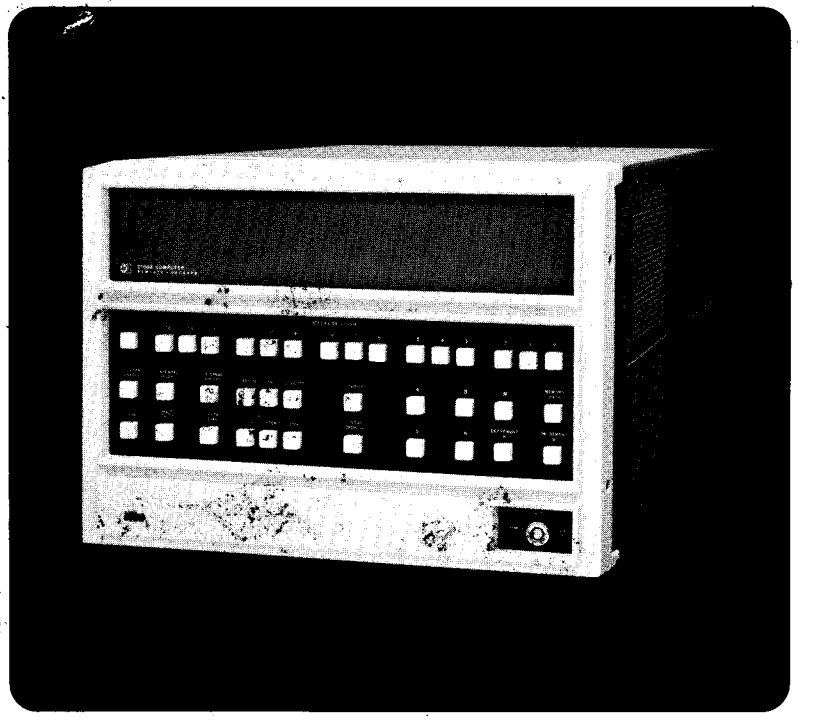
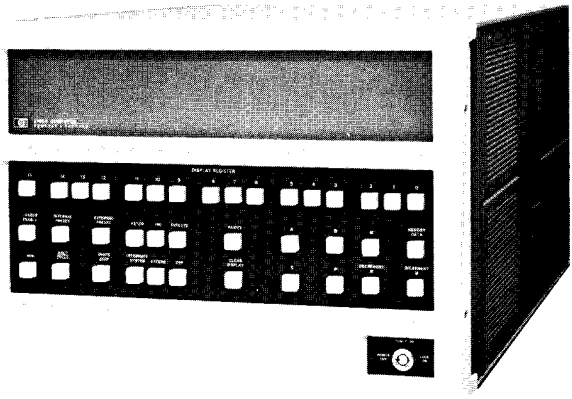
HEWLETT PACKARD

2100 computer



microprogramming guide



- **POWERFUL HARDWARE**
  A proven architecture implemented by a microprocessor in the heart of the control section.
- **EXPANDABLE MAINFRAME MEMORY**
  Lets you choose from 4K to 32K *all in mainframe.*
- **STANDARD FEATURES**
  Includes extended arithmetic instructions, power fail interrupt, memory parity check and memory protect.
- **FLEXIBLE INPUT/OUTPUT**
  14 internal I/O channels, externally expandable to 45.
- **FULL INTERRUPT SYSTEM**
  Interrupt priority easily established or changed for all devices.
- **COMPREHENSIVE SOFTWARE**
  Proven software packages for generating and executing your programs.

2100 computer

The Hewlett-Packard 2100 is a general-purpose digital computer designed for a wide range of small computer applications.

Features built-in to the 2100 include extended arithmetic instructions, power fail interrupt with automatic restart, memory parity check with interrupt and memory protect. Besides the standard built-in features, dual-channel Direct Memory Access (DMA) and Floating Point Hardware are also available. Under DMA control, data can be transferred to or from computer memory at rates greater than one million sixteen-bit words per second. Floating Point Hardware provides a typical ten-fold speed increase for scientific, computer bound algorithms.

A minimum 2100 provides 4096 words of core memory, self-contained power supply and 14 input/output channels.

You can select a wide range of memory sizes up to 32K words, all in mainframe. By including an HP 2155 Extender, you add another 31 input/output channels and power supply.

The 2100 automatically inherits a comprehensive range of proven software packages, including assemblers, compilers, operating systems and subroutines. A complete line of standard computer peripherals and I/O interface kits are also available, permitting complete systems to be tailored around the 2100. Added to these capabilities, you can also depend on the HP reputation for high quality and world-wide customer support. The result is a cost-effective computer that can meet your data processing problems today and continue meeting them as your needs expand.

MICROPROGRAMMING GUIDE

for

Hewlett-Packard Model 2100 Computer

HEWLETT-PACKARD COMPANY
11000 WOLFE ROAD, CUPERTINO, CALIFORNIA U.S.A.

5951-3028

Printed: FEB 1972

PREFACE

This handbook is a complete guide to microprogramming for the Hewlett-Packard 2100 Computer. With the information given here, you will be able to expand the already powerful capability of your 2100 by adding custom-tailored instructions to the existing set of microprogrammed operations. This capability of expanding the Central Processor Unit, in addition to the extraordinary expansion features of the memory and I/O sections, contributes to the total flexibility and unusual adaptability of the 2100.

Essentially, this handbook is self-contained. Microassembler documentation is, of course, required in order to format and assemble your microprograms correctly. Beyond this, however, no other reference documents will be needed for most microprogramming projects.

It is intended that this handbook should be read in sequence, from beginning to end, before attempting to use the information as a reference.

While Hewlett-Packard cannot assume responsibility for the effectiveness of microprograms written and implemented according to the recommendations outlined herein, further information and assistance can be obtained by contacting a Hewlett-Packard field office. Sales and Service offices throughout the world are listed at the back of this handbook.

CONTENTS

1 THE MICROPROGRAMMED COMPUTER CONCEPT 1-1
Comparison of Conventional vs Microprogram Control 1-1
What a Microprogram Is 1-4
The 2100 Approach 1-10
Physical Organization 1-11
Timing 1-17
The Steps to Implement a New Microprogram 1-21
Practical Considerations 1-23
General Factors 1-23
WCS/ROM Implementations 1-24

2 CONTROLLABLE FUNCTIONS IN THE 2100 2-1
General Control Functions 2-1
Control 2-1
Arithmetic Logic 2-4
Memory 2-9
Input/Output 2-12
Effects of Microinstruction Fields 2-13
R-Bus Field 2-13
S-Bus Field 2-13
Function Field 2-14
Store Field 2-15
Special Field 2-16
Skip Field 2-17

3 ACCESS SCHEME 3-1
The MAC Instruction Group 3-1
Mapping 3-3
Standard Jump Table 3-6
Secondary Entry Points 3-9
Non-Standard Jump Tables 3-12
Assigning Addresses 3-13
Software Access 3-14
Use of Module 0 3-15

4 THE 2100 MICROPROGRAMMING LANGUAGE 4-1
Microinstruction Word Format 4-1
Assembly Format 4-2
Micro-Order Instruction Set 4-4
R-Bus Field 4-4
S-Bus Field 4-5
Function Field 4-7
Store Field 4-11
Special Field 4-12
Skip Field 4-14

5 MICROPROGRAMMING METHODS 5-1
Introduction 5-1
Example Microprogram 5-1
Programming Aids and Restrictions 5-5

APPENDIX
Microprogram Listing for Basic Instruction Set A-1

ILLUSTRATIONS

1. Simplified Block Diagrams of Computer Systems 1-2
2. Microprogrammable Controls 1-6
3. Microprogrammed Generation of Controls 1-8
4. Elementary Microprogram 1-9
5. Control Store Locations 1-12
6. Four-Module Control Store 1-14
7. External Control Store 1-15
8. Timing Considerations 1-18
9. Microprogram Implementation 1-22
10. 2100 Block Diagram, Part A 2-18
11. 2100 Block Diagram, Part B 2-19
12. 2100 Block Diagram, Part C 2-20
13. 2100 Block Diagram, Part D 2-21
14. Binary Machine Codes for Extensions 3-2
15. Primary Entry Point Codes 3-4
16. Standard Jump Tables 3-7
17. Secondary Jump Tables 3-10
18. Microinstruction Formats 4-1
19. Sample Assembly Coding 4-4

TABLES

1. Primary Entry Point Mapping 3-5
2. Assembly Language Access 3-14
3. Microinstruction Coding 4-3
4. SWP Microprogram 5-2
5. Storing/Reading Locations 0 and 1 5-8

COMPARISON OF CONVENTIONAL vs MICROPROGRAM CONTROL

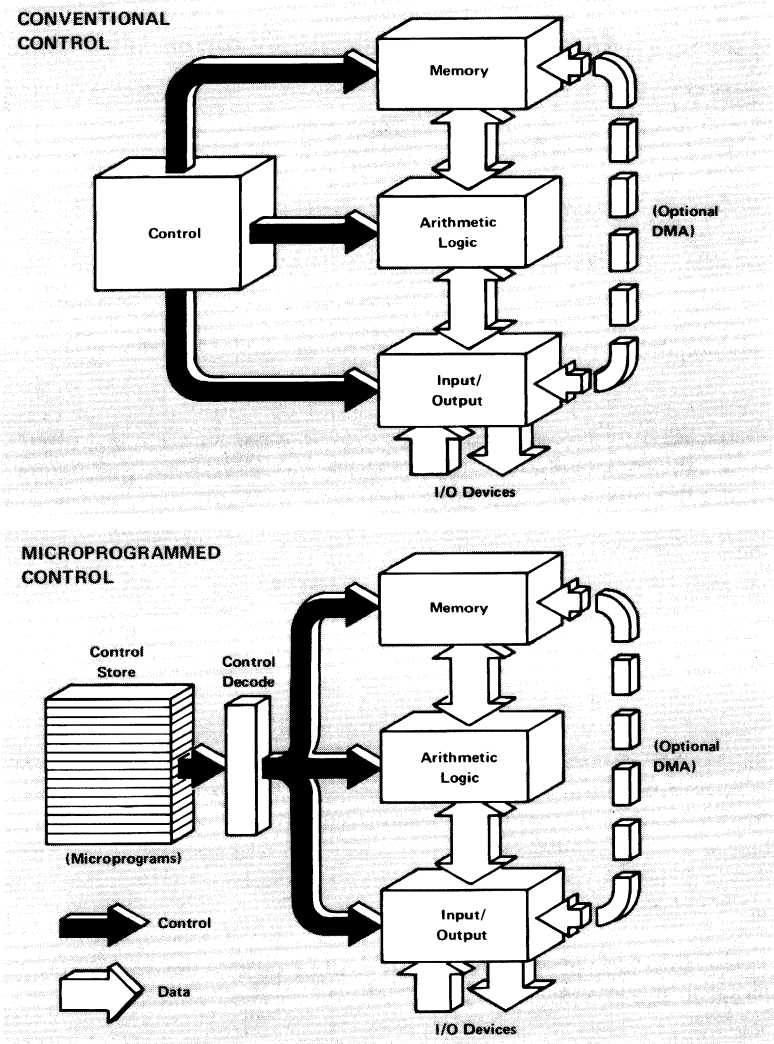
Functionally, a computer is comprised of four major sections:

a. the memory section
b. the arithmetic logic section
c. the control section
d. the input/output section.

