zero
7, LE	GJCLE+	Test condition codes for less than or equal to zero (FALZERO ∨ FALMINUS)
8, SKIP	FSKIP+	Staticized version of skip net
9, RSCNEM1	GRSC6Z-	(RSC) not equal to - 1; used for looping as in shift instructions
10, FETCH1	GJCFSP+	Any condition that require the fetch cycle be stopped: master clear, control panel, external interrupt, power failure.
11, SYSCLRNOT	FYSYSCLR-	Not master clear
12, CP	FRUN-	Control panel request
13, REL,AM1	FAM1 +	Relative mode, address mode #1
14, INDIRECTEND	GJCNMI+	End of indirect addressing chain
15, FCBIT	FCBIT+	Carry bit
16, STACKOP, RS15	GJCSOP +	Stack operation; (F01-- ∨ F02--) ∧ RS15
17, RELDONE	GJCRELD +	Effective address completed for relative mode; (F01--∧F02--)
18, PUSH, RS16	RS16+	Push on to stack register (ie:RS16=1); (pop= (RS16=0))
19, FETCHDONE, F01NOT	GJCNMAF+	Fetch completed (used as F01--in PI/O instructions)
20, OTANOT	$\overline{\text{F01.F02}}$	$\overline{\text{F01.F02}}$
21, FUII	FUII+	FUII Register/Flop; (used to indicate implemented instructions;set by field #11=2 and field #12=SETFUII; reset by field #10=15
22, DANOT20	GJCPA20-	DeviceAddress not equal to 20
23, READYNOTANDNE20	GJCRDY20-	Device not ready (external signal) and device address not equal to 20.
24,READYANDF02	FPCREDY∧F02	Device ready andOTA (F02)
25, FDRQ	FDRQ+	DMX request pending
26, RSC11	RSC11+	RSC11
27, RSC12	RSC12+	RSC12
28, DP	FDPM+	Double precision mode
29, DMX, STATREQ	FREQSEN-	Last DMX enable did not catch a DMX request
30, DMANOTVALID	DMA-VALID	Not a DMA request
31, DMTNOTVALID	DMT-VALID	Not a DMT request
32, FALSE	-	Do not jump
33, RM01NOT	RM01-	Not minus or not indirect
34, RMLTM240	GDL240+	RM01-16<-240
35, BPSP1NOT	BPCMOD2-	I/O Mode line
36, EQ	FALZERO +	Test condition codes for equal to zero
37, OUTPUT	BPCINMD-	DM in output mode (set by external controller)
38, LT	FALMINUS+	Test condition codes for "less than" or minus
39, GT	GJCLE-	Test condition codes for greater than (FALMINUS--∧FALZERO--).
40, DMCVALID	DMCVALID-	DMC mode of transfer selected by external device
48, EVIM	FVIM+	Vectored interrupt mode (set by field #11=2, field #12 bit 54=1,bit 61=1 and RM15=1;reset by field #11=2 and either field #12=8 or bit 54=1 and bit 61=0 and RM15=0)
49, MC	FMCHK+	Machine check error detected
50, MCNOT	FMCHK-	Machine check error not detected
51, VERIFY	MCV + (PLUPN)	A Micro verification check requested
52, RXM	FRXM +	Restricted execution mode
53, DMTOUTNVAL		DMT output is not valid
54, DMTINNVAL		DMT input is not valid
55, NOTRF16		RF16 is not set this cycle requires a 280 ns clock
56, SPECIALVALID	DMCSPLVD	Special DMX mode (=00) selected by external device
41-47,	-	Reserved
57-63	-	Reserved

Figure 12A

Field #12: EMIT ACTION (for field #11 = 2)

Field #12* RCM 47-50 =	Emit Action
0, NOP, (blank)	None
1, LOADSERIALINT	Load the Serial Interface: BD13-16 → FS0 1-4
2, JST	Replace BD01,02 with BB01, 02 in 16S mode; BD01 with BB01 in 32S or 32R mode; and make no replacements in 64R mode; In normal use as a JST, field #1=RF, Field #2=RM field #6=M, field #7=RP
3, MPYLOGIC	If Link=1, AL01-16→BD01-16; if link=0, RF01-16→BD01-16.
4, DIVLOGIC	If AL01 or FALMINUS =1, RF01-16→BD01-16; if=0, AL01-16→BD01-16
5,	Reserved
6, LOADCAM	RM01-16→entry part of CAM;RY8-16→Associative part of CAM
7, CLEARCAM	CAM forced to "no match" condition
8, CLRNVKEYS	Clear the non-visible keys (RXM, FSYSCLR, PAM and EINTM) if triggered by RCM54=1
9, PLOADVKEYS	BD01-08→Visible keys if triggered by RCM56=1
10, SETFUII	1→FUII (double word fetch cycle flag)
11, LOADF01F02	BB01→F01 (indirect addressing); BB02→F02 (indexed addressing)
13	Reserved
14	Reserved
15	Reserved
RCM51-64	
51	Reserved
52	Reserved
53	Reserved
54, TRIGNVKEYS	Trigger non-visible keys (required for CLRNVKEYS & defines RCM57-64 as non-visible keys
55	Reserved
56, TRIGVKEYS	Trigger visible keys (required for PLOADVKEYS & defines RCM52-64 as visible keys)

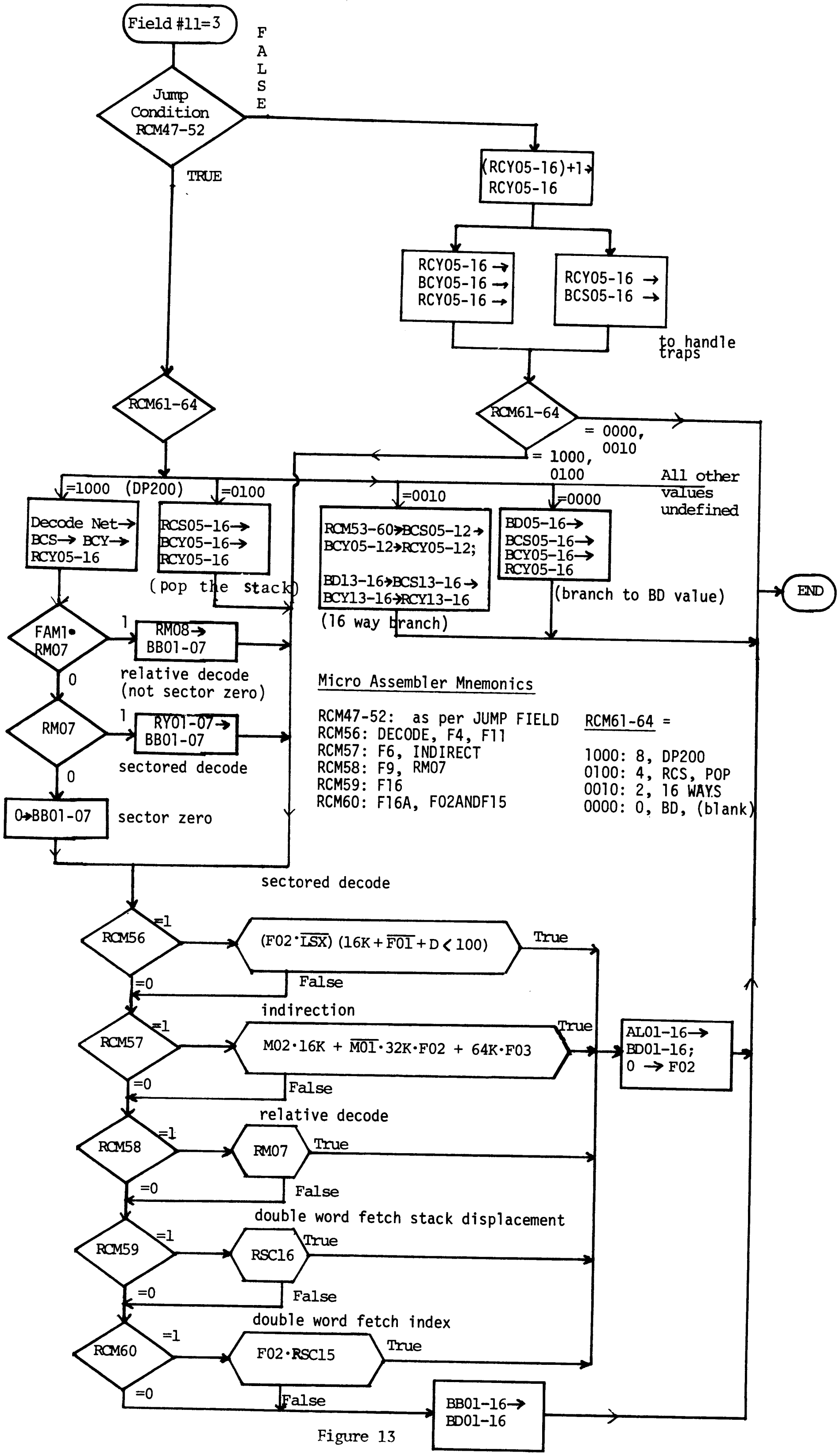
*Enclose multiple parameter of field #12 in parentheses

Figure 12B

EMIT ACTION (continued)

Field #12 RCM 57-64 (Visible Keys)	Enable conditions to update visible keys (if RCM56=1, TRIGVKEYS) Condition	Condition set by	Condition reset by
57	Spare		
58, DP	Double precision mode	RM15=1	RM15 = 0
59	Spare		
60	Spare		
61, AM1	Address mode bit #1	RM07=1	RM07= 0
62, AM2	Address mode bit #2 AM1 AM2 Mode 0 0 16S 0 1 32S 1 0 64R 1 1 32R	RM15=1	RM15=0
63	Spare		
64	Spare		
Non Visible Keys	Enable conditions to update non-visible keys (if RCM 54=1,TRIGNVKEYS		
57	Spare		
58	Spare		
59, PAM	Paged address mode	RM15=1	RM15=0
60, EINTM	External interrupts	RM08-1	RM08=0
61, VIM	Vector interrupt mode	RM15=1	RM15=0
62, MCHK	Machine check mode	parity error	RM15=0
63, PARIM	Parity error vector enabled	RM15=1	RM15=0
64, RXM	Restricted execution mode	RCM64	CLRNVKEYS

Field #12: JUMP STACK/FETCH LOGIC (for field #11 = 3)



Micro Assembler Mnemonics

RCM47-52: as per JUMP FIELD
RCM56: DECODE, F4, F11
RCM57: F6, INDIRECT
RCM58: F9, RM07
RCM59: F16
RCM60: F16A, F02ANDF15

RCM61-64 =

1000: 8, DP200
0100: 4, RCS, POP
0010: 2, 16 WAYS
0000: 0, BD, (blank)

Figure 13

APPENDIX B

MICROCODING MACROS

RR Macros

Format:

```
(Label)  RR  <1>  =   <2>  <3>  <4>    etc.
```

Symbols:

Argument #	Definition	Symbols	Restriction
<1>	Source	Any BB source	
		Any Register in the Register File	No "indirect" Addressing
		(Blank), NULL	
<2>	Destination	RM, RY or	
		Any Register in the Register File	
		(Blank), NULL	
<3>	Independent Action	Same as field 10 of CPU	
<4>	Emit	Same as fields 11, 12 of CPU	

Argument <2> may be ORed, i.e., (RM,RA) is legal.

Special Symbols: (Optional)

= may be used to define RCM data fields, i.e., RCM = '17120 will put an OTA 1720 with proper parity on the D Bus.

C = may be used to define anything in CPU macro argument 9 (HSM + Carry)

TR = may be used to define anything in CPU macro argument 3 (trap control). Overrides normal RR selection of traps.

Examples:

```
RR     RCM  =  3412  =>  RA

RR     RM   => RB  NOP JUMP ON NE TO S12

RR     RA   => RY
```

ALU Macro

Format 1:

```
(Label)  ALU  <1>  <2>  <3> => <4>   <5>   <6>, etc.
```

Symbols:

Argument #	Definition	Symbols	Restrictions
<1>	Argument 1	Any Register in the Register File	No "indirect" addressing
		Zero	Register File can't be destination
<2>	Operator	AND	Logical AND
		OR	Logical OR
		XOR	Logical XOR
		NOR	Logical $\overline{\text{OR}}$
		ADD	1 + 3
		PLUS	1 + 3
		SUB	1 - 3
		MINUS	1 - 3
		SUB1	1 - 3 - 1
		MINUS1	1 - 3 - 1
<3>	Argument 2	Any BB Source	

ALU Macro

<4>	Destination	Same as RM, RY Any Register in the Register File
<5>	IAC	Same as field 10 of CPU
<6>	Emit	Same as CPU 11, 12

Special Symbols: (Optional)

= Same as RR

C = Same as RR

TR = Same as RR

+ Followed by CPU Field 5 symbol to force carry in or logic select (normal ALU select for field 5 is appropriate for command). "INC RA + 0" is a way to get RA to the condition code.

FLIP BYTES This selects the Byte swapped output of the ALU.

Format 2:

```
(Label)  ALU  <1>  <3> = <4>  <5>  <6>  etc.
```

Symbols:

Argument #	Definition	Symbols	Restrictions
1	Operator	INC	
		NOT	
		DEC	Only for Registers
		CON	Argument 2 must be ZERO (or 0) or MINUS1 (or -1). Generates the constant

APPENDIX B (cont)

ALU Macro

Argument #	Definition	Symbols	Restrictions
<3>	Argument	Any BB source or Any Register in the Register File	
<4> to <6>		Same as Format 1 ALU 4 6	

Special Symbols: (Optional)

(All used for Format 1 apply)

Examples:

```
ALU    RA    PLUS RCM    =    3412 =  RA

ALU    INC RM => RM   SETCC  JUMP ON NE TO S12

ALU    RM   MINUS ZERO    =>RM

ALU    RS   PLUS   RM =>(RS, RY)

ALU    INC   RA   + 0=>NULL   SETCC
```