Some textbooks separate input and output so as to form five distinct sections. However, the section of prime interest in this handbook is the control section. The advent of a microprogrammable architecture, as used in the Hewlett-Packard 2100 Computer, represents a departure from the conventional method of implementing the control function.

Figure 1 compares the basic structure of computer systems using a conventional control section and one using a microprogrammed control section. Note that, except for the control section, the two systems are identical.

However, in the conventional system, although the block picture looks simpler, the control logic is in fact comparatively more complex. All control functions are implemented by means of a large number of specialized logic circuits scattered throughout the entire computer. A complete set of timing signals, which break the basic machine cycle into discrete "time periods", must also be routed throughout the computer so that the controls may be generated in a desired sequence. This implementation permits the basic instruction set to be executed in a most efficient manner. However, any function beyond the original



2177-1

Figure 1. Simplified Block Diagrams of Computer Systems

design would be very difficult to incorporate. Even minor changes might reveal unforeseen consequences, due to logic interdependency, long after the modification is made. Major changes mean extensive and costly redesign.

In the microprogrammed system, the control logic is relatively simple, and it becomes easy for either the original manufacturer or the user to incorporate new functions (e.g., more machine instructions). The complexity of the microprogrammed system is in the coding of the microprograms. But even here, the systematic design of this approach, once understood, simplifies the process of comprehending and visualizing the various control functions.

Basically, microprogrammed control consists of two parts: control store and control decode. The hardware for both is centrally located, rather than distributed as in the case of conventional control. The control store may be a read-only memory (ROM), as provided in the 2100 to implement the basic instruction set, or it may be a writable control store (typically a semiconductor random-access memory). Stored within this memory, either permanently or semi-permanently, are the microprograms which control the operation of the computer. For the most part, the microprograms are dedicated to the execution of the machine instructions (one microprogram per machine instruction); however some microprograms perform other functions, as will be seen later in this handbook.

The control decode accepts one microinstruction word at a time from control store. Each such word consists of a number of "micro-orders" (six in the case of the 2100). Each micro-order is decoded to activate one or a set of specific control lines to perform a given function. Thus, in the 2100, up to six control functions may be simultaneously activated by control decode.

Taken together, control store and control decode are sometimes referred to as a microprocessor.

Although not shown in figure 1, there is obviously a need for some means to address the word locations in the control store. Also, the address will have to be incremented in order to advance through a

microprogram, and it will have to be altered for microprogram jumps. These features will be discussed in section 2.

At this point, however, it can be seen that control signals are generated from a decoded microinstruction word. It should also be apparent that most timing requirements are automatically taken care of by the fact that only one microinstruction word at a time may be executed.

WHAT A MICROPROGRAM IS

As applied in this handbook, a microprogram is a program-structured sequence of commands which resides in hardware and can be translated by hardware into hardware controls. This mergence of software (i.e., programs) into a hardware medium leads to the generic term "firmware"; this term is used when speaking of microprograms as a physical entity.

To illustrate the concept of a microprogram, it is best to look at the hardware functions that are to be controlled and work backward. First we'll extract a portion of the 2100 Computer logic (figure 2), then show how the logic can be controlled by a microprogram (figure 3), and finally develop an actual microprogram to perform the intended function (figure 4).

Note: In order to keep things simple at this level of discussion, the following descriptions are not strictly valid for the 2100 Computer. Specifically, we will neglect the complications arising out of the fact that the A- and B-registers are addressable as memory locations. The actual corresponding microprogram can be seen in the listing in the appendix of this handbook; see ROM addresses 144, 145, and 146.

The intended function in our example is to add the contents of a memory location to the contents of the A-register — i.e., to execute the

been fetched from memory; as a result, the operand address is presently residing in the Scratch Pad 1 register. (Fetching is also done by a microprogram, although this will not be duscussed here.)

Referring to figure 2, note that nine separate controls are necessary to execute the ADA instruction. Also note the logic symbology key at the bottom of figure 2. The S-bus, shown bold in the figure, is the major data communication path in the 2100.

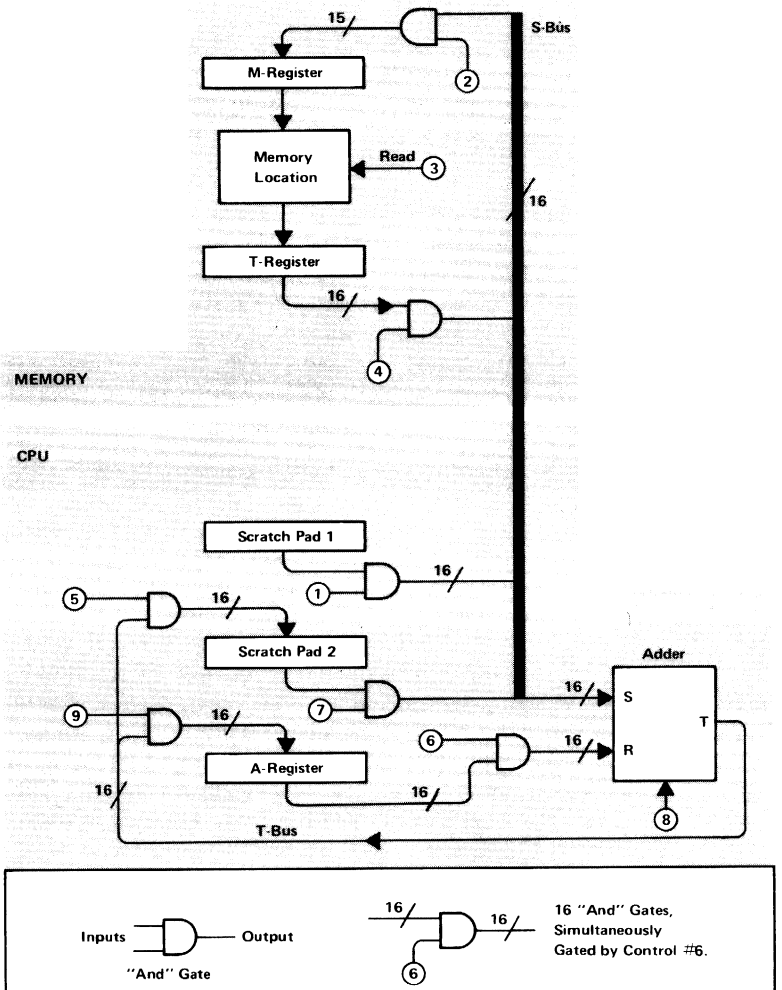
The first step in the execution of ADA is to ask memory to fetch the addressed operand. (The implicit operand is assumed to be present in the A-register.) First, the operand address in Scratch Pad 1 is to be read out to the S-bus (1) and stored in the M-register (2); also, memory must be given a command (3) to read the contents of the addressed location. After this, there will presumably be a short delay while memory goes through its cycle. Then, when the operand is in the T-register, the next step may proceed.

The second step is to bring the operand from memory into the central processor unit. (The CPU is the unit which contains the control and arithmetic logic sections.) This step consists of reading the T-register contents out to the S-bus (4), routing it through the adder with an "IOR", and storing from the T-bus into Scratch Pad 2 (5). Once this is done, the final step may proceed.

The third step is to add the two operands and deposit the result in the A-register. This is done by reading out the contents of the A-register to the R-bus (6), which is one input to the adder, reading out the contents of Scratch Pad 2 to the S-bus (7), which is the other input to the adder, and issuing an "add" command to the adder (8). The result on the T-bus is stored into the A-register (9), and the execution is complete.

By analyzing the types of actions that occur in the preceding steps, a systematic approach can be made. Simply group the actions according to the type of control required. That is, controls that:

a. read something onto the R-bus
b. read something onto the S-bus
c. cause the adder to do a specific function



2177-2A

Figure 2. Microprogrammable Controls

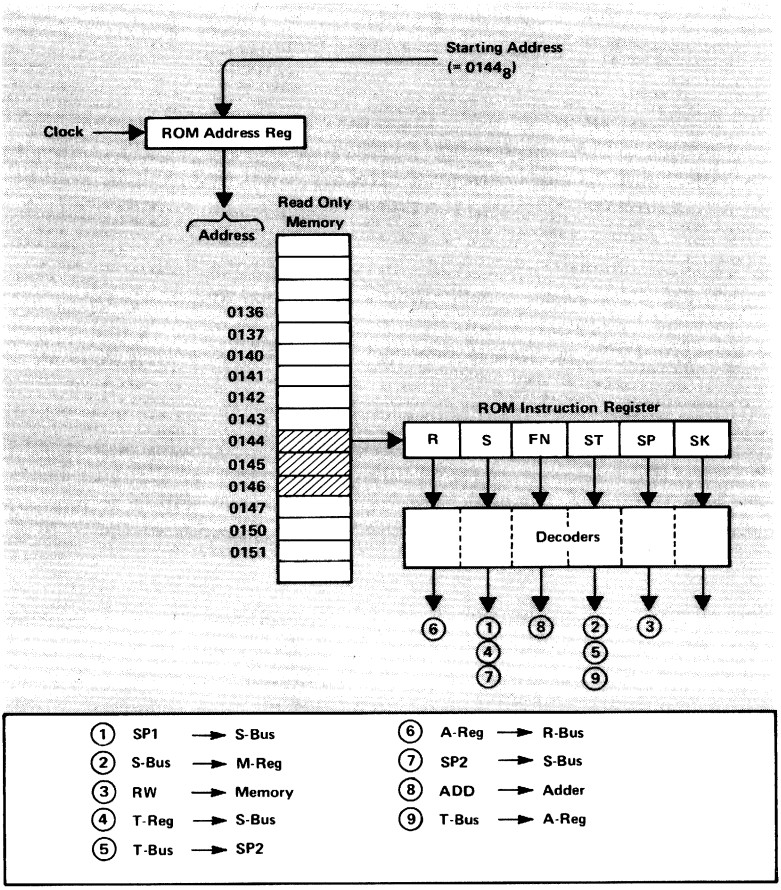
d. store something into a register
e. cause special functions (e.g., read memory).

Refer now to figure 3. Here, the nine actions (circled numbers from figure 2) have been grouped in five categories directly corresponding to the above list. (A sixth essential category, SK for skip, is not used in this example but is shown in the figure.) Thus, action 6 reads something onto the R-bus, actions 1, 4, and 7 read something onto the S-bus, action 8 is an arithmetic (add) function, actions 2, 5, and 9 store something into a register, and action 3 is a special function.

If each of these actions is decoded from an instruction register, as shown in figure 3, the instruction register will be divided into six separate "fields". Several bits in each field (the average is four) permits the selection of a specific number of sources, destinations, or functions. For example, actions 1, 4, and 7 will read (respectively) Scratch Pad 1, the T-register, and Scratch Pad 2 onto the S-bus. These actions are encoded, respectively, by the following binary codes in the S field: 1011, 0010, and 1010.

Now it is possible to encode all of the nine actions into an instruction word format. Since the S-bus and Store fields are each used three separate times, there will have to be a minimum of three microinstruction words. It will also be shown shortly that no more than three words are required. Thus our microprogram will consist of three storable microinstruction words. These are shown in figure 3 as occupying locations 0144, 0145, and 0146 of a read-only memory.

Now then, to execute the ADA instruction, assuming that the three microinstruction words have been correctly coded, it is only necessary to read three words, in succession, into the ROM Instruction Register. This is done by supplying the starting address of 0144 (which is derived from the ADA instruction code) to the ROM Address Register, and then permitting the system clock (which has a period of 196 nanoseconds) to increment the ROM Address Register three times. As each word is read into the ROM Instruction Register, it is immediately decoded and control signals go out to enable the appropriate gates and functions. After this (on the third clock), the ROM Address Register is

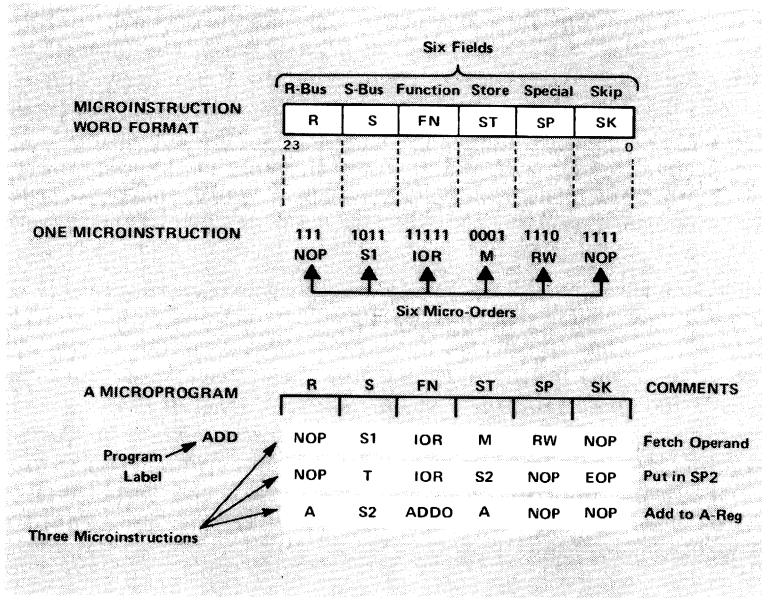


2177-3A

Figure 3. Microprogrammed Generation of Controls

normally forced to address 0000 (the starting address of the fetch phase microprogram), rather than proceeding to 0147.

Figure 4 defines some of the terms relating to a microprogram, and shows the written microprogram resulting from the example described



2177-4A

Figure 4. Elementary Microprogram

in the preceding paragraphs. Note that the microinstruction word format has 24 bits, the least significant bit (0) on the right and the most significant bit (23) on the left. The six fields are distributed left to right as shown; the R field has three bits, the Function field has five bits, and the remaining fields each have four bits. One microinstruction (the first one of the microprogram) is shown in full binary form, with the corresponding mnemonic below each code. The individual command, in mnemonic or binary form, is termed a "micro-order".

The three microinstructions which comprise the microprogram (lower part of figure 4) directly correspond to the three steps which were outlined earlier in the discussion of figure 2. Thus the first microinstruction sends the address in Scratch Pad 1, via the S-bus, to the M-register and requests memory to read (RW) the contents of that

location. The second microinstruction transfers the operand from the T-register (via the S-bus and adder) to Scratch Pad 2. The third microinstruction reads the A-register contents to the R-bus, reads Scratch Pad 2 to the S-bus, adds the two, and stores the result in the A-register. Note that it is possible, due to the nature of the flip-flop elements, to specify an A-register read and an A-register store in the same microinstruction word. (This characteristic is not true of Scratch Pad registers, as explained later.)

Other comments on the microprogram: since the Function field (FN) does not have a NOP (No Operation) code, an Inclusive "OR" (IOR) is specified; this has no effect in these cases because an IOR of one bus with NOP (zeros) on the other bus does not affect the data passing through the adder from the first bus. The EOP (End of Phase) microorder causes the ROM Address Register to return to address 0000 after executing the final microinstruction. It is always located in the microinstruction just prior to the final microinstruction. The program label "ADD" is assigned and used during assembly of the microprograms by a microassembler.

THE 2100 APPROACH

The 2100 Computer is not a wholly microprogrammed machine. In order to maintain software and peripheral compatibility with the earlier 2116, 2115, and 2114 computers, the 2100 was designed to emulate its predecessors. As a result, some of the controls are hardware-generated, rather than originating from the microprogram. Thus the control section is a hybrid firmware/hardware implementation, and as such bears special considerations which will now be pointed out. First, the overall organization will be described, followed by a timing discussion.

PHYSICAL ORGANIZATION

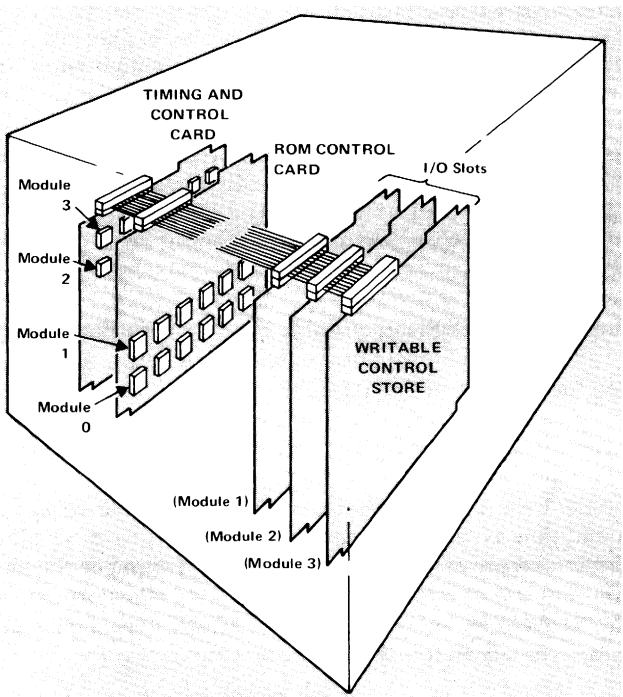
The control store for the 2100 is configured into four modules of 256 words each. This gives a total of 1024 available addressable locations. Module 0 is fully occupied by the basic 2100 instruction set, leaving modules 1, 2, and 3 for extensions to the basic set. Although not recommended, it is also possible to substitute a different module 0 to replace the existing basic instruction set.

Figure 5 shows the approximate layout of modules within the 2100. Note that module 0 consists of six integrated-circuit packs located on the ROM Control card. Module 1, if present would consist of an additional six packs on the ROM Control card. Modules 2 and 3, if present, would each consist of six additional packs located on the Timing and Control card. If modules 2 or 3 are used, a flat cable with a pair of edge-connectors is required to connect these modules to the ROM Control card.

The Hewlett-Packard Floating Point package is an option designed to occupy module 1. If this option is present, only modules 2 and 3 are available for special extensions.

Any mix of modules may be present in the computer, except that module 0 must always be present. For example, modules 0 and 3 could be present, with 1 and 2 absent. Thus it is possible to allow for the future addition of the Floating Point option while proceeding to use special microprogramming in modules 2 and 3. Jumper connections on the ROM Control card are manually set so as to allow proper addressing of the modules. Modules must be physically located in their proper positions, as indicated in figure 5.

Figure 5 also shows the arrangement for adding modules in the form of Writable Control Store cards. These cards are designed to be installed in computer I/O slots, so that they may be loaded (written into) via the I/O system. That is, machine I/O instructions are used to load words consisting of coded microinstructions from the accumulators into the 256 word locations on the card. The locations are then accessed by the ROM Control card by means of a flat cable and top-edge connections, as shown in the figure.



Note: Approximate Representation

2177-5

Figure 5. Control Store Locations

When Writable Control Store modules are used, they are assigned module numbers by manually setting a switch located on each Writable Control Store card. The corresponding hard-wired module locations on the ROM Control or Timing and Control cards then may not be used. If IC packs are present in these modules, they must be removed (except for module 0) before using WCS as those modules.

Whether the modules consist of hard-wired integrated-circuit packs or Writable Control Store, or a mixture of both, the addressing capability limits the maximum number of modules to four.