APPENDIX C

PRIME 300 MICROCODE

```
            (0001) *       300D,U-CODE,MHJ,FEBRUARY 1974
            (0002) *       PRIME 300 MICRO-CODE
            (0003) *       PRIME COMPUTER INC.,SRC0768.001
            (0004) *       COPYRIGHT 1974,PRIME COMPUTER INC.,NATICK MASS.
            (0005) *
      000001 (0006) P300   XSET     1
            (0007) *      MA,SRCLIB,MHJ,FEBRUARY 1974
            (0008) *      PRIME MICRO-CODE ASSEMBLER
            (0009) *      PRIME COMPUTER INC.,0702.002
            (0010) *      COPYRIGHT 1974,PRIME COMPUTER INC.,NATICK MASS.
            (0011) *
            (0012) *
            (0013) *           **********************************
            (0014) *           *                                *
            (0015) *           *            P R I M E           *
            (0016) *           *                                *
            (0017) *           *      MICRO-CODE ASSEMBLER      *
            (0018) *           *                                *
            (0019) *           *          SRC0702.002           *
            (0020) *           *                                *
            (0021) *           **********************************
            (0022) *
            (0023) *
            (0855)         IDNT     (PRIME 300D MICRO-CODE),(JANUARY 2,1974) ;
            (0856)                  (FLOATING POINT),( IS INCLUDED)
            (0857) *
            (0858) *                FETCH CYCLE
            (0859) *
            (0860) *       BEGIN FETCH CYCLE, TEST FOR NEED TO INTERRUPT PROCESSING
            (0861) *
            (0862) F1      RR       RP => RY  CLEARFUII ;
            (0863)                  JUMP ON FETCH1 TO FHALT
000: 0200 780F 0004 A030
            (0864) *
            (0865) *       INCREMENT P COUNTER, READ INSTRUCTION
            (0866) *
            (0867) F3      CPU      AL    0     ALL    INC     1      M      RP ;
            (0868)                  CLMCLFCLI MREAD ,  ;
            (0869)                  JUMP ON REL TO F9
001: 8304 7F84 0004 D004
            (0870) *
            (0871) *       SECTORED DECODE, BUILD EA WITH PRE-INDEXING IF NEEDED
            (0872) *
            (0873) F4      CPU      BB    RM    NONE   ADD     0      XM     0 ;
            (0874)                  RY280       NOP    EAF ;
            (0875)                  S FETCHDONE TO DP200 DECODE
002: EC61 0E0D 000D 3108
            (0876) *
            (0877) *       INDIRECTION REQUIRED
            (0878) *
            (0879) F5      CPU      0     0     ALL    0       0      0      0 ;
            (0880)                  RMMRDY      MREAD NOP ;
            (0881)                  GO TO F6
003: 0300 0784 0004 0010
            (0882) *
            (0883) *       RELATIVE DECODE, BUILD RELATIVE EA
            (0884) *
            (0885) F9      CPU      BB    RM    NONE   ADD     0      M      RP ;
            (0886)                  RY280       NOP    EAF ;
            (0887)                  S FETCHDONE TO DP200 RM07
004: EC60 7E0D 000D 3048
            (0888) *
            (0889) *       TEST FOR DOUBLE WORD OR STACK OPERATION
            (0890) *
            (0891) F10     RR       ,,     NOP ;
            (0892)                  JUMP ON RMLTM240 TO F13
005: 0200 0004 0006 2008
            (0893) *
            (0894) *       PRE-INDEX FOR RELATIVE MODE
            (0895) *
            (0896) F11     CPU      BB    RY    ALL    ADD     0      XM     0 ;
            (0897)                  RY240       NOP    EAF ;
            (0898)                  S FETCHDONE TO POP F11
006: E861 030D 000D 3104
            (0899) *
            (0900) *       INDIRECTION REQUIRED
            (0901) *
            (0902) F12     CPU      0     0     ALL    0       0      0      0 ;
            (0903)                  RMMRDY      MREAD NOP ;
            (0904)                  GO TO F6
007: 0300 0784 0004 0010
            (0905) *
            (0906) F13     RR       RP => RY  NOP TR= ALL ;
            (0907)                  JUMP ON STACKOP TO F17
008: 0300 7804 0005 000D
            (0908) *
            (0909) *       SECOND OPERAND FETCH
            (0910) *
            (0911) F14     CPU      AL    0     ALL    INC     1      M      RP ;
            (0912)                  RMRFMRDY  MREAD EAF ;
            (0913)                  EAC  SETFUII
009: 8304 758D 000A 8000
            (0914) *
            (0915) *       DOUBLE STACK INDEX, IF REQUIRED
            (0916) *
            (0917) F16     CPU      BB    RM    ALL    ADD     0      M      RS ;
            (0918)                  RY240       NOP    EAF ;
            (0919)                  S RELDONE TO POP F16
00A: EF60 330D 000D 1024
            (0920) *
```

```
                   (0921) *      DOUBLE INDEX ON RX, IF REQUIRED
                   (0922) *
                   (0923) F16A   CPU    BB    RY   ALL   ADD    0     XM    0 ;
                   (0924)               RY240      NOP   EAF ;
                   (0925)               S FETCHDONE TO POP F16A
008: E861 030D 000D 3014
                   (0926) *
                   (0927) *      DOUBLE INDIRECT
                   (0928) *
                   (0929) F16B   CPU    0     0    ALL   0      0     0     0 ;
                   (0930)               RMMRDY.    MREAD NOP ;
                   (0931)               GO TO F6
00C: 0300 0784 0004 0010
                   (0932) *
                   (0933) *      STACK OPERATION PROCESSING
                   (0934) *
                   (0935) F17    RR     RS => RY   NOP ;
                   (0936)               JUMP ON PUSH TO F19
00D: 0200 3804 0005 2014
                   (0937) *
                   (0938) *      PRE-DECREMENT OR POP PROCESSING
                   (0939) *
                   (0940) F18    CPU    AL    0    ALL   INC 1  M     RS ;
                   (0941)               RF200      NOP   EAF ;
                   (0942)               S ON RELDONE, POP
00E: 8304 3A0D 000D 1004
                   (0943) F18A   CPU    ,,ALL,,,,,RMMRDY   MREAD
00F: 0300 0784 0000 0000
                   (0944) *
                   (0945) *      INDIRECTION AND POST INDEXING
                   (0946) *
                   (0947) F6     CPU    RF    RCM  NONE  0      0     M     RA ;
                   (0948)               RF160      ,LOADRSC  DATA  -8
010: 1000 190E 0003 FEF8
                   (0949) F6A     CPU    BB    RM   NX    ADD    0     XM    0 ;
                   (0950)               RY240      NOP   EAF ;
                   (0951)               S INDIRECTEND TO POP INDIRECT
011: EE61 030D 000C E084
                   (0952) *
                   (0953) *      MULTIPLE LEVELS OF INDIRECTION
                   (0954) *
                   (0955) F7     CPU    0     0    ALL   0      0     0     0 ;
                   (0956)               RMMRDY     MREAD INCRSC ;
                   (0957)               JUMP ON RSCNEM1 TO F6A
012: 0300 0785 0004 9011
                   (0958) F7A    RR     ,,REST     GO TO F6A
013: 0200 0006 0004 0011
                   (0959) *
                   (0960) *      POST-INCREMENT STACK OP---PUSH
                   (0961) *
                   (0962) F19    CPU    AL    0    ALL   DEC    0     M     RS ;
                   (0963)               RYRF240 NOP  EAF ;
                   (0964)               S ON RELDONE TO POP
014: 83F0 3D0D 000D 1004
                   (0965) *
                   (0966) *      INDIRECT PUSH
                   (0967) *
                   (0968) F20    CPU    0     0    ALL   0      0     0     0 ;
                   (0969)               RMMRDY     MREAD NOP ;
                   (0970)               GO TO F6
015: 0300 0784 0004 0010
                   (0971)       EJCT
```

```
                     (0972) *
                     (0973) *      EXTERNAL INTERUPT
                     (0974) *
                     (0975) INT3   RP       ,,CLEARFUII JUMP ON FUII TO INT3
016: 0200 000F 0005 5016
                     (0976)        CPU      BB   BPA  NX    0      0     0     0 ;
                     (0977)                 RY280      NOP   NOP ;
                     (0978)                 JUMP ON BPSP1 TO MI1
017: F600 0E04 0004 30BF
                     (0979)        ALU      CON ZERO => RM  TR= NONE ,JUMP ON FVIM TO NVECT
018: 803C 0204 0007 001A
                     (0980)        CPU      BB   RCM  NONE  0      0     RY    ICAI ;
                     (0981)                 RY200      ,,    DATA '63
019: F002 8804 0002 0133
                     (0982) NVECT  CPU      ,,ALL,,,RY,ICPN,RMMRDY  AREAD,, ;
                     (0983)                 MODAL  CLRNVKEYS  (TRIGNVKEYS,EINTM)
01A: 0302 47C4 000A 0410
                     (0984)        CPU      AL   RM   NONE  BB     L     0     0 ;
                     (0985)                 RY240 ,    SETCC  GO TO AVECT+2
01B: 8C5C 0307 0004 0079
                     (0986) *
                     (0987) *       MEMORY INCREMENT.  THE LOCATION SPECIFIED BY THE BPA IS
                     (0988) * READ AND THEN INCREMENTED.  AFTER THIS , IT IS WRITTEN
                     (0989) * BACK.  IF IT INCREMENTED TO 0, END OF BLOCK IS GENERATED
                     (0990) * ALONG WITH ICPN.  IF NOT, END OF BLOCK (BPCEOB) IS
                     (0991) * GENERATED AS ZERO.  BPCEOB = CARRY OUT OF BIT 1 OF THE ALU.
                     (0992) *
                     (0993) MI2    ALU      INC RM => RM    SETCC
01C: 8E65 1207 0000 0000
                     (0994)        RR       ,,C= AWRITE , JUMP ON NE TO *+2
01D: 0200 00F4 0004 401F
                     (0995)        CPU      0    RM   1     MINUS1 1     RY    ICPN ;
                     (0996)                 280  ,    ,    GO TO CP8
01E: 0DC6 4604 0004 0020
                     (0997)        CPU      0    RM   1     MINUS1 0     RY    ICPN ;
                     (0998)                 280  ,    ,    GO TO CP8
01F: 0DC2 4604 0004 0020
                     (0999)       EJCT
                     (1000) *      BEGIN PROCESSING THE FUNCTIONS
                     (1001) *
                     (1002) *
                     (1003)        ORG      $20
                     (1004) *               1    STOP/STEP  (OTHERS FOLLOW IN SEQUENCE)
                     (1005) *
                     (1006) CP8    RR       RP => RY  NOP  TR= ALL ;
                     (1007)                 GO TO F3
020: 0300 7804 0004 0001
                     (1008) CP9    RR       YSAVE => RY      NOP ;
                     (1009)                 GO TO CP1
021: 0200 C804 0004 0035
                     (1010) CP10   ALU      INC YSAVE => YSAVE      , ;
                     (1011)                 JUMP ON LT TO CP9
022: 8204 CA04 0006 6021
                     (1012) CP11   RR       YSAVE => RY      , GO TO CP19
023: 0200 C804 0004 002C
                     (1013) CP12   RR       RA => RM  , GO TO BOOT
024: 0200 1204 0004 0091
                     (1014) *
                     (1015) *       CLEAR THE ACTIVE REGISTER.  BIT 1 CLEARS THE DATA.
                     (1016) *
                     (1017) CP13   CPU      AL   0    NX    ZERO   L     RSC   0 ;
                     (1018)                 280  NOP  SETCC ;
                     (1019)                 JUMP ON LT TO CP15
025: 823F 0607 0006 6027
                     (1020) *
                     (1021) *       ADDRESS CHANGES THE RSC ADDRESSED REGISTER FROM
                     (1022) * MSAVE TO YSAVE.
                     (1023) *
                     (1024) CP14   CPU      BB   RCM  NX    0      0     RSC   0 ;
                     (1025)                 200  NOP  LOADRSC ;
                     (1026)                 DATA $C
026: F203 000E 0002 010C
                     (1027) *
                     (1028) *       THE SAVED REGISTER SELECTED IS OUTPUT TO THE LIGHTS,
                     (1029) * THEN THE DATA SWITCHES ARE ORED INTO IT.
                     (1030) *
                     (1031) CP15   CPU      RF   0    NX    ,,     RSC   DATA  RM200
027: 0203 2204 0000 0000
                     (1032) CP16   CPU      0    0    NX    0      0     RSC   (DATA,STROBE) ;
                     (1033)                 280
028: 0203 A604 0000 0000
                     (1034)        CPU      BB   RCM  NX    0      0     RSC   0 ;
                     (1035)                 RY200      ,,DATA '131720
029: F203 0804 0001 6600
                     (1036) CP17   CPU      AL   BPD  NX    OR     L     RSC 0 RF280 , ;
                     (1037)                 ,JUMP ON NE TO CP3
02A: 9A4F 0C04 0004 4037
                     (1038) CP18   CPU      AL   0    NX    ZERO   L     RSC   0 ;
```

```
                   (1039)                RF280     ,, GO TO CP3
028: 823F 0C04 0004 0037
                   (1040) CP19   CPU     RF   0    NX    0      0     M     MSAVE ;
                   (1041)                RM280     AWRITE       ,GO TO CP3
02C: 0200 D4F4 0004 0037
                   (1042)       EJCT
                   (1043) *
                   (1044) *      TEST FOR CONTROL PANEL REQUEST, IF NOT, DO EXTERNAL INT.
                   (1045) *
                   (1046) FHALT3 RR      ,, TR= ALL  ,JUMP ON CP TO CP1
02D: 0300 0004 0004 C035
                   (1047) *
                   (1048) *      EXTERNAL INTERRUPT PROCESSING.
                   (1049) * IENB IS VALID FOR THE THIRD STEP.
                   (1050) * IENB BEGINS 480 NS EARLIER AT THE START
                   (1051) * OF THE SECOND CYCLE.  BPSP1 IS TESTED, IF TRUE
                   (1052) * A MEMORY INCREMENT OPERATION IS PERFORMED.  IF FALSE, AN
                   (1053) * EXTERNAL INTERRUPT IS TAKEN.  THE VECTOR ADDRESS IS TAKEN
                   (1054) * FROM BPA IN VECTORED MODE, AND GENERATED AS '63
                   (1055) * IF IN COMPATIBLE MODE.
                   (1056) *
                   (1057) INTEX  CPU     0    0    NX    0      0     RY    IEN ;
                   (1058)                200  NOP  NOP ;
                   (1059)                EAC SETFUII
02E: 0202 2004 000A 8000
                   (1060)        CPU     0    0    NX    0      0     RY    IEN ;
                   (1061)                280 NOP NOP GO TO INT3
02F: 0202 2604 0004 0016
                   (1062)       EJCT
                   (1063) *
                   (1064) *
                   (1065) *      HALTED FETCH CYCLE, FIND WHAT NEEDS PROCESSING
                   (1066) *
                   (1067) *
                   (1068) *      MASTER CLEAR INVERSE TEST
                   (1069) *
                   (1070) FHALT  RR      ,,, TR= ALL  JUMP ON SYSCLRNOT TO FHALT2
030: 0300 0004 0004 8074
                   (1071) *
                   (1072) *      MASTER CLEAR ROUTINE CLEARS CARRY, MODALS, AND REGISTERS
                   (1073) *      0 TO '37. '1000 IS PUT INTO THE P COUNTER, AND THE CONTROL
                   (1074) *      PANEL ROUTINE WHICH FOLLOWS READS '1000 INTO THE LIGHTS.
                   (1075) *
                   (1076) *
                   (1077) MC1    RR      RCM = '1000 => RY      C= BD01      LOADRSC
031: F200 083E 0000 0500
                   (1078) MC2    CPU     AL   0    NX    ZERO   L     RSC   0 ;
                   (1079)                RF280     NOP   INCRSC ;
                   (1080)                JUMP ON RSCNEM1 TO *
032: 823F 0C05 0004 9032
                   (1081) MC3    CPU     AL   0    NX    ZERO   L     0     0 ;
                   (1082)                RM200     NOP   CLEARFUII ;
                   (1083)                JUMP ON VERIFY TO VIRY1
033: 823C 020F 0007 30C4
                   (1084) MC4    CPU     BB   RY   NX    0      0     M     RP ;
                   (1085)                RF280     NOP   NOP ;
                   (1086)                EAC CLRNVKEYS (54,56,58,59,60,61,62,63)
034: EA00 7C04 000A 057E
                   (1087)       EJCT
```

```
                           *
                           *
                           *      CONTROL PANEL ROUTINE.  THIS PROGRAM FIRST READS THE
                           * FUNCTION SWITCHES (INA 1520).  THE INFORMATION IS
                           * ENCODED IN THE LOW ORDER FOUR BITS OF THE WORD AND THE
                           * SIGN BIT.  A SIXTEEN WAY BRANCH IS THEN DONE WITH THE
                           * FIRST 8 BRANCHES DEFINED AS FOLLOWS
                           *              1     RUN OR STOP STEP
                           *              2     FETCH THIS
                           *              3     FETCH OR STORE NEXT (BIT 1 IS SET FOR FETCH NEXT)
                           *              4     STORE THIS
                           *              5     AUTO LOAD
                           *              6     CLEAR ADDRESS OR DATA (BIT 1 IS SET FOR DATA)
                           *              7     ADDRESS (NO SPECIAL FUNCTION--LIGHTS =ADDRESS)
                           *              8     DATA (NO SPECIAL FUNCTION--LIGHTS = ADDRESS)
                           *
                           *
                           *      REGISTERS YSAVE AND MSAVE HOLD THE WORKING
                           * COPIES OF THE ADDRESS AND DATA LIGHTS, RESPECTIVELY.  THESE
                           * ARE UPDATED EACH TIME THROUGH THE LOOP AND DISPLAYED ON THE
                           * NEXT PASS.  INA 1720 IS USED TO READ THE DATA SWITCHES AND
                           * OR THEM INTO THE SAVED REGISTERS.  OTA 1720 IS THEN USED TO
                           * DUMP OUT THE OLD IMAGE.
                           *
                           *
                           *
                           CP1     CPU      BB    RY    NX     0      0      M      YSAVE ;
                                            RMRFMRDY  AREAD NOP
035: EA00 C5C4 0000 0000
                           CP2     RR       RM => MSAVE    TR= NX
036: EE00 0A04 0000 0000
                           CP3     RR       RCM = '131520 => RY                   INA FUNCTION
037: F200 0804 0001 6750
                           CP4     CPU      BB    RCM   NX     0      0      RY     PIO ;
                                            280   NOP   LOADRSC ;
                                            DATA $D
038: F202 160E 0002 000D
                           CP5     CPU      BB    BPD   NX     0      0      RSC    STROBE ;
                                            RM280                     FUNCTION SWITCH TO RM
039: FA03 8404 0000 0000
                           CP6     CPU      BB    RCM   NONE   0      0      RSC    0 ;
                                            RY200      ,,DATA '171720
03A: F003 0804 0003 E600
                           CP7     CPU      AL    RM    NONE   BB     L      RSC    0 ;
                                            RM280      NOP   SETCC ;
                                            S ON TRUE 16WAYS TO CP8
03B: 8C5F 0407 000C 0022
                           *
                           *      REGISTER SAVE FOR DMX ROUTINE.
                           *
                           DMX2    RR       RY  =>  YSAVE
03C: E800 C904 0000 0000
                                   CPU      AL    RSC   NX     BB     L      M      RSCSAVE ;
                                            RF200      ,,JUMP ON DMTNOTVALID TO DMA
03D: 865C EA04 0005 F04B
                                   EJCT
```

```
                    (1135) *
                    (1136) *      DMT.  HIGHEST SPEED OF ALL TRANSFERS, THE PERIPHERAL
                    (1137) * ADDRESS LINES ARE READ INTO RY AND DIRECTLY SELECT THE
                    (1138) * MEMORY LOCATION TO BE USED.  INPUT OR OUTPUT TRANSFER
                    (1139) * IS DETERMINED BY TESTING THE INPUT/OUTPUT LINE AS IS
                    (1140) * TRUE OF ALL DMX.  THE SIGNALS GENERATED ARE THE SAME
                    (1141) * AS FOR DMA EXCEPT THAT THE END OF RANGE SIGNAL IS UN=
                    (1142) * DEFINED FOR THIS TRANSFER TYPE.  THIS MEANS THE CONTROL-
                    (1143) * ER DETERMINES IF THE TRANSFER ENDS THE BLOCK.
                    (1144) *
                    (1145) *
                    (1146) DMT     CPU      BB   BPA  NX     0      0     RSC   (CPN,STROBE) ;
                    (1147)                  RY200      0     LOAD256K ;
                    (1148)                  JUMP ON OUTPUT TO TOUT1
03E: F603 C808 0006 5046
                    (1149)         CPU      ,,,,,RSC STROBE ,,,GO TO TIN1
03F: 0203 8004 0004 0042
                    (1150) *
                    (1151) *      CYCLE TO CYCLE FULLY OVERLAPPED DMT INPUT LOOP.
                    (1152) * NOTE THAT THE STROBE HAS ALREADY BEGUN BEFORE
                    (1153) * THE CYCLE CAN EXIT.  THIS FORCES ALL OF THE OTHER
                    (1154) * ALGORITHMS TO BE WRITTEN WITH STROBE ALWAYS EN-
                    (1155) * ABLED.
                    (1156) *
                    (1157) TIN     CPU      ,,,,,RSC CPN
040: 0203 4004 0000 0000
                    (1158)         CPU      BB   BPA  NX     0      0     RSC   STROBE ;
                    (1159)                  RY200      NOP   LOAD256K ;
                    (1160)                  JUMP ON DMTINNVAL TO TEXIT
041: F603 8808 0007 604A
                    (1161) TIN1    CPU      BB   BPD  NX     0      0     RSC   (ENB,STROBE) ;
                    (1162)                  RM280      AWRITE       NOP ;
                    (1163)                  GO TO TIN
042: FA03 94F4 0004 0040
                    (1164) *
                    (1165) *      DMT OUTPUT LOOP.
                    (1166) *
                    (1167) TOUT    CPU      0    0    NX     0      0     RSC   (STROBE,DATA) ;
                    (1168)                  RMMRDY
043: 0203 A704 0000 0000
                    (1169)         CPU      BB   BPA  NX     0      0     RSC   (CPN,DATA) ;
                    (1170)                  RY240      NOP   LOAD256K ;
                    (1171)                  JUMP ON DMTOUTNVAL TO TEXIT
044: F603 6308 0007 504A
                    (1172)         CPU      RF   0    NONE   0      0     RSC   STROBE ;
                    (1173)                  RF160
045: 0003 8904 0000 0000
                    (1174) TOUT1   CPU      0    0    NX     0      0     RSC   (ENB,STROBE) ;
                    (1175)                  280  AREAD      NOP ;
046: 0203 96C4 0004 0043
                    (1177)         EJCT
                    (1178) *      THIS CODE TRANSFERS CONTROL FROM ONE MODE OF DMX
                    (1179) * TRANSFER TO ANOTHER OR RESUMES NORMAL USER LEVEL CODE
                    (1180) * EXECUTION.
                    (1181) *
                    (1182) DMA1    CPU      RF   0    NX     0      0     RSC   (DATA,CPN) ;
                    (1183)                  RF240      NOP   NOP ;
                    (1184)                  JUMP ON DMXSTATREQ TO REST
047: 0203 6804 0005 D059
                    (1185) *
                    (1186) *      START OF DMC LOOP
                    (1187) *
                    (1188) DMC     CPU      BB   BPA  NX     0      0     RSC   STROBE ;
                    (1189)                  RY200      NOP   NOP ;
                    (1190)                  JUMP ON DMCVALID TO DMC1
048: F603 8804 0006 805C
                    (1191)         CPU      ,,,,,RSC STROBE ,,,JUMP ON DMANOTVALID TO DMT
049: 0203 8004 0005 E03E
                    (1192) TEXIT   CPU      ,,,,,RSC STROBE ,,,JUMP ON DMXSTATREQ TO REST
04A: 0203 8004 0005 D059
                    (1193) *
                    (1194) *
                    (1195) *      DMA.  THE ADDRESS LINES ARE READ INTO THE SHIFT COUNTER.
                    (1196) * THIS DIRECTLY SELECTS THE FIRST DMA REGISTER.  IT IS INCRE-
                    (1197) * MENTED BY $10 AND TESTED TO SEE IF ZERO IS CROSSED BY THE
                    (1198) * END OF BLOCK SIGNAL.  THE SECOND REGISTER IS ACCESSED AND AN
                    (1199)'* INPUT OR OUTPUT TRANSFER IS BEGUN TO THE LOCATION READ.  A
                    (1200) * BPCENB IS GENERATED BEFORE THE COMPLETION OF THE TRANSFER TO
                    (1201) * DETECT CONSECUTIVE TRANSFERS.  IF CONSECUTIVE, LOOP BACK,
                    (1202) * IF NOT, EXIT.
                    (1203) *
                    (1204) *
                    (1205) DMA     CPU      0    BPA  NX     0      0     RSC   STROBE ;
                    (1206)                  200  NOP  LOADRSC ;
                    (1207)                  JUMP ON DMANOTVALID TO DMC
04B: 1603 800E 0005 E048
                    (1208)         CPU      AL   RCM  NX     ADD    0     RSC   (STROBE,CPN) ;
                    (1209)                  RYRF240    NOP   INCRSC ;
                    (1210)                  DATA $10
04C: 9263 C005 0002 0010
                    (1211)         CPU      RF   0    NX     0      0     RSC   STROBE ;
                    (1212)                  RY200      NOP   LOAD256K ;
                    (1213)                  JUMP ON OUTPUT TO DOUT1
04D: 0203 8808 0006 5058
                    (1214)         CPU      BB   BPD  NONE   0      0     RSC   (ENB,STROBE) ;
                    (1215)                  RM280      AWRITE       NOP ;
                    (1216)                  GO TO DIN
04E: F803 94F4 0004 004F
                    (1217) *
                    (1218) *      CYCLE TO CYCLE, FULLY OVERLAPPED DMA INPUT ROUTINE.
                    (1219) *
                    (1220) DIN    CPU      AL   0    NX    INC    1     RSC   0 ;
                    (1221)                 RF200
04F: 8207 0A04 0000 0000
                    (1222)        RR       BPA =>  ,LOADRSC,  JUMP ON DMANOTVALID TO DMA1
050: F600 000E 0005 E047
                    (1223)        CPU      AL   RCM  NX    ADD    0     RSC   (CPN,STROBE) ;
                    (1224)                 RYRF240   NOP   INCRSC ;
                    (1225)                 DATA $10
051: 9263 CD05 0002 0010
                    (1226)        CPU      RF   0    NX    0      0     RSC   STROBE ;
                    (1227)                 RY200     NOP   LOAD256K ;
                    (1228)                 JUMP ON OUTPUT TO DOUT1
052: 0203 8808 0006 5058
                    (1229) DIN1   CPU      BB   BPD  NONE  0      0     RSC   (ENB,STROBE) ;
                    (1230)                 RM280     AWRITE       NOP ;
                    (1231)                 GO TO DIN
053: F803 94F4 0004 004F
                    (1232) *
                    (1233) *      DMA CYCLE-CYCLE OUTPUT ROUTINE.
                    (1234) *
                    (1235) DOUT   CPU      RF   ,NONE,,,RSC       STROBE      RF160
054: 0003 8904 0000 0000
                    (1236)        CPU      0    BPA  NX    0      0     RSC   (STROBE,DATA) ;
                    (1237)                 RMMRDY  NOP  LOADRSC ;
                    (1238)                 JUMP ON DMANOTVALID TO DMA1
055: 1603 A70E 0005 E047
                    (1239)        CPU      AL   RCM  NX    ADD    0     RSC   (CPN,DATA) ;
                    (1240)                 RYRF240   NOP   INCRSC ;
                    (1241)                 DATA $10
056: 9263 6D05 0002 0010
                    (1242)        CPU      RF   0    NX    0      0     RSC   STROBE ;
                    (1243)                 RY200     NOP   LOAD256K ;
                    (1244)                 JUMP ON INPUT TO DIN1
057: 0203 8808 0004 5053
                    (1245) DOUT1  CPU      AL   0    NONE  INC    1     RSC   (STROBE,ENB) ;
                    (1246)                 RF280     AREAD NOP ;
                    (1247)                 GO TO DOUT
058: 8007 9CC4 0004 0054
                    (1248)        EJCT
                    (1249) *
                    (1250) *      RESTORE ROUTINE.  RETURN TO NORMAL PROCESSING.
                    (1251) *
                    (1252) REST   RR       YSAVE  =>  RY
059: 0200 C804 0000 0000
                    (1253)        RR       RSCSAVE => RM
05A: 0000 E104 0000 0000
                    (1254)        CPU      RF   RM   ALL   0      0     M     MSAVE ;
                    (1255)                 RM280     NOP   LOADRSC ;
                    (1256)                 S ON TRUE, POP
05B: 0F00 D40E 000C 0004
                    (1257) *
                    (1258) *      DMC.  SLOWEST OF ALL THE TRANSFERS, DMC HAS AS ITS VIRTUE
                    (1259) * THE CAPABILITY OF USING ANY OF THE FIRST 64K MAIN
                    (1260) * MEMORY LOCATIONS AS A CHANNEL.  IDENTICAL IN FUNCTION
                    (1261) * TO DMA, THE TWO LOCATIONS MAKING UP A CHANNEL CONTAIN
                    (1262) * THE CURRENT AND ENDING MEMORY ADDRESSES, AND MUST BE ACCESSED
                    (1263) * TESTED AND UPDATED EACH CYCLE.
                    (1264) *
                    (1265) *
                    (1266) DMC1   CPU      0    RCM  NX    0      0     RSC   STROBE ;
                    (1267)                 RMMRDY    AREAD LOADRSC ;
                    (1268)                 DATA '17
05C: 1203 87CE 0002 010F
                    (1269)        CPU      BB   RCM  NX    0      0     RSC   STROBE ;
                    (1270)                 RF200     NOP   NOP ;
                    (1271)                 DATA 1
05D: F203 8A04 0002 0001
                    (1272)        CPU      AL   RM   NX    ADD    0     RSC   STROBE ;
                    (1273)                 RM280     AWRITE
05E: 8E63 84F4 0000 0000
                    (1274)        CPU      AL   RY   NX    ADD    0     RSC   STROBE ;
                    (1275)                 RY240     ,,GO TO DMC3
05F: 8A63 8304 0004 0086
                    (1276)        EJCT
```

```
                  (1277)        ORG      $60
                  (1278) *
                  (1279) *
                  (1280) *       TRAP ENTRY POINTS.  EVERY OTHER LOCATION FROM HERE TO
                  (1281) * $7E IS A POTENTIAL TRAP ENTRY POINT.  THOSE LOCATIONS
                  (1282) * WHERE TRAPS ARE NOT IMPLEMENTED ARE AVAILABLE FOR OTHER
                  (1283) * CODING.  PRIOITY IS FROM 6F TO 60, THEN FROM
                  (1284) * 7F TO 70.  THEREFORE MISSING MEMORY MODULE IS HIGHEST
                  (1285) * PRIORITY, AND DMA IS LOWEST.
                  (1286) *
                  (1287) *
                  (1288) *
                  (1289) *       FETCH PAGE TRAP.
                  (1290) *
                  (1291) FPAGE  CPU     BB   RCM  NONE  ,,,,280       ,PUSHBD ;
                  (1292)                DATA FPAGE3
060: F000 060C 0002 008A
                  (1293)        RR      RY => YSAVE TR= NONE  ,  GO TO PAGE+1
061: E800 CA04 0004 007D
                  (1294) *
                  (1295) *       WRITE ADDRESS TRAP
                  (1296) *
                  (1297) WRITE  CPU     BB   RM   1     ,,RY   0     RF240 ;
                  (1298)                ,,S ON TRUE, POP
062: ED02 0804 000C 0004
                  (1299) PAGE3  CPU     AL   RM   NONE  BB     L     M      17 ;
                  (1300)                RMRFMRDY  AREAD FORCERD ;