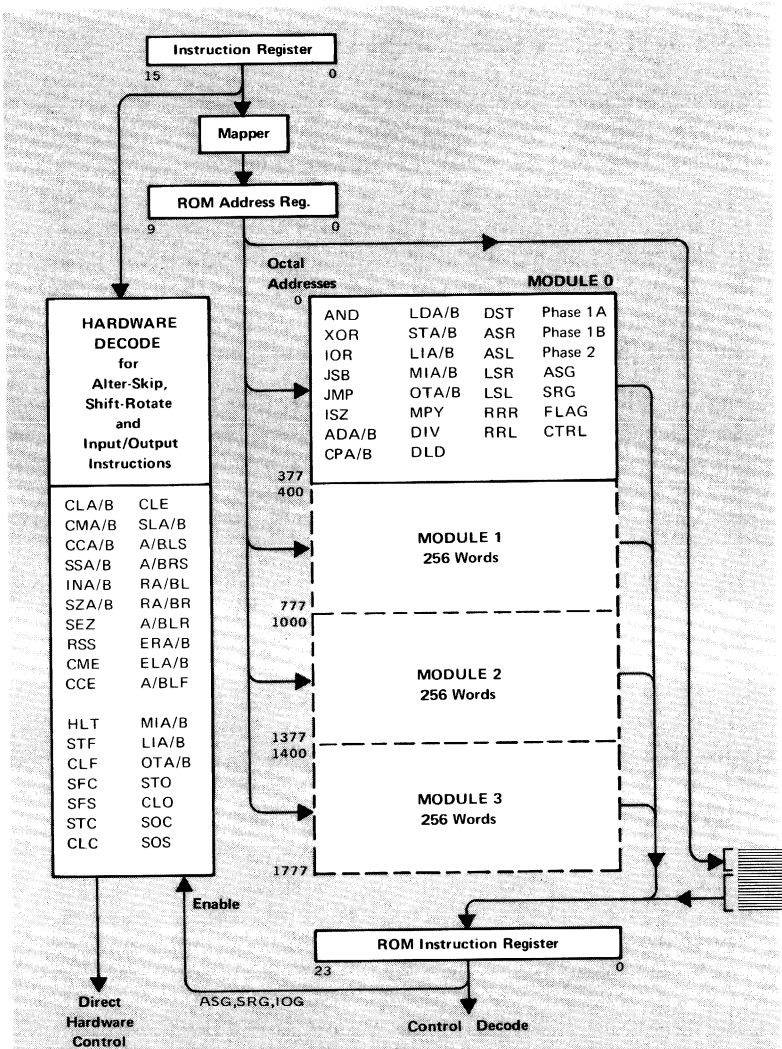
Figures 6 and 7 show the organization in simplified block form. In figure 6, the four module blocks correspond to the four hard-wired module locations shown earlier in figure 5. The basic machine instructions which are microprogrammed are listed in the module 0 block, along with some of the other major routines. The optional modules (1, 2, and 3), are represented with broken-line boxes.

Octal addresses for each module, which come from the ROM Address Register, are listed at the left side of each module block. When one specific location is addressed, the contents of that location are loaded into the ROM Instruction Register.

Note that the ROM Address Register has ten bits (0-9), and the ROM Instruction Register has 24 bits (0-23). In addition to the above routing, the ten address bits can also be sent to an external control store in the I/O section, and the 24 microinstruction bits can be returned via the same cable to the ROM Instruction Register.

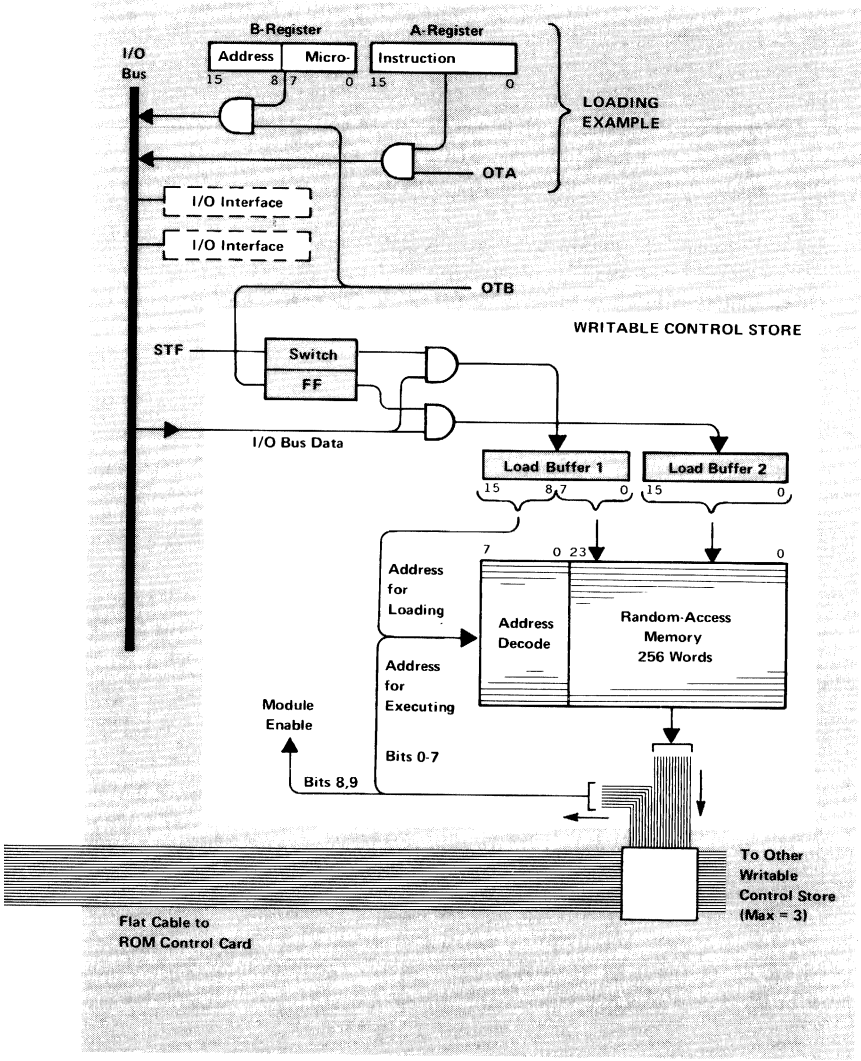
The ROM address, which establishes the starting point for a given routine (e.g., the LDA phase 3 routine) is derived from a mapper, which generates the appropriate ten bits from the machine instruction code in the Instruction Register.

In the block on the left side of figure 6 are listed the machine instructions which are decoded by hardware. While these instructions do not have dedicated execute routines in firmware, the hardware decoders must be enabled by special microprograms; these micropro-



2177-6A

Figure 6. Four-Module Control Store



2177-7

Figure 7. External Control Store

grams also provide appropriately timed reading and storing of registers for those instructions which use registers. The alter-skip and shift-rotate decoders must be enabled by ASG and SRG microprogram routines (respectively). In the I/O group, the decoder for flag instructions (STF, CLF, SFC, SFS, STO, CLO, SOC, SOS) must be enabled by the Flag routine; control instructions (STC, CLC) must be enabled by the CTRL routine; input instructions must be enabled by either LI* or MI* routines; and output instructions must be enabled by the OT* routine. In general, hardware decoding was chosen for these instructions to permit rapid execution. The bit testing and synchronization required for these instructions would have resulted in decreased performance if done purely by firmware.

Figure 7 shows the general configuration of one Writable Control Store unit. As indicated, the storage capability is provided by a Random Access Memory (RAM) of 256-word capacity. Each location stores one 24-bit word, and is addressed by an integral 8-bit address decoder. Since the storage elements are writable (as opposed to read-only), the locations may be loaded by the computer.

A loading example (top of figure 7) assumes that the RAM address is contained in the eight most significant bits of the B-register; the microinstruction is contained in the remaining eight bits plus the 16 bits of the A-register.

The load is initialized by issuing a Set Flag (STF) instruction to the appropriate select code location of the card. This sets a Switch flip-flop to enable the loading of Load Buffer 1. Then an OTB instruction transfers the contents of the B-register to Load Buffer 1, via the I/O bus, and clears the Switch flip-flop. Next, an OTA instruction transfers the contents of the A-register to Load Buffer 2. Finally a Set Control signal (STC, not shown) loads the 24-bit microinstruction from the two Load Buffers into the addressed RAM location.

To execute microinstructions from RAM, the ten address bits from the ROM Address Register are sent out to Writable Control Store via the

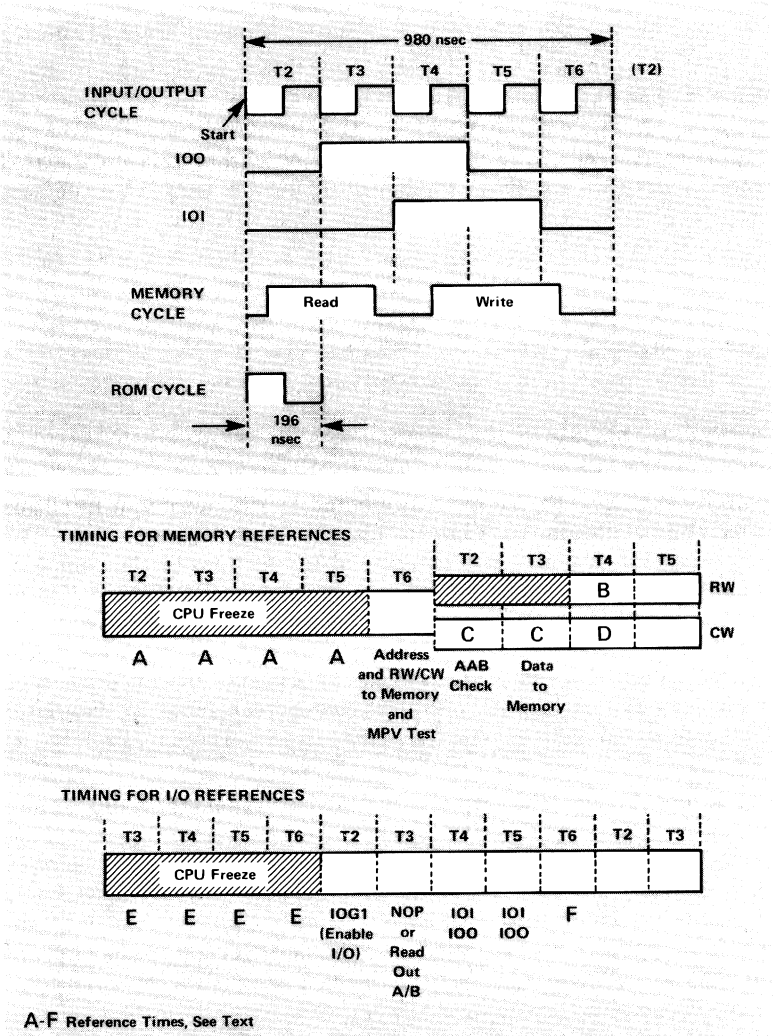
flat cable connected to the card. Bits 8 and 9 of the address enable the particular module according to a manually-set switch, and bits 0-7 are used on the card as an address. This eight-bit address reads out one of the 256 RAM locations via output line drivers to the flat cable. This cable routes the RAM word to the ROM Instruction Register. (Although not shown in figure 7, the RAM output can also be read back to the I/O bus by an input instruction for diagnostic checking purposes.)

TIMING

When writing microprograms for the 2100, it cannot be assumed that microinstructions can always be executed at the maximum rate (one every 196 nanoseconds), although this will commonly be the case. The reason for this is that the memory and I/O sections of the computer operate on a cycle that is five times as long as the ROM cycle (980 vs 196 nanoseconds). Thus whenever a microprogram reference to memory or I/O is made, execution of the microprogram must be synchronized with the memory or I/O cycle. That is, the microprogram must be delayed at certain points until an appropriate point in the longer cycle is detected.

There are two kinds of delays that achieve synchronization. One is automatic, called the CPU freeze. The microprogrammer has no control over this type of delay, but should be aware of its existence. The other type of delay is deliberately microprogrammed by inserting NOP (No Operation) microinstructions. It is the microprogrammer's responsibility to know when to insert a NOP delay and how to apply it properly.

Figure 8 illustrates the basic timing considerations for microprogramming. Note that the I/O cycle begins at the start of time T2 and ends at the end of time T6. (The timing nomenclature purposely omits



2177-8A

Figure 8. Timing Considerations

T0 and T1 in order to maintain documentation uniformity with the large number of existing interfaces.) Output transfers are made during T3 and T4 (IOO signal), and input transfers are made during T4 and T5 (IOI signal).

The memory cycle also begins at the start of T2 and ends at the end of T6. Memory read (or clear) starts early in T2 and ends in T3. Memory write begins in the middle of T4 and ends in T6. Thus, when reading from memory, data will be available in the T-register during T4, and when writing into memory, data must be loaded into the T-register before T4.

The ROM cycle, shown for comparison, can cause five microinstructions to be executed in the same length of time required for one memory or I/O cycle.

The lower part of figure 8 illustrates the various delays required for memory and I/O references.

MEMORY REFERENCES. Memory references are caused when an RW (Read/Write) or a CW (Clear/Write) micro-order is specified in the Special field of a microinstruction. If such a microinstruction is decoded during any time period designated T2 through T5 (see A), a CPU freeze will delay execution of that microinstruction until T6. At that time the address and the RW or CW signal is sent to memory. (For CW, a memory protect violation test is also made at this time.)

During T2 and T3, data transfers take place. For reading (RW), a second CPU freeze occurs if, after issuing the RW, you attempt to read the T-register (Memory Data) while the read half-cycle is not complete. The T-register can be read either by a T or COND in the S-bus field with AAF and BAF both clear in the case of COND. Otherwise, no freeze will occur. This allows the microprogrammer to use the time periods T2 and T3 following an RW for additional microcoding. The data from memory is available at T4 (point B) in either case and must be read out of the T-register in either T4 or T5. A NOP delay is not

necessary. For writing, however, provision is made to test for a possible attempt to store into a protected memory location. Thus the second CPU freeze does not occur during T2 and T3. Instead, two lines of microcoding must be inserted (**C** in figure 8) to be executed at times T2 and T3, before further microprogramming can continue at point **D** (T4). During T2 an addressable A/B check may be made, assuming there is no memory protect violation, and at T3 data is sent to memory. (Refer to ST* listing, addresses 0134, 0135, and 0136.)

Note: The preceding two paragraphs assume that the memory reference operand for RW is in core memory and the operand address for CW refers to a location in core memory. If the operand for RW were in the A- or B-register, no CPU freeze occurs. However this is not a common occurrence; the shorter possible execution time (example: as short as 1.568 microsecond for ADA 1) is not usually listed as a 2100 Computer specification.

I/O REFERENCES. I/O references are caused when an IOG1 micro-order is specified in the Special field of a microinstruction. If such a microinstruction is decoded during T3 through T6 (see **E**), a CPU freeze will delay execution of that microinstruction until T2. At that time the IOG1 signal enables the I/O decoders. The next three time periods (T3, T4, T5) must be coded with NOP microinstructions to allow time for the hardware to fully decode and execute the current I/O instruction. The remainder of the microprogram may then continue at point **F** (T6). However, if the microprogram involves the input or output of data, IOI or IOO micro-orders (respectively) must be encoded in the S-bus or Store fields, respectively, of the microinstruction during T4 and T5. Additionally, in the case of output, the data must be read out of the A- or B-register to the S-bus during T3, T4, and T5, by encoding CAB in the R-bus field and RRS in the S-bus field. In the case of input (LIA/B, MIA/B), CAB must be encoded in the STOR field during T5.

In summary: associate T6 with the start of memory references, and T2 with the start of I/O references.

THE STEPS TO IMPLEMENT A NEW MICROPROGRAM

This handbook tells only how to write microprograms. However, a written microprogram is of no use until it is stored in the appropriate binary form in ROM or RAM. The task of getting your microprogram into the required form is simplified by the use of Hewlett-Packard software which is specially prepared for this purpose. Software documentation is separately available to describe in detail the procedures required to implement your microprograms.

Figure 9 illustrates the overall process. Briefly, the process is as follows:

First, the microprograms are written using the guidelines given in this handbook. These are then punched or recorded in a format suitable for the HP 2100 micro-assembler. The assembler accepts the cards or tape and produces an interim punched tape or disc file, and a microprogram listing. This interim tape or file is then loaded into core memory by a microprogram editor. The editor provides several useful features, including:

a. the ability to output the block of microprograms in memory to Writable Control Store;

b. the ability to examine any word in Writable Control Store;

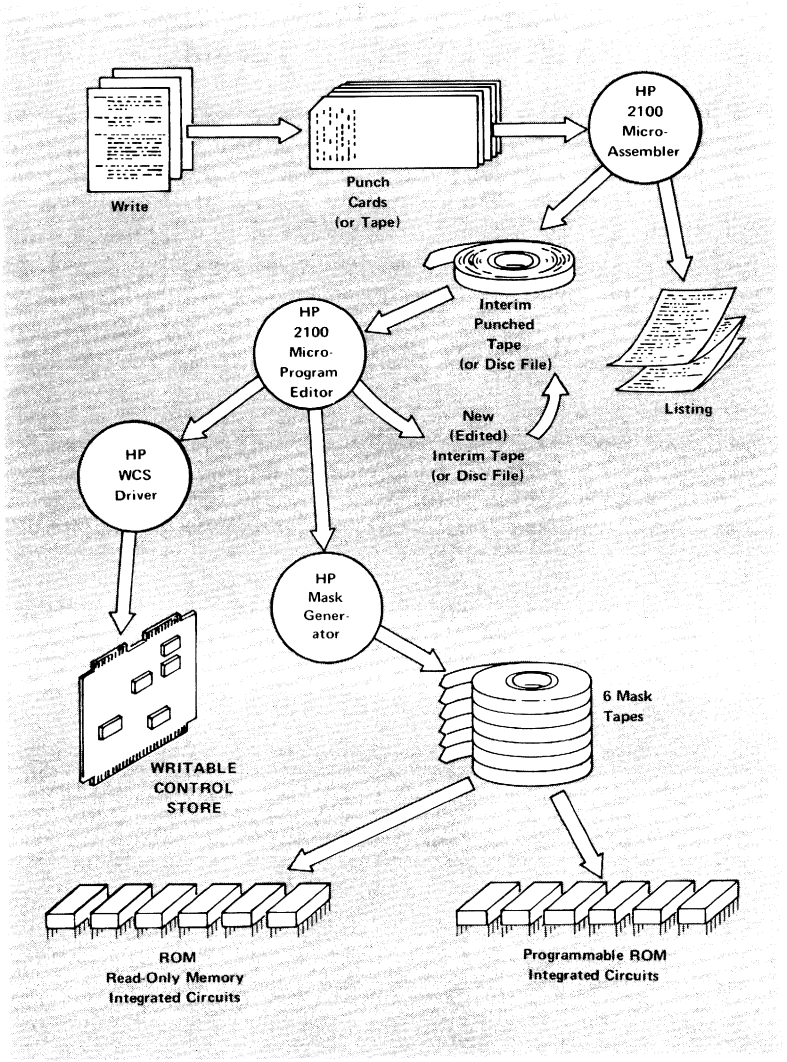
c. the ability to alter any word in memory, and hence in WCS;

d. the ability to produce a new, edited interim tape or file;

e. an output suitable for use by a WCS driver;

f. an output suitable for use by a mask generator program to produce mask tapes for the manufacture of ROM or programmable ROM.

Depending on the desired end result, the HP software will give you either a fully loaded Writable Control Store or a set of six mask tapes. With WCS, the driver used to originally load the card is also callable by FORTRAN and ALGOL programs. This makes it possible to dynamically modify the microprograms.



2177-9

Figure 9. Microprogram Implementation

The six mask tapes would be used to make the six integrated-circuit packages for one hard-wired ROM module. Each IC has a 256-location storage capability for four bits; thus six IC's are required in order to form the 24-bit word length. The programmable ROM version is the same as ROM except in the way it is manufactured.

Once produced, the ROM packages are to be installed in the reserved IC locations on the ROM Control and/or Timing and Control printed-circuit cards. Refer to the 2100 Computer manuals for locations and procedures.

PRACTICAL CONSIDERATIONS

The decision to extend or replace the 2100 firmware bears careful consideration. After all, considerable cost will be involved — not only in hardware investment (primarily the control store hardware), but also in the time requirements for a microprogrammer to acquire sufficient knowledge to be able to generate correct microcode, and then to write, debug, and implement his microprograms. Also to be borne in mind is the fact that much of your software will have to be modified to recognize new function codes.

The benefits to be gained by special microprogramming must necessarily outweigh the cost considerations. While it is beyond the scope of this handbook to enumerate specific applications, some of the basic benefits are listed in the following paragraphs.

GENERAL FACTORS

SPEED. Microprogramming can increase system speed in many ways. A frequently-used software subroutine, for example, will execute many times faster when implemented as a microprogram. Since six additional CPU registers become accessible (see section 2), the number of memory references can be greatly reduced. This can be particularly significant in

real-time systems, or systems which are compute-bound (i.e., I/O runs faster than computation in a serial input-compute-output application).

MEMORY SPACE. By converting software routines into firmware, core space is freed for other purposes. The routines remain instantly callable, as opposed to the technique of relegating routines to mass storage in order to gain core space.

SPECIAL FUNCTIONS. The software instruction set can be enriched to perform functions that are application oriented. Thus the general-purpose computer can become a special-purpose machine, uniquely adapted to a specific environment. (However, due to hardware restrictions, the 2100 cannot be made to emulate other systems.) Because of the relative inaccessibility of firmware contents (as compared to software), proprietary packages can receive a high degree of security.