                  (1301)                GO TO PAGE4
063: 8C5C F5C9 0004 0080
                  (1302) *
                  (1303) *       READ ADDRESS TRAP
                  (1304) *
                  (1305) READ   CPU     RF   0    1     ,,RY   0     RM280 ;
                  (1306)                ,,S ON TRUE, POP
064: 0102 0404 000C 0004
                  (1307)        ORG     $66
                  (1308) *
                  (1309) *       FETCH READ ADDRESS TRAP
                  (1310) *
                  (1311) FREAD  CPU     RF   0    1     ,,RY   0     RM200 ;
                  (1312)                ,,EAC LOADF01F02
066: 0102 0204 000A C000
                  (1313)        CPU     RF,,1,,M,0,RF240,,,S ON TRUE, POP
067: 0100 0804 000C 0004
                  (1314) *
                  (1315) *       RESTRICT EXECUTION MODE TRAP.  THOSE INSTRUCTIONS WHICH
                  (1316) *
                  (1317) * ARE RESTRICTED FORCE THIS TRAP IF THE MODE IS ENABLED.
                  (1318) *
                  (1319) RXM    RR      RCM = '62  => RY TR= NONE
068: F000 0804 0002 0032
                  (1320)        CPU     ,,NONE,,,,,RMMRDY,AREAD,,GO TO AVECT1
069: 0000 07C4 0004 0077
                  (1321) *
                  (1322) *       CENTRAL PROCESSOR PARITY .
                  (1323) *
                  (1324) CPPAR  ALU     CON 0 => RM  CLEARFUII TR= NONE  JUMP ON VERIFY TO VIRY1
06A: 803C 020F 0007 30C4
                  (1325)        RR      RP => RY  TR= NONE , GO TO MC4
06B: 0000 7804 0004 0034
                  (1326)        ORG     $6C
                  (1327) *
                  (1328) *       MEMORY PARITY ERROR
                  (1329) *
                  (1330) MEMPAR RR      RCM = '67  => RY  TR= NONE
06C: F000 0804 0002 0037
                  (1331)        ALU     CON 0  => RM  TR= 1    , GO TO MEMP3
06D: 813C 0204 0004 008C
                  (1332) *
                  (1333) *       MISSING MEMORY MODULE
                  (1334) *
                  (1335) MMOD   RR      RCM = '71  => RY  TR= NONE
06E: F000 0804 0002 0139
                  (1336)        RR      ,,TR= NONE      ,GO TO MEMPAR+1
06F: 0000 0004 0004 006D
                  (1337) *
                  (1338) *       DMX INSTRUCTION INTERRUPT ENTRY POINT
                  (1339) *
                  (1340) DMX1   RR      RM => MSAVE  TR= NX
070: EE00 DA04 0000 0000
                  (1341)        CPU     ,,NONE,,,RSC ENB 280 ,,GO TO DMX2
071: 0003 1604 0004 003C
                  (1342) *
                  (1343) *       PAGE WRITE PROTECT VIOLATION
                  (1344) *
                  (1345) WRITEP RR      RCM = '73 => RY  TR= 1
072: F100 0804 0002 003B
                  (1346)        CPU     ,,NONE,,,,,RMMRDY,AREAD,,GO TO AVECT1
073: 0000 07C4 0004 0077
                  (1347) *
                  (1348) *       POWER FAILURE INTERRUPT
                  (1349) *
                  (1350) FHALT2 ALU     CON ZERO => RM ,JUMP ON PFLNOT TO FHALT3
074: 823C 0204 0005 902D
                  (1351) PFL    RR      RCM = '60  => RY  TR= NONE
075: F000 0804 0002 0130
                  (1352)        CPU     AL   0    NONE  INC    1     M      RP ;
                  (1353)                RMRFMRDY  AREAD  NOP ;
                  (1354)                EAC  CLRPWFL  (TRIGNVKEYS,EINTM)
076: 8004 75C4 0008 0410
```

```
                   (1355) *
                   (1356) *       ABSOLUTE VECTOR   ---   THE FIRST ENTRY ALSO BACKS THE
                   (1357) * PROGRAM COUNTER UP BEFORE VECTORING.   THE LOCATION IS
                   (1358) * READ.  IF ZERO, THE PROCESSER HALTS, IF NOT, A JST TO THE
                   (1359) * LOCATION READ IS DONE.  NOTE THAT THE ADDRESS IS
                   (1360) * INTERPRETED AS AN ABSOLUTE 64K POINTER, NO INDEXING OR
                   (1361) * INDIRECTION.
                   (1362) *
          000167   (1363) AVECT  EQU     *
                   (1364) AVECT1 ALU     DEC RP  => RP TR= 1    CLEARFUII ;
                   (1365)                JUMP ON FUII TO *
077: 81F0 7A0F 0005 5077
                   (1366)        CPU     AL   RM   NONE  BB     L      ,,RY240 ,SETCC ;
                   (1367)                EAC CLRNVKEYS  TRIGNVKEYS
078: 8C5C 0307 000A 0400
                   (1368)        RR      RP => RM  TR= ALL  ,   ;
                   (1369)                JUMP ON EQ TO CP1
079: 0300 7204 0006 4035
                   (1370)        RR      RY => RP  C= AWRITE  TR= ALL    ,GO TO CAS5
07A: E800 7AF4 0004 01E5
                   (1371) *
                   (1372) *       PAGE TRAP .  PROCESS TO UPDATE CAM IF POSSIBLE
                   (1373) * AND CONTINUE THE INSTRUCTION.  IF NO PAGE IN MEMORY
                   (1374) * BACK UP THE P-COUNTER , AND PAGE FAULT INTERRUPT.
                   (1375) *
                   (1376)        ORG     $7C
                   (1377) PAGE   RR      RY => YSAVE
07C: E600 C904 0000 0000
                   (1378)        ALU     PMAR OR RY  => RY TR= 0   FLIP BYTES  DISABLERHBB ;
                   (1379)                GO TO PAGE3
07D: 684C 830B 0004 0063
                   (1380) *
                   (1381) *       POST PROCESSING OF CENTRAL PROCESSOR PARITY ERROR
                   (1382) * AFTER MICRO-VERIFY HAS SUCESSFULLY BEEN EXECUTED.
                   (1383) *
                   (1384) CPPAR3 RR      RCM = '70   => RY      TR= NONE
07E: F000 0804 0002 0038
                   (1385)        CPU     ,,NONE,,,,,RMMRDY,AREAD,,GO TO AVECT+1
07F: 0000 07C4 0004 0078
                   (1386) PAGE4  RR      YSAVE => RY TR= NONE ,     ;
                   (1387)                JUMP ON RM01NOT TO PAGE7
080: 0000 C804 0006 1083
                   (1388)        CPU     RF   0    0    0    0     M     17 ;
                   (1389)                RM280     ,,EAC LOADCAM
081: 0000 F404 0009 8000
                   (1390)        CPU     ,,ALL     ,,,,RMMRDY   MWRITE ;
                   (1391)                HSMRESUME  S ON TRUE,POP
082: 0300 07BA 000C 0004
                   (1392) PAGE7  RR      RY => EAS  TR= 1
083: E900 AA04 0000 0000
                   (1393)        RR      RCM = '64  => RY       TR= 1
084: F100 0804 0002 0034
                   (1394)        CPU     ,,NONE,,,,,RMMRDY,AREAD,,GO TO AVECT1
085: 0000 07C4 0004 0077
                   (1395) *
                   (1396) *       DMC EXECUTION
                   (1397) *
                   (1398) DMC3   CPU     BB   RM   NX    0     0     RSC   STROBE ;
                   (1399)                RMRFMRDY  AREAD
086: EE03 85C4 0000 0000
                   (1400)        CPU     RF   RM   NX    SUB   0     RSC   (CPN,STROBE) ;
                   (1401)                RY240     ,,JUMP ON INPUT TO DMC2
087: 0E93 C304 0004 5089
                   (1402)        CPU     AL,,NX,DEC,0,RSC  (STROBE) RY240 ;
                   (1403)                ,,GO TO DOUT1
088: 82F3 8304 0004 0058
                   (1404) DMC2   CPU     AL   0    NX    DEC   0     RSC   STROBE ;
                   (1405)                RY240  ,,GO TO DIN1
089: 82F3 8304 0004 0053
                   (1406) *
                   (1407) *       FETCH PAGE FINISH OFF.  IT IS NECESSARY TO LOAD F01 AND2
                   (1408) *
                   (1409) FPAGE3 CPU     BB   RM   NONE  ,,,,RM200    ,,EAC LOADF01F02
08A: EC00 0204 000A C000
                   (1410)        CPU     RF,,1,,,M,0,RF240,,,S ON TRUE, POP
08B: 0100 0804 000C 0004
                   (1411) *
                   (1412) *       FINISH THE MEMORY PARITY ERROR TRAP
                   (1413) *
                   (1414) MEMP3  CPU     AL   0    NONE  RF     0     M     RX ;
                   (1415)                RF200     ,,EAC NOP    (TRIGNVKEYS,MCHK)
08C: 8000 0A04 0008 0404
                   (1416)        CPU     AL   0    1     RF     0     M     RA ;
                   (1417)                RMRFMRDY  AREAD  ,GO TO AVECT+1
08D: 8100 15C4 0004 0078
                   (1418) FOUT1  ALU     RX ADD RCM = $20  => RX  TR= NONE  SETCC
08E: 9060 0A07 0002 0020
                   (1419)        RR      ,,,TR= NONE   JUMP ON NE TO *-1
08F: 0000 0004 0004 408E
                   (1420)        RR      ,,,TR= NONE   GO TO VIRY1
090: 0000 0004 0004 00C4
                   (1421)        EJCT
                   (1422) *      CONTROL PANEL BOOT.  THIS PROGRAM READS IN 512 WORDS FROM
                   (1423) * THE CONTROL PANEL AND PUTS THEM INTO LOCATIONS 6
                   (1424) * THROUGH '56 IN VIRTUAL MEMORY.  THE LOCATION POINTED TO
                   (1425) * BY THE P COUNTER IS THEN EXECUTED.
                   (1426) *
                   (1427) *      METHOD OF OPERATION.  AN OTA '1720 SENDS THE CONTROL
                   (1428) * PANEL THE LOCATION TO BE READ FROM ITS MEMORY.  AN INA
                   (1429) * '1420 IS THEN ISSUED TO GET THE CONTENTS OF THE LOCATION
                   (1430) * JUST ACCESSED.  THIS IS WRITTEN INTO MEMORY, THE POINTERS
                   (1431) * ARE INCREMENTED AND THE PROCESS REPETED UNTIL ALL 512 WORDS
                   (1432) * ARE TRANSFERRED.
                   (1433) *
                   (1434) *
                   (1435) *
                   (1436) BOOT   RR      RCM = 0  => RM
091: F000 0104 0002 0100
                   (1437)        RR      RCM = 0  => RB
092: F000 2904 0002 0100
                   (1438)        RR      RCM = 5  => RS
093: F000 3904 0002 0105
                   (1439) BOOT1  CPU     BB   RCM  NX    0      0     RY    PIO ;
                   (1440)                RY200     ,,DATA '171720
094: F202 1804 0003 E6D0
                   (1441)        CPU     BB   RCM  NX    0      0     RSC   (DATA,STROBE) ;
                   (1442)                RY280     ,,  DATA '131420
095: F203 AE04 0001 6610
                   (1443)        CPU     AL   0    NX    ZERO   L     RSC      0 ;
                   (1444)                280  NOP  NOP ;
                   (1445)                EAC  PLOADVKEYS TRIGVKEYS
096: 823F 0604 000A 4100
                   (1446)        CPU     AL   BPD  NX    BB     L     RSC   0 ;
                   (1447)                RM280
097: 9A5F 0404 0000 0000
                   (1448)        ALU     INC RS => (RY,RS)
098: 8204 3D04 0000 0000
                   (1449)        ALU     RS MINUS RCM = '56  => NULL  C= MWRITE SETCC
099: 9294 3087 0002 012E
                   (1450)        ALU     INC RB  => RM
09A: 8204 2204 0000 0000
                   (1451)        ALU     INC  RB => RB ;
                   (1452)                JAMF   JUMP ON LT TO BOOT1
09B: 8204 2A00 0006 6094
                   (1453)        EJCT
                   (1454) DIV27  ALU     DEC RB  => RM TR= ALL   ,JUMP ON GT TO IAB
09C: 83F0 2204 0006 71A2
                   (1455)        RR      RY => RB  ,GO TO LDA+1
09D: EA00 2A04 0004 01DF
                   (1456) DIV17  CPU     ALLS RM   NX    ADD    0     M      RA ;
                   (1457)                200  DIVER        , ;
                   (1458)                EAC DIVLOGIC
09E: AE60 1014 0009 0000
                   (1459)        CPU     RFLS 3    ALL   3      0     M      RB ;
                   (1460)                RF200     LINK  , ;
                   (1461)                JUMP ON FCBIT TO DIVER
09F: 2F30 2A44 0004 F146
                   (1462)        CPU     ALLS RM   NX    ADD    0     M      RA ;
                   (1463)                RF280     LINK  ,EAC DIVLOGIC
0A0: AE60 1C44 0009 0000
                   (1464)        CPU     RFLS 3    ALL   3      0     M      RB ;
                   (1465)                RF200     LINK  INCRSC ;
                   (1466)                JUMP ON RSCNEM1 TO *-1
0A1: 2F30 2A45 0004 90A0
                   (1467) * SET RSC = 1 SO IAB CODE AT END WORKS
                   (1468)        CPU     ALLS RM   NX    ADD    0     M      RA ;
                   (1469)                200  LINK INCRSC EAC DIVLOGIC
0A2: AE60 1045 0009 0000
                   (1470)        CPU     AL   RM   NX    ADD    0     M      RA ;
                   (1471)                RYRF240   ,,EAC DIVLOGIC
0A3: 8E60 1D04 0009 0000
                   (1472)        CPU     RFLS 3    ALL   6      0     M      RB ;
                   (1473)                RF200
0A4: 2F60 2A04 0000 0000
                   (1474)        CPU     AL   0    NX    ANOT   L     M      RB ;
                   (1475)                RF200
0A5: 82FC 2A04 0000 0000
                   (1476)        ALU     INC RB => RB  TR= ALL   ,JUMP ON GE TO XCB
0A6: 8304 2A04 0004 61A8
                   (1477)        ALU     RA ADD RM  => RY       SETCC   GO TO DIV27
0A7: 8E60 1307 0004 009C
                   (1478) DIV4   CPU     ,,ALL RF   0      M        RA ;
                   (1479)                200  ,SETCC      JUMP ON LT TO DIV17
0A8: 0300 1007 0006 609E
                   (1480)        CPU     ALLS RM   NX    SUB    1     M      RA ;
                   (1481)                200  DIVER       ,EAC DIVLOGIC
0A9: AE94 1014 0009 0000
                   (1482)        CPU     RFLS 3    ALL   3      0     M      RB ;
                   (1483)                RF200     LINK  ,JUMP ON FCBIT TO DIVER
0AA: 2F30 2A44 0004 F146
                   (1484)        CPU     ALLS RM   NX    SUB    1     M      RA ;
                   (1485)                RF280     LINK  ,EAC DIVLOGIC
0AB: AE94 1C44 0009 0000
                   (1486)        CPU     RFLS 3    ALL   3      0     M      RB ;
                   (1487)                RF200     LINK  INCRSC ;
```

```
                  (1488)             JUMP ON RSCNEM1 TO *-1
0AC: 2F30 2A45 0004 90A8
                  (1489)      CPU    ALLS RM   NX   SUB    1     M     RA ;
                  (1490)             200  LINK INCRSC EAC DIVLOGIC
0AD: AE94 1045 0009 0000
                  (1491)      CPU    AL   RM   NX   SUB   1     M     RA ;
                  (1492)             RYRF240   ,,EAC DIVLOGIC
0AE: 8E94 1D04 0009 0000
                  (1493)      CPU    RFLS 3    ALL   6      0     M     RB ;
                  (1494)             RF200     ,,JUMP ON GE TO XCB
0AF: 2F60 2A04 0004 61A8
                  (1495)      ALU    RA SUB RM  => RY SETCC
0B0: 8E94 1307 0000 0000
                  (1496)      CPU    AL   KEYS ALL   RF     1     M     RB ;
                  (1497)             RM200     ADVFL , ;
                  (1498)             JUMP ON GT IAB
0B1: 8704 2274 0006 71A2
                  (1499)      CPU    BB   RY   NX    0      0     M     RB ;
                  (1500)             RF200     ,,GO TO LDA+1
0B2: EA00 2A04 0004 01DF
                  (1501) *
                  (1502) *    UNIMPLEMENTED INSTRUCTION VECTOR
                  (1503) *
                  (1504) UII3 RR     RCM = '66   => RY
0B3: F200 0804 0002 0136
                  (1505)      CPU    BB   RM   NX    0      0     M     11 ;
                  (1506)             RMRFMRDY  AREAD   NOP ;
                  (1507)             GO TO AVECT1
0B4: EE00 95C4 0004 0077
                  (1508) *
                  (1509) *    FINISH ILLEGAL INSTRUCTION VECTOR
                  (1510) *
                  (1511) ILL3 RR     RCM = '72  => RY
0B5: F200 0804 0002 013A
                  (1512)      RR     ,,TR= ALL ,GO TO UII3+1
0B6: 0300 0004 0004 0084
                  (1513) STA3 ALU    INC RY => RY EAF
0B7: 8A65 130D 0000 0000
                  (1514)      CPU    RF   0    ALL   0      0     M     RB ;
                  (1515)             RM280     MWRITE      JAMF
0B8: 0300 2480 0000 0000
                  (1516) ADD3 ALU    INC RY => RY  EAF
0B9: 8A65 130D 0000 0000
                  (1517)      CPU    BB   RM   ALL   0      0     M     13 ;
                  (1518)             RMRFMRDY  MREAD
0BA: EF00 B584 0000 0000
                  (1519)      ALU    RB PLUS RM => RB C= ADVFL
0BB: 8E60 2A74 0000 0000
                  (1520)      ALU    RB AND RCM = $7FFF  => RB  TR= ALL
0BC: 931C 2A04 0000 FFFF
                  (1521)      RR     13 => RM
0BD: 0000 B104 0000 0000
                  (1522)      ALU    RA PLUS RM + CBIT  => RA JAMF  C= ADVFL
0BE: 8E68 1A70 0000 0000
                  (1523) MI1  CPU    ,,NX,,,RY,0,RMMRDY,AREAD,,GO TO MI2
0BF: 0202 07C4 0004 001C
                  (1524)      EJCT
```

```
                          *
                          *
                          *
                          *
                          *         MICRO-VERIFICATION ROUTINES.  THESE ROUTINES ARE DIVIDED
                          * UP INTO '13 TESTS.  THE TEST NUMBER IS LOCATED IN RY AT
                          * LEAT AT THE END OF THE TEST WHEN CHECKS ARE DONE
                          * TO SEE IF THE TEST WAS SUCESSFUL OR NOT.  IF A TEST FAILS, THE
                          * TEST NUMBER IS DISPLAYED ON THE CONTROL PANEL LIGHTS VIA THE
                          * THE BMA PATH.  THESE ROUTINES ARE EXECUTED FOR THREE DIFFERENT
                          * FUNCTIONS:
                          *                  MASTER CLEAR:
                          *
                          *         THE TESTS ARE RUN UNTIL THEY ASE ALL PASSED.  UPON COMPLETION,
                          *         THE CONTROL PANEL ROUTINE IS ENTERED.  FAILURE OF ANY TEST
                          *         CAUSES A DELAY, FOLLOWED BY A RETRY OF ALL THE TESTS.
                          *
                          *                  MACHINE CHECK:
                          *
                          *         ENTERED ON THE DETECTION OF A CPU PARITY ERROR, THE TESTS ARE
                          *         RUN EXACTLY AS IN MASTER CLEAR EXCEPT THAT WHEN THE TESTS ARE
                          *         SUCESSFULLY COMPLETED, THE MACHINE CHECK VECTOR IS
                          *         EXECUTED.
                          *
                          *                  VERIFY  (VIRY):
                          *
                          *         THIS IS AN EXPLICIT USER REQUESTED EXECUTION OF THE TEST
                          *         ROUTINES.  SUCCESS IS REWARDED BY SKIPPING
                          *         THE NEXT SEQUENTIAL INSTRUCTION, FAILURE IS SHOWN
                          *         BY PUTTING THE FAILED TEST NUMBER INTO RA AND EXECUTING
                          *         THE NEXT SEQUENTIAL INSTRUCTION.
                          *
                          *
                          *         TEST 0:  TESTS ALU =0 ,SUB ,=0 CONDITIONAL BRANCH, RCM =0,
                          *                  CONTROL UNIT AND SETCC.
                          *
                          *
                                    ORG      $C4
                          VIRY1     ALU      CON 0 => RM        TR= NONE
0C4: 803C 0204 0000 0000
                                    CPU      AL   RM   NONE  ZERO   L  M      RA  RYRF240   ,LOADRSC ;
                                             MODAL NOP (54,56,57,58,59,60,61,63,64)
0C5: 8C3C 1D0E 0008 05FB
                                    CPU      0    RCM  NONE  SUB    1      M      RA ;
                                             200   ,SETCC      DATA 0
0C6: 1094 1007 0002 0100
                                    CPU      0    RY   NONE  SUB    1      M      RA ;
                                             200   ,SETCC ;
                                             JUMP ON NE TO F1
0C7: 0894 1007 0004 4000
                          *
                          *         TEST 1:   TESTS RY => BB
                          *
                                    ALU      CON 0  => RX  ,TR= NONE  EAC CLRNVKEYS (54,56,62)
0C8: 803C 0A04 000A 0504
                                    ALU      INC RY => RY  ,TR= NONE  ;
                                             JUMP ON NE TO FOUT
0C9: 8865 1304 0004 40FC
                          *
                          *         TEST 2:   TESTS MACHINE CHECK RESET
                          *
                                    ALU      INC RY => RY  TR= NONE , ;
                                             JUMP ON MC TO FOUT
0CA: 8865 1304 0007 10FC
                          *
                          *         TEST 3:   TESTS SHIFT COUNTER INCREMENTING AND
                          *                  THE RSCNOTEQM1 TEST. ALSO RX+1 => RX, AND COUNTS FOR
                          *                  PROPER NUMBER OF INCREMENTS.  (THIS TEST WILL SHOW AS #2
                          *                  IF RSCNEM1 IS ALWAYS TRUE.)
                          *
                                    ALU      INC RX => RX  TR= NONE  INCRSC ;
                                             JUMP ON RSCNEM1 TO *
0CB: 8004 0A05 0004 90CB
                                    ALU      RX MINUS RCM = $40   => NULL  TR= NONE  SETCC
0CC: 9094 0007 0002 0040
                                    ALU      INC RY => RY  TR= NONE , ;
                                             JUMP ON NE TO FOUT
0CD: 8865 1304 0004 40FC
                          *
                          *         TESTS 4,5,6:   THESE TESTS VERIFY THAT EACH OF THE REGISTERS
                          *                  CAN DETECT BAD PARITY.  4 TESTS RY, 5 RM, AND 6 RF.  ALSO
                          *                  CHECKED:  16 WAY BRANCH, MC- TEST.
                          *
                                    CPU      BB   RCM  NONE  ,,,,RY200     ,, ;
                                             EMIT  $100  $003
0CE: F000 0804 0002 0003
                          VIRY12    ALU      CON 0 => RM  TR= NX
0CF: 823C 0204 0000 0000
                                    CPU      AL   0    NONE  ZERO   L      ,,RF160
0D0: 803C 0904 0000 0000
                                    ALU      INC RY => RY  TR= NONE , ;
                                             JUMP ON MCNOT TO FOUT
0D1: 8865 1304 0007 20FC
                                    RR       ,,,TR= NONE  MODAL NOP (TRIGNVKEYS,MCHK)
0D2: 0000 0004 0008 0404
                                    CPU      BB   RY   NONE  ,,,,280       ,, ;
                                             S ON TRUE, 16WAYS TO VIRY12+1
0D3: E800 0604 000C 00D2
                                    RR       RCM => RM TR= NONE ,GO TO VIRY12
0D4: F000 0204 0004 00CF
```

```
                    (1608)        RR      RCM => RX TR= NONE ,GO TO VIRY12
0D5: F000 0A04 0004 00CF
                    (1609) *
                    (1610) *      TEST 7:   PARITY ON LEFT SHIFT,ROTATE
                    (1611) *
                    (1612)        ALU     CON -1  => RA , EAC LOADSERIALINT
0D6: 82CC 1A04 0008 4000
                    (1613) * LOGICAL LEFT SHIFT
                    (1614)        CPU     RFLS 7    NX    7      0      M    RA ;
                    (1615)                RF200     RF01
0D7: 3E70 1A54 0000 0000
                    (1616) * LOGICAL LEFT ROATATE
                    (1617)        CPU     RFLS 1    NX    8      0      M    RA ;
                    (1618)                RF200    ,, ;
                    (1619)                JUMP ON MC TO VIRY12-5
0D8: 2680 1A04 0007 10CA
                    (1620) * ROTATE 64 TIMES
                    (1621)        RR      RA => RA  INCRSC ;
                    (1622)                JUMP ON RSCNEM1 TO *-1
0D9: 0200 1A05 0004 90D8
                    (1623) *
                    (1624) *      TEST 7 (CONT):   TEST C BIT FOR SET, LOAD AND RESET
                    (1625) * CBIT SHOULD BE SET SO RA IS TESTED AGAINST -2
                    (1626) *
                    (1627)        ALU     RA MINUS RCM = -2  + CBIT  => RA ;
                    (1628)                C= BD01  SETCC
0DA: 9298 1A37 0003 FEFE
                    (1629)        CPU     RF   0    NX    MINUS1 L      M    RP ;
                    (1630)                RM200     ,SETCC ;
                    (1631)                JUMP ON NE TO VIRY12-2
0DB: 02CC 7207 0004 40CD
                    (1632) *
                    (1633) *      TEST '10:   wRITE BITS THROUGH REGISTER FILE.  CHECK
                    (1634) *              PATTERN USING EXCLUSIVE ORS TO VERIFY PROPPER
                    (1635) *              OPERATION.  ZERO OUT FILES.
                    (1636) *
                    (1637)        RR      RCM = $FFC0  => RX  TR= NX
0DC: F200 0A04 0003 FFC0
                    (1638)        CPU     AL   0    NONE RF      1     RSC   0 ;
                    (1639)                RY240     ,INCRSC ;
                    (1640)                JUMP ON EQ TO *+2
0DD: 8007 0305 0006 40DF
                    (1641)        CPU     BB   RY   NONE BB      L     RSC   0 ;
                    (1642)                RF200     ,SETCC ;
                    (1643)                GO TO *-1
0DE: E85F 0A07 0004 00DD
                    (1644)        RR      RCM = $FFE1  => RY
0DF: F200 0804 0003 FFE1
                    (1645)        ALU     INC RY  => RY INCRSC ;
                    (1646)                JUMP ON NE TO *+4
0E0: 8A65 1305 0004 40E4
                    (1647)        CPU     RF   0    NX    INC CBIT     RSC   0 ;
                    (1648)                200  ,SETCC
0E1: 0206 0007 0000 0000
                    (1649)        CPU     AL   RY   NX    XOR    L     RSC   0 ;
                    (1650)                RF200     ,SETCC ;
                    (1651)                JUMP ON NE TO *-2
0E2: 8A6F 0A07 0004 40E0
                    (1652)        CPU     AL   RY   NX    ZERO   L      M    RA ;
                    (1653)                RF200     ,LOADRSC
0E3: 8A3C 1A0E 0000 0000
                    (1654)        RR      RM  => RP  TR= NX
0E4: EE00 7A04 0000 0000
                    (1655)        RR      RCM = '10  => RY
0E5: F200 0804 0002 0008
                    (1656)        RR      ,,,JUMP ON NE TO FOUT
0E6: 0200 0004 0004 40FC
                    (1657) *
                    (1658) *      TEST '11:   I/O BUS TEST.  ALTENATING ONES AND ZEROS ARE
                    (1659) *              SENT OUT BOTH BPA AND BPD AND READ BACK IN.  THIS
                    (1660) *              CHECKS CPU LOU C IN THE PATH AND 'STUCK AT' FAULTS ON THE
                    (1661) *              CONTROLERS.
                    (1662) *
                    (1663)        CPU     BB   RCM  NX    0      0     RSC   0 ;
                    (1664)                RYRF240   ,,DATA $A5A5
0E7: F203 0D04 0003 46A5
                    (1665)        CPU     BB   RY   NX    0      0     RY    PIO ;
                    (1666)                RM200
0E8: EA02 1204 0000 0000
                    (1667)        CPU     BB   BPA  NX    XOR    L     RSC   DATA ;
                    (1668)                RY240     ,SETCC
0E9: F66F 2307 0000 0000
                    (1669)        CPU     BB   BPD  NX    XOR    L     RSC   DATA ;
                    (1670)                RM200     ,SETCC ;
                    (1671)                JUMP ON NE TO *+4
0EA: FA6F 2207 0004 40EE
                    (1672)        CPU     RFRS 2    NX    2      0     RSC   0 ;
                    (1673)                RYRF240   ,, ;
                    (1674)                JUMP ON NE TO *+3
0EB: 4A23 0D04 0004 40EE
                    (1675)        CPU     BB   BPA  NX    XOR    L     RSC   DATA ;
                    (1676)                RY240     ,SETCC
0EC: F66F 2307 0000 0000
                    (1677)        CPU     BB   BPD  NX    XOR    L     RSC   DATA ;
                    (1678)                RM200     ,SETCC ;
                    (1679)                JUMP ON EQ TO *+3
0ED: FA6F 2207 0006 40F0
                    (1680)        RR      RCM = '11  => RY
0EE: F200 0804 0002 0109
                    (1681)        CPU     AL   0    NX    ZERO   L     RSC   0 ;
```

```
                   (1682)              RF200     ,,GO TO FOUT
0EF: 823F 0A04 0004 00FC
                   (1683) *
                   (1684) *      TEST '12:   MEMORY TEST.  LOCATION 5 (UNDER THE REGISTERS)
                   (1685) *             IS TESTED TO SEE IF ONES AND ZEROS CAN BE WRITTEN AND READ.
                   (1686) *             NOTE THAT A MISSING FIRST 8K CAN CAUSE A MISSING MEMORY
                   (1687) *             TRAP.
                   (1688) *
                   (1689)        RR     RCM = 5   => RY
0F0: F200 0804 0002 0105
                   (1690)        CPU    BB   RM    NONE  0      0      M      RX ;
                   (1691)               RF200      AWRITE
0F1: EC00 0AF4 0000 0000
                   (1692)        CPU    BB   RY    NONE  0      0      0      0  ;
                   (1693)               RY200
0F2: E800 0804 0000 0000
                   (1694)        CPU    ,,NONE     ,,,,RMMRDY   AREAD
0F3: 0000 07C4 0000 0000
                   (1695)        ALU    RX XOR RM => RX SETCC
0F4: 8E6C 0A07 0000 0000
                   (1696)        RR     RCM = '12  => RY  TR= NONE
0F5: F000 0804 0002 010A
                   (1697)        RR     ,,,JUMP ON NE TO FOUT
0F6: 0200 0004 0004 40FC
                   (1698) *
                   (1699) *      TEST '13:   PARITY TEST.  AFTER VERIFYING PARITY WORKS IN
                   (1700) *             TESTS 4,5, AND 6, THE REST OF THE TESTS ARE RUN WITH
                   (1701) *             PARITY ENABLED.  IF MACHINE CHECK GETS SET DURING THOSE
                   (1702) *             TESTS, '13 WILL FAIL.
                   (1703) *
                   (1704)        ALU    INC RY => RY,, JUMP ON MC TO FOUT
0F7: 8A65 1304 0007 10FC
                   (1705) *
                   (1706) *      SUCCESSFUL TEST COMPLETION
                   (1707) *
                   (1708)        CPU    BB   RCM   NX    RF      0      M      RP ;
                   (1709)               RY200      BD01  SETCC  DATA '1000
0F8: F200 7837 0000 0500
                   (1710)        ALU    CON 0 => RM ,JUMP ON EQ TO MC4
0F9: 823C 0204 0006 4034
                   (1711)        RR     ,,CLEARFUII  JUMP ON FUII TO CAS5
0FA: 0200 000F 0005 51E5
                   (1712)        RR     ,,,GO TO CPPAR3
0FB: 0200 0004 0004 007E
                   (1713) *
                   (1714) *      FAILED TEST COMPLETION/RETRY
                   (1715) *
                   (1716) FOUT   CPU    BB   RY    NONE  0      0      M      RA ;
                   (1717)               RF200      ,, ;
                   (1718)               JUMP ON FUII TO F1
0FC: E800 1A04 0005 5000
                   (1719)        ALU    CON 0  => RX   TR= NONE ,  GO TO FOUT1
0FD: 803C 0A04 0004 008E
                   (1720)        EJCT
```

```
                        (1721)          ORG      $100
                        (1722) ENTR     ALU      RS MINUS RY  => RY
100: 8A94 3304 0000 0000
                        (1723)          CPU      RF,,ALL,,,M,RS,RM280,MWRITE
101: 0300 3484 0000 0000
                        (1724)          RR       RY => RS  JAMF
102: E800 3900 0000 0000
                        (1725) FLX1     RR       RM  => RX  ,GO TO FLX2
103: EE00 0A04 0004 012A
                        (1726)          ORG      $104
                        (1727) RTN      RR       RS => RY