EXPANDED CAPABILITY. Through microprogramming, six additional registers become accessible. Software instructions may be invented to reference and use these registers. In addition, due to the three-operand format of the microinstruction word, instructions can be created which perform some function with the contents of two registers and store the result in a third register. A flag bit, also not otherwise accessible, may also be used.

As can be expected, certain restrictions limit the operations that may be performed. These are described later in section 5. However, one consideration should be mentioned at this point: a microprogram normally inhibits all I/O interrupts until it is fully executed. (A CJMP may be used to circumvent this restriction.) This fact can become significant if a routine is very long — e.g., if it contains potentially endless loops or numerous links to other microprograms. Microprograms should be kept short. Direct Memory Access (DMA) is not, however, held off by the microprogram.

WCS/ROM IMPLEMENTATIONS

Deciding whether to use Writable Control Store or a permanent Read-

Only Memory again involves the factor of cost. WCS is convenient, ready to install as a plug-in unit, but is more costly than ROM. Although ROM involves a manufacturing step, the cost factor is usually decisive when a quantity of units are to be made. For low quantities, a programmable Read-Only Memory is often a suitable compromise. It is relatively easy to produce, although the cost per unit for large quantities is somewhat higher than for ROM.

The primary advantage of WCS is that it is modifiable — even dynamically modifiable. An executing software program can actually alter the functions performed by a microprogram during run time, based upon any internal or external stimuli that may be desired. The modifiability feature also means that a preliminary microprogram may be tested and debugged under actual run conditions. Execution speed is the same whether operating from WCS or ROM. For this reason, WCS is frequently used to check out microprograms before they are permanently committed to ROM. This also permits division of labor on microprogramming projects; several microprogrammers can independently test their microprograms before integration into a total set.

The disadvantages of WCS, apart from cost, are that: 1) each module uses up one I/O slot, and 2) the stored information is volatile — i.e., the contents are lost in the event of a power failure. This requires an automatic restart routine to be written which would reload WCS when power is restored. If automatic restart is used, firmware routines must be short enough to run to completion and still allow time for power fail interrupt and execution of the service routine before power is gone.

GENERAL CONTROL FUNCTIONS

Figures 10 through 13 represent a four-part block diagram of the 2100 Computer. Each part corresponds to one of the major component parts of a computer, as illustrated earlier in figure 1.

The block diagram is specifically configured to show where in the machine the various micro-orders have their effect. Later, under the heading, "Effects of Microinstruction Fields", most of the important controls will be discussed. First, each of the blocks in the four parts will be described briefly.

CONTROL

The control section of the computer includes the read-only memory and its addressing and decoding logic. Refer to figure 10.

INSTRUCTION REGISTER. The Instruction Register is 16 bits wide. It accepts software instruction codes, in binary, from memory via the S-bus. The microprograms are responsible for reading memory data onto the S-bus, and for storing the S-bus data into the Instruction Register. Major destinations of Instruction Register outputs are: 1) the Phase 3 Mapper; 2) the SRG/ASG Decoder, for register reference instructions, and; 3) the I/O Instruction Decoder (see figure 13), for I/O group instructions. Other outputs are used for phase control, operand addressing (IR0-9), and A/B-register references (IR11).

PHASE CONTROL. The Phase Control logic controls the state of the computer. Operation begins in the fetch phase, and thereafter the Phase

Control logic determines the next state (or phase) based upon the current state and the type of instruction (IR11-15), plus indirect and interrupt detection logic (not shown). The End-of-Phase (EOP) signal from the microprogram commands the Phase Control logic to switch to the next phase. Phase 1A is the fetch phase; phase 1B is the interrupt fetch phase; phase 2 is the indirect phase; phase 3 is the execute phase.

PHASE 3 MAPPER. The Phase 3 Mapper is enabled by bits 4 through 15 of the Instruction Register and the SPH3 (Set Phase 3) signal from the Phase Control logic. The mapper accepts these bits from the Instruction Register and translates this information into a 10-bit ROM address. This address is the starting location in ROM for the microprogram which executes the current instruction.

SRG/ASG DECODER. This decoder provides the necessary hardware controls to enable execution of phase 3 for shift-rotate and alter-skip group instructions. It, in turn, is enabled by an SRG or ASG signal (not shown) from the microprogram (i.e., Special field decoder).

ROM ADDRESS REGISTER. The ROM Address Register supplies one address at a time to ROM. Its contents can be forced to a particular value from one of seven sources and, unless inhibited by RPT (Repeat) or overridden by a jump, the content increments on every clock cycle (196 nanoseconds). The ROM Address Register is loaded for phase 1A by applying no data input to the register and enabling the parallel-load terminal with EOP (End of Phase). This forces an address of 0000 (all addresses are in octal), which is the starting location of the fetch routine. For phases 1B or 2, EOP again enables the parallel-load terminal, and addresses 4 or 14 are forced. For phase 3, the address from the Phase 3 Mapper is loaded into the register, again with EOP; this is the starting address for the routine which executes the instruction which has been fetched (in phase 1A) into the Instruction Register. For microprogram jumps (JMP, JSB, and CJMP) a 10-bit address from the ROM Instruction Register, consisting of bits 0 through 7 plus 12 and 17, is forced into the ROM Address Register. The JMP, JSB, and CJMP signals (instead of EOP) enable the parallel-load terminal. The

contents of the Save Register (discussed next) can also be loaded into the ROM Address Register; this loading is caused by an RSB (Return from Subroutine) signal, which also (though not shown) enables the parallel-load terminal.

SAVE REGISTER AND JSB FF. The Save Register provides a means for returning from a microprogram subroutine. It is controlled by the JSB (Jump to Subroutine) flip-flop. Normally, the JSB flip-flop is clear. In this condition, the Save Register automatically copies the content of the ROM Address Register on every cycle. Then, when a JSB micro-order appears in the Function field of a microinstruction, a JSB signal sets the JSB flip-flop. This inhibits the Save Register from further copying of ROM Address Register contents; thus the return address for the subroutine is saved. Later, when the microprogram completes the subroutine, it generates an RSB (Return from Subroutine) signal. This signal forces a parallel-load of the Save Register contents into the ROM Address Register, and clears the JSB flip-flop. The microprogram thus continues following the point where the JSB was given.

READ-ONLY MEMORY. Four modules of ROM provide 256 locations each, for a total of 1024 possible locations. Module 0 (octal addresses 0 through 377) contains the microprograms for the basic instruction set. The remaining three modules are optional. As explained in section 1, modules can be located in the I/O section and connected to the ROM Address Register and ROM Instruction Register by direct cabling.

ROM INSTRUCTION REGISTER. This register receives the contents of the ROM location addressed by the ROM Address Register. The ROM Instruction Register is 24 bits wide, divided into six fields as shown. Mostly, the outputs of the register go directly to the ROM field decoders. The EOP micro-order, however, is separately decoded, and the address bits for a microprogram jump (0-7, 12, 17) come directly from the register.

ROM FIELD DECODERS. There is a separate decoder for each field of the ROM word: R-bus, S-bus, Function, Store, Special, and Skip. The signals they generate depend on the coding of micro-orders in each

field. In addition, the output of the R-bus and Store fields can be affected by AAF and BAF input signals and the A/B bit (IR11) from the Instruction Register. The Function field can be affected by bit 0 of the A-register or a carry-out (COUT) from the arithmetic logic unit (ALU). A full discussion of the effects of microinstruction fields is given later in this section.

MICRO-SKIP TEST LOGIC. A microinstruction skip is accomplished by inhibiting most of the decoder outputs on the next clock cycle. This will cause the next microinstruction to have no effect. Some signals, such as the entire R- and S-bus field signals, are permitted to occur since they do not change anything. The skip signal to the inhibit gates is generated by the micro-skip test logic. The decoded Skip field (SK) specifies which condition is to be tested for a possible skip. Nine such input conditions (see figure 11) are shown: COUT, TBZ, OVF, FLG, ALU0, ALU15, Ctr = 17, AAF, BAF. (These are defined later.) An RSS (Reverse Skip Sense) causes the skip to occur if the tested condition is false. For example, TBZ alone would cause a skip if the T-bus is zero; TBZ with RSS would cause a skip if the T-bus is not zero.

ARITHMETIC LOGIC

The arithmetic logic section contains the data registers and data manipulating logic. Refer to figure 11.

ADDRESSABLE A/B. The Addressable A/B logic contains two flip-flops, the outputs of which are designated as AAF and BAF (Addressable A flip-flop and Addressable B flip-flop). One or the other of these flip-flops may be set (or neither), but not both. If AAF is set, it indicates that the A-register is being addressed as a memory reference. Similarly, if BAF is set, it indicates that the B-register is being referenced as a memory location. Either flip-flop is conditionally set by an AAB or RW signal from the Special field of the microinstruction. Thus, AAB or RW will set AAF if T-bus bits 1 through 14 are "0" and ALU0 is "0" (this is the address of the

A-register), and will set BAF if T-bus bits 1 through 14 are "0" and ALU0 is "1" (the address of the B-register).

A-/B-REGISTERS. The A-register and B-register are 16-bit accumulators which are accessible to both software programming and firmware microprogramming. They are loaded from the T-bus by signals from the Store field, and can be read out to the R-bus by signals from the R-bus field. The two registers are capable of being right-shifted as a 32-bit quantity, in a single clock cycle, by specifying R1 (Right one place) in the Special field, with ARS, CRS, LGS, or MPY in the Function field and B specified in the R-bus and Store fields. The B-register contents (most significant 16 bits) are shifted by routing through the ALU and the shifter; the A-register shifts internally in the register itself. Either register may also be individually shifted left or right through the ALU and the shifter, by specifying L1 or R1 in the Special field. All of these shifts are microprogrammable, and are in addition to the non-microprogrammed shifts generated by the shift-rotate group of instructions. (See note following Q- and F-register description.)

Q-/F-REGISTERS. The Q-register and F-register are 16-bit accumulators which, in the basic computer, are not accessible to software programming. Special microprograms must be written if it is desired to have instructions which reference these registers. Under microprogram control, the Q- and F-registers are loaded from the T-bus and read out to the R-bus. The two registers are capable of being left-shifted as a 32-bit quantity, in a single clock cycle, by specifying L1 (Left one place) in the Special field, with ARS, CRS, LGS, or DIV in the Function field and F specified in the R-bus and Store fields. The F-register contents (most significant 16 bits) are shifted by routing through the ALU and the shifter; the Q-register shifts internally in the register itself. The Q-register may also be individually shifted left or right through the ALU and the shifter, by specifying L1 or R1 in the Special field. Since the F-register is used as a fence register by the memory protect feature, any microprograms which use the F-register must save the contents on entry and restore the contents on exit, assuming that the system does use memory protect.