104: 0200 3804 0000 0000
                        (1728)          CPU      ,,ALL,,,,,RMMRDY,MREAD
105: 0300 0784 0000 0000
                        (1729)          ALU      INC RM => RY
106: 8E65 1304 0000 0000
                        (1730)          CPU      BB   RM   ALL   BB     L     M     11 ;
                        (1731)                   RMRFMRDY  MREAD SETCC
107: EF5C 9587 0000 0000
                        (1732)          RR       11 => RY   ,JUMP ON EQ TO *+3
108: 0200 9804 0006 4108
                        (1733)          RR       RY => RS
109: E800 3904 0000 0000
                        (1734)          RR       RM => RP  JAMF  TR= NX
10A: EE00 7A00 0000 0000
                        (1735)          RR       RCM = '75  => RY
10B: F200 0804 0002 003D
                        (1736)          RR       RP  => RM  ,GO TO UII3+1
10C: 0200 7204 0004 00B4
                        (1737) FGEN     CPU      RF   0    NX    0      0     M        0 ;
                        (1738)                   RF240     0     NOP    GO TO FGEN1
10D: 0200 0B04 0004 0406
                        (1739)          ORG      $110
                        (1740) *
                        (1741) *        INSTRUCTION EXECUTION SPACE
                        (1742) *
              000420    (1743) EVMX     EQU      *
              000420    (1744) ERMX     EQU      *
              000420    (1745) EVMJ     EQU      *
                        (1746) ERMJ     RR       RP => RY  NOP  JUMP ON RXM TO RXM
110: 0200 7804 0007 4068
                        (1747)          CPU      BB   RM   ALL   0      0     M     13 ;
                        (1748)                   RMRFMRDY  MREAD NOP ;
                        (1749)                   MODAL     NOP   (TRIGNVKEYS,RXM,EINTM)
111: EF00 B584 0008 0411
                        (1750)          RR       RM => RY  EAF   EAC  SETFUII
112: EE00 080D 000A 8000
                        (1751)          CPU      AL   0    ALL   INC    1     M        RP ;
                        (1752)                   RMRFMRDY  MREAD NOP ;
                        (1753)                   JUMP ON RSC12 TO EVMX1
113: 8304 7584 0005 B1F2
                        (1754)          RR       RM => RP  TR= NX
114: EE00 7A04 0000 0000
                        (1755)          RR       13 => RM  TR= NX   ,EAC CLEARCAM
115: 0200 B204 0009 C000
                        (1756) EPMJ6    RR       ,,JAMF TR= ALL   MODAL NOP (TRIGNVKEYS,PAM)
116: 0300 0000 0008 0420
                        (1757) ERM      RR       ,,RESTJAMF    MODAL NOP (TRIGNVKEYS,RXM,EINTM)
117: 0200 0002 0008 0411
                        (1758) NRM      RR       RCM = 0   =>  RM  C= BD01  LOADRSC
118: F000 013E 0002 0100
                        (1759)          CPU      RF   0    ALL   0      0     M     RA ;
                        (1760)                   200  SOVFL
119: 0300 1064 0000 0000
                        (1761)          CPU      RFLS 7    NX    3      0     M        RB ;
                        (1762)                   RF200     LINK  , ;
                        (1763)                   JUMP ON FCBIT TO *+2
11A: 3E30 2A44 0004 F11C
                        (1764)          CPU      RFLS 3    ALL   7      0     M     RA ;
                        (1765)                   RF200     SOVFL INCRSC ;
                        (1766)                   JUMP ON RSCNEM1 TO *-2
11B: 2F70 1A65 0004 9119
                        (1767)          CPU      RFRS 0    NX    6      0     M     RB ;
                        (1768)                   RF200
11C: 4260 2A04 0000 0000
                        (1769)          RR       RSC => 13
11D: E400 B904 0000 0000
                        (1770)          ALU      13 AND RCM =  $3F  => (RY,13)  TR= TDMX
11E: 911C BD04 0002 013F
                        (1771)          RR       ,,,GO TO OTK+2
11F: 0200 0004 0004 0194
              000440    (1772) EPMX     EQU      *
              000440    (1773) LPMX     EQU      *
              000440    (1774) LPMJ     EQU      *
                        (1775) EPMJ     RR       RP => RY NOP  JUMP ON RXM TO RXM
120: 0200 7804 0007 4068
                        (1776)          CPU      BB   RM   ALL   0      0     M     13 ;
                        (1777)                   RMRFMRDY  MREAD  ,GO TO ERMJ+2
121: EF00 B584 0004 0112
                        (1778) XEC      RR       ,,,JUMP ON FETCH1 TO *+2
122: 0200 0004 0004 A124
                        (1779)          CPU      RF   0    NX    0      0     M     RA ;
                        (1780)                   CLMCLFCLI  MREAD  ,GO TO F9
123: 0200 1F84 0004 0004
                        (1781)          ALU      DEC RP => RP CLEARFUII JUMP ON FUII TO *
124: 82F0 7A0F 0005 5124
                        (1782)          RR       ,,JAMF
125: 0200 0000 0000 0000
                        (1783) VIRY     RR       ,,REST    EAC SETFUII
126: 0200 0006 000A 8000
```

```
                   (1784)         RR       ,,,GO TO VIRY1
127: 0200 0004 0004 00C4
          000450   (1785) CEA3    EQU      *
                   (1786) EAA     RR       RY => RA  TR= NX  JAMF
128: EA00 1A00 0000 0000
                   (1787) FLX     CPU      ,,ALL,,,,,RMMRDY MREAD  ,GO TO FLX1
129: 0300 0784 0004 0103
                   (1788) FLX2    CPU      RFLS 7    NX    3      0     M     RX ;
                   (1789)                  RF200     ,JAMF
12A: 3E30 0A00 0000 0000
                   (1790) JEQ     ALU      INC RA + 0  => NULL SETCC
12B: 8200 1007 0000 0000
                   (1791)         RR       ,,JAMF JUMP ON EQ TO JMP
12C: 0200 0000 0006 41D0
                   (1792) JNE     ALU      INC RA + 0  => NULL SETCC
12D: 8200 1007 0000 0000
                   (1793)         RR       ,,JAMF JUMP ON NE TO JMP
12E: 0200 0000 0004 41D0
                   (1794) JLE     ALU      INC RA + 0  => NULL SETCC
12F: 8200 1007 0000 0000
                   (1795)         RR       ,,JAMF JUMP ON LE TO JMP
130: 0200 0000 0004 71D0
                   (1796) JGT     ALU      INC RA + 0  => NULL SETCC
131: 8200 1007 0000 0000
                   (1797)         RR       ,,JAMF JUMP ON GT TO JMP
132: 0200 0000 0006 71D0
                   (1798) AOA     ALU      INC RA => RA C= AOVFL  JAMF  DATA 0
133: 8204 1A70 0002 0100
                   (1799) JLT     ALU      INC RA + 0  => NULL SETCC
134: 8200 1007 0000 0000
                   (1800)         RR       ,,JAMF JUMP ON LT TO JMP
135: 0200 0000 0006 61D0
                   (1801) JGE     ALU      INC RA + 0  => NULL SETCC
136: 8200 1007 0000 0000
                   (1802)         RR       ,,JAMF JUMP ON GE TO JMP
137: 0200 0000 0004 61D0
                   (1803) JDX     ALU      DEC RX => RX  SETCC ;
                   (1804)                  GO TO JNE+1
138: 82F0 0A07 0004 012E
                   (1805) JIX     ALU      INC RX => RX  SETCC ;
                   (1806)                  GO TO JNE+1
139: 8204 0A07 0004 012E
                   (1807) JSX     RR       RP => RM
13A: 0000 7104 0000 0000
                   (1808)         RR       RM => RX ,GO TO JMP
13B: EE00 0A04 0004 01D0
                   (1809) CREP    RR       RY => 13
13C: E800 B904 0000 0000
                   (1810)         ALU      INC RS => RY
13D: 8204 3304 0000 0000
                   (1811)         CPU      RF   0    ALL   0      0     M     RP ;
                   (1812)                  RM280     MWRITE
13E: 0300 7484 0000 0000
                   (1813)         RR       13 => RY ,GO TO JMP
13F: 0200 B804 0004 01D0
                   (1814) *
                   (1815) *       UNIMPLEMENTED INSTRUCTION VECTOR (UII)
                   (1816) *
                   (1817) UII     RR       RY => EAS TR= ALL
140: E800 AA04 0000 0000
                   (1818)         RR       RP => RM  ,GO TO UII3
141: 0200 7204 0004 00B3
                   (1819) *
                   (1820) *       ILLEGAL INSTRUCTION VECTOR (ILL)
                   (1821) *
                   (1822) ILL     RR       RY => EAS TR= NX
142: EA00 AA04 0000 0000
                   (1823)         RR       RP => RM  ,GO TO ILL3
143: 0200 7204 0004 00B5
                   (1824) DIV     CPU      0    RCM  ALL   0      0     0     0 ;
                   (1825)                  RMMRDY    MREAD LOADRSC ;
                   (1826)                  DATA -15
144: 1300 078E 0003 FEF1
                   (1827)         CPU      0    RM   NX    XOR    L     M     RA ;
                   (1828)                  200  ,SETCC     GO TO DIV4
145: 0E6C 1007 0004 00A8
                   (1829) DIVER   CPU      RFRS 0    NX    6      0     M     RB ;
                   (1830)                  RF200     0     JAMF
146: 4260 2A00 0000 0000
                   (1831) LDX     CPU      ,,ALL,,,,,RMMRDY   MREAD
147: 0300 0784 0000 0000
                   (1832)         CPU      BB   RM   NX    0      0     XM    0 ;
                   (1833)                  RF200     ,JAMF
148: EE01 0A00 0000 0000
                   (1834) *
                   (1835) *
                   (1836) *         INSTRUCTION EXECUTION.  IN THIS SECTION, INSTRUCTIONS
                   (1837) * (INDICATED BY THE LABELS) ARE EXECUTED.
                   (1838) *
                   (1839) *
                   (1840) *         SEVERAL GEN B INSTRUCTIONS DECODE TO THE SAME PLACE. AS
                   (1841) * A RESULT, THEY MUST BE SPECIALLY DECODED BY TESTING FOR DIF-
                   (1842) * FERENT OP CODES IN RM.
                   (1843) *
                   (1844) GENB    RR       ,,,JUMP ON RSC12 TO RMC
149: 0200 0004 0005 B187
                   (1845)         RR       ,,,JUMP ON RS16 TO GENB1
14A: 0200 0004 0005 2183
                   (1846) HLT     RR       RP => RY REST GO TO CP1
14B: 0200 7806 0004 0035
```

```
                   (1847) SUB     CPU       ,,ALL,,,,,RMMRDY   MREAD , JUMP ON DP TO SUB3
14C: 0300 0784 0005 C16C
                   (1848)         ALU       RA SUB RM => RA C= ADVFL JAMF
14D: 8E94 1A70 0000 0000
                   (1849) JST     CPU       ,,ALL,,,,,RMMRDY   MREAD
14E: 0300 0784 0000 0000
                   (1850)         CPU       AL    RM    NX    RF      0      M      RP ;
                   (1851)                   RM200       ,,EAC JST
14F: 8E00 7204 0008 8000
                   (1852) JST1    CPU       BB    RY    ALL   0       0      M      RP ;
                   (1853)                   RF200       MWRITE        ,GO TO CAS5
150: E800 7A84 0004 01E5
                   (1854)         EJCT
                   (1855) *
                   (1856) *       PROGRAMMED INPUT/OUTPUT.  INA,OTA,SKS,AND OCP ARE
                   (1857) * ALL EXECUTED USING THIS COMMON CODE. INSTRUCTION
                   (1858) * SPECIFIC CODE IS EXECUTED AFTER TESTING THE TWO HIGH
                   (1859) * ORDER BITS OF THE OP CODE.  TH  TESTS ON DEVICE
                   (1860) * ADDRESS 20 ARE TO AVOID TESTING READY, CLEAN UP PARITY
                   (1861) * AND NOT SKIP FOR DEVICE 20 DEVICES INCLUDING THE CONTROL
                   (1862) * PANEL THE REAL TIME CLOCK, AND THE MASK CHANGE COMMANDS.
                   (1863) *
                   (1864) INA     CPU       0     RCM   NX    0       0      RY     PIO ;
                   (1865)                   280   NOP   LOADRSC ;
                   (1866)                   DATA 1
151: 1202 160E 0002 0001
                   (1867)         CPU       RF    0     NX    0       0      RSC    0 ;
                   (1868)                   RM200       NOP   NOP ;
                   (1869)                   JUMP ON F01NOT TO SKS
152: 0203 0204 0005 3157
                   (1870)         CPU       BB    BPD   NX    0       0      RSC    0 ;
                   (1871)                   RF280       0     NOP ;
                   (1872)                   JUMP ON READYNOTANDNE20 TO INA1
153: FA03 0C04 0005 7156
                   (1873)         CPU       0     0     NONE  0       0      RSC    STROBE ;
                   (1874)                   200   NOP   NOP ;
                   (1875)                   JUMP ON BPSP1NOT TO INA2
154: 0003 8004 0006 318A
                   (1876)         CPU       AL    0     NONE  RF      0      RSC    0 ;
                   (1877)                   RM200       NOP   NOP ;
                   (1878)                   JUMP ON DANOT20 TO CAS5
155: 8003 0204 0005 61E5
                   (1879) INA1    CPU       AL    RM    NONE  BB      L      RSC    0 ;
                   (1880)                   RF200       NOP   JAMF
156: 8C5F 0A00 0000 0000
                   (1881) SKS     CPU       ,,NX,,,RSC,0,280,,,JUMP ON READYANDF02 TO SKS1
157: 0203 0604 0005 81EC
                   (1882)         CPU       ,,NX,,,RSC,STROBE,,,,GO TO INA1
158: 0203 8004 0004 0156
          000531   (1883) PIO     EQU       *
                   (1884) OTA     RR        RM => RY  TR= ALL  REST  JUMP ON OTANOT TO INA
159: EF00 0806 0005 4151
                   (1885)         CPU       0     RCM   NX    0       0      RY     PIO ;
                   (1886)                   280   0     LOADRSC  DATA 1
15A: 1202 160E 0002 0001
                   (1887)         CPU       RF    RCM   NX    0       0      RSC    0 ;
                   (1888)                   RM200       NOP   LOADRSC  DATA 7
15B: 1203 020E 0002 0007
                   (1889)         CPU       0     0     NX    0       0      RSC    DATA ;
                   (1890)                   280   NOP   NOP ;
                   (1891)                   JUMP ON READYNOTANDNE20 TO F1
15C: 0203 2604 0005 7000
                   (1892)         CPU       ,,,,,RSC  (DATA,STROBE)  200 ,,JUMP ON DANOT20 TO OTA1
15D: 0203 A004 0005 615F
                   (1893)         CPU       ,,ALL,,,RSC DATA  200 ,JAMF
15E: 0303 2000 0000 0000
                   (1894) OTA1    CPU       AL    0     ALL   INC     1      RSC    DATA ;
                   (1895)                   RF200  NOP  EAF ;
                   (1896)                   GO TO F1
15F: 8307 2A0D 0004 0000
                   (1897)         EJCT
```

```
                  (1898)        ORG      $160
                  (1899) LDA3   ALU      INC RY => RY EAF
160: 8A65 130D 0000 0000
                  (1900)        CPU      BB   RM   ALL  0      0     M     RA ;
                  (1901)                 RMRFMRDY  MREAD
161: EF00 1584 0000 0000
                  (1902)        RR       RM => RB  TR= NX       JAMF
162: EE00 2A00 0000 0000
                  (1903) FLD    CPU      BB   RCM  ALL  0      0     M        VSC ;
                  (1904)                 RMRFMRDY  MREAD NOP   DATA  $00FF
163: F300 6584 0002 01FF
                  (1905)        CPU      BB   RM   NX   0      0     M        FLTH ;
                  (1906)                 RF200     NOP  NOP    GO TO FLD1
164: EE00 4A04 0004 0456
                  (1907) FAD   CPU     0    0    ALL  0      0     0        0 ;
                  (1908)                 RMMRDY    MREAD NOP   GO TO FAD1
165: 0300 0784 0004 040B
                  (1909) FSB    CPU      0    0    ALL  0      0     0        0 ;
                  (1910)                 RMMRDY    MREAD NOP   GO TO FSB1
166: 0300 0784 0004 040D
                  (1911) FMP    CPU      0    0    ALL  0      0     0        0 ;
                  (1912)                 RMMRDY    MREAD NOP   GO TO FMP1
167: 0300 0784 0004 0417
                  (1913) FDV   CPU     0    0    ALL  0      0     0        0 ;
                  (1914)                 RMMRDY    MREAD NOP   GO TO FDV1
168: 0300 0784 0004 042B
                  (1915) FST    CPU      RF   0    NX   0      0     M        FLTH ;
                  (1916)                 RM200     NOP  NOP    GO TO FST1
169: 0200 4204 0004 0464
                  (1917) FCS    CPU      BB   RCM  ALL  0      0     M        11 ;
                  (1918)                 RMRFMRDY  MREAD NOP   DATA  $00FF
16A: F300 9584 0002 01FF
                  (1919)        CPU      BB   RM   NX   XOR    L     M        FLTH ;
                  (1920)                 280  BD01 SETCC GO TO FCS1
16B: EE6C 4637 0004 046B
                  (1921) SUB3   ALU      INC RY => RY EAF
16C: 8A65 130D 0000 0000
                  (1922)        CPU      BB   RM   ALL  0      0     M     13 ;
                  (1923)                 RMRFMRDY  MREAD
16D: EF00 8584 0000 0000
                  (1924)        ALU      RB MINUS RM  => RB SETCC
16E: 8E94 2A07 0000 0000
                  (1925)        RR       13 => RM  TR= NX ,   JUMP ON GE SUB+1
16F: 0200 8204 0004 6140
                  (1926)        ALU      RB AND RCM = $7FFF  => RB  TR= ALL
170: 931C 2A04 0000 FFFF
                  (1927)        ALU      RA SUB RM + 0  => RA  C= AOVFL  JAMF
171: 8E90 1A70 0000 0000
                  (1928) PIM    ALU      RA AND RCM = $8000  => RM
172: 921C 1204 0001 0100
                  (1929)        ALU      RB OR RM  => RM ,GO TO LDA+1
173: 8E4C 2204 0004 01DF
                  (1930) PID    ALU      INC RA => RY + 0 SETCC
174: 8200 1307 0000 0000
                  (1931)        ALU      CON -1 => RA    ,JUMP ON GE TO XCA+1
175: 82CC 1A04 0004 61A3
                  (1932)        RR       RY => RB
176: E800 2904 0000 0000
                  (1933)        ALU      RB AND RCM = $7FFF  => RB    JAMF
177: 921C 2A00 0000 FFFF
                  (1934) IMA    RR       RA => RM
178: 0000 1104 0000 0000
                  (1935)        CPU      BB   RM   ALL  0      0     M     13 ;
                  (1936)                 RMRFMRDY  MREAD
179: EF00 B584 0000 0000
                  (1937)        RR       RM => RA  TR= NX
17A: EE00 1A04 0000 0000
                  (1938)        CPU      RF   0    ALL  0      0     M     13 ;
                  (1939)                 RM280     MWRITE      JAMF
17B: 0300 64B0 0000 0000
                  (1940) MPY    CPU      0    RCM  ALL  0      0     M     0 ;
                  (1941)                 RMMRDY    MREAD LOADRSC ;
                  (1942)                 DATA -15
17C: 1300 078E 0003 FEF1
                  (1943)        CPU      RFRS 5    NX   6      0     M     RA ;
                  (1944)                 RY240     LINK
17D: 5660 1344 0000 0000
                  (1945)        RR       RY => RB
17E: E800 2904 0000 0000
                  (1946)        RR       RCM = 0  => RA
17F: F000 1904 0002 0100
                  (1947)        CPU      ALRS RM   NX   ADD    0     M     RA ;
                  (1948)                 RF240     LINK ,EAC MPYLOGIC
180: CE60 1844 0008 C000
                  (1949)        CPU      RFRS 0    ALL  6      0     M     RB ;
                  (1950)                 RF200     LINK INCRSC ;
                  (1951)                 JUMP ON RSCNEM1 TO *-1
181: 4360 2A45 0004 9180
                  (1952)        CPU      AL   RM   NX   SUB    1     M     RA ;
                  (1953)                 RF240     AOVFL JAMF  EAC MPYLOGIC
182: 8E94 1870 0008 C000
                  (1954) GENB1  RR       ,,JAMF    JUMP ON RSC11 TO *+1
183: 0200 0000 0005 A184
                  (1955)        ALU      VSC AND RCM = $FF => RM  TR= ALL
184: 931C 6204 0002 01FF
                  (1956) SCA    RR       RM => RA  JAMF ;
                  (1957)                 JUMP ON RS15 TO *+1
185: EE00 1A00 0005 0186
                  (1958) INK    ALU      RA OR KEYS => RA       DISABLERHBB GO TO F1
186: 864C 1A08 0004 0000
```

```
                   (1959) RMC    RR      ,,RESTJAMF    MODAL NOP (TRIGNVKEYS,MCHK)
187: 0200 0002 0008 0404
          000610   (1960) DBL    EQU     *
                   (1961) SGL    RR    ,,JAMF      MODAL NOP (TRIGVKEYS,DP)
188: 0200 0000 0008 0140
          000611   (1962) E16S   EQU     *
          000611   (1963) E32S   EQU     *
          000611   (1964) E32R   EQU     *
                   (1965) E64R   RR      ,,JAMF    MODAL NOP (TRIGVKEYS,AM1,AM2)
189: 0200 0000 0008 010C
                   (1966) INA2   CPU     AL   RM   NX    OR     L     RSC   0 ;
                   (1967)                RF200     0     NOP ;'
                   (1968)                GO TO CASS
18A: 8E4F 0A04 0004 01E5
                   (1969) CEA    CPU     BB   RCM  NONE  ,,,,280       ,PUSHBD ;
                   (1970)                DATA CEA3
18B: F000 060C 0000 0328
                   (1971)        RR      RA => RM , GO TO F6
18C: 0200 1204 0004 0010
          000615   (1972) EMCM   EQU     *
                   (1973) LMCM   RR      ,,,JUMP ON RXM TO RXM
18D: 0200 0004 0007 4068
                   (1974)        RR      ,,JAMF MODAL NOP (TRIGNVKEYS,PARIM)
18E: 0200 0000 0008 0402
                   (1975) IAB2   RR      RY => RB  TR= NX        ,GO TO LDA+1
18F: EA00 2A04 0004 01DF
          000620   (1976) ENB    EQU     *
                   (1977) INH    RR      ,,,JUMP ON RXM TO RXM
190: 0200 0004 0007 4068
                   (1978)        RR      ,,JAMF    MODAL NOP (TRIGNVKEYS,EINTM)
191: 0200 0000 0008 0410
                   (1979) OTK    RR      RA => , TR= NX  C= BD01         ;
                   (1980)                EAC PLOADVKEYS TRIGVKEYS
192: 0200 1034 000A 4100
                   (1981)        ALU     RA AND RCM = $FF => RY  TR= ALL
193: 931C 1304 0002 01FF
                   (1982)        ALU     VSC AND RCM = $FF00 => VSC
194: 921C 6A04 0003 FF00
                   (1983)        ALU     VSC OR  RY  => VSC     JAMF
195: 8A4C 6A00 0000 0000
                   (1984) *
                   (1985) *      NON-VISIBLE KEYS ARE TRIGGERED ON CAI BECAUSE THIS
                   (1966) * INHIBITS INTERRUPTS FOR THE FOLLOWING INSTRUCTION.
                   (1987) * (JST CONCATINATION ALSO PERFORMS THE SAME FUNCTION.
                   (1988) *
          000626   (1989) ESIM   EQU     *
                   (1990) EVIM   RR      ,,,JUMP ON RXM TO RXM
196: 0200 0004 0007 4068
                   (1991)        RR      ,,JAMF    MODAL NOP (TRIGNVKEYS,VIM)
197: 0200 0000 0008 0408
                   (1992) SVC    RR      RCM = '65 => RY
198: F200 0804 0002 0135
                   (1993)        CPU     ,,NONE,,,,,RMMRDY,AREAD,,GO TO AVECT+1
199: 0000 07C4 0004 0078
                   (1994) ISI    RR      SI => RA  TR= NX
19A: E200 1A04 0000 0000
                   (1995)        ALU     RA AND RCM = $F => RA  TR= 0  RESTJAMF
19B: 901C 1A02 0002 010F
                   (1996) OSI    RR      ,,,JUMP ON RXM TO RXM
19C: 0200 0004 0007 4068
                   (1997)        RR      RA => , TR= NX  JAMF   EAC LOADSERIALINT
19D: 0200 1000 0008 4000
                   (1998) CRL    RR      RCM = 0 => RA   TR= NX
19E: F200 1A04 0002 0100
                   (1999) CRB    RR      RCM = 0 => RB   TR= NX JAMF
19F: F200 2A00 0002 0100
                   (2000) CHS    ALU     RA XOR RCM = $8000 => RA     JAMF
1A0: 926C 1A00 0001 0100
                   (2001) SSP    ALU     RA AND RCM = $7FFF => RA     JAMF
1A1: 921C 1A00 0000 FFFF
          000642   (2002) IAB    EQU     *
                   (2003) XCA    RR      RA => RY  TR= NX  , JUMP ON RS16 TO XCB
1A2: 0200 1804 0005 21A8
                   (2004)        RR      RY => RB  TR= NX ,GO TO CRA
1A3: EA00 2A04 0004 01CE
                   (2005) SOA    ALU     RA MINUS RCM = 1  => RA  C= AOVFL  JAMF
1A4: 9294 1A70 0002 0001
                   (2006) IRX    CPU     AL   ,,INC 1    XM     0     RF200 ,SETCC
1A5: 8205 0A07 0000 0000
                   (2007)        RR      ,,JAMF    JUMP ON EQ TO CASS
1A6: 0200 0000 0006 41E5
                   (2008) RCB    RR      RCM = 0 ,C= BD01        JAMF
1A7: F200 0030 0002 0100
                   (2009) XCB    RR      RB => RM  TR= NX  , JUMP ON RS16 TO IAB2
1A8: 0200 2204 0005 218F
                   (2010)        RR      RM => RA  TR= NX        ,GO TO CRB
1A9: EE00 1A04 0004 019F
                   (2011) DRX    CPU     AL   ,,DEC 0    XM     0     RF200 ,SETCC ;
                   (2012)                GO TO IRX+1
1AA: 82F1 0A07 0004 01A6
                   (2013) CAZ    CPU     RF   ,,RF 0     M      RA    RF200 ;
                   (2014)                ,SETCC    GO TO CAS5-2
1AB: 0200 1A07 0004 01E3
                   (2015) CSA    ALU     RA AND RCM = $7FFF  => RA    C= RF01 JAMF
1AC: 921C 1A50 0000 FFFF
                   (2016) A2A    CPU     AL   RCM  NX    ADD    0     M           RA ;
                   (2017)                RF240     AOVFL JAMF   DATA 2
1AD: 9260 1B70 0002 0002
                   (2018) S2A    CPU     AL   RCM  NX    SUB    1     M           RA ;
                   (2019)                RF240     AOVFL JAMF   DATA 2
```

```
1AE: 9294 1870 0002 0002
                        (2020) CMA    ALU     NOT RA => RA     JAMF
1AF: 82FC 1A00 0000 0000
                        (2021) TCA    ALU     NOT RA => RA     ,GO TO AOA
1B0: 82FC 1A04 0004 0133
                        (2022) SSM    ALU     RA OR RCM = $8000  => RA     JAMF
1B1: 924C 1A00 0001 0100
                        (2023) SCB    RR      RCM = $8000 => RY       C= BD01     JAMF
1B2: F200 0830 0001 0100
                        (2024) CAR    ALU     RA AND RCM = $FF00  => RA    JAMF
1B3: 921C 1A00 0003 FF00
                        (2025) CAL    ALU     RA AND RCM = $00FF  => RA    JAMF
1B4: 921C 1A00 0002 01FF
                        (2026) ICL    ALU     RA AND RCM = $FF00  => RA    FLIP BYTES JAMF
1B5: 721C 1A00 0003 FF00
                        (2027) ICR    ALU     RA AND RCM = $00FF  => RA    FLIP BYTES JAMF
1B6: 721C 1A00 0002 01FF
                        (2028) STX    CPU     RF   0    ALL    0       0      XM     0 ;
                        (2029)                RM280   MWRITE  JAMF
1B7: 0301 0480 0000 0000
                        (2030) ACA    ALU     INC RA + CBIT => RA    C= AOVFL     JAMF
1B8: 8208 1A70 0000 0000
                        (2031) ICA    ALU     INC RA + L  => RA      FLIP BYTES   JAMF
1B9: 620C 1A00 0000 0000
                        (2032) *
                        (2033) *      LOGIC COMMON ENTRY.  ALL LOGICIZE INSTRUCTIONS BEGIN HERE.
                        (2034) *
                        (2035) LOGIC  CPU     BB   RM   NX    DEC     1      M      RA ;
                        (2036)                280   ,SETCC     S TRUE 16WAYS TO $1C0
1BA: EEF4 1607 000C 01C2
                        (2037) LRL    CPU     RFRS 5    ALL    6       ,M     RA     RF200 ;
                        (2038)                LINKS
1BB: 5760 1A14 0000 0000
                        (2039)        CPU     RFRS 1    NX     6       ,M     RB     RF200 ;
                        (2040)                LINKS INCRSCF ;
                        (2041)                JUMP ON RSCNEM1 TO *-1
1BC: 4660 2A11 0004 91BB
                        (2042) LRS    CPU     RFRS 3    ALL    6       ,M     RA     RF200 ;
                        (2043)                LINKS
1BD: 4F60 1A14 0000 0000
                        (2044)        CPU     RFRS 0    NX     6       ,M     RB     RF200 ;
                        (2045)                LINKS INCRSCF ;
                        (2046)                JUMP ON RSCNEM1 TO *-1
1BE: 4260 2A11 0004 91BD
                        (2047) LRR    CPU     RFRS    1    NX    6      ,M      RB     200 ;
                        (2048)                LINK
1BF: 4660 2044 0000 0000
                        (2049)        CPU     RFRS 1    ALL    6       ,M     RA     RF200 ;
                        (2050)                LINKS
1C0: 4760 1A14 0000 0000
                        (2051)        CPU     RFRS 1    NX     6       ,M     RB     RF200 ;
                        (2052)                LINKS INCRSCF ;
                        (2053)                JUMP ON RSCNEM1 TO *-2
1C1: 4660 2A11 0004 91BF
                        (2054) ARL    CPU     RFRS    5    ALL  6      ,M      RA     RF200 ;
                        (2055)                LINKS INCRSCF ;
                        (2056)                JUMP ON RSCNEM1 TO *
1C2: 5760 1A11 0004 91C2
                        (2057) ARS    CPU     RFRS 3    ALL    6       ,M     RA     RF200 ;
                        (2058)                LINKS INCRSCF ;
                        (2059)                JUMP ON RSCNEM1 TO *
1C3: 4F60 1A11 0004 91C3
                        (2060) ARR    CPU     RFRS 2    ALL    6       ,M     RA     RF200 ;
                        (2061)                LINKS INCRSCF ;
                        (2062)                JUMP ON RSCNEM1 TO *
1C4: 4B60 1A11 0004 91C4
                        (2063) LLR    CPU     RFLS 3    NX     7       ,M     RA     200 ;
                        (2064)                LINK
1C5: 2E70 1044 0000 0000
                        (2065)        CPU     RFLS 3    ALL    7       ,M     RB     RF200 ;
                        (2066)                RF01
1C6: 2F70 2A54 0000 0000
                        (2067)        CPU     RFLS 3    NX     7       ,M     RA     RF200 ;
                        (2068)                RF01 INCRSCF ;
                        (2069)                JUMP ON RSCNEM1 TO *-2
1C7: 2E70 1A51 0004 91C5
                        (2070)        EJCT
```

```
                    (2071)          ORG      $1C8
                    (2072) LLT      ALU      CON ZERO => RA  JAMF ;
                    (2073)                   JUMP ON LT TO LT
1C8: 823C 1A00 0006 61CF
                    (2074) LLE      ALU      CON ZERO => RA  JAMF ;
                    (2075)                   JUMP ON LE TO LT
1C9: 823C 1A00 0004 71CF
                    (2076) LNE      ALU      CON ZERO => RA  JAMF ;
                    (2077)                   JUMP ON NE TO LT
1CA: 823C 1A00 0004 41CF
                    (2078) LEQ      ALU      CON ZERO => RA  JAMF ;
                    (2079)                   JUMP ON EQ TO LT
1CB: 823C 1A00 0006 41CF
                    (2080) LGE      ALU      CON ZERO => RA  JAMF ;
                    (2081)                   JUMP ON GE TO LT
1CC: 823C 1A00 0004 61CF
                    (2082) LGT      ALU      CON ZERO => RA  JAMF ;
                    (2083)                   JUMP ON GT TO LT
1CD: 823C 1A00 0006 71CF
          000716    (2084) LF       EQU      *
                    (2085) CRA      RR       RCM = 0  => RA  TR= NX JAMF
1CE: F200 1A00 0002 0100
                    (2086) LT       RR       RCM = 1  => RA  TR= NX JAMF
1CF: F200 1A00 0002 0001
                    (2087) JMP      RR       RY => RP  TR= NX   JAMF
1D0: EA00 7A00 0000 0000
                    (2088) LLL      CPU      RFLS 7    ALL    7       ,M     RB     RF200 ;
                    (2089)                   RF01
1D1: 3F70 2A54 0000 0000
                    (2090)          CPU      RFLS 3    NX     7       ,M     RA     RF200 ;
                    (2091)                   RF01 INCRSCF ;
                    (2092)                   JUMP ON RSCNEM1 TO *-1
1D2: 2E70 1A51 0004 91D1
                    (2093) LLS      RR       RCM = 0  => ,C= BD01
1D3: F200 0034 0002 0100
                    (2094)          CPU      RFLS 7    NX     3       ,M     RB     RF200 ;
                    (2095)                   LINK
1D4: 3E30 2A44 0000 0000
                    (2096)          CPU      RFLS 3    ALL    6       ,M     RA     RF200 ;
                    (2097)                   SOVFL     INCRSCF ;
                    (2098)                   JUMP ON RSCNEM1 TO *-1
1D5: 2F60 1A61 0004 91D4
                    (2099) ALL      CPU      RFLS 7    ALL    7       ,M     RA     RF200 ;
                    (2100)                   RF01 INCRSCF ;
                    (2101)                   JUMP ON RSCNEM1 TO *
1D6: 3F70 1A51 0004 91D6
                    (2102) ALS      RR       RCM = 0  => ,C= BD01
1D7: F200 0034 0002 0100
                    (2103)          CPU      RFLS 7    ALL    7       ,M     RA     RF200 ;
                    (2104)                   SOVFL     INCRSCF ;
                    (2105)                   JUMP ON RSCNEM1 TO *
1D8: 3F70 1A61 0004 91D8
                    (2106) ALR      CPU      RFLS 1    ALL    8       ,M     RA     RF200 ;
                    (2107)                   RF01 INCRSCF ;
                    (2108)                   JUMP ON RSCNEM1 TO *
1D9: 2780 1A51 0004 91D9
                    (2109)          EJCT
```

```
                   (2110) SKP     CPU      BB   RCM  NX    RF      0      M      RA ;
                   (2111)                  RY200     ,SETCC        DATA '131620
1DA: F200 1807 0001 6790
                   (2112) *
                   (2113) *      THE SKIP INSTRUCTION MUST READ THE CONTROL PANEL
                   (2114) * SENSE SWITCHES.  THIS MEANS THE I/O COMMAND MUST BE
                   (2115) * FORMED IN RY AND THE SIGNALS GENERATED.  THE SKIP HARD-
                   (2116) * WARE WILL WORK CORRECTLY ONLY IF RA IS SELECTED FROM THE
                   (2117) * REGISTER FILES, THE CONDITION CODES HAVE BEEN PRESET TO THE
                   (2118) * CONTENTS OF RA, AND BD HAS THE SENSE SWITCHES ON IT.  IF THE
                   (2119) * ABOVE IS TRUE IN ONE CYCLE, THEN FSKIP IS DEFINED TO HAVE
                   (2120) * THE CORRECT SENSE FOR TESTING ON THE NEXT CYCLE.
                   (2121) *
                   (2122) *
                   (2123)         CPU      BB   RCM  NX    ,,RY   PIO   200 ;
                   (2124)                  ,LOADRSC  DATA 1
1DB: F202 100E 0002 0001
                   (2125)         CPU      BB   BPD  NX    RF      0      RSC   0 280
1DC: FA03 0604 0000 0000
                   (2126)         RR       RA => ,   JAMF ;
                   (2127)                  JUMP ON SKIP TO CAS5
1DD: 0200 1000 0004 81E5
                   (2128) LDA     CPU      ,,ALL,,,,,RMMRDY   MREAD , ;
                   (2129)                  JUMP ON DP TO LDA3
1DE: 0300 0784 0005 C160
                   (2130)         RR       RM => RA  TR= NX        JAMF
1DF: EE00 1A00 0000 0000
                   (2131) CAS     CPU      AL   KEYS ALL   BB     L      M      13 ;
                   (2132)                  RMRFMRDY  MREAD
1E0: 875C B584 0000 0000
                   (2133)         ALU      RA MINUS RM => NULL  C= AOVFL  SETCC
1E1: 8E94 1077 0000 0000
                   (2134)         RR       13 => NULL  NOP  C= RF01 ;
                   (2135)                  JUMP ON FCBIT TO CAS6
1E2: 0200 B054 0004 F1F1
                   (2136)         RR       ,,JAMF    JUMP ON LE TO CAS4
1E3: 0200 0000 0004 71E4
                   (2137) CAS4    ALU      INC RP => RP  JAMF ;
                   (2138)                  JUMP ON NE TO CAS5
1E4: 8204 7A00 0004 41E5
                   (2139) CAS5    ALU      INC RP => RP TR= ALL  NOP   GO TO F1
1E5: 8304 7A04 0004 0000
                   (2140) IRS     CPU      ,,ALL,,,,,RMMRDY MREAD
1E6: 0300 0784 0000 0000
                   (2141)         ALU      INC RM => RM  SETCC
1E7: 8E65 1207 0000 0000
                   (2142)         CPU      ,,ALL,,,,,280   MWRITE JAMF  JUMP ON EQ TO CAS5
1E8: 0300 0680 0006 41E5
                   (2143) ANA     CPU      ,,ALL,,,,,RMMRDY   MREAD
1E9: 0300 0784 0000 0000
                   (2144)         ALU      RA AND RM => RA JAMF
1EA: 8E1C 1A00 0000 0000
                   (2145) STA     CPU      RF   0    ALL   0       0      M      RA ;
                   (2146)                  RM280     MWRITE        JAMF ;
                   (2147)                  JUMP ON DP TO STA3
1EB: 0300 1480 0005 C0B7
                   (2148) SKS1    CPU      ,,NX,,,RSC STROBE 200 ,,GO TO INA2
1EC: 0203 8004 0004 018A
                   (2149) ERA     CPU      ,,ALL,,,,,RMMRDY   MREAD
1ED: 0300 0784 0000 0000
                   (2150)         ALU      RA XOR RM => RA JAMF
1EE: 8E6C 1A00 0000 0000
                   (2151) ADD     CPU      ,,ALL,,,,,RMMRDY   MREAD  ,JUMP ON DP TO ADD3
1EF: 0300 0784 0005 C0B9
                   (2152)         ALU      RA ADD RM => RA C= AOVFL     JAMF
1F0: 8E60 1A70 0000 0000
                   (2153) CAS6    RR       ,,JAMF  JUMP ON GT TO CAS4
1F1: 0200 0000 0006 71E4
                   (2154) EVMX1   RR       RM  => 11    TR= NX
1F2: EE00 9A04 0000 0000
                   (2155)         RR       13  => RM  TR= NX  ,EAC CLEARCAM
1F3: 0200 B204 0009 C000
                   (2156)         RR       11  => NULL ,MODAL  NOP  (TRIGNVKEYS,PAM)
1F4: 0200 9004 0008 0420
                   (2157)         CPU      RF   0    NX    0       0      M          11 ;
                   (2158)                  280  NOP  NOP   S ON TRUE BD
1F5: 0200 9604 000C 0000
                   (2159) DFLD    CPU      ,,ALL,,,,,RMMRDY,MREAD,,GO TO DFLD1
1F6: 0300 0784 0004 0500
                   (2160) DFST    CPU      RF,,ALL,,,M,FLTH,RM280,MWRITE,,GO TO DFST1
1F7: 0300 44B4 0004 0507
                   (2161) DFAD    CPU      ,,ALL,,,,,RMMRDY,MREAD,,GO TO DFAD1
1F8: 0300 0784 0004 050D
                   (2162) DFSB    CPU      ,,ALL,,,,,RMMRDY,MREAD,,GO TO DFSB1
1F9: 0300 0784 0004 0516
                   (2163) DFMP    CPU      ,,ALL,,,,,RMMRDY,MREAD,,GO TO DFMP1
1FA: 0300 0784 0004 054B
                   (2164) DFDV    CPU      ,,ALL,,,,,RMMRDY,MREAD,,GO TO DFDV1
1FB: 0300 0784 0004 0570
                   (2165) DFCS    CPU      ,,ALL,,,,,RMMRDY,MREAD,,GO TO DFCS1
1FC: 0300 0784 0004 0564
                   (2166) CAI     RR       ,,,JUMP ON RXM TO RXM
1FD: 0200 0004 0007 4068
                   (2167)         CPU      ,,,,,RY ICAI,,,JAMF  MODAL NOP TRIGNVKEYS
1FE: 0202 8000 0008 0400
                   (2168)         EJCT
```

```
                  (2169) *     P3FLT,U-CODE,MHJ,FEBRUARY 1974
                  (2170) *     FLOATING POINT PROCESSOR - PRIME COMPUTER
                  (2171) *     PRIME COMPUTER INC.,SRC0769.001
                  (2172) *     COPYRIGHT 1974,PRIME COMPUTER INC.,NATICK MASS.
                  (2173) *
                  (2174) *
                  (2175) *
                  (2176) *
          000002  (2177) P300  XSET     2
                  (2178)       CLOCK
                  (2179)       RRCLK
                  (2180)       ALUCLK
                  (2181)        ORG     $400
                  (2182) *
                  (2183) *      FLOATING POINT GENERICS DECODE TO THIS GENERAL
                  (2184) * AREA.  FLOATING SKIPS ARE FIRST.
                  (2185) *
          002000  (2186) FGEN2  EQU     *
                  (2187) FSEQ   RR      ,,JAMF   JUMP ON EQ TO CAS5
400: 0200 0600 0006 41E5
                  (2188) FSNE   RR      ,,JAMF   JUMP ON NE TO CAS5
401: 0200 0600 0004 41E5
                  (2189) FSMI   RR      ,,JAMF  JUMP ON LT TO CAS5
402: 0200 0600 0006 61E5
                  (2190) FSPL   RR      ,,JAMF   JUMP ON GE TO CAS5
403: 0200 0600 0004 61E5
                  (2191) FSLE   RR      ,,JAMF   JUMP ON LE TO CAS5
404: 0200 0600 0004 71E5
                  (2192) FSGT   RR      ,,JAMF   JUMP ON GT TO CAS5
405: 0200 0600 0006 71E5
                  (2193) FGEN1  RR      RCM = $37  => 11
406: F200 9A04 0002 0037
                  (2194)         ALU    11 AND RM  => 11  C= BD01  TR= ALL
407: 8F1C 9A34 0000 0000
                  (2195)         ALU    11 OR     RCM = FGEN2 => RM  TR= ALL
408: 934C 9204 0000 0900
                  (2196)         CPU    BB   RM   NX    0     0     0     0 ;
                  (2197)                200
409: EE00 0004 0000 0000
                  (2198)         CPU    BB   RM   NX    RF    0     M     FLTH ;
                  (2199)                280  NOP  SETCC ;
                  (2200)                S ON TRUE, BD
40A: EE00 4607 000C 0000
                  (2201) FAD1    CPU    BB   RCM  0     0     0     0     0 ;
                  (2202)                280  NOP  PUSHBD  DATA  FAD4
40B: F000 060C 0000 095A
                  (2203)         ALU    INC  RY => RY  TR= ALL  ,GO TO LOAD
40C: 8865 1304 0004 0419
                  (2204) FSB1    CPU    BB   RCM  0     0     0     0     0 ;
                  (2205)                280  NOP  PUSHBD  DATA FSB4
40D: F000 060C 0000 095F
                  (2206)         ALU    INC RY => RY  TR= ALL  ,GO TO LOAD
40E: 8865 1304 0004 0419
                  (2207) *
                  (2208) *       FLOATING COMPLEMENT INSTRUCTION
                  (2209) *
                  (2210)         ORG    (FGEN2 .OR. $10)
                  (2211) FCM     ALU    NOT FLTL => FLTL  ,JUMP ON FCBIT TO FLEX
410: 82FC 5B04 0004 F4CE
                  (2212)         ALU    FLTL ADD RCM = 1  => FLTL C= COUT  SETCC
411: 9260 5A27 0002 0001
                  (2213)         ALU    NOT FLTH => FLTH  TR= ALL ;
                  (2214)                ,JUMP ON NE TO NRML
412: 83FC 4804 0004 444B
                  (2215)         CPU    AL RCM    NX    INC    1     M     FLTH ;
                  (2216)                RF240     AOVFL SETCC   GO TO NRML
413: 9204 4877 0004 044B
                  (2217) FRN     ALU    FLTL ADD RCM = $80  => FLTL  C= COUT
414: 9260 5A24 0002 0080
                  (2218)         CPU    AL   RCM  NX    INC    C     M     FLTH ;
                  (2219)                RF240     AOVFL SETCC GO TO NRML
415: 9208 4877 0004 044B
                  (2220) FLOT1   RR     RM => FLTL  ,GO TO NRML
416: EE00 5B04 0004 044B
                  (2221) FMP1    CPU    BB   RCM  0     0     0     0     0 ;
                  (2222)                280  NOP  PUSHBD  DATA FMP4
417: F000 060C 0000 087A
                  (2223)         ALU    INC RY => RY  TR= ALL ,GO TO LOAD
418: 8865 1304 0004 0419
                  (2224) *
                  (2225) *
                  (2226) *       LOAD SUBROUTINE.  THIS ROUTINE ASSUMES RM CONTAINS THE
                  (2227) * FIRST WORD FROM MEMORY, RY CONTAINS THE ADDRESS OF THE SECOND
                  (2228) * WORD, AND THE STACK CONTAINS THE RETURN ADDRESS.  THE FIRST
                  (2229) * WORD IS PUT INTO 13 AS IS.  THE SECOND WORD IS BROKEN INTO
                  (2230) * THE REST OF THE FRACTION AND THE EXPONENT AND PLACED INTO
                  (2231) * RY AND 11 FOR THE EXPONENT, AND 12 AND RM FOR THE LOW
                  (2232) * ORDER PART OF THE FRACTION.
                  (2233) *
                  (2234) *
                  (2235) LOAD   CPU     BB   RM   ALL   0      0     M     13 ;
                  (2236)                RMRFMRDY  MREAD
419: EF00 8584 0000 0000
                  (2237)        RR      RCM = $FF00  => 12
41A: F200 AA04 0003 FF00
                  (2238)        RR      RCM = $00FF  => 11
41B: F200 9A04 0002 01FF
                  (2239)        ALU     11 AND RM => (11,RY)
41C: 8E1C 9D04 0000 0000
                  (2240)        ALU     12 AND RM  => 12  CLEARFUII
```

```
41D: 8E1C AA0F 0000 0000
                    (2241)         RR       12 => RM  TR= ALL  ,S ON TRUE, POP
41E: 0300 A404 000C 0004
                    (2242)         ORG      (FGEN2 .OR. $20)
                    (2243) FLOT    RR       RCM = 128+30  => VSC  C= BD01
420: F200 6A34 0002 009E
                    (2244)         CPU      RFLS 7    NX    7      0     M     RB  RM200
421: 3E70 2204 0000 0000
                    (2245)         RR       RA => RY
422: 0200 1804 0000 0000
                    (2246)         RR       RY => FLTH , GO TO FLOT1
423: EA00 4804 0004 0416
                    (2247) INT     ALU      VSC MINUS RCM = 127  => NULL SETCC  C= AOVFL
424: 9294 6077 0002 007F
                    (2248)         CPU      RFRS 5    NX    6      0     M     FLTL ;
                    (2249)                  RY240     NOP   NOP ;
                    (2250)                  JUMP ON LE TO CRL
425: 5660 5304 0004 719E
                    (2251)         ALU      VSC MINUS RCM = 128+30 => RM  SETCC  TR= ALL
426: 9394 6207 0002 009E
                    (2252)         CPU      RF   RM   NX    0      0     M     FLTH ;
                    (2253)                  RM200     NOP   LOADRSC  JUMP ON FCBIT TO CRL
427: 0E00 420E 0004 F19E
                    (2254) INT1    RR       RM => RA ,JUMP ON GT TO FLEX9
428: EE00 1804 0006 74C8
                    (2255)         CPU      AL   RY   ALL   BB     L     M     RB ;
                    (2256)                  RF240     BD01  SETCC ;
                    (2257)                  JUMP ON LT TO INT2
429: 885C 2B37 0006 64BF
                    (2258)         RR       ,,,GO TO F1
42A: 0200 0604 0004 0000
                    (2259) FDV1    CPU      BB   RCM  0     0      0     0     0 ;
                    (2260)                  280  NOP  PUSHBD  DATA FDV4
42B: F000 060C 0000 098D
                    (2261)         ALU      INC RY => RY  ,GO TO LOAD  TR= ALL
42C: 8865 1304 0004 0419
                    (2262) NRM7    CPU      RFRS 1    NX    6      0     M     FLTL ;
                    (2263)                  RF200     NOP   NOP
42D: 4660 5A04 0000 0000
                    (2264)         ALU      11 7 RSC + L  => RY
42E: 867C 9304 0000 0000
                    (2265)         ALU      VSC MINUS RY => VSC  C= AOVFL  ,GO TO FLEX8
42F: 8A94 6B74 0004 04CD
                    (2266)         ORG      (FGEN2 .OR. $30)
                    (2267) FRAC    ALU      VSC MINUS RCM = 128-31  => NULL SETCC C= AOVFL
430: 9294 6077 0002 0061
                    (2268)         CPU      RFRS 5    NX    6      0     M     FLTL ;
                    (2269)                  RY240     ,,JUMP ON LE TO CRL
431: 5660 5304 0004 719E
                    (2270)         ALU      VSC MINUS RCM = 128 => RM SETCC  TR= ALL
432: 9394 6207 0002 0080
                    (2271)         CPU      AL   RM   NX    SUB    1     XM    DISABLE ;
                    (2272)                  RM280     NOP   LOADRSC ;
                    (2273)                  GO TO FRAC1
433: 8E95 140E 0004 0482
                    (2274) *
                    (2275) * DOUBLE PRECISION FLOATING POINT COMPLEMENT
                    (2276) *
                    (2277) DFCM    ALU      NOT FLTL => FLTL ,JUMP ON FCBIT TO DFLEX
434: 82FC 5804 0004 F5C8
                    (2278)         ALU      NOT RB => RB
435: 82FC 2A04 0000 0000
                    (2279)         ALU      INC RB => RB C= COUT  SETCC
436: 8204 2A27 0000 0000
                    (2280)         ALU      NOT FLTH => FLTH , JUMP ON NE TO DNRML
437: 82FC 4804 0004 453A
                    (2281)         ALU      INC FLTL + C => FLTL C= COUT
438: 8208 5A24 0000 0000
                    (2282)         ALU      INC FLTH + C => FLTH C= AOVFL ;
                    (2283)                  SETCC GO TO DNRML
439: 8208 4B77 0004 053A
                    (2284) *
                    (2285) *
                    (2286) *       ADJUST SUBROUTINE.  THIS SUBROUTINE TAKES THE TWO NUMBERS
                    (2287) * FOUND IN THE FLOATING ACCUMULATOR AND THE RESULTS OF THE LOAD
                    (2288) * SUBROUTINE AND MAKES THE EXPONENTS THE SAME SO THEY CAN BE ADDED
                    (2289) * OR SUBTRACTED.  THE TWO NUMBERS ARE LEFT IN THE FLOATING
                    (2290) * ACCUMULATOR AND IN 12,13,AND RM FOR THE LOW ORDER HALF.
                    (2291) * THE VSC CONTAINS THE FINAL ADJUSTED EXPONENT.
                    (2292) *
                    (2293) *
                    (2294) ADJUST ALU       VSC SUB RY => RY ,JUMP ON FCBIT TO ADJ9
43A: 8A94 6304 0004 F443
                    (2295)         RR       RY => NULL  LOADRSC  S ON EQ, POP
43B: EA00 060E 000E 4004
                    (2296)         RR       RCM = 32 => YSAVE
43C: F200 CA04 0002 0020
                    (2297)         CPU      AL   RY   NX    SUB    1     XM    DISABLE ;
                    (2298)                  RY240     NOP   NOP ;
                    (2299)                  JUMP ON LE TO ADJ8
43D: 8A95 1304 0004 7444
                    (2300) *
                    (2301) * ABOVE BRANCH IS TAKEN IF THE ACCUMULATOR MUST BE RIGHT SHIFTED.
                    (2302) *
                    (2303) * NOW TEST THE SHIFT COUNTER AGAINST 31 TO SEE IF
                    (2304) * THE TWO NUMBERS ARE WITHIN RANGE, AND IF SO LOAD RSC AND
                    (2305) * UNNORMALIZE THE NEW ARGUMENT.
                    (2306) *
                    (2307) *
                    (2308)         ALU      YSAVE ADD RY => NULL  SETCC ;
```

```
                   (2309)                C= AOVFL
43E: 8A60 C077 0000 0000
                   (2310)        RR      RY => NULL LOADRSC
43F: EA00 000E 0000 0000
                   (2311)        CPU     RFRS 3    ALL    6      0      M      13 ;
                   (2312)                RF240     LINK   NOP ;
                   (2313)                JUMP ON LE TO NRML
440: 4F60 8844 0004 744B
                   (2314)        CPU     RFRS 1    NX     6      0      M      12 ;
                   (2315)                RF240     NOP    INCRSC ;
                   (2316)                JUMP ON RSCNEM1 TO *-1
441: 4660 AB05 0004 9440
                   (2317)        RR      12 => RM  ,S ON TRUE, POP
442: 0200 A404 000C 0004
                   (2318) *
                   (2319) * NEW ARGUMENT IS GREATER OR EQUAL TO ACC.
                   (2320) *
                   (2321) *
                   (2322) ADJ9   ALU     ZERO MINUS RY => RY ,GO TO MOVE
443: 8A95 1304 0004 0449
                   (2323) ADJ8   ALU     INC 13 + 0 => NULL SETCC
444: 8200 B007 0000 0000
                   (2324)        ALU     YSAVE SUB RY => NULL  SETCC ;
                   (2325)                S ON EQ POP
445: 8A94 C607 000E 4004
                   (2326)        CPU     RFRS 3    ALL    6      0      M      FLTH ;
                   (2327)                RF240     LINK   NOP ;
                   (2328)                JUMP ON LE TO MOVE
446: 4F60 4844 0004 7449
                   (2329)        CPU     RFRS 1    NX     6      0      M      FLTL ;
                   (2330)                RF240     NOP    INCRSC ;
                   (2331)                JUMP ON RSCNEM1 TO *-1
447: 4660 5805 0004 9446
                   (2332)        ALU     VSC ADD RY  => VSC  ,S ON TRUE,POP
448: 8A60 6604 000C 0004
                   (2333) MOVE   CPU     BB   RCM  NX     ZERO   L      M      FLTL ;
                   (2334)                RF200     ,,DATA 0
449: F23C 5A04 0002 0100
                   (2335)        CPU     AL   0    NX     ZERO   L      M      FLTH ;
                   (2336)                RF240     ,,GO TO MOVE-1
44A: 823C 4804 0004 0448
                   (2337) *
                   (2338) *
                   (2339) *      NORMALIZE SUBROUTINE.  THIS ROUTINE TAKES THE FLOATING
                   (2340) * ACCUMULATOR AND NORMALIZES THE RESULTS, ADJUSTING THE
                   (2341) * VSC AS NEEDED.  THE ROUTINE EXPECTS THE CARRY BIT TO
                   (2342) * BE SET IF THE ACCUMULATOR IS ALREADY OVER SHIFTED.  THAT
                   (2343) * IS, THE VALUE MUST BE RIGHT SHIFTED ONE PLACE.
                   (2344) * IF THE CBIT CAN BE SET, THEN THE CONDITION CODE MUST ALSO
                   (2345) * BE SET TO REFLECT THE VALUE OF THE FACCUM.
                   (2346) * IF OVERFLOW OCCURRED, THE CONDITION CODE WILL IN FACT SHOW
                   (2347) * THE REVERSE OF THE CORRECT SIGN. NRML DEPENDS ON THIS.
                   (2348) * THE ROUTINE THEN EXITS TO THE FETCH CYCLE.
                   (2349) *
                   (2350) *
                   (2351) NRML   CPU     RFLS 3    ALL    7      0      M      FLTH ;
                   (2352)                RY240  SOVFL      ,JUMP ON FCBIT TO NRM10
44B: 2F70 4364 0004 F452
                   (2353)        ALU     CON 0 => RY C= BD01 ,JUMP ON FCBIT TO F1
44C: 823C 0334 0004 F000
                   (2354)        RR      RCM = -32 => 11 LOADRSC
44D: F200 9A0E 0003 FEE0
                   (2355) NRM18  CPU     RFLS 7    ALL    7      0      M      FLTL ;
                   (2356)                RF240     LINK   ,JUMP ON FCBIT TO NRM7
44E: 3F70 5844 0004 F42D
                   (2357)        CPU     RFLS 3    Nx     7      0      M      FLTH ;
                   (2358)                RF240     NOP    INCRSC ;
                   (2359)                JUMP ON RSCNEM1 TO NRM17
44F: 2E70 4805 0004 9451
                   (2360)        RR      RCM = 0  => VSC  C= BD01  JAMF
450: F200 6A30 0002 0100
                   (2361) NRM17  CPU     RFLS 3    ALL    7      0      M      FLTH ;
                   (2362)                RY240     SOVFL ,GO TO NRM18
451: 2F70 4364 0004 044E
                   (2363) NRM10  ALU     VSC PLUS RCM = 1  => VSC C= AOVFL
452: 9260 6A74 0002 0001
                   (2364)        CPU     RFRS 5    ALL    6      0      M      FLTH ;
                   (2365)                RF240     LINK   NOP ;
                   (2366)                JUMP ON LT TO *+2
453: 5760 4844 0006 6455
                   (2367)        ALU     FLTH OR RCM = $8000  => FLTH
454: 924C 4A04 0001 0100
                   (2368)        CPU     RFRS 1    ALL    6      0      M      FLTL ;
                   (2369)                RF240     NOP    JAMF ;
                   (2370)                JUMP ON FCBIT TO FLEX
455: 4760 5800 0004 F4CE
                   (2371) *
                   (2372) *
                   (2373) *      EXECUTION OF THE FLOATING POINT MEMORY REFERENCE
                   (2374) * INSTRUCTIONS.
                   (2375) *
                   (2376) *
                   (2377) FLD1   ALU     INC RY  => RY
456: 8A65 1304 0000 0000
                   (2378)        CPU     BB   RCM  ALL    0      0      M      FLTL ;
                   (2379)                RMRFMRDY  MREAD ,DATA $FF00
457: F300 5584 0003 FF00
                   (2380)        ALU     FLTL AND RM  => FLTL
458: 8E1C 5A04 0000 0000
                   (2381)        ALU     VSC AND RM => VSC  JAMF
```

```
459: 8E1C 6A00 0000 0000
                    (2382) FAD4   CPU      BB   RCM  0     0      0     0     0 ;
                    (2383)                 280  NOP  PUSHBD  DATA FAD6
45A: F000 060C 0000 095C
                    (2384)       ALU   VSC MINUS RY => NULL SETCC C= AOVFL  GO TO ADJUST
45B: 8A94 6677 0004 043A
                    (2385) FAD6 ALU   FLTL ADD RM => FLTL C= COUT
45C: 8E60 5A24 0000 0000
                    (2386)       RR    13 => RM
45D: 0200 B204 0000 0000
                    (2387)       ALU   FLTH ADD RM + C => FLTH  C= AOVFL SETCC ;
                    (2388)             GO TO NRML
45E: 8E68 4B77 0004 044B
                    (2389) FSB4   CPU      BB   RCM  0     0      0     0     0 ;
                    (2390)                 280  NOP  PUSHBD  DATA FSB6
45F: F000 060C 0000 0861
                    (2391)       ALU   VSC MINUS RY => NULL SETCC C= AOVFL GO TO ADJUST
460: 8A94 6677 0004 043A
                    (2392) FSB6 ALU   FLTL SUB RM => FLTL  C= COUT
461: 8E94 5A24 0000 0000
                    (2393)         RR      13 => RM
462: 0200 B204 0000 0000
                    (2394)       ALU   FLTH SUB RM  + C => FLTH C= AOVFL SETCC ;
                    (2395)             GO TO NRML
463: 8E98 4B77 0004 044B
                    (2396) FST1 ALU   VSC AND RCM = $FF00  => NULL SETCC C= MWRITE TR= ALL
464: 931C 60B7 0003 FF00
                    (2397)         ALU INC RY => RY
465: 8A65 1304 0000 0000
                    (2398)         ALU      VSC AND RCM = $00FF => RM  C= BD01
466: 921C 6234 0002 01FF
                    (2399)         RR       RM => 11
467: EE00 9A04 0000 0000
                    (2400)         ALU      FLTL AND RCM = $FF00 => RM
468: 921C 5204 0003 FF00
                    (2401)         ALU      11 OR RM  => RM  C= MWRITE  TR= ALL ;
                    (2402)              JAMF  JUMP ON NE TO FLEX1
469: 8F4C 9480 0004 44CF
                    (2403) FDV13  CPU      AL   RM    NX    DEC    0     M     VSC ;
                    (2404)                 RF240     AOVFL ,GO TO FCM
46A: 8EF0 6B74 0004 0410
                    (2405) *
                    (2406) *
                    (2407) *    FLOATING COMPARE.  THIS INSTRUCTION MUST COMPARE:  FIRST,
                    (2408) * SIGN, THEN EXPONENT, FINALLY UPPER THEN LOWER MAGNITUDE.
                    (2409) FCS1   ALU      INC RY => RY ,JUMP ON LT TO FCS2
46B: 8A65 1304 0006 6477
                    (2410) * SIGNS ARE EQUAL
                    (2411)        CPU      BB   RM   ALL   BB     L      M      13 ;
                    (2412)                 RMRFMRDY  MREAD  SETCC
46C: EF5C B587 0000 0000
                    (2413)        ALU      11 AND RM => RY  ,JUMP ON LT TO FCS3
46D: 8E1C 9304 0006 64C8
                    (2414) * EXPONENT TEST IS BACKWARDS FOR NEG #'S.
                    (2415) * TEST FOR ZERO IN EITHER ARGUMENT.
                    (2416)        ALU      INC FLTH + 0  => NULL  SETCC  JUMP ON EQ TO FCS7
46E: 8200 4607 0006 4479
                    (2417)        ALU      VSC MINUS RY  => NULL  SETCC C= AOVFL ;
                    (2418)                 TR= ALL  JUMP ON EQ TO FCS2+1
46F: 8B94 6677 0006 4478
                    (2419) FCS4   ALU      11  7  RM  => 11  ,JUMP ON FCBIT TO FCS2+1
470: 8E7C 9B04 0004 F478
                    (2420)        RR       RX => RX , JUMP ON GT TO F1
471: 0200 0B04 0006 7000
                    (2421) FCS5   RR       13 => RM  NOP  JUMP ON NE TO FCS2+1
472: 0200 B404 0004 4478
                    (2422) * EXPONENTS EQUAL
                    (2423)        ALU      FLTH MINUS RM => NULL SETCC
473: 8E94 4007 0000 0000
                    (2424)        RR       11 => RM  NOP  TR= ALL JUMP ON GT TO F1
474: 0300 9404 0006 7000
                    (2425)        ALU      FLTL MINUS RM => NULL C= COUT  SETCC ;
                    (2426)                 JUMP ON LT TO FCS2+1
475: 8E94 5627 0006 6478
                    (2427) * HIGH ORDER MAGNITUDES ARE EQUAL
                    (2428)        RR       ,,,JUMP ON EQ TO CAS5
476: 0200 0604 0006 41E5
                    (2429) FCS2   RR       RX => RX , JUMP ON FCBIT TO F1
477: 0200 0B04 0004 F000
                    (2430)        ALU      RP PLUS RCM = 2 => RP JAMF
478: 9260 7A00 0002 0002
                    (2431) FCS7   RR       RX => RX  JAMF  JUMP ON EQ TO CAS5
479: 0200 0B00 0006 41E5
                    (2432) *
                    (2433) *
                    (2434) *       FLOATING POINT MULTIPLY.  THE ALGORITHM USED IS BASICALLY THAT
                    (2435) * OF SHIFT AND ADD.  ONLY 24 ITTERATIONS ARE REQUIRED FOR FULL
                    (2436) * ACCURACY.  IN ADDITION, THE FIRST 7 ITTERATIONS ARE SINGLE
                    (2437) * PRECISION ONLY.  THE REMAINING 16 ARE FULL DOUBLE
                    (2438) * PRECISION.
                    (2439) *
                    (2440) *
                    (2441) FMP4   ALU      11 MINUS RCM = 128 => RY
47A: 9294 9304 0002 0080
                    (2442)        ALU      VSC PLUS RY => VSC C= AOVFL
47B: 8A60 6A74 0000 0000
                    (2443)        RR       FLTH => RM
47C: 0200 4204 0000 0000
                    (2444)        RR       FLTL => RY  , JUMP ON FCBIT TO  FLEX
47D: 0200 5304 0004 F4CE
```

```
                (2445)          CPU     AL   RCM  ALL   ZERO  L     M     FLTH ;
                (2446)                  RF200   NOP  LOADRSC  DATA -7
47E: 933C 4A0E 0003 FFF9
                (2447)          CPU     ALFB 0    NONE  RF     0     M     12 ;
                (2448)                  RF200
47F: 6000 AA04 0000 0000
                (2449)          CPU     RFRS 0    NONE  6      0     M     12 ;
                (2450)                  RF200  LINK
480: 4060 AA44 0000 0000
                (2451) *  STD MPY FOR FIRST 7 BITS.  (SORT OF)
                (2452)          CPU     ALRS RM   NX    ADD    0     M     FLTH ;
                (2453)                  RF240     NOP   ,EAC MPYLOGIC
481: CE60 4B04 0008 C000
                (2454)          CPU     RFRS 0    ALL   6      0     M     12 ;
                (2455)                  RF240     LINKS  INCRSC ;
                (2456)                  JUMP ON RSCNEM1 TO *-1
482: 4360 AB15 0004 9481
                (2457) * CBIT = RF16 = LAST BIT FROM 12 (NEW LOW)
                (2458)          CPU     AL   RCM  ALL   ZERO  L     M     FLTL ;
                (2459)                  RF200     NOP   LOADRSC ;
                (2460)                  DATA -16
483: 933C 5A0E 0003 FFF0
                (2461) *  SET UP FOR FULL DOUBLE PRECISION MULTIPY FOR THE
                (2462) * REMAINING PORTION OF THE LOOP.  (16 TIMES + SUB)
                (2463) *
                (2464)          RR      ,,,  JUMP ON FCBIT TO FMP5
484: 0200 0604 0004 F487
                (2465) * SHIFT ONLY
                (2466)          CPU     RFRS      3     ALL    6     0     M FLTH ;
                (2467)                  RF240     LINK  NOP    GO TO FMP7
485: 4F60 4B44 0004 0489
                (2468) FMP6   CPU       RFRS 5    ALL   6      0     M     13 ;
                (2469)                  RF280     NOP   NOP ;
                (2470)                  JUMP ON NOTRF16 TO *-1
486: 5760 BC04 0007 7485
                (2471) FMP5   ALU       FLTL PLUS RY  => FLTL  C= COUT
487: 8A60 5A24 0000 0000
                (2472)        CPU       ALRS RM   ALL   6      C     M     FLTH ;
                (2473)                  RF240     LINK  NOP
488: CF68 4B44 0000 0000
                (2474) FMP7   CPU       RFRS 1    ALL   6      0     M     FLTL ;
                (2475)                  RF240     NOP   INCRSC ;
                (2476)                  JUMP ON RSCNEM1 TO FMP6
489: 4760 5B05 0004 9486
                (2477) *TEST FOR LAST MOVE -- ZERO CBIT
                (2478)        CPU       RF   0    NX    0      0     M     13 ;
                (2479)                  RF280     BD01  NOP ;
                (2480)                  JUMP ON NOTRF16 TO NRML
48A: 0200 BC34 0007 744B
                (2481) * FINAL SUBTRACT
                (2482)        ALU       FLTL MINUS RY => FLTL    C= COUT
48B: 8A94 5A24 0000 0000
                (2483)        ALU       FLTH MINUS RM + C => FLTH  C= AOVFL ;
                (2484)                  SETCC  GO TO NRML
48C: 8E98 4B77 0004 044B
                (2485) *
                (2486) *
                (2487) *      DIVIDE INSTRUCTION.  ROUTINE EXECUTES A NON-PREFORMING
                (2488) * STYLE DIVIDE.  BEFORE STARTING THE LOOP, THE NUMBERS ARE
                (2489) * BOTH MADE POSITIVE.  THERE ARE 31 ITTERATIONS OF THE