Note: Special care should be taken when microprogramming long shifts on the B-/A-registers and any shifts on the F-/Q-registers, particularly if attempting unconventional operations. For example, if you do not intend to store the result of a long shift (i.e., do not specify B or F in the Store field), the A-register would be shifted anyway, but not the B-register; whereas, in the case of the F- and Q-registers, neither would be shifted. Also note that the F-register is not intended to be shifted by itself as an individual register. This is because the F-register is permanently linked with the Q-register. Thus on left shifts, the most significant bit of the Q-register automatically shifts into the least significant bit of the F-register; furthermore there is the possibility of an "OR"-tie with the Flag content if the LWF micro-order is specified. This can, of course, be circumvented by clearing the unwanted low-order bits; however, considerable caution and close study of the computer logic are advisable before attempting such operations. Right-shifting of the F-register will not have any serious consequences.

P-REGISTER. The P-register is a 16-bit program counter, which contains the memory address of the next instruction to be fetched. It is initially loaded manually from the front panel. Thereafter, in run mode, its contents are incremented at the start of each execute phase (phase 3) by an INP (Increment P) signal from the Phase Control logic. During the execute phase of skip instructions (alter-skip group) the register may be incremented a second time by an INP signal from the Skip Carry logic. During the execute phase of JMP and JSB instructions, a different address from Scratch Pad 1 (SP1) is loaded into the P-register. The address in SP1 is either a direct address obtained during the fetch phase (from Instruction Register bits 0-9, conditionally combined with P-register bits 10-15, depending on the state of page bit IR10) or a final indirect address obtained from the T-register during an

indirect phase. The transfer from SP1 to P-register occurs by way of the S-bus, ALU, and T-bus.

SCRATCH PAD REGISTERS. Like the Q- and F-registers, the four Scratch Pad registers (SP1, SP2, SP3, SP4) are accessible to software only by special microprogramming. These are 16-bit registers, normally used for temporary storage of information during the execution of a microprogram. They are loaded from the T-bus, and can be read onto the S-bus. Information is not normally carried over in these registers from one execute phase to another; only the A- and B-register have this capability. Unlike the A-, B-, Q-, F-, and P-registers, which use edge-triggered storage elements, the Scratch Pad storage elements are latches. This means that it is not possible to read the contents of one Scratch Pad and store back into that Scratch Pad in the same cycle (i.e., same microinstruction). Another register, such as another Scratch Pad would have to be specified for storing, if it is desired to preserve the T-bus information.

MULTIPLEXER. A four-input multiplexer is used to select one of four registers (A-, B-, Q-, and F-registers) for reading onto the R-bus. The multiplexer is controlled by the R-bus field of the decoded microinstruction. Note that, by specifying an RRS (Read R-bus to S-bus) in the S-bus field, it is possible to read the output of the multiplexer onto the S-bus, as well as onto the R-bus.

ARITHMETIC LOGIC UNIT (ALU). The ALU performs one of eight arithmetic or logical functions on the combined R- and S-bus inputs. If nothing is read onto one of the buses, its state is all-zero, and the specified function essentially operates on only the remaining bus input. The function is specified by the Function field of the decoded microinstruction. The eight functions are: IOR ("inclusive OR"), XOR ("exclusive OR"), AND (logical "AND"), NOR ("OR" and complement), ADD (two's complement add), SUB (subtract S-bus from R-bus, two's complement), INC (add S-bus and R-bus, increment the result), and DEC (subtract S-bus from R-bus, one's complement; decrement R-bus if S-bus is zero). The output bits of the ALU are designated ALU0 through ALU15.

SHIFTER. The 16 ALU bits are applied to the shifter, which routes each bit onto the numerically corresponding T-bus line unless a shift signal is applied by either the SRG/ASG decoder or the Special field of the decoded microinstruction. The SRG/ASG decoder can generate all three basic shift signals: R1 (Right one), L1 (Left one), and L4 (Left four). The Special field decoder can supply two of these: R1 and L1. The various types of shifts (arithmetic, logical, etc.) are enabled by controlling the data bit that is inserted into either the high end (ALX16) or the low end (LSI) of the shifter. See Shift Linkage paragraph.

SHIFT LINKAGE. The shift linkage logic takes some combination of three input data bits (Flag, Extend, and either ALX16 or LSI), and outputs one bit to either ALX16 or LSI depending on the direction of the shift. The shift linkage is controlled by shift-type signals from either the SRG/ASG decoder or the Function field.

RFE LOGIC. The RFE (Rotate Flag and Extend bits) logic exchanges the contents of the Flag and Extend flip-flops on receiving an RFE signal from the Function field decoder.

FLAG LOGIC. The Flag logic controls the state of the Flag flip-flop, which is used by microprograms for temporary storage of a single data bit or status bit. This is not the same flag referred to in the I/O group. The state of the bit (FLG) may be tested by the micro-skip test logic and, as mentioned above, its content may be exchanged with the Extend bit content. (FLG was also used in the shift linkage for implementing the shift and rotate instructions of the basic instruction set.) The Flag flip-flop may be set or cleared by SFLG or CFLG signals from the Function field decoder, or an LWF (Link with Flag) micro-order may be used to cause the Flag flip-flop to save the bit shifted off either end of a word by the shifter. That is, if shifting left (L1), the Flag will assume the state of ALU15 (which would be lost from the word shifted to the T-bus). Similarly, if shifting right (R1), the Flag will assume the state of ALU0. The LWF micro-order also inserts the Flag content into the vacated bit position at the other end of the shifted word.

EXTEND LOGIC. The Extend logic controls the state of the Extend flip-flop. The bit contained in this flip-flop is accessible to software by way of the shift-rotate and alter-skip groups of instructions. For micro-programming purposes, the Extend bit will be set by a carry out (COUT) from the ALU, when enabled by an ADDO (ADD, with Overflow enabled) or an INCO (Increment, with Overflow enabled) micro-order in the Function field. Microprograms cannot directly set or clear the Extend bit; indirectly it may be controlled by rotating with the Flag bit, which is directly controllable (see RFE and Flag logic).

OVERFLOW LOGIC. The Overflow logic controls the state of the Overflow flip-flop. The state of the bit contained in this flip-flop can be controlled by software (STO and CLO instructions) and may be tested for skips (SOS and SOC instructions). For microprogramming purposes, the Overflow bit may be directly set or cleared by SOV and CLO micro-orders, and may be enabled to check for possible ALU overflow by ADDO and INCO micro-orders. ALU overflow normally is tested by comparing ALU15 with bit 15 of the R- and S-buses ("anded"); if a sign change occurs, Overflow will be set. The Overflow bit output (OVF) is one of the conditions which may be tested by the micro-skip test logic.

COUNTER. A five-bit hardware counter is available for use by micro-programs. Typically it would be used for loop counting. The four least significant bits (0-3) may be parallel-loaded from the S-bus (with bit 4 cleared) by a CNTR micro-order in the Special field, and all five bits may be read out to the S-bus by a CNTR micro-order in the S-bus field. The counter is incremented by ICTR or CTRI in the Skip field. The latter of these two micro-orders, CTRI, enables the micro-skip test logic to test the counter contents for a full count. A full count is represented by all ones in the four least significant bits (i.e., octal 17).

MEMORY

The memory section of the computer contains the memory core modules, plus the memory addressing logic, timing circuits, and a data holding register. Refer to figure 12.

M-REGISTER. The M-register contains the memory address of the core location which is to be accessed on a given memory cycle. The register has 15 bit positions. (Bit 15 of an address word is used as an indirect address indicator and thus never forms part of an address.) Addresses may be loaded from the S-bus by an M micro-order in the Store field, and the current M-register contents may be read out to the S-bus by an M micro-order in the S-bus field. However, the main output of the M-register is to the Memory Address Decoder. The Direct Memory Access option may also load the M-register.

MEMORY ADDRESS DECODER. The Memory Address Decoder converts the address bit pattern in the M-register to a selection of matrix-arranged addressing lines.

X-Y DRIVER/SWITCHES. A set of Driver/Switch circuits, selected by the lines from the Memory Address Decoder, supplies the current required to access one particular memory location. Timing signals cause the current direction to reverse (for storing) from the "read" direction to the "write" direction.

CORE. Up to four core modules may be used in optional configurations to provide various storage capacities from 4K (4096 words) to 32K (32,768 words). For reading, the X-Y Driver/Switches provide a read current to the 17 cores in the selected location; the Sense Amplifiers (SA) detect the state changes caused by the read currents. For writing, the X-Y Driver/Switches provide a write current to the 17 cores in the selected location; the bit pattern supplied by the Inhibit Drivers (described next) causes the T-register contents to be copied into the core location. (The 17th bit, used by parity-checking logic, is being ignored in these discussions since parity generation and checking are not microprogrammable functions.)

INHIBIT DRIVERS. The Inhibit Drivers (ID) are used when writing data into core. During the write operation, the Driver/Switches attempt to drive all cores in the selected location to the "1" state. The Inhibit Drivers supply opposing current to any core which has a corresponding "0"-state bit in the T-register. Thus the T-register contents will be copied into the core location.

SENSE AMPLIFIERS. The Sense Amplifiers (SA) are used when reading data from core. During the read operation, the Driver/ Switches drive all cores in the selected location to the "0" state. Any cores that change state obviously were "1"s. The Sense Amplifiers detect this change of state, and set the corresponding bit in the T-register. The T-register is always cleared by hardware at the start of any read operation.

T-REGISTER. The T-register is a 16-bit register that holds the data that is read out of and written into a memory location during memory read and write operations. It is automatically loaded with the contents of a core location during the read operation of a read/write (RW) cycle, or will be cleared by the clear operation of a clear/write (CW) cycle. Under microprogram control its contents may be loaded from or read onto the S-bus. A T micro-order in the Store field loads the register, and a T micro-order in the S-bus field reads the contents. The Direct Memory Access option may also load or read the T-register.

MEMORY TIMING. For microprogramming purposes, the memory timing circuits may be viewed as the block which translates the RW (Read/Write) and CW (Clear/Write) micro-orders into memory cycle sequences. A typical microprogrammed read would consist of specifying RW in the Special field of one microinstruction, and then reading the contents of the T-register (T in S-bus field) in the next microinstruction. As explained in section 1, a CPU freeze will occur, so that the read signal will not actually occur until data is present in the T-register. A typical microprogrammed write would consist of specifying CW in the Special field and, in the same microinstruction, M in the Store field. This will load the S-bus data (the address) into the M-register and start the clear operation. Then, after one microinstruction for memory-protect violation checking (or a forced skip), a T in the Store field may be used to load the T-register from the S-bus. Note: the T-register may not be loaded by a microinstruction prior to issuing CW, since a DMA transfer could destroy the data.