                (2490) * DIVIDE LOOP BUT ONLY 17 OF THEM ARE IN FACT FULL DOUBLE
                (2491) * PRECISION, THE REST BEING SINGLE PRECISION.
                (2492) *
                (2493) *
                (2494) *
                (2495) FDV4   ALU       INC 13 + 0 => RM  SETCC
48D: 8200 B207 0000 0000
                (2496)        ALU       FLTH XOR RM => NULL  SETCC ;
                (2497)                  JUMP ON EQ TO FLEX2
48E: 8E6C 4607 0006 44D1
                (2498) * DIVISION BY ZERO OVERFLOW EXIT.
                (2499) * NOW MOVE EXPONENT TO RY
                (2500)        RR        RCM = -17 => NULL LOADRSC
48F: F200 000E 0003 FEEF
                (2501) * DO EXPONENT CALCULATION
                (2502)        ALU       11 MINUS RCM = 129 => RY
490: 9294 9304 0002 0181
                (2503)        ALU       VSC MINUS RY => VSC C= AOVFL ;
                (2504)                  , JUMP ON GE TO *+2
491: 8A94 6B74 0004 6493
                (2505) * SET FUII AS A 'COMPLEMENT RESULTS' FLAG
                (2506)        RR        ,,,EAC SETFUII
492: 0200 0004 000A 8000
                (2507) * SKIP ON LIKE SIGNS, SET UP FOR RM COMP TEST.
                (2508)        ALU       INC RM + 0 => NULL  SETCC ;
                (2509)                  JUMP ON FCBIT TO FLEX
493: 8E61 1607 0004 F4CE
                (2510)        RR        12 => RY , JUMP ON GE TO FDV5
494: 0200 A304 0004 6498
                (2511) * MUST TWO'S COMPLEMENT RM[RY .
                (2512)        ALU       ZERO MINUS RY => NULL C= COUT
495: 8A95 1024 0000 0000
                (2513)        ALU       ZERO MINUS RM + C => RM  SETCC
496: 8E99 1207 0000 0000
                (2514)        ALU       ZERO MINUS RY => RY ,JUMP ON LT TO FDV13
497: 8A95 1304 0006 646A
                (2515) * ABOVE EXIT IS FOR -(1/2)**N---COMPLEMENT THE ACCUMULATOR
                (2516) * AND EXIT
                (2517) * TEST ACC FOR NEG
                (2518) FDV5   ALU       INC FLTH + 0 => NULL SETCC
```

```
498: 8200 4007 0000 0000
                   (2519)        ALU      CON 0 => 12 , JUMP ON GE TO FDV6
499: 823C AB04 0004 64AA
                   (2520) * COMPLEMENT ACC
                   (2521)        ALU      NOT FLTL => FLTL
49A: 82FC 5A04 0000 0000
                   (2522)        ALU      INC FLTL => FLTL SETCC
49B: 8204 5A07 0000 0000
                   (2523)        ALU      NOT FLTH => FLTH , JUMP ON NE TO FDV6
49C: 82FC 4B04 0004 44AA
                   (2524)        ALU      INC FLTH => FLTH SETCC
49D: 8204 4A07 0000 0000
                   (2525)        RR       RX => RX , JUMP ON GE TO FDV6
49E: 0200 0B04 0004 64AA
                   (2526)        ALU      VSC PLUS RCM = 1 => VSC  C= AOVFL
49F: 9260 6A74 0002 0001
                   (2527)        CPU      RFRS 5    ALL    6      0      M      FLTH ;
                   (2528)                 RF240     ,,JUMP ON FCBIT TO FLEX
4A0: 5760 4B04 0004 F4CE
                   (2529)        RR       RX => RX , GO TO FDV6
4A1: 0200 0B04 0004 04AA
                   (2530) * FINNALLY DOUBLE PRECISION DIVIDE LOOP CAN BE EXECUTED.
                   (2531) *
                   (2532) * TEST FOR SUB OR SHIFT AND SHIFT IN
                   (2533) * A QUOTIENT BIT FROM THE LAST ITTERATION.
                   (2534) *
                   (2535) FDV10  CPU      RFLS 3    NX     3      0      M      13 ;
                   (2536)                 RF240     NOP    INCRSC ;
                   (2537)                 JUMP ON LT TO FDV7
4A2: 2E30 BB05 0006 64A5
                   (2538) * IF SHIFT, GO TO SHIFT.  THEN, TEST LOW ORDER HALF.
                   (2539) * ABORT LOW TEST IF HIGH TEST WAS CONCLUSIVE.
                   (2540)        ALU      FLTL MINUS RY => NULL  , ;
                   (2541)                 C= COUT  JUMP ON GT TO FDV8
4A3: 8A94 5624 0006 74A7
                   (2542) * LOW TEST REQUIRED - SHIFT ON LT
                   (2543)        RR       ,,,JUMP ON FCBIT TO FDV8
4A4: 0200 0604 0004 F4A7
                   (2544) * SHIFT STEP PROCESSING -- NO SUBTRACT
                   (2545) *
                   (2546) FDV7   CPU      RFLS 7    NX     7      0      M      FLTL ;
                   (2547)                 RF200     LINK  NOP
4A5: 3E70 5A44 0000 0000
                   (2548)        CPU      RFLS 3    ALL    7      0      M      FLTH ;
                   (2549)                 RF240     LINK NOP  GO TO FDV6
4A6: 2F70 4B44 0004 04AA
                   (2550) * PERFORM SUBTRACT AND SHIFT
                   (2551) FDV8   ALU      FLTL MINUS RY => FLTL  C= COUT
4A7: 8A94 5A24 0000 0000
                   (2552)        CPU      RFLS 7    ALL    7      0      M      FLTL ;
                   (2553)                 RF200     LINK  NOP
4A8: 3F70 5A44 0000 0000
                   (2554)        CPU      ALLS RM   ALL    SUB    C      M      FLTH ;
                   (2555)                 RF240     LINK
4A9: AF98 4B44 0000 0000
                   (2556) FDV6   ALU      FLTH MINUS RM => NULL SETCC ;
                   (2557)                 JUMP ON RSCNEM1 TO FDV10
4AA: 8E94 4607 0004 94A2
                 . (2558)        CPU      RFLS 3    NX     3      0      M      FLTL ;
                   (2559)                 RF200
4AB: 2E30 5A04 0000 0000
                   (2560) *DOUBLE PRECISION HALF FINISHED.  NOW DO SINGLE PRECISIONPART.
                   (2561) FDV11  RR       RCM = -15  => NULL  LOADRSC
4AC: F200 000E 0003 FEF1
                   (2562)        CPU      AL   RCM  NX     AND    L      M      13 ;
                   (2563)                 RY240     NOP    SETCC DATA $7FFF
4AD: 921C B307 0000 FFFF
                   (2564)        CPU      ALLS RM   ALL    SUB    1      M      FLTH ;
                   (2565)                 RF280     LINK   ,EAC DIVLOGIC
4AE: AF94 4C44 0009 0000
                   (2566)        CPU      RFLS 3    ALL    7      0      M      FLTL ;
                   (2567)                 RF240     LINK   INCRSC ;
                   (2568)                 JUMP ON RSCNEM1 TO *-1
4AF: 2F70 5B45 0004 94AE
                   (2569) *RESULTS OF DIVIDE NOW IN RY AND FLTL
                   (2570)        RR       RY => FLTH C= BD01 , JUMP ON FUII TO FCM
4B0: EA00 4B34 0005 5410
                   (2571)        CPU      RFLS 3    ALL    7      0      M      FLTH ;
                   (2572)                 RY240     SOVFL  ,GO TO NRML+1
4B1: 2F70 4364 0004 044C
                   (2573) * PATCH SPACE
                   (2574) FRAC1  RR       RY => RB  TR= ALL  C= BD01
4B2: EB00 2A34 0000 0000
                   (2575)        RR       FLTH => RY  ,JUMP ON LE TO FRAC3
4B3: 0200 4304 0004 74BC
                   (2576)        RR       RY => RA
4B4: EA00 1A04 0000 0000
                   (2577)        ALU      VSC MINUS RCM = 128+31  => NULL ;
                   (2578)                 SETCC  TR= ALL
4B5: 9394 6007 0002 019F
                   (2579)        RR       RM => NULL LOADRSC  JUMP ON GT TO CRL
4B6: EE00 060E 0006 719E
                   (2580)        CPU      RFLS 7    ALL    3      0      M      RB ;
                   (2581)                 RF200     LINK
4B7: 3F30 2A44 0000 0000
                   (2582)        CPU      RFLS 3    NX     3      0      M      RA ;
                   (2583)                 RF240     NOP    INCRSC ;
                   (2584)                 JUMP ON RSCNEM1 TO *-1
4B8: 2E30 1B05 0004 94B7
                   (2585) FRAC2  ALU      RA XOR RCM = $8000  => NULL  SETCC
```

```
4B9: 926C 1007 0001 0100
                  (2586)          ALU      INC RB + 0 => NULL  SETCC ;
                  (2587)                   JUMP ON NE TO F1
4BA: 8200 2607 0004 4000
                  (2588)          RR       RX => RX JAMF  JUMP ON EQ TO CRA
4BB: 0200 0800 0006 41CE
                  (2589) FRAC3    CPU      BB    RY    ALL    ZERO    L      M      RA ;
                  (2590)                   RF240       NOP    SETCC ;
                  (2591)                   JUMP ON EQ TO FRAC2
4BC: EB3C 1B07 0006 44B9
                  (2592)          CPU      RFRS 3      ALL    6       0      M      RA ;
                  (2593)                   RF200       LINK
4BD: 4F60 1A44 0000 0000
                  (2594)          CPU      RFRS 0      NX     6       0      M      RB ;
                  (2595)                   RF240       NOP    INCRSCF ;
                  (2596)                   JUMP ON RSCNEM1 TO *-1
4BE: 4260 2801 0004 94BD
                  (2597) INT2     CPU      RFRS 3      ALL    6       0      M      RA ;
                  (2598)                   RF240       LINK   ,JUMP ON FCBIT TO INT6
4BF: 4F60 1B44 0004 F4C7
                  (2599)          CPU      RFRS 0      NX     6       0      M      RB ;
                  (2600)                   RF240       LINKS  INCRSC ;
                  (2601)                   JUMP ON RSCNEM1 TO *-1
4C0: 4260 2B15 0004 94BF
                  (2602)          ALU      CON ZERO  => NULL  C= BD01  ,JUMP ON FCBIT TO INT4
4C1: 823C 0634 0004 F4C6
                  (2603) INT5     RR       ,,JAMF  JUMP ON LT TO *+1
4C2: 0200 0600 0006 64C3
                  (2604)          ALU      RB PLUS RCM = 1  => RB  C= AOVFL
4C3: 9260 2A74 0002 0001
                  (2605)          ALU      INC RA + C => RA
4C4: 8208 1A04 0000 0000
                  (2606)          ALU      RB AND RCM = $7FFF  => RB C= BD01  JAMF
4C5: 921C 2A30 0000 FFFF
                  (2607) INT4     ALU      INC RA + 0 => NULL SETCC  GO TO INT5
4C6: 8200 1607 0004 04C2
                  (2608) INT6     ALU      INC RA + 0 => NULL SETCC GO TO INT2+1
4C7: 8200 1607 0004 04C0
                  (2609) FCS3     ALU      VSC MINUS RY => NULL C= AOVFL SETCC
4C8: 8A94 6077 0000 0000
                  (2610)          ALU      11  7 RM => 11 , JUMP ON FCBIT TO F1
4C9: 8E7C 9B04 0004 F000
                  (2611)          RR       RX => RX JAMF JUMP ON GE TO FCS5
4CA: 0200 0800 0004 6472
                  (2612) * FLOATING POINT EXCEPTION VECTOR.  ENTRY POINTS ARE:
                  (2613) *        FLEX     OVERFLOW
                  (2614) *        FLEX1    STORE EXCEPTION
                  (2615) *        FLEX2    DIVIDE BY ZERO
                  (2616) *        FLEX9    INT EXCEPTION
                  (2617) *
                  (2618) * REGISTER 11 IS USED TO SHOW THE TYPE OF EXCEPTION:
                  (2619) *        $100     OVERFLOW
                  (2620) *        $101     DIVIDE BY ZERO
                  (2621) *        $102     STORE EXCEPTION   REGISTER 12 = EFFECTIVE ADDRESS
                  (2622) *        $103     INT EXCEPTION
                  (2623) *
                  (2624) * IF 74 IS ZERO, THE VECTOR ABORTS, EXECUTING THE NEXT
                  (2625) * SEQUENTIAL INSTRUCTION WITH THE CBIT SET.
                  (2626) * . THE ABSOLUTE MAPPED VECTOR HANDLING
                  (2627) * IS IDENTICAL TO THAT OF THE UII.
                  (2628) *
                  (2629) *
                  (2630) FLEX9    RR       RCM = $FFFC C= BD01 TR= ALL => 11
4CB: F300 9A34 0003 FFFC
                  (2631)          RR       ,,,GO TO FLEX4
4CC: 0200 0604 0004 04D2
                  (2632) FLEX8    RR       ,,JAMF JUMP ON FCBIT TO FLEX
4CD: 0200 0600 0004 F4CE
                  (2633) FLEX     ALU      CON  MINUS1 => 11  C= BD01 , GO TO FLEX4
4CE: 82CC 9B34 0004 04D2
                  (2634) FLEX1    RR       RCM = $FFFD  => 11  C= BD01
4CF: F200 9A34 0003 FEFD
                  (2635)          RR       RY => 12 , GO TO FLEX4
4D0: EA00 AB04 0004 04D2
                  (2636) FLEX2    RR       RCM = $FFFE => 11 C= BD01
4D1: F200 9A34 0003 FEFE
                  (2637) FLEX4    ALU      11 XOR RCM = $FEFF => 11
4D2: 926C 9A04 0001 FDFF
                  (2638) FLEX5    RR       RCM = '74 => RY
4D3: F200 0804 0002 013C
                  (2639)          CPU      AL    0     ALL    DEC     0      M      EAS ;
                  (2640)                   RMRFMRDY    AREAD ,EAC CLRNVKEYS TRIGNVKEYS
4D4: 83F0 A5C4 000A 0400
                  (2641)          CPU      AL    RM    NX     BB      L      0      0 ;
                  (2642)                   RY240       NOP    SETCC
4D5: 8E5C 0307 0000 0000
                  (2643)          CPU      AL    RY    NX     INC     0      M      RP ;
                  (2644)                   RM280       NOP    JAMF ;
                  (2645)                   JUMP ON NE TO JST1
4D6: 8A00 7400 0004 4150
                  (2646) *
                  (2647) * DOUBLE PRECISION  EXECUTION SPACE.  FOR EASE OF DEBUG,
                  (2648) * EXECUTION BEGINS AT $500.
                  (2649) *
                  (2650)          ORG      $500
                  (2651) DFLD1    ALU      INC RY => RY
500: 8A65 1304 0000 0000
                  (2652)          CPU      BB    RM    ALL    0       0      M      FLTH ;
                  (2653)                   RMRFMRDY    MREAD
501: EF00 4584 0000 0000
```

```
                  (2654)         ALU     INC RY => RY
502: 8A65 1304 0000 0000
                  (2655)         CPU     BB   RM   ALL   0      0      M      FLTL ;
                  (2656)                 RMRFMRDY  MREAD
503: EF00 5584 0000 0000
                  (2657)         ALU     INC RY => RY
504: 8A65 1304 0000 0000
                  (2658)         CPU     BB   RM   ALL   0      0      M      RB ;
                  (2659)                 RMRFMRDY  MREAD
505: EF00 2584 0000 0000
                  (2660)         RR      RM => VSC ,  GO TO F1
506: EE00 6804 0004 0000
                  (2661) DFST1   ALU INC RY => RY
507: 8A65 1304 0000 0000
                  (2662)         CPU     RF,,ALL,,,M,FLTL,RM280,MWRITE
508: 0300 5484 0000 0000
                  (2663)         ALU     INC RY => RY
509: 8A65 1304 0000 0000
                  (2664)         CPU     RF,,ALL,,,M,RB,RM280,MWRITE
50A: 0300 2484 0000 0000
                  (2665)         ALU     INC RY => RY
50B: 8A65 1304 0000 0000
                  (2666)         CPU     RF,,ALL,,,M,VSC,RM280,MWRITE,JAMF
50C: 0300 64B0 0000 0000
                  (2667) DFAD1   CPU     BB   RCM  NONE  0      0      0      0 ;
.                 (2668)                 280  ,PUSHBD  DATA DFAD4
50D: F000 060C 0002 0B0F
                  (2669)         ALU     INC RY => RY ,GO TO DGET
50E: 8A65 1304 0004 051F
                  (2670) DFAD4   CPU     BB   RCM  NONE  0      0      0      0 ;
                  (2671)                 280  ,PUSHBD  DATA DFAD6
50F: F000 060C 0002 0B11
                  (2672)         ALU     VSC MINUS RM => RM SETCC C= AOVFL ;
                  (2673)                 GO TO DFIX
510: 8E94 6477 0004 0524
                  (2674) DFAD6   ALU     RB PLUS RY => RB C= COUT
511: 8A60 2A24 0000 0000
                  (2675)         RR      12 => RM
512: 0200 A204 0000 0000
                  (2676)         ALU     FLTL PLUS RM + C'  => FLTL  C= COUT  TR= ALL
513: 8F68 5A24 0000 0000
                  (2677)         RR      13 => RM
514: 0200 B204 0000 0000
                  (2678)         ALU     FLTH PLUS RM + C  => FLTH  C= AOVFL ;
                  (2679)                 SETCC  GO TO DNRML
515: 8E68 4B77 0004 053A
                  (2680) DFSB1   CPU     BB   RCM  NONE  0      0      0      0 ;
                  (2681)                 280  ,PUSHBD  DATA DFSB4
516: F000 060C 0002 0B18
                  (2682)         ALU     INC RY => RY ,GO TO DGET
517: 8A65 1304 0004 051F
                  (2683) DFSB4   CPU     BB   RCM  NONE  0      0      0      0 ;
                  (2684)                 280  ,PUSHBD  DATA DFSB6
518: F000 060C 0002 0A1A
                  (2685)         ALU     VSC MINUS RM => RM C= AOVFL ;
                  (2686)                 SETCC     GO TO DFIX
519: 8E94 6477 0004 0524
                  (2687) DFSB6   ALU     RB MINUS RY => RB C= COUT  TR= ALL
51A: 8B94 2A24 0000 0000
                  (2688)         RR      12 => RM
51B: 0200 A204 0000 0000
                  (2689)         ALU     FLTL MINUS RM + C  => FLTL  C= COUT  TR= ALL
51C: 8F98 5A24 0000 0000
                  (2690)         RR      13 => RM
51D: 0200 B204 0000 0000
                  (2691)         ALU     FLTH MINUS RM + C  => FLTH  C= AOVFL ;
                  (2692)                 SETCC  GO TO DNRML
51E: 8E98 4B77 0004 053A
                  (2693) *
                  (2694) *
                  (2695) *  SUBROUTINE DLOAD.  ROUTINE LOADS THE FULL 4 WORD
                  (2696) * FLOATING POINT ARGUMENT INTO 13,12,11,AND RM, RESPECTIVLY.
                  (2697) * THE SUBROUTINE EXPECTS THE FIRST WORD TO BE IN RM.  RY IS
                  (2698) * ASSUMED TO CONTAIN A POINTER TO THE SECOND WORD IN MEMORY.
                  (2699) * NOTE THAT THE EXPONENT IS LEFT IN RM.
                  (2700) *
                  (2701) *
                  (2702) DGET    CPU     BB   RM   ALL   0      0      M      13 ;
                  (2703)                 RMRFMRDY  MREAD
51F: EF00 6584 0000 0000
                  (2704)         ALU     INC RY => RY
520: 8A65 1304 0000 0000
                  (2705)         CPU     BB   RM   ALL   0      0      M      12 ;
                  (2706)                 RMRFMRDY  MREAD
521: EF00 A584 0000 0000
                  (2707)         ALU     INC RY => RY
522: 8A65 1304 0000 0000
                  (2708)         CPU     BB   RM   ALL   0      0      M      11 ;
                  (2709)                 RMRFMRDY  MREAD ,S ON TRUE, POP
523: EF00 9584 000C 0004
                  (2710) *
                  (2711) *
                  (2712) * ADJUST SUBROUTINE.  CALLED DFIX, THIS ROUTINE ADJUSTS TWO
                  (2713) * NUMBERS OF UNEQUAL EXPONENTS BY 'UN-NORMALIZING' THE
                  (2714) * NUMBER WITH SMALLER EXPONENT.  IF THE TWO NUMBERS ARE TOO
                  (2715) * FAR APART, AND THE NEW ARGUMENT IS SMALLER, THE ROUTINE
                  (2716) * ABORTS, RETURNING TO THE FETCH CYCLE.  IF THE OLD ARGUMENT
                  (2717) * IS THE SMALLER,  THEN IT IS ZEROED,AND THE EXPONENT (VSC)
                  (2718) * IS MADE EQUAL TO THAT OF THE NEW ARGUMENT.
                  (2719) *
```

```
                (2720) * IF THE TWO NUMBERS CAN BE COMBINED, VSC IS SET TO THE
                (2721) * LARGER EXPONENT, AND THE SMALLER NUMBER IS SHIFTED RIGHT.
                (2722) *
                (2723) * RY IS ALWAYS LOADED WITH THE LSW OF THE NEW ARGUMENT.
                (2724) *
                (2725) DFIX    ALU       ZERO MINUS RM => RM LOADRSC  TR= ALL ;
                (2726)                   JUMP ON FCBIT TO DFIX1
524: 8F95 140E 0004 F52F
                (2727)         RR        11 => RY TR= ALL ,S ON EQ TO POP
525: 0300 9304 000E 4004
                (2728)         RR        RCM = 48 => YSAVE
526: F200 CA04 0002 0130
                (2729) DFIX3   ALU       INC 13 + 0 => NULL SETCC ;
                (2730)                   JUMP ON LT TO DFIX2
527: 8200 6607 0006 6532
                (2731) * NEW ARGUMENT IS SMALLER, SHIFT IT RIGHT OR ABORT.
                (2732) *
                (2733)         ALU       INC FLTH + 0 => NULL SETCC
528: 8200 4007 0000 0000
                (2734)         ALU       YSAVE PLUS RM => NULL SETCC  TR= ALL ;
                (2735)                   JUMP ON EQ TO DFIX4-1
529: 8F60 C607 0006 4536
                (2736)         ALU       ZERO PLUS RM => NULL C= AOVFL ;
                (2737)                   LOADRSC  JUMP ON LE TO DNRML
52A: 8E61 167E 0004 753A
                (2738) *
                (2739) * DE-NORMALIZE CAN BE DONE.
                (2740) *
                (2741)         CPU       RFRS 3    ALL    6      0      M      13 ;
                (2742)                   RF200     LINK
52B: 4F60 8A44 0000 0000
                (2743)         CPU       RFRS 1    NX     6      0      M      12 ;
                (2744)                   RF200     LINK
52C: 4660 AA44 0000 0000
                (2745)         CPU       RFRS 1    NX     6      0      M      11 ;
                (2746)                   RF240     NOP   INCRSC ;
                (2747)                   JUMP ON RSCNEM1 TO *-2
52D: 4660 9805 0004 952B
                (2748)         RR        11 => RY , S ON TRUE, POP
52E: 0200 9304 000C 0004
                (2749) * IF EXPONENT OVERFLOW IS DETECTED HERE, ONE OF THE
                (2750) * TWO NUMBERS WILL BE PRESERVED UNCHANGED BECAUSE THE
                (2751) * EXPONENTS ARE TOO FAR APART FOR AN ADD OR SUBTRACT.
                (2752) * THEREFORE, FORCE THE SIGN THE RIGHT WAY AND ALSO FORCE
                (2753) * THE MAXIMUN EXPONENT DIFFERENCE ALLOWED TO BE ZERO
                (2754) * TO ALLOW THE REST OF THE ADJUST ROUTINE TO ACT CORRECTLY.
                (2755) *
                (2756) *
                (2757) DFIX1   RR        RCM = $8000 => YSAVE
52F: F200 CA04 0001 0100
                (2758)         ALU       YSAVE MINUS RM => NULL SETCC
530: 8E94 C007 0000 0000
                (2759)         RR        11 => RY  , GO TO DFIX3
531: 0200 9304 0004 0527
                (2760) *
                (2761) *
                (2762) * TEST FOR VALID ALIGN.  IF VALID, UN-NORMALIZE THE
                (2763) * FACC.  IF NOT VALID, ZERO THE FACC AND RETURN.
                (2764) * IN BOTH CASES, ADD THE EXPONENT DIFFERENCE TO THE
                (2765) * VSC.
                (2766) *
                (2767) *
                (2768) DFIX2   ALU       YSAVE MINUS RM => NULL SETCC TR= ALL ;
                (2769)                   S ON EQ, POP
532: 8F94 C607 000E 4004
                (2770)         CPU       RFRS 3    ALL    6      0      M      FLTH ;
                (2771)                   RF240     LINK  NOP ;
                (2772)                   JUMP ON LE TO DFIX4
533: 4F60 4844 0004 7537
                (2773)         CPU       RFRS 1    NX     6      0      M      FLTL ;
                (2774)                   RF200     LINK
534: 4660 5A44 0000 0000
                (2775)         CPU       RFRS 1    NX     6      0      M      R8 ;
                (2776)                   RF240     NOP   INCRSC ;
                (2777)                   JUMP ON RSCNEM1 TO *-2
535: 4660 2805 0004 9533
                (2778)         ALU       VSC PLUS RM => VSC ,S ON TRUE, POP
536: 8E60 6804 000C 0004
                (2779) * ZERO OUT THE FACC
                (2780) *
                (2781) DFIX4   ALU       CON 0 => FLTH
537: 823C 4A04 0000 0000
                (2782)         ALU       CON 0 => FLTL
538: 823C 5A04 0000 0000
                (2783)         ALU       CON 0 => R8 , GO TO DFIX4-1
539: 823C 2804 0004 0536
                (2784) *
                (2785) *
                (2786) * NORMALIZE SUBROUTINE.  THIS DOUBLE PRECSION NORMALIZE
                (2787) * ROUTINE OPERATES EXACTLY AS THE SINGLE PRECISION ROUTINE
                (2788) * DOES.  IF THE CBIT IS SET COMMING INTO THE ROUTINE, THE
                (2789) * VALUE IS TAKEN TO BE OVERSHIFTED, AND THE NUMBER IS
                (2790) * RIGHT SHIFTED ONE PLACE.  THE SIGN BIT IS RECONSTRUCTED
                (2791) * FROM THE CONDITION CODE AT ENTRY.  THE CODE IS ASSUMED
                (2792) * TO BE THE OPPOSITE SIGN OF THE PROPPER NUMBER.
                (2793) *
                (2794) * IF THE NUMBER IS ALREADY NORMALIZED, A RETURN TO FETCH IS
                (2795) * DONE.  OTHERWISE, THE FACC IS SHIFTED LEFT UNTIL
                (2796) * THE NUMBER IS NORMALIZED, THE EXPONENT IS REDUCED
                (2797) * APPROPRIATELY, AND A RETURN TO FETCH IS DONE.
```

```
                          *
                          *
                          DNRML  RR      RCM = -48 => NULL LOADRSC
53A: F200 000E 0003 FED0
                                 CPU     RFLS 3    ALL    7       0      M      FLTH ;
                                         RY240  SOVFL   , JUMP ON FCBIT TO DNRM1
53B: 2F70 4364 0004 F540
                                 CPU     RFLS 7    NX     7       0      M      RB ;
                                         RF240     LINK   , JUMP ON FCBIT TO DNRM2
53C: 3E70 2B44 0004 F545
                                 CPU     RFLS 3    ALL    7       0      M      FLTL ;
                                         RF200     LINK
53D: 2F70 5A44 0000 0000
                                 CPU     RFLS 3    NX     3       0      M      FLTH ;
                                         RF240     NOP    INCRSC ;
                                         JUMP ON RSCNEM1 TO *-3
53E: 2E30 4805 0004 953B
                                 ALU     CON 0 => VSC C= B001 ,GO TO F1
53F: 823C 6834 0004 0000
                          DNRM1  ALU     VSC PLUS RCM = 1  => VSC  C= AOVFL
540: 9260 6A74 0002 0001
                                 CPU     RFRS 5    ALL    6      0      M      FLTH ;
                                         RF240     LINK   NOP ;
                                         JUMP ON LT TO *+2
541: 5760 4B44 0006 6543
                                 ALU     FLTH OR RCM = $8000 => FLTH
542: 924C 4A04 0001 0100
                                 CPU     RFRS 1    ALL    6       0      M      FLTL ;
                                         RF200     LINK
543: 4760 5A44 0000 0000
                                 CPU     RFRS 1    NX     6       0      M      RB ;
                                         RF240     NOP    JAMF ;
                                         JUMP ON FCBIT TO DFLEX
544: 4660 2B00 0004 F5C8
                          DNRM2  RR      RCM = $FFC0 => 11
545: F200 9A04 0003 FFC0
                                 CPU     RFRS 1    ALL    6       0      M      RB ;
                                         RF200     NOP    NOP
546: 4760 2A04 0000 0000
                                 ALU     11 OR RSC => 11
547: 864C 9A04 0000 0000
                                 ALU     11 PLUS RCM = 48  => RY
548: 9260 9304 0002 0130
                                 ALU     VSC MINUS RY => VSC C= AOVFL
549: 8A94 6A74 0000 0000
                                 RR      ,,JAMF JUMP ON FCBIT TO DFLEX
54A: 0200 0600 0004 F5C8
                          *
                          *
                          * DOUBLE PRECISION FLOATING POINT MULTIPLY.  TECHNIQUE USED IS
                          * A SIMPLE EXTENSION OF THE SINGLE PRECISION MULTIPLY.
                          * THE EXPONENT IS UPDATED FIRST, THEN 16 BITS OF SINGLE
                          * PRECISION MULTIPLY, THEN 16 BITS OF DOUBLE PRECISION,
                          * AND FINALLY 15 BITS OF TRIPPLE PRECISION SHIFT AND/OR
                          * MULTIPLY.  FINALLY, A TRIPLE PRECISION SUBTRACT
                          * IS DONE, IF NECESSARY, FOLLOWED BY NORMALIZE AS NEEDED
                          *
                          *
                          DFMP1  CPU     BB   RCM  NONE  0       0      0      0 ;
                                         280  NOP  PUSHBD  DATA DFMP4
54B: F000 060C 0002 054D
                                 ALU     INC RY => RY , GO TO DGET
54C: 8A65 1304 0004 051F
                          *
                          * FIRST, FIX THE EXPONENT
                          *
                          DFMP4  ALU     VSC MINUS RCM = 128 => VSC  C= AOVFL
54D: 9294 6A74 0002 0080
                                 ALU     VSC PLUS RM => VSC C= AOVFL , ;
                                         JUMP ON FCBIT TO DFM1
54E: 8E60 6874 0004 F55C
                          DFM2   RR      FLTL => RY , JUMP ON FCBIT TO DFLEX
54F: 0200 5304 0004 F5C8
                                 RR      FLTH => RM
550: 0200 4204 0000 0000
                                 CPU     AL   RCM  ALL   ZERO   L      M      FLTH ;
                                         RF200     ,LOADRSC DATA -16
551: 933C 4A0E 0003 FFF0
                                 CPU     RFRS 5    NX     6       0      M      11 ;
                                         RF200     LINK
552: 5660 9A44 0000 0000
                                 CPU     ALRS RM   NX     ADD     0      M      FLTH ;
                                         RF240     NOP , EAC MPYLOGIC
553: CE60 4804 0008 C000
                                 CPU     RFRS 5    ALL    6       0      M      11 ;
                                         RF240     LINKS INCRSC ;
                                         JUMP ON RSCNEM1 TO *-1
554: 5760 9B15 0004 9553
                          *
                          * SINGLE PRECISION PART DONE,  DO DOUBLE PRECISION.
                          *
                                 CPU     AL   RCM  ALL   ZERO   L      M      FLTL ;
                                         RF200     NOP   LOADRSC  DATA -16
555: 933C 5A0E 0003 FFF0
                                 CPU     RFRS 5    NX     6       0      M      12 ;
                                         RF280     LINKS NOP ;
                                         JUMP ON NOTRF16 TO DFM3
556: 5660 AC14 0007 7556
                                 ALU     FLTL PLUS RY => FLTL C= COUT
557: 8A60 5A24 0000 0000
```

```
                 (2868)         CPU      ALRS RM    ALL    ADD     C       M      FLTH ;
                 (2869)                  RF240      LINK
558: CF68 4844 0000 0000
                 (2870) DFM5    CPU      RFRS 1     NX     6       0       M      FLTL ;
                 (2871)                  RF240      NOP    INCRSC ;
                 (2872)                  JUMP ON RSCNEM1 TO *-3
559: 4660 5805 0004 9556
                 (2873)         RR       RY => 11 , GO TO DFM4
55A: EA00 9804 0004 055E
                 (2874) *
                 (2875) * SHIFT PORTION OF DOUBLE PRECISION PART
                 (2876) *
                 (2877) DFM3    CPU      RFRS 3     ALL    6       0       M      FLTH ;
                 (2878)                  RF240      LINK   NOP     GO TO DFM5
55B: 4F60 4844 0004 0559
                 (2879) *
                 (2880) * EXPONENT OVERFLOW TESTS
                 (2881) *
                 (2882) DFM1    ALU      CON 0 => NULL C= BD01 , JUMP ON FCBIT TO DFM2
55C: 823C 0634 0004 F54F
                 (2883)         RR       ,,,GO TO DFLEX
55D: 0200 0604 0004 05C8
                 (2884) *
                 (2885) * FINALLY DO TRIPLE PRECISION MULTIPLY
                 (2886) *
                 (2887) DFM4    RR       RCM = -15 => NULL LOADRSC
55E: F200 000E 0003 FEF1
                 (2888) DFM7    CPU      RFRS 5     NX     6       0       M   .  13 ;
                 (2889)                  RF280      LINKS NOP ;
                 (2890)                  JUMP ON NOTRF16 TO DFM8
55F: 5660 8C14 0007 756F
                 (2891)         RR       RB => RY
560: 0200 2804 0000 0000
                 (2892)         ALU      12 PLUS RY => 12 C= COUT
561: 8A60 AA24 0000 0000
                 (2893)         RR       11 => RY
562: 0200 9804 0000 0000
                 (2894)         ALU      FLTL PLUS RY + C  => FLTL C= COUT
563: 8A68 5A24 0000 0000
                 (2895)         CPU      ALRS RM    ALL    ADD     C       M      FLTH ;
                 (2896)                  RF240      LINK
564: CF68 4844 0000 0000
                 (2897) DFM6    CPU      RFRS 1     ALL    6       0       M      FLTL ;
                 (2898)                  RF200      LINK
565: 4760 5A44 0000 0000
                 (2899)         CPU      RFRS       1      NX      6       0      M 12 ;
                 (2900)                  RF240      NOP    INCRSC ;
                 (2901)                  JUMP ON RSCNEM1 TO DFM7
566: 4660 A805 0004 955F
                 (2903) * NOW DO FINAL SUBTRACT
                 (2904) *
                 (2905)         CPU      RF   0     ALL    0       0       M      13 ;
                 (2906)                  RF280      BD01  NOP ;
                 (2907)                  JUMP ON NOTRF16 TO DFM9
567: 0300 8C34 0007 756D
                 (2908)         RR       RB => RY
568: 0200 2804 0000 0000
                 (2909)         ALU      12 MINUS RY => 12 C= COUT
569: 8A94 AA24 0000 0000
                 (2910)         RR       11 => RY
56A: 0200 9804 0000 0000
                 (2911)         ALU      FLTL MINUS RY + C  => FLTL  C= COUT
56B: 8A98 5A24 0000 0000
                 (2912)         ALU      FLTH MINUS RM + C => FLTH C= AOVFL  SETCC
56C: 8E98 4A77 0000 0000
                 (2913) DFM9    RR       12 => RM
56D: 0200 A204 0000 0000
                 (2914)         RR       RM => RB , GO TO DNRML
56E: EE00 2804 0004 053A
                 (2915) *
                 (2916) * SHIFT PART OF TRIPLE PRECISION MPY LOOP.
                 (2917) *
                 (2918) DFM8    CPU      RFRS 3     ALL    6       0       M      FLTH ;
                 (2919)                  RF240      LINK   NOP GO TO DFM6
56F: 4F60 4844 0004 0565
                 (2920) *
                 (2921) *
                 (2922) * DIVIDF -- DOUBLE PRECISION FLOATING POINT
                 (2923) *
                 (2924) DFDV1   CPU      BB   RCM  0      0       0       0      0 ;
                 (2925)                  280  NOP  PUSHBD   DATA DDV4
570: F000 060C 0002 0AC7
                 (2926)         ALU      INC RY => RY ,GO TO DGET
571: 8A65 1304 0004 051F
                 (2927) * DIVIDE BY ZERO TEST
                 (2928) DDV30   ALU      FLTH XOR RY => NULL SETCC ;
                 (2929)                  JUMP ON EQ TO DFLEX1
572: 8A6C 4607 0006 45CA
                 (2930)         ALU      VSC PLUS RCM = 129 => VSC C= AOVFL ;
                 (2931)                  CLEARFUII
573: 9260 6A7F 0002 0181
                 (2932)         RR       RX => RX NOP JUMP ON GE TO *+2
574: 0200 0804 0004 6576
                 (2933) * SET FUII FOR NEGATIVE RESULTS
                 (2934)         RR       ,,,EAC SETFUII
575: 0200 0004 000A 8000
                 (2935) * EXPONENT CALCULATION
                 (2936)         ALU      VSC MINUS RM => VSC ,C= AOVFL TR= ALL ;
                 (2937)                  JUMP ON FCBIT TO DDV5
```

```
576: 8F94 6B74 0004 F5A5
                     (2938) * EXPONENTS DONE EXCEPT FOR OVERFLOW
                     (2939) * TEST NOW TO SEE IF EITHER OF THE TWO ARGUMENTS
                     (2940) * IS NEGATIVE.  IF IT IS, THEN COMPLEMENT IT
                     (2941) *
                     (2942) DDV6   ALU     INC 13 + 0 => RM  SETCC ;
                     (2943)                JUMP ON FCBIT TO DFLEX
577: 8200 8407 0004 F5C8
                     (2944)        ALU     INC FLTH + 0 => NULL SETCC ;
                     (2945)                JUMP ON GE TO DDV7
578: 8200 4607 0004 6570
                     (2946) * COMPLEMENT NEW ARGUMENT
                     (2947) *
                     (2948)        ALU     NOT 11 => 11
579: 82FC 9A04 0000 0000
                     (2949)        ALU     NOT 13 => RM
57A: 82FC 8204 0000 0000
                     (2950)        ALU     INC 11 => 11 C= COUT
57B: 8204 9A24 0000 0000
                     (2951)        ALU     NOT 12 => 12 , TR= ALL ;
                     (2952)                JUMP ON FCBIT TO DDV8
57C: 83FC A804 0004 F5A7
                     (2953) * NOW TEST NEW ARGUMENT
                     (2954) *
                     (2955) DDV7   RR      12 => RY ,JUMP ON GE TO DDV11 TR= ALL
57D: 0300 A304 0004 6582
                     (2956) * COMPLEMEMT ACCUMULATOR
                     (2957) *
                     (2958)        ALU     NOT RB => RB
57E: 82FC 2A04 0000 0000
                     (2959)        ALU     NOT FLTL => FLTL
57F: 82FC 5A04 0000 0000
                     (2960)        ALU     INC RB => RB C= COUT TR= ALL
580: 8304 2A24 0000 0000
                     (2961)        ALU     NOT FLTH => FLTH , JUMP ON FCBIT TO DDV10
581: 82FC 4804 0004 F5A8
                     (2962) * NOW SET UP AND EXECUTE 16 ITTERATIONS OF
                     (2963) * THE TRIPPLE PRECISION DIVIDE LOOP.
                     (2964) *
                     (2965) DDV11  ALU     FLTH MINUS RM => NULL SETCC TR= ALL
582: 8F94 4007 0000 0000
                     (2966)        RR      RCM = -17 => NULL LOADRSC
583: F200 000E 0003 FEEF
                     (2967) * RM=NH/RY,12=NM/11=NL/13=POSITIVE
                     (2968) *
                     (2969) *
                     (2970) * COMPARE PART OF NON-PERFORMING DIVIDE
                     (2971) *
                     (2972) DDV15  CPU     RFLS 3     ALL    3      0      M     13 ;
                     (2973)                RF240      NOP    INCRSC ;
                     (2974)                JUMP ON LT TO DDV13
584: 2F30 8805 0006 6581
                     (2975)        ALU     FLTL MINUS RY => NULL SETCC ;
                     (2976)                C= COUT JUMP ON GT TO DDV14
585: 8A94 5627 0006 75C6
                     (2977)        RR      11 => RY , JUMP ON FCBIT TO *+2
586: 0200 9304 0004 F588
                     (2978)        RR      12 => RY , GO TO DDV13
587: 0200 A304 0004 0581
                     (2979)        ALU     RB MINUS RY => NULL C= COUT  , ;
                     (2980)                JUMP ON NE TO DDV12
588: 8A94 2624 0004 458B
                     (2981)        RR      RX => RX , JUMP ON FCBIT TO DDV12
589: 0200 0804 0004 F58B
                     (2982)        RR      12 => RY , GO TO DDV13
58A: 0200 A304 0004 0581
                     (2983) *END OF DIVIDE TEST.
                     (2984) *
                     (2985) * SUBTR ACT -- SHIFT
                     (2986) *
                     (2987) DDV12  ALU     RB MINUS RY => RB C= COUT
58B: 8A94 2A24 0000 0000
                     (2988)        CPU     RF    0    ALL    ZERO   L      M     12 ;
                     (2989)                RY200      NOP    SETCC
58C: 033C A807 0000 0000
                     (2990)        ALU     FLTL MINUS RY + C => FLTL C= COUT
58D: 8A98 5A24 0000 0000
                     (2991)        CPU     RFLS 7     ALL    7      0      M     RB ;
                     (2992)                RF200      LINK
58E: 3F70 2A44 0000 0000
                     (2993)        CPU     RFLS 3     NX     7      0      M     FLTL ;
                     (2994)                RF200      LINK
58F: 2E70 5A44 0000 0000
                     (2995)        CPU     ALLS RM    ALL    SUB    C      M     FLTH ;
                     (2996)                RF240      LINK
590: AF98 4844 0000 0000
                     (2997) *
                     (2998) * START OF NEXT ITTERATION
                     (2999) *
                     (3000) DDV9   ALU     FLTH MINUS RM => NULL SETCC ;
                     (3001)                JUMP ON RSCNEM1 TO DDV15
591: 8E94 4607 0004 9584
                     (3002) * DOUBLE PERECISION DIVIDE.  SIMILAR TO THAT IN FDV.
                     (3003) *
                     (3004) * ON ENTRY: RM=NH,RY=NM,13=HIGH ANSWER,LINK=16TH QUOTIENT BIT.
                     (3005) *
                     (3006)        RR      RCM = -17 => NULL LOADRSC
592: F200 000E 0003 FEEF
                     (3007) *
                     (3008) * HIGH ORDER CONDITION CODE ALREADY TESTED UPON ENTRY
```

```
                         (3009) *
                         (3010) DDV23  CPU     RFLS 3    NX     7       0      M      12 ;
                         (3011)                RF240     NOP    INCRSC ;
                         (3012)                JUMP ON LT TO DDV21
593: 2E70 A805 0006 6596
                         (3013)        ALU     FLTL MINUS RY => NULL C= COUT , ;
                         (3014)                JUMP ON GT TO DDV24
594: 8A94 5624 0006 7598
                         (3015)        RR      RX => RX , JUMP ON FCBIT TO DDV24
595: 0200 0804 0004 F596
                         (3016) * DOUBLE WORD SHIFT
                         (3017) DDV21  CPU     RFLS 7    NX     7       0      M      FLTL ;
                         (3018)                RF200     LINK
596: 3E70 5A44 0000 0000
                         (3019)        CPU     RFLS 3    ALL    7       0      M      FLTH ;
                         (3020)                RF240     LINK   NOP ;
                         (3021)                GO TO DDV20
597: 2F70 4844 0004 0596
                         (3022) DDV24  ALU     FLTL MINUS RY => FLTL C= COUT
598: 8A94 5A24 0000 0000
                         (3023)        CPU     RFLS 7    ALL    7       0      M      FLTL ;
                         (3024)                RF200     LINK
599: 3F70 5A44 0000 0000
                         (3025)        CPU     ALLS RM   ALL    SUB     C      M      FLTH ;
                         (3026)                RF240     LINK
59A: AF98 4844 0000 0000
                         (3027) DDV20  ALU     FLTH MINUS RM => NULL SETCC ;
                         (3028)                JUMP ON RSCNEM1 TO DDV23
59B: 8E94 4607 0004 9593
                         (3029) *
                         (3030) * FINISHED DOUBLE PART. NOW DO SINGLE PART.
                         (3031) *
                         (3032) *
                         (3033) * LINK CONTAINS 32ND QUOTIENT BIT.  RM=NH,13=Q1,12=Q2
                         (3034) * BUILD Q3 IN RB. PUT 13 IN RY, INITIALIZE AND DIVIDE FLTH/RM
                         (3035) *
                         (3036)        CPU     RF   RCM  ALL    ZERO    L      M      RB ;
                         (3037)                RF200     NOP    LOADRSC ;
                         (3038)                DATA -15
59C: 133C 2A0E 0003 FEF1
                         (3039)        CPU     RFLS 3    NX     7       0      M      RB ;
                         (3040)                RF200     LINK
59D: 2E70 2A44 0000 0000
                         (3041)        CPU     AL   RCM  ALL    AND     L      M      13 ;
                         (3042)                RY240     NOP    SETCC DATA $7FFF
59E: 931C 8307 0000 FFFF
                         (3043) * SINGLE PRECISION DIVIDE LOOP
                         (3044) *
                         (3045)        CPU     ALLS RM   ALL    SUB     1      M      FLTH ;
                         (3046)                RF280     LINK ,EAC DIVLOGIC
59F: AF94 4C44 0009 0000
                         (3047)        CPU     RFLS 3    NX     7       0      M      RB ;
                         (3048)                RF240     LINK   INCRSC ;
                         (3049)                JUMP ON RSCNEM1 TO *-1
5A0: 2E70 2845 0004 959F
                         (3050) * DONE WITH DIVIDE
                         (3051) *
                         (3052) * RESULTS IN RY,12,RB
                         (3053) *
                         (3054)        RR      RY => FLTH C= BD01
5A1: EA00 4A34 0000 0000
                         (3055)        RR      12 => RM  TR= ALL  ,JUMP ON FUII TO *+2
5A2: 0300 A404 0005 55A4
                         (3056)        RR      RM => FLTL ,GO TO DNRML
5A3: EE00 5804 0004 053A
                         (3057)        ALU     NOT RM => FLTL ,GO TO DFCM+1
5A4: 8EAC 5804 0004 0435
                         (3058) *
                         (3059) * ASSORTED DIVIDE NON-STRAIGHT LINE CODE
                         (3060) *
                         (3061) DDV5   ALU     CON 0 => NULL C= BD01 , ;
                         (3062)                JUMP ON FCBIT TO DDV6
5A5: 823C 0634 0004 F577
                         (3063)        RR      ,,,GO TO DFLEX
5A6: 0200 0604 0004 05C8
                         (3064) DDV8   ALU     INC 12 + C => 12 C= COUT
5A7: 8208 AA24 0000 0000
                         (3065)        ALU     INC RM + C => RM SETCC
5A8: 8E69 1207 0000 0000
                         (3066)        ALU     INC FLTH + 0 => NULL SETCC ;
                         (3067)                JUMP ON GE TO DDV7
5A9: 8200 4607 0004 6570
                         (3068)        CPU     AL   RCM  NX     DEC     0      M      VSC ;
                         (3069)                RF240     AOVFL , GO TO DFCM
5AA: 92F0 6874 0004 0434
                         (3070) *
                         (3071) * FINISH COMPLEMENT
                         (3072) *
                         (3073) DDV10  ALU     INC FLTL + C => FLTL C= COUT
5AB: 8208 5A24 0000 0000
                         (3074)        ALU     INC FLTH + C => FLTH SETCC
5AC: 8208 4A07 0000 0000
                         (3075)        ALU     FLTH MINUS RM => NULL SETCC  TR= ALL ;
                         (3076)                JUMP ON GE DDV11+1
5AD: 8F94 4607 0004 6583
                         (3077)        ALU     VSC PLUS RCM = 1 => VSC C= AOVFL
5AE: 9260 6A74 0002 0001
                         (3078)        CPU     RFRS 5    ALL    6       0      M      FLTH ;
                         (3079)                RF240     NOP    NOP ;
                         (3080)                JUMP ON FCBIT TO DFLEX
```

```
5AF: 5760 4804 0004 F5C8
                  (3081)         ALU      FLTH MINUS RM => NULL  SETCC GO TO DDV11+1
5B0: 8E94 4607 0004 0583
                  (3082) *
                  (3083) * SHIFT STEP -- TRIPPLE SUBTRACT
                  (3084) *
                  (3085) DDV13  CPU      RFLS 7    ALL   7       0     M      RB ;
                  (3086)                 RF200     LINK
5B1: 3F70 2A44 0000 0000
                  (3087)         CPU      RFLS 3    NX    7       0     M      FLTL ;
                  (3088)                 RF200     LINK
5B2: 2E70 5A44 0000 0000
                  (3089)         CPU      RFLS 3    ALL   7       0     M      FLTH ;
                  (3090)                 RF240     LINK  NOP ;
                  (3091)                 GO TO DDV9
5B3: 2F70 4844 0004 0591
                  (3092) *
                  (3093) * DOUBLE PRECISION COMPARE
                  (3094) *
                  (3095)         ORG      $5B4
                  (3096) DFCS1  CPU      BB   RM   NX    XOR     L     M      FLTH ;
                  (3097)                 200  BD01 SETCC
5B4: EE6C 4037 0000 0000
                  (3098)         ALU      INC RY => RY , JUMP ON LT TO FCS2
5B5: 8A65 1304 0006 6477
                  (3099)         CPU      BB   RM   ALL   BB      L     M      13 ;
                  (3100)                 RMRFMRDY  MREAD SETCC
5B6: EF5C 8587 0000 0000
                  (3101)         RR       RY => 12
5B7: EA00 AA04 0000 0000
                  (3102)         ALU      12 PLUS RCM = 2  => RY
5B8: 9260 A304 0002 0002
                  (3103) * FOR NEG REVERSE EXPONENT TEST
                  (3104)         CPU      BB   RM   ALL   0       0     M      11 ;
                  (3105)                 RMRFMRDY  MREAD  NOP    ;
                  (3106)                 JUMP ON LT TO CS1
5B9: EF00 9584 0006 65C3
                  (3107) *
                  (3108) * EXPONENT TEST
                  (3109) *
                  (3110)         ALU      INC FLTH + 0 => NULL SETCC JUMP ON EQ TO FCS7
5BA: 8200 4607 0006 4479
                  (3111)         ALU      VSC MINUS RM => RM SETCC ;
                  (3112)                 C= AOVFL  JUMP ON EQ TO FCS2+1
5BB: 5E94 6477 0006 4478
                  (3113) * IF OVERFLOW - REVERSE TEST
                  (3114)         RR       13 => RM , JUMP ON FCBIT TO CS3
5BC: 0200 8404 0004 F5C5
                  (3115) *
                  (3116)         ALU      INC 12 => RY , JUMP ON GT TO F1
5BD: 8204 A304 0006 7000
                  (3117) *
                  (3118) * EXPONENTS EQUAL
                  (3119) *
                  (3120) CS4    ALU      FLTH MINUS RM => NULL SETCC ;
                  (3121)                 JUMP ON NE TO FCS2+1
5BE: 8E94 4607 0004 4476
                  (3122)         RR       11 => RM , JUMP ON GT TO F1
5BF: 0200 9404 0006 7000
                  (3123)         ALU      FLTL MINUS RM => NULL SETCC TR= ALL  C= COUT ;
                  (3124)                 JUMP ON LT TO FCS2+1
5C0: 8F94 5627 0006 6478
                  (3125) *
                  (3126) * HIGH ORDER PART EQUAL -- TEST MIDDLE
                  (3127) *
                  (3128)         CPU      ,,ALL,,,,,RMMRDY MREAD NOP JUMP ON NE TO FCS2
5C1: 0300 0784 0004 4477
                  (3129)         ALU      RB MINUS RM => NULL C= COUT SETCC GO TO FCS2-1
5C2: 8E94 2627 0004 0476
                  (3130) * NEGATIVE EXPONENTS, REVERSE TEST
                  (3131) CS1    ALU      VSC MINUS RM => NULL SETCC C= AOVFL
5C3: 6E94 6077 0000 0000
                  (3132) * IF OVERFLOW, DO NOT REVERSE TEST.
                  (3133)         RR       13 => RM , JUMP ON FCBIT TO CAS4-1
5C4: 0200 8404 0004 F1E3
                  (3134) * REVERSE TEST
                  (3135) CS3    ALU      INC 12 => RY JAMF JUMP ON GE TO CS4
5C5: 8204 A300 0004 658E
                  (3136) * PATCH SPACE
                  (3137) DDV14  RR       11 => RY , GO TO DDV12
5C6: 0200 9304 0004 0588
                  (3138) DDV4   ALU      INC 13 + 0 => RY  SETCC  GO TO DDV30
5C7: 8200 6307 0004 0572
                  (3139) *
                  (3140) * DOUBLE PRECISION FLOATING EXCEPTION
                  (3141) *
                  (3142) DFLEX  RR       RCM = $200 => 11
5C8: F200 9A04 0000 0500
                  (3143)         ALU      CON -1 => NULL C= BD01 , GO TO FLEX5
5C9: 82CC 0634 0004 0403
                  (3144) DFLEX1 RR       RCM = $201 => 11
5CA: F200 9A04 0000 0401
                  (3145)         ALU      CON -1 => NULL C= BD01 , GO TO FLEX5
5CB: 82CC 0634 0004 0403
         002714   (3146)         END
```

```
A2A        01AD   A 2017
ACA        01B8   A 2030
ADD        01EF   A 2151
ADD3       0069   A 1516  2151
ADJ8       0444   A 2299  2323
ADJ9       0443   A 2294  2322
ADJUST     043A   A 2294  2384  2391
ALL        01D6   A 2101
ALR        01D9   A 2108
ALS        01D7   A 2102
ANA        01E9   A 2143
AOA        0133   A 1798  2021
ARL        01C2   A 2056
ARR        01C4   A 2062
ARS        01C3   A 2059
AVECT      0077   A 0965  1363  1385  1417  1993
AVECT1     0077   A 1320  1346  1365  1394  1507
BOOT       0091   A 1013  1436
BOOT1      0094   A 1440  1452
CAI        01FD   A 2166
CAL        01B4   A 2025
CAR        01B3   A 2024
CAS        01E0   A 2132
CAS4       01E4   A 2136  2138  2153  3133
CAS5       01E5   A 1370  1711  1853  1878  1968  2007  2014  2127  2138
                    2139  2142  2187  2188  2189  2190  2191  2192  2428
                    2431
CAS6       01F1   A 2135  2153
CAZ        01AB   A 2014
CEA        01B6   A 1970
CEA3       0128   A 1785  1970
CHS        01A0   A 2000
CMA        01AF   A 2020
CP1        0035   A 1009  1046  1115  1369  1846
CP10       0022   A 1011
CP11       0023   A 1012
CP12       0024   A 1013
CP13       0025   A 1019
CP14       0026   A 1026
CP15       0027   A 1019  1031
CP16       0028   A 1033
CP17       002A   A 1037
CP18       002B   A 1039
CP19       002C   A 1012  1041
CP2        0036   A 1116
CP3        0037   A 1037  1039  1041  1117
CP4        0038   A 1120
CP5        0039   A 1122
CP6        003A   A 1124
CP7        003B   A 1127
CP8        0020   A 0996  0998  1007  1127
CP9        0021   A 1009  1011
CPPAR      006A   A 1324
CPPAR3     007E   A 1384  1712
CRA        01CE   A 2004  2085  2588
CRB        019F   A 1999  2010
CREP       013C   A 1809
CRL        019E   A 1996  2250  2253  2269  2579
CS1        05C3   A 3106  3131
CS3        05C5   A 3114  3135
CS4        058E   A 3121  3135
CSA        01AC   A 2015
DBL        0188   A 1960
DDV10      05A8   A 2961  3073
DDV11      0582   A 2955  2965  3076  3081
DDV12      0588   A 2980  2981  2987  3137
DDV13      0581   A 2974  2978  2982  3086
DDV14      05C6   A 2976  3137
DDV15      0584   A 2974  3001
DDV20      059B   A 3021  3028
DDV21      0596   A 3012  3018
DDV23      0593   A 3012  3028
DDV24      0598   A 3014  3015  3022
DDV30      0572   A 2929  3136
DDV4       05C7   A 2925  3138
DDV5       05A5   A 2937  3062
DDV6       0577   A 2943  3062
DDV7       0570   A 2945  2955  3067
DDV8       05A7   A 2952  3064
DDV9       0591   A 3001  3091
DFAD       01F8   A 2161
DFAD1      050D   A 2161  2668
DFAD4      050F   A 2668  2671
DFAD6      0511   A 2671  2674
DFCM       0434   A 2277  3057  3069
DFCS       01FC   A 2165
DFCS1      0584   A 2165  3097
DFDV       01FB   A 2164
DFDV1      0570   A 2164  2925
DFIX       0524   A 2673  2686  2726
DFIX1      052F   A 2726  2757
DFIX2      0532   A 2730  2769
DFIX3      0527   A 2730  2759
DFIX4      0537   A 2735  2772  2781  2783
DFLD       01F6   A 2159
DFLD1      0500   A 2159  2651
DFLEX      05C8   A 2277  2820  2827  2848  2883  2943  3063  3080  3142
DFLEX1     05CA   A 2929  3144
DFM1       055C   A 2847  2882
DFM2       054F   A 2846  2882
```

```
DFM3      055B   A 2866   2878
DFM4      055E   A 2873   2887
DFM5      0559   A 2872   2878
DFM6      0565   A 2898   2919
DFM7      055F   A 2890   2901
DFM8      056F   A 2890   2919
DFM9      056D   A 2907   2913
DFMP      01FA   A 2163
DFMP1     054B   A 2163   2840
DFMP4     054D   A 2840   2845
DFSB      01F9   A 2162
DFSB1     0516   A 2162   2681
DFSB4     0518   A 2681   2684
DFSB6     051A   A 2684   2687
DFST      01F7   A 2160
DFST1     0507   A 2160   2661
DGET      051F   A 2669   2682   2703   2841   2926
DIN       004F   A 1216   1221   1231
DIN1      0053   A 1231   1244   1405
DIV       0144   A 1826
DIV17     009E   A 1458   1479
DIV27     009C   A 1454   1477
DIV4      00A8   A 1479   1828
DIVER     0146   A 1461   1483   1830
DMA       004B   A 1133   1207
DMA1      0047   A 1184   1222   1238
DMC       0048   A 1190   1207
DMC1      005C   A 1190   1268
DMC2      0089   A 1401   1405
DMC3      0086   A 1275   1399
DMT       003E   A 1148   1191
DMX1      0070   A 1340
DMX2      003C   A 1131   1341
DNRM1     0540   A 2802   2811
DNRM2     0545   A 2804   2821
DNRML     053A   A 2280   2283   2679   2692   2737   2800   2914   3056
DOUT      0054   A 1235   1247
DOUT1     0058   A 1213   1228   1247   1403
DRX       01AA   A 2012
E16S      0189   A 1962
E32R      0189   A 1964
E32S      0189   A 1963
E64R      0189   A 1965
EAA       0128   A 1786
EMCM      018D   A 1972
ENB       0190   A 1976
ENTR      0100   A 1722
EPMJ      0120   A 1775
EPMJ6     0116   A 1756
EPMX      0120   A 1772
ERA       01ED   A 2149
ERM       0117   A 1757
ERMJ      0110   A 1746   1777
ERMX      0110   A 1744
ESIM      0196   A 1989
EVIM      0196   A 1990
EVMJ      0110   A 1745
EVMX      0110   A 1743
EVMX1     01F2   A 1753   2154
F1        0000   A 0863   1570   1718   1891   1896   1958   2139   2258   2353
                   2420   2424   2429   2587   2610   2660   2810   3116   3122
F10       0005   A 0892
F11       0006   A 0898
F12       0007   A 0904
F13       0008   A 0892   0907
F14       0009   A 0913
F16       000A   A 0919
F16A      000B   A 0925
F16B      000C   A 0931
F17       000D   A 0907   0936
F18       000E   A 0942
F18A      000F   A 0943
F19       0014   A 0935   0964
F20       0015   A 0970
F3        0001   A 0869   1007
F4        0002   A 0875
F5        0003   A 0881
F6        0010   A 0881   0904   0931   0948   0970   1971
F6A       0011   A 0951   0957   0958
F7        0012   A 0957
F7A       0013   A 0958
F9        0004   A 0869   0887   1780
FAD       0165   A 1908
FAD1      0406   A 1908   2202
FAD4      045A   A 2202   2383
FAD6      045C   A 2383   2385
FCM       0410   A 2211   2404   2570
FCS       016A   A 1918
FCS1      046B   A 1920   2409
FCS2      0477   A 2409   2418   2419   2421   2426   2429   3098   3112   3121
                   3124   3128   3129
FCS3      04C8   A 2413   2609
FCS4      0470   A 2419
FCS5      0472   A 2421   2611
FCS7      0479   A 2416   2431   3110
FDV       0168   A 1914
FDV1      0428   A 1914   2260
FDV10     04A2   A 2537   2557
FDV11     04AC   A 2561
FDV13     046A   A 2404   2514
```

```
FDV4      048D   A 2260   2495
FDV5      0498   A 2510   2518
FDV6      04AA   A 2519   2523   2525   2529   2549   2557
FDV7      04A5   A 2537   2547
FDV8      04A7   A 2541   2543   2551
FGEN      010D   A 1738
FGEN1     0406   A 1738   2193
FGEN2     0400   A 2186   2195   2210   2242   2266
FHALT     0030   A 0863   1070
FHALT2    0074   A 1070   1350
FHALT3    002D   A 1046   1350
FLD       0163   A 1904
FLD1      0456   A 1906   2377
FLEX      04CE   A 2211   2370   2444   2509   2528   2632   2633
FLEX1     04CF   A 2402   2634
FLEX2     04D1   A 2497   2636
FLEX4     04D2   A 2631   2633   2635   2637
FLEX5     04D3   A 2638   3143   3145
FLEX8     04CD   A 2265   2632
FLEX9     04C8   A 2254   2630
FLOT      0420   A 2243
FLOT1     0416   A 2220   2246
FLX       0129   A 1787
FLX1      0103   A 1725   1787
FLX2      012A   A 1725   1789
FMP       0167   A 1912
FMP1      0417   A 1912   2222
FMP4      047A   A 2222   2441
FMP5      0487   A 2464   2471
FMP6      0486   A 2470   2476
FMP7      0489   A 2467   2476
FOUT      00FC   A 1576   1581   1592   1603   1656   1682   1697   1704   1718
FOUT1     008E   A 1418   1719
FPAGE     0060   A 1292
FPAGE3    008A   A 1292   1409
FRAC      0430   A 2267
FRAC1     0482   A 2273   2574
FRAC2     0489   A 2585   2591
FRAC3     048C   A 2575   2591
FREAD     0066   A 1312
FRN       0414   A 2217
FSB       0166   A 1910
FSB1      040D   A 1910   2205
FSB4      045F   A 2205   2390
FSB6      0461   A 2390   2392
FSEQ      0400   A 2187
FSGT      0405   A 2192
FSLE      0404   A 2191
FSMI      0402   A 2189
FSNE      0401   A 2188
FSPL      0403   A 2190
FST       0169   A 1916
FST1      0464   A 1916   2396
GENB      0149   A 1844
GENB1     0183   A 1845   1954
HLT       0148   A 1846
IAB       01A2   A 1454   1498   2002
IAB2      018F   A 1975   2009
ICA       0189   A 2031
ICL       0185   A 2026
ICR       0186   A 2027
ILL       0142   A 1822
ILL3      0085   A 1511   1823
IMA       0176   A 1934
INA       0151   A 1866   1864
INA1      0156   A 1872   1880   1882
INA2      018A   A 1875   1968   2148
INH       0190   A 1977
INK       0186   A 1958
INT       0424   A 2247
INT1      0428   A 2254
INT2      048F   A 2257   2598   2608
INT3      0016   A 0975   0975   1061
INT4      04C6   A 2602   2607
INT5      04C2   A 2603   2607
INT6      04C7   A 2598   2608
INTEX     002E   A 1059
IRS       01E6   A 2140
IRX       01A5   A 2006   2012
ISI       019A   A 1994
JDX       0138   A 1804
JEQ       0128   A 1790
JGE       0136   A 1801
JGT       0131   A 1796
JIX       0139   A 1806
JLE       012F   A 1794
JLT       0134   A 1799
JMP       0100   A 1791   1793   1795   1797   1800   1802   1808   1813   2087
JNE       012D   A 1792   1804   1806
JST       014E   A 1849
JST1      0150   A 1853   2645
JSX       013A   A 1807
LDA       01DE   A 1455   1500   1929   1975   2129
LDA3      0160   A 1899   2129
LDX       0147   A 1831
LEQ       01C8   A 2079
LF        01CE   A 2084
LGE       01CC   A 2081
LGT       01CD   A 2083
LLE       01C9   A 2075
```

```
        LLL      01D1   A 2089
        LLR      01C5   A 2064
        LLS      01D3   A 2093
        LLT      01C8   A 2073
        LMCM     01BD   A 1973
        LNE      01CA   A 2077
        LOAD     0419   A 2203   2206   2223   2236   2261
        LOGIC    01BA   A 2036
        LPMJ     0120   A 1774
        LPMX     0120   A 1773
        LRL      01B8   A 2038
        LRR      01BF   A 2048
        LRS      01BD   A 2043
        LT       01CF   A 2073   2075   2077   2079   2081   2083   2086
        MC1      0031   A 1077
        MC2      0032   A 1080
        MC3      0033   A 1083
        MC4      0034   A 1086   1325   1710
        MEMP3    008C   A 1331   1415
        MEMPAR   006C   A 1330   1336
        MI1      008F   A 0978   1523
        MI2      001C   A 0993   1523
        MMOD     006E   A 1335
        MOVE     0449   A 2322   2328   2334   2336
        MPY      017C   A 1942
        NRM      0118   A 1758
        NRM10    0452   A 2352   2363
        NRM17    0451   A 2359   2362
        NRM18    044E   A 2356   2362
        NRM/     0420   A 2263   2356
        NRML     0448   A 2214   2216   2219   2220   2313   2352   2388   2395   2480
                          2484   2572
        NVECT    001A   A 0979   0983
        OSI      019C   A 1996
        OTA      0159   A 1884
        OTA1     015F   A 1892   1896
        OTK      0192   A 1771   1980
        PAGE     007C   A 1293   1377
        PAGE3    0063   A 1301   1379
        PAGE4    0080   A 1301   1387
        PAGE7    0083   A 1387   1392
        PFL      0075   A 1351
        PID      0174   A 1930
        PIM      0172   A 1928
        PIO      0159   A 1883
        RCB      01A7   A 2008
        READ     0064   A 1306
        REST     0059   A 1184   1192   1252
        RMC      0187   A 1844   1959
        RIN      0104   A 1727
        RXM      0068   A 1319   1746   1775   1973   1977   1990   1996   2166
        S2A      01AE   A 2019
        SCA      0185   A 1957
        SCB      0182   A 2023
        SGL      0188   A 1981
        SKP      01DA   A 2111
        SKS      0157   A 1869   1881
        SKS1     01EC   A 1881   2148
        SDA      0144   A 2005
        SSM      0181   A 2022
        SSP      01A1   A 2001
        STA      01E8   A 2147
        STA3     0087   A 1513   2147
        STX      0187   A 2029
        SUB      014C   A 1847   1925
        SUB3     016C   A 1847   1921
        SVC      0198   A 1992
        TCA      0180   A 2021
        TEXIT    004A   A 1160   1171   1192
        TIN      0040   A 1157   1163
        TIN1     0042   A 1148   1163
        TOUT     0043   A 1168   1176
        TOUT1    0046   A 1148   1176
        UII      0140   A 1817
        UII3     0083   A 1504   1512   1736   1818
        VIRY     0126   A 1783
        VIRY1    00C4   A 1083   1324   1420   1563   1784
        VIRY12   00CF   A 1600   1606   1607   1608   1619   1631
        WRITE    0062   A 1298
        WRITEP   0072   A 1345
        XCA      01A2   A 1931   2083
        XCB      01A8   A 1476   1494   2003   2009
        XEC      0122   A 1776
 IDNT [MACRO]
 CPU  [MACRO]
 RR [MACRO]
 ALU  [MACRO]
 FRRD$  [MACRO]
 EMIT$  [MACRO]
 SET$ [MACRO]
 GEN$ [MACRO]
 LST1$  [MACRO]
 CNT$ [MACRO]
 RRSV$  [MACRO]
 AVF$ [MACRO]
 GBP$ [MACRO]
 ORG  [MACRO]
 SYM$ [MACRO]
 SYM1$  [MACRO]
 CLST$  [MACRO]
```

APPENDIX D

TRAPS - INTERRUPTS SUMMARY

Traps

There are 12 different μ-code traps that cause the μ-code to break normal sequence. They are:

1. 60 FPAGE Trap.

   The CAM must be filled with the new page pointer if available. If not, generate a Page Fault interrupt identical to page trap except F01 and F02 are loaded.

2. 62 Write Address Trap.

   Put RM into the appropriate RF instead of HSM.

3. 64 Read Address Trap.

   Put RF into RM as appropriate instead of HSM to RM.

4. 66 Fetch Read Address Trap

   Same as 3, but load R01 and F02 also.

5. 68 Restricted Execution Trap.

   Generates RXM vector.

6. 6A CP Parity.

   Machine Check generates the Machine Check Interrupt.

7. 6C Memory Parity.

   Generates Memory Parity interrupt.

8. 6E Missing Memory Module.

   Generates Missing Module interrupt.

9. 70 DMX.

   Performs a DMX transfer without changing user execution flow.

10. 72 Page Write Violation.

    Generates a Page Write Violation interrupt.

11. 7C Page Trap

    Identical to 1 only F01 and F02 are not changed.

Interrupt Vector Summary

Loc	Name	Deposited P Counter	Vector Type	Register 11	Register 12
60	PFAIL	Next	Absolute[1]	(none)	(none)
62	Restrict Ex Violation	This	Absolute	(none)	(none)
63	Ext INT	Next	Absolute[1]	(none)	(none)
64	Page Fault	This	Absolute	(none)	effective addr
65	SVC	Next	Absolute	(none)	(none)
66	UII	This	Absolute	Next P	effective addr
67	Mem Parity	Next[2]	Absolute	(none)	(none)
70	CPU Parity	Next[2]	Absolute	(none)	(none)
71	Missing Mem Mod	Next[2]	Absolute	(none)	(none)
72	ILL	This	Absolute	Next P	effective addr
73	Page Write	This	Absolute	(none)	effective addr
74	FLEX	Next	Absolute[3]	Flag Reg[4]	effective addr
75	PSU	This	Absolute	(none)	(none)

[1] External interrupts are inhibited automatically.

[2] Next is true only if the error occurred during code execution. DMA or DMC errors can abort an instruction in mid-execution.

[3] Identical to other vectors except that non-implementation ([74] = 0) causes the vector to not be executed and the instruction sequence is continued with C Bit = 1.

[4] Flag Register:

Left byte:

Single precision Floating Point	= 1
Double precision Floating Point	= 2

Right byte:

Overflow/Underflow	= 0
Divide by Zero	= 1
Store exception	= 2
INT exception	= 3

APPENDIX E

INSTRUCTIONS FOR USE OF PROM, μ-CODE PROGRAM

1. This program will handle up to 512 u-code words.

2. See comment at beginning of listing.

3. Legal responses to "MODULE" must be followed by a carriage return and are as follows:

   a) Any legal CPU or XCS module. (1-1L, 1-2L, ..., 7-2U)

   b) The letter "N" (next) may be used for sequential module selection except before the first module of a bank. (1-1L or 1-1U)

   c) The letter "Q" (quit) returns user to operating system.

4. A legal response causes the system to "answer back" the legal module. At this time, the inputs to the PROM Writer are valid and PROM may be verified or programmed.

See enclosed example of a typical working session.

THE FOLLOWING IS AN EXAMPLE OF A WORKING SESSION WHERE THE REQUIREMENTS ARE TO PROGRAM MODULES 1-1L THROUGH 3-1U AND MODULES 6-1U, 7-1U, AND 7-2U.

NOTE: THE SYSTEM PRINTS THE PROMPT 'MODULE = ' AND THE USER RESPONDS WITH THE MODULE NAME OR WITH 'N' TO SPECIFY THE NEXT MODULE NAME IN THE SEQUENCE.

```
MODULE = 1-1L
1-1L
MODULE = N
1-2L
MODULE = N
2-1L
MODULE = N
2-2L
MODULE = N
3-1L
MODULE = N
3-2L
MODULE = N
4-1L
MODULE = N
4-2L
MODULE = N
5-2L
MODULE = N
6-1L
MODULE = N
6-2L
MODULE = N
7-1L
MODULE = N
7-2L
MODULE = 1-1U
MODULE = N
```

```
THE FOLLOWING IS AN EXAMPLE OF A WORKING SESSION WHERE THE


1-2U
MODULE = N
2-1 L
MODULE = N
2-2U
MODULE = N
3-1U
MODULE = 5-1L
MODULE =        (SINCE 5-1L IS NOT A LEGAL MODULE THE RESPONSE IS NOT ACCEPTED
MODULE = 6-1U
6-1U
MODULE = 7-1U
7-1U
MODULE = N
7-2U
MODULE = Q
```

```
                   (0001) *      PROM, U-CODE, JMG-KRR, 11 JUNE 74
                   (0002) *      PROGRAM OR VERIFY P-ROM FOR WCS OR CENTRAL PROCESSOR
                   (0003) *      PRIME COMPUTERS INC., SRCXXXX.000
                   (0004) *      COPYRIGHT 1974, PRIME COMPUTERS INC., NATICK MASS
                   (0005) *
                   (0006) *
                   (0007) *      THIS ROUTINE IS A DOS COMMAND TO PROGRAM OR VERIFY P-ROM.
                   (0008) *
                   (0009) *      THE DOS COMMAND IS:
                   (0010) *              PROM FILENAME
                   (0011) *
                   (0012) *      FILENAME MUST BE A 'SAVE' FILE WITH A LOW OF '10000.
                   (0013) *
          000221   (0014)        ENT     X                COMMON FOR RDCOM
                   (0015) *
                   (0016)        REL
                   (0017) *
          010000   (0018) BUFF   EQU     '10000           BUFFER FOR CODE TO BE LOADED TO WCS
          000024   (0019) DEV    EQU     '24              WCS DEVICE CODE
                   (0020) *
000000:            (0021) BEGN   ELM
                   (0022) *
000001: 10.000000E (0023)        CALL    CMREAD           GET FILE NAME
000002: 00.010000A (0024)        DAC     BUFF
                   (0025) *
000003: 10.000000E (0026)        CALL    SEARCH           CLOSE UNIT 1
000004: 00.000227  (0027)        DAC     =4
000005: 00.000230  (0028)        DAC     =0
000006: 00.000231  (0029)        DAC     =1
000007:   000000   (0030)        OCT     0
                   (0031) *
000010: 10.000000E (0032)        CALL    SEARCH           OPEN UNIT 1 FOR READING
000011: 00.000231  (0033)        DAC     =1
000012: 00.010003A (0034)        DAC     BUFF+3
000013: 00.000231  (0035)        DAC     =1
000014:   000000   (0036)        OCT     0
                   (0037) *
000015: 10.000000E (0038)        CALL    PRWFIL           READ FILE INTO MEMORY
```

```
000016: 00.000231  (0039)         DAC      =1             KEY TO READ
000017: 00.000231  (0040)         DAC      =1
000020: 00.000232  (0041)         DAC      =BUFF
000021: 00.000233  (0042)         DAC      =2048
000022: 00.000234  (0043)         DAC      =9             STEP PAST SAVE VECTOR
000023: 00.000235  (0044)         DAC      =ALT           MOST READS WILL BE SHORT
000024:    000000  (0045)         OCT      0
                   (0046) *
000025: 10.00000E  (0047) ALT     CALL     SEARCH         CLOSE UNIT 1
000026: 00.000227  (0048)         DAC      =4
000027: 00.000230  (0049)         DAC      =0
000030: 00.000231  (0050)         DAC      =1
000031:    000000  (0051)         OCT      0
                   (0052) *
                   (0053) *       DETERMINE NEXT MODULE TO BE LOADED INTO WCS
                   (0054) *
           000032  (0055) NEXTMODULE  EQU  *            CHOOSE A MODULE
                   (0056) *
000032: 10.00000E  (0057)         CALL     TNOUA          ASK FOR MODULE
000033: 00.000236  (0058)         DAC      =C'MODULE =  '
000034: 00.000234  (0059)         DAC      =9
000035:    000000  (0060)         OCT      0
                   (0061) *
000036: 10.00000E  (0062)         CALL     RDCOM          GET RESPONSE
000037: 00.000317  (0063)         DAC      CMBUFF
                   (0064) *
000040: 02.000317  (0065)         LDA      CMBUFF         VERIFY 1ST CHAR VALID
000041:    141050  (0066)         CAL
000042: 11.000243  (0067)         CAS      =R'Q'          QUIT, RETURN TO OPERATING SYSTEM
000043:    100000  (0068)         SKP
000044: 01.000125  (0069)         JMP      DONE
000045: 11.000244  (0070)         CAS      =R'N'          NEXT-USE NEXT SEQUENTIAL MODULE
000046:    100000  (0071)         SKP
000047: 01.000131  (0072)         JMP      NEXT
000050: 11.000245  (0073)         CAS      ='260          MUST BE INTEGER 1-7
000051: 11.000246  (0074)         CAS      ='270
000052: 01.000032  (0075)         JMP      NEXTMODULE     NON VALID RESPONSE
000053: 01.000032  (0076)         JMP      NEXTMODULE
```

```
000054:    141340  (0077)         ICA
000055:    140104  (0078)         XCA                     SAVE THIS CHARACTER IN LEFT HALF OF B-REG
                   (0079) *
000056: 02.000320  (0080)         LDA      CMBUFF+1       2ND CHAR MUST BE '-'
000057:    141050  (0081)         CAL
000060: 05.000247  (0082)         ERA      =R'-'
000061:    100040  (0083)         SZE
000062: 01.000032  (0084)         JMP      NEXTMODULE
                   (0085) *
000063: 02.000321  (0086)         LDA      CMBUFF+2       3RD CHAR MUST BE 1 OR 2
000064: 11.000245  (0087)         CAS      ='260
000065: 11.000250  (0088)         CAS      ='263
000066: 01.000032  (0089)         JMP      NEXTMODULE
000067: 01.000032  (0090)         JMP      NEXTMODULE
000070: 05.00002A  (0091)         ERA      2              VALID MODULE
000071: 04.000226  (0092)         STA      MODULE         SAVE FOR MODULE CHECK
                   (0093) *
000072: 02.000322  (0094)         LDA      CMBUFF+3       4TH CHAR MUST BE 'U' OR 'L'
000073:    141050  (0095)         CAL
000074: 11.000251  (0096)         CAS      =R'U'
000075:    100000  (0097)         SKP
000076: 01.000106  (0098)         JMP      NM20
000077: 11.000252  (0099)         CAS      =R'L'
000100: 01.000032  (0100)         JMP      NEXTMODULE
000101:    100000  (0101)         SKP
000102: 01.000032  (0102)         JMP      NEXTMODULE
000103: 04.000224  (0103)         STA      UL             SAVE FOR TYPE-OUT
000104:    140040  (0104)         CRA
000105: 01.000110  (0105)         JMP      NM40
                   (0106) *
000106: 04.000224  (0107) NM20    STA      UL             SAVE FOR TYPE-OUT
000107: 02.000253  (0108)         LDA      ='2000         SPECIFY UPPER CONTROL BANK
000110: 04.000225  (0109) NM40    STA      ULBASE         SAVE FOR BASE CALCULATION
                   (0110) *
000111: 35.000316  (0111)         LDX      TBLCNT         CHECK FOR VALID MODULE NAME
000112: 02.000254  (0112)         LDA      =TABLE
000113: 04.000223  (0113)         STA      TBLPT          TABLE POINTER AT START OF MODULE TABLE
000114: 42.000223  (0114) NM50    LDA*     TBLPT          FETCH MODULE NAME
```

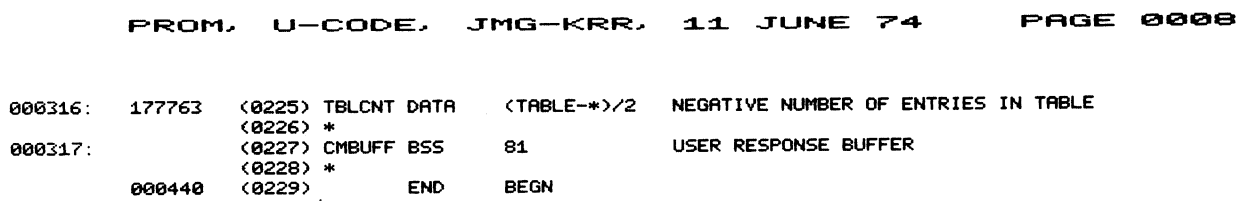
```
000115: 05.000226  (0115)        ERA     MODULE
000116:   101040   (0116)        SNZ
000117: 01.000141  (0117)        JMP     MODULELOAD    MATCH, GO LOAD WCS
000120: 12.000223  (0118)        IRS     TBLPT         BUMP TWICE TO POINT AT NEXT ENTRY
000121: 12.000223  (0119)        IRS     TBLPT
000122:   140114   (0120)        IRX
000123: 01.000114  (0121)        JMP     NM50
000124: 01.000032  (0122)        JMP     NEXTMODULE    NO MATCH, INVALID RESPONSE
                   (0123) *
          000125   (0124) DONE   EQU     *             RETURN TO OPERATING SYSTEM
000125:   031724   (0125)        OCP     '1700+DEV     LEAVE WCS BOARD INITIALIZED
000126: 10.000000E (0126)        CALL    EXIT
000127:            (0127)        ELM
000130: 01.000000  (0128)        JMP     BEGN          RESTART
                   (0129) *
          000131   (0130) NEXT   EQU     *             USE NEXT SEWUENTIAL MODULE
000131: 02.000223  (0131)        LDA     TBLPT
000132:   140304   (0132)        A2A
000133: 04.000223  (0133)        STA     TBLPT
000134: 11.000255  (0134)        CAS     =TABLE-1      MUST BE WITHIN TABLE BOUNDS
000135: 11.000256  (0135)        CAS     =TBLCNT
000136: 01.000032  (0136)        JMP     NEXTMODULE
000137: 01.000032  (0137)        JMP     NEXTMODULE
000140: 01.000141  (0138)        JMP     MODULELOAD
                   (0139) *
                   (0140) *      LOAD MODULE POINTED AT BY TBLPT INTO WCS
                   (0141) *
          000141   (0142) MODULELOAD  EQU  *
                   (0143) *
000141: 35.000257  (0144)        LDX     =-256        1K CHIP IS 256X4
000142: 02.000232  (0145)        LDA     =BUFF        SET BASE TO START OF UPPER OR LOWER BANK
000143: 06.000225  (0146)        ADD     ULBASE
000144: 04.060222  (0147)        STA     BASE
000145:   031724   (0148)        OCP     '1700+DEV     INITIALIZE WCS
                   (0149) *
000146: 02.000223  (0150) ML10   LDA     TBLPT         FETCH START BIT OF THIS 64 BIT ENTRY
000147:   141206   (0151)        A1A
000150: 42.000001A (0152)        LDA*    1
```

```
000151:   040074   (0153)        LRL     4             SPLIT INTO 16 BIT WORD AND BIT WITHIN WORD
000152: 06.000222  (0154)        ADD     BASE
000153: 04.000220  (0155)        STA     TEMP          SAVE WORD
000154:   140040   (0156)        CRA
000155:   041074   (0157)        LLL     4             GET BIT POSITON BACK
000156: 06.000260  (0158)        ADD     =3            MAKE INTO A POSITIVE SHIFT COUNT
000157:   140407   (0159)        TCA
000160: 03.000261  (0160)        ANA     ='77          NEGATIVE SHIFT COUNT IN 6 BITS
000161: 06.000262  (0161)        ADD     ='041600      ALR WITH CALCULATED COUNT
000162: 04.000164  (0162)        STA     ML20
000163: 42.000220  (0163)        LDA*    TEMP          FETCH WORD
000164:   000000   (0164) ML20   OCT     0             SHIFT
000165:   170124   (0165)        OTA     '100+DEV      LOAD RAM
000166:   000000   (0166)        HLT
000167:   170124   (0167)        OTA     '100+DEV
000170:   000000   (0168)        HLT
000171:   170124   (0169)        OTA     '100+DEV
000172:   000000   (0170)        HLT
000173:   170124   (0171)        OTA     '100+DEV
000174:   000000   (0172)        HLT
                   (0173) *
000175: 02.000222  (0174)        LDA     BASE          MOVE BASE BY 64 BITS
000176: 06.000227  (0175)        ADD     =4
000177: 04.000222  (0176)        STA     BASE
000200:   140114   (0177)        IRX
000201: 01.000146  (0178)        JMP     ML10          GO LOOP
                   (0179) *
000202: 02.000263  (0180)        LDA     ='40000       LET PROM DEVICE ACCESS RAM
000203:   170324   (0181)        OTA     '300+DEV
000204:   000000   (0182)        HLT
                   (0183) *
000205: 42.000223  (0184)        LDA*    TBLPT         TYPE MODULE NUMBER TO USER
000206:   040070   (0185)        LRL     8
000207: 10.000000E (0186)        CALL    T1OB
000210: 02.000247  (0187)        LDA     =R'-'
000211: 10.000000E (0188)        CALL    T1OB
000212:   041070   (0189)        LLL     8
000213: 10.000000E (0190)        CALL    T1OB
```

```
000214: 02.000224  (0191)          LDA     UL
000215: 10.000000E (0192)          CALL    T10B
000216: 10.000000E (0193)          CALL    TONL
000217: 01.000032  (0194)          JMP     NEXTMODULE     READY FOR NEXT REQUEST
                   (0195) *
                   (0196) *        STORAGE
                   (0197) *
000220:   000000   (0198) TEMP     OCT     0
000221:   000000   (0199) X        OCT     0              COMMON FOR RDCOM
000222:   000000   (0200) BASE     OCT     0
000223:   000000   (0201) TBLPT    OCT     0              POINTER INTO MODULE TABLE
000224:   000000   (0202) UL       OCT     0              CONTAINS 'U' OR 'L'
000225:   000000   (0203) ULBASE   OCT     0              CONTAINS 0 OR '2000
000226:   000000   (0204) MODULE   OCT     0              TRIAL MODULE
                   (0205) *
000227: 00.000004A (0206)          FIN
000230: 00.000000A
000231: 00.000001A
000232: 00.010000A
000233: 00.004000A
000234: 00.000011A
000235: 00.000025
000236: 00.146717A
000237: 00.142325A
000240: 00.146305A
000241: 00.120275A
000242: 00.120240A
000243: 00.000321A
000244: 00.000316A
000245: 00.000260A
000246: 00.000270A
000247: 00.000255A
000250: 00.000263A
000251: 00.000325A
000252: 00.000314A
000253: 00.002000A
000254: 00.000264
000255: 00.000263
```

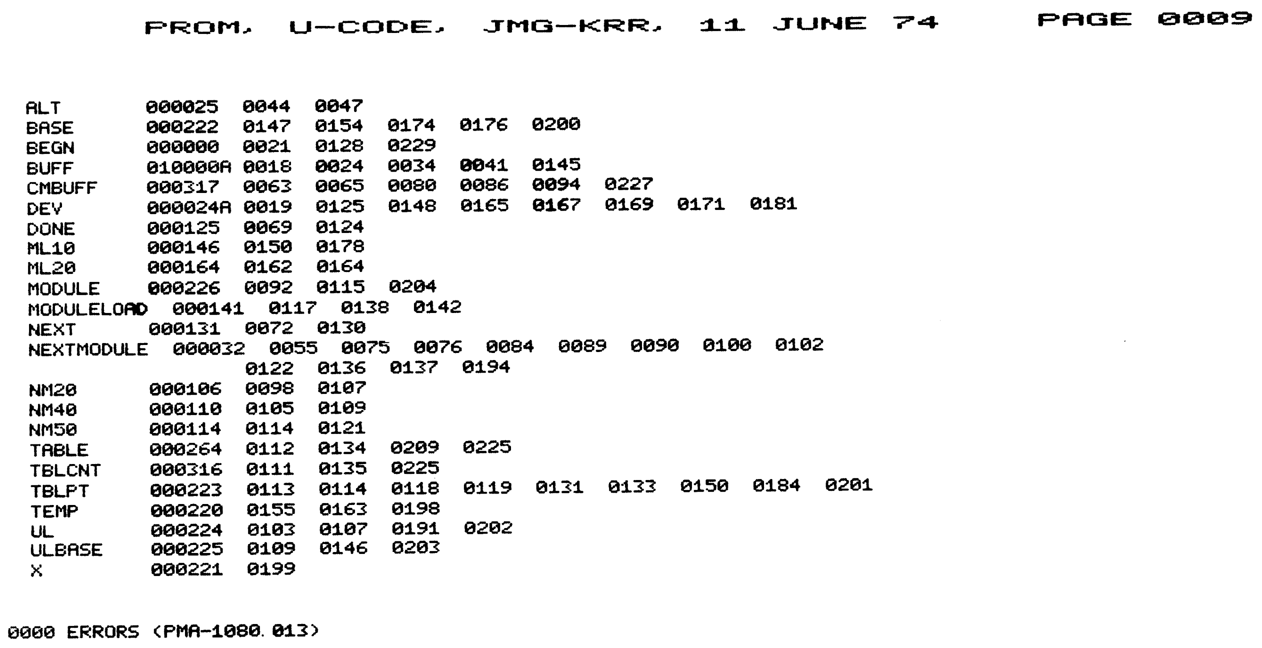
```
000256: 00.000316
000257: 00.177400A
000260: 00.000003A
000261: 00.000077A
000262: 00.041600A
000263: 00.040000A

                   (0207) *
                   (0208) *
          000264   (0209) TABLE   EQU      *        CONTAINS MODULE NAME AND 1ST BIT OF 64 FOR THAT MODULE
                   (0210) *
000264:   130661   (0211)         DATA     C'11',1
000265:   000001
000266:   130662   (0212)         DATA     C'12',5
000267:   000005
000270:   131261   (0213)         DATA     C'21',9
000271:   000011
000272:   131262   (0214)         DATA     C'22',13
000273:   000015
000274:   131661   (0215)         DATA     C'31',17
000275:   000021
000276:   131662   (0216)         DATA     C'32',21
000277:   000025
000300:   132261   (0217)         DATA     C'41',25
000301:   000031
000302:   132262   (0218)         DATA     C'42',29
000303:   000035
000304:   132662   (0219)         DATA     C'52',45
000305:   000055
000306:   133261   (0220)         DATA     C'61',49
000307:   000061
000310:   133262   (0221)         DATA     C'62',53
000311:   000065
000312:   133661   (0222)         DATA     C'71',57
000313:   000071
000314:   133662   (0223)         DATA     C'72',61
000315:   000075
                   (0224) *
```



```
        PROM, U-CODE, JMG-KRR, 11 JUNE 74        PAGE 0008


000316:   177763   (0225) TBLCNT DATA    (TABLE-*)/2   NEGATIVE NUMBER OF ENTRIES IN TABLE
                   (0226) *
000317:            (0227) CMBUFF BSS     81            USER RESPONSE BUFFER
                   (0228) *
          000440   (0229)        END     BEGN
```



```
        PROM, U-CODE, JMG-KRR, 11 JUNE 74        PAGE 0009


 ALT        000025  0044  0047
 BASE       000222  0147  0154  0174  0176  0200
 BEGN       000000  0021  0128  0229
 BUFF       010000A 0018  0024  0034  0041  0145
 CMBUFF     000317  0063  0065  0080  0086  0094  0227
 DEV        000024A 0019  0125  0148  0165  0167  0169  0171  0181
 DONE       000125  0069  0124
 ML10       000146  0150  0178
 ML20       000164  0162  0164
 MODULE     000226  0092  0115  0204
 MODULELOAD  000141  0117  0138  0142
 NEXT       000131  0072  0130
 NEXTMODULE  000032  0055  0075  0076  0084  0089  0090  0100  0102
                    0122  0136  0137  0194
 NM20       000106  0098  0107
 NM40       000110  0105  0109
 NM50       000114  0114  0121
 TABLE      000264  0112  0134  0209  0225
 TBLCNT     000316  0111  0135  0225
 TBLPT      000223  0113  0114  0118  0119  0131  0133  0150  0184  0201
 TEMP       000220  0155  0163  0198
 UL         000224  0103  0107  0191  0202
 ULBASE     000225  0109  0146  0203
 X          000221  0199

0000 ERRORS (PMA-1080.013)
```

INDEX

INDEX (Cont)