INPUT/OUTPUT

The input/output section includes I/O instruction decoding and device interfacing. Refer to figure 13.

I/O INSTRUCTION DECODER. As mentioned in section 1, the decoding for I/O instructions is mostly done in hardware. Thus all Instruction Register bits, IR0-IR15, are shown applied to the I/O instruction decoder. The decoder is enabled by an IOG1 micro-order in the Special field. The control and timing logic has not been shown in detail, since it is not particularly relevant to microprogramming.

S-REGISTER AND DISPLAY. In the run mode, the S-register may be addressed by I/O instructions as select code 01. Thus its contents may be transferred to and from the S-bus in the same manner as I/O interface data (see next paragraph). The Display register, in the run mode, is locked to the S-register, thus providing a convenient means of display and modification via the front panel.

I/O BUS. For input and output of data, it is necessary for the microprogram to gate the data between the S-bus and the I/O bus at the appropriate time. (Refer to figure 8 for timing details, explained earlier.) Hardware-decoded controls will take care of the remaining operations involved in data transfers (i.e., addressing the interface cards via select codes, and transferring data between the cards and the I/O bus). An IOO micro-order in the Store field will read the S-bus onto the I/O bus, and an IOI micro-order in the S-bus field will read the I/O bus onto the S-bus.

CENTRAL INTERRUPT REGISTER. Whenever an interrupt occurs, the interrupt address (select code) is loaded into the 6-bit Central Interrupt Register. The contents of this register are used by the PH1B (phase 1B, or interrupt fetch phase) microprogram, for transferring computer control to an interrupt subroutine in software.

SKF SIGNAL. The SKF (Skip Flag) signal is one of the conditions which the Skip Carry logic can use for a possible software instruction skip. It is not used in microprogramming.

EFFECTS OF MICROINSTRUCTION FIELDS

Most of the micro-orders have been mentioned in the preceding block-diagram discussion. In order to present a perspective from the point of view of the microprogram, the following descriptions provide a summary of controls that are available to the microprogrammer. Refer to figures 10 through 13. A complete listing of the microprogram language instruction set is given in section 4.

R-BUS FIELD

The R-bus field, having only three bits, is the smallest of the six fields. Basically, it selects one of the four R-bus registers (A, B, Q, F) for reading onto the R-bus. This is done by specifying an A, B, Q, or F micro-order. Also, the A- and B-registers may be conditionally selected by specifying AAB or CAB. For AAB, A or B will be selected depending on whether an AAF or BAF signal is being supplied by the Addressable A/B logic; if neither is present, the A-register will be read. For CAB, the selection depends only on the state of IR11 ("0" for A, "1" for B).

S-BUS FIELD

The S-bus field basically selects one of the five S-bus registers (P, SP1, SP2, SP3, SP4) for reading onto the S-bus. The respective micro-orders are P, S1, S2, S3, and S4. Also (in figure 11), an RRS micro-order can read the R-bus value to the S-bus, and CNTR can read the counter contents (five bits) to the S-bus.

In figure 10, note that a COND (Conditional) micro-order can force the R-bus field to conditionally read the A- or B-register, depending on whether AAF or BAF is set (AAB must be coded in the R-bus field for this case); hardware logic also enables RRS so that the data will be routed to the S-bus. If neither AAF nor BAF is set, then the T-register contents are read onto the S-bus. Also, as shown in figure

10, a CR or CL micro-order can read an eight-bit constant from the ROM Instruction Register (RIR0-7) to the S-bus. The CR micro-order reads this data to the right half of the bus (bits 8 through 15), and CL reads it to the left half (bits 0 through 7). In each case the unused bits are cleared. An ADR micro-order reads Instruction Register bits 0 through 9 (normally the operand address for memory reference instructions) to the S-bus, and (not shown) conditionally may also read P-register bits 10 through 15. See ADR micro-order definition.

As shown in figures 12 and 13, the S-bus field can also read the T-register (T micro-order), the M-register (M micro-order), the I/O bus (IOI micro-order), and the Central Interrupt Register (CIR micro-order).

FUNCTION FIELD

The Function field controls operations in five separate areas: the ALU, the Overflow logic, the Flag logic, the shift linkage logic, and the micro-jump logic. The following five paragraphs summarize the controls in these areas.

ALU OPERATIONS. Eight functions can be performed by the ALU on command of micro-orders from the Function field. These are: IOR ("inclusive OR"), XOR ("exclusive OR"), AND (logical "AND"), NOR (logical "NOR"), ADD (two's complement add), SUB (two's complement subtract), INC (add R- and S-bus inputs and increment the sum), DEC (one's complement subtract, or decrement R-bus input). The micro-orders ADDO and INCO are the same as ADD and INC except for the additional logic they enable (see next paragraph).

OVERFLOW. Two of the Function field micro-orders directly control the Overflow flip-flop: SOV sets Overflow, and CLO clears Overflow. The ADDO and INCO micro-orders enable the Overflow logic, so that Overflow can be set by arithmetic overflow in the ALU. Note that

ADDO and INCO also enable the Extend logic for the software ADA/B instruction. Extend is then set by a carry out (COUT) from the ALU.

SHIFT OPERATIONS. Long shifts (32 bits) and multiply and divide steps can be specified by ARS, CRS, LGS, MPY, and DIV in the Function field. These are complex operations, involving other logic (not shown in figure 11) in addition to the essential shift linkage (shown). Refer to definitions for full explanation.

FLAG LOGIC. The Flag flip-flop can be directly set or cleared by SFLG or CFLG micro-orders in the Function field. By specifying LWF (Link with Flag) in the Function field, plus L1 or R1 in the Special field, the content of the Flag flip-flop can be rotated with the ALU output (i.e., circular shift), left or right. Also, RFE may be used to exchange Flag and Extend contents.

MICRO-JUMPS. As shown in figure 10, a JMP, JSB, or CJMP micro-order will force a jump address into the ROM Address Register. This address is taken from the eight least significant bits of the current microinstruction (RIR0-7) plus RIR12 and RIR17. The Special and Skip fields cannot be used in this case. For special reasons that will be explained in section 3, the four least significant bits of the S-bus (SB0-3) are "OR"-tied into the ROM Address Register. Additionally, for JSB only, the Save Register will be caused to save the former ROM Address Register contents; later, an RSB (Return from Subroutine) micro-order in the Function field can restore the former address from the Save Register to the ROM Address Register.

STORE FIELD

Mostly, the Store field is used to store the data on the T-bus into a specifiable register. Four exceptions (next paragraph) involve the S-bus instead of the T-bus. As shown in figure 11, the nine R- and S-bus registers (A, B, Q, F, P, SP1, SP2, SP3, SP4) can be loaded from the T-bus on command from the Store Field decoder, by A, B,

Q, F, P, S1, S2, S3, or S4 micro-orders. The A- and B-registers may also be loaded conditionally by AAB or CAB micro-orders. For AAB, the selection will depend on whether an AAF or BAF signal is being supplied by the Addressable A/B logic. If neither is present, no store will occur. For CAB, the selection depends on the state of IR11 ("0" for A, "1" for B).

The IR, T, and M micro-orders store the data on the S-bus into (respectively) the Instruction Register, T-register, or M-register. (Refer to figures 10 and 12.) The IOO micro-order gates the data on the S-bus onto the I/O bus (figure 13).

SPECIAL FIELD

The Special field provides for several miscellaneous operations, described in the following paragraphs. Special functions provided for the SRG/ASG decoder are disregarded.

SKIP SENSE. An RSS (Reverse Skip Sense) in the Special field will cause the micro-skip test logic to test for a condition which is the complement of that specified in the Skip field (e.g., skip on cleared condition instead of set condition).

ADDRESSABLE A/B. An AAB or RW micro-order enables the setting of AAF or BAF, depending on whether ALU0 is "0" or "1", respectively, with T-bus bits 1 through 14 all "0". When AAF or BAF is set as a result of an RW (Read/Write) micro-order, this indicates that the data to be read is in the A- or B-register; thus a memory fetch (and synchronization) will not be necessary, although it is still executed. The following microinstruction, which normally contains AAB and COND in the R- and S-bus fields respectively, makes the decision as to where to get the data.

SHIFTS. An L1 (Left one) or R1 (Right one) micro-order in the Special field causes the shifter to shift the ALU data bits left or right one place onto the T-bus. By using LWF (Link with Flag) in the Function field, the vacated bit position can be filled with the current

bit state of the Flag flip-flop. For long shifts (ARS, CRS, LGS, MPY, and DIV), L1 and R1 enable shifting of the higher order word. (The lower order word is shifted internally in the appropriate register.)

COUNTER. The CNTR micro-order in the Special field causes bits 0 through 3 of the S-bus to be loaded into the counter, and bit 4 of the counter to be cleared.

MEMORY. An RW micro-order causes the memory to execute a read/write cycle; this will place memory data into the T-register. A CW micro-order causes the memory to execute a clear/write cycle, which stores the contents of the T-register into memory. The CW command requires a "true" skip condition (specified in the Skip field); otherwise the command will be inhibited.

SKIP FIELD

The Skip field permits one of several conditions to be specified for possible microinstruction skip decisions. Some special operations (see next paragraph) are also coded in this field. Ten skip conditions are shown in figures 10 and 11, specified by the following micro-orders: CTR (skip if counter bits 0-3 are all "1"), CTRI (same as CTR, but also increment the counter), NEG (skip if ALU15 is "1"), ODD (skip if ALU0 is "1"), COUT (skip if there is a carry out from the ALU), FLG, (skip if Flag is set), OVF (skip if Overflow is set), TBZ (skip if T-bus is all "0"), AAB (skip if AAF or BAF is set), and NAAB (skip if the T-bus value is not the address of the A- or B-register). In addition, although not shown, there is a test for no memory-protect violation (NMPV) and an unconditional (UNC) skip micro-order.

There are three special functions available in the Skip field. The RPT micro-order causes the next microinstruction to be repeated until a condition specified in the Skip field of that microinstruction is met. The ICTR micro-order increments the counter. The EOP micro-order commands the Phase Control logic to switch to the next phase after executing the next microinstruction.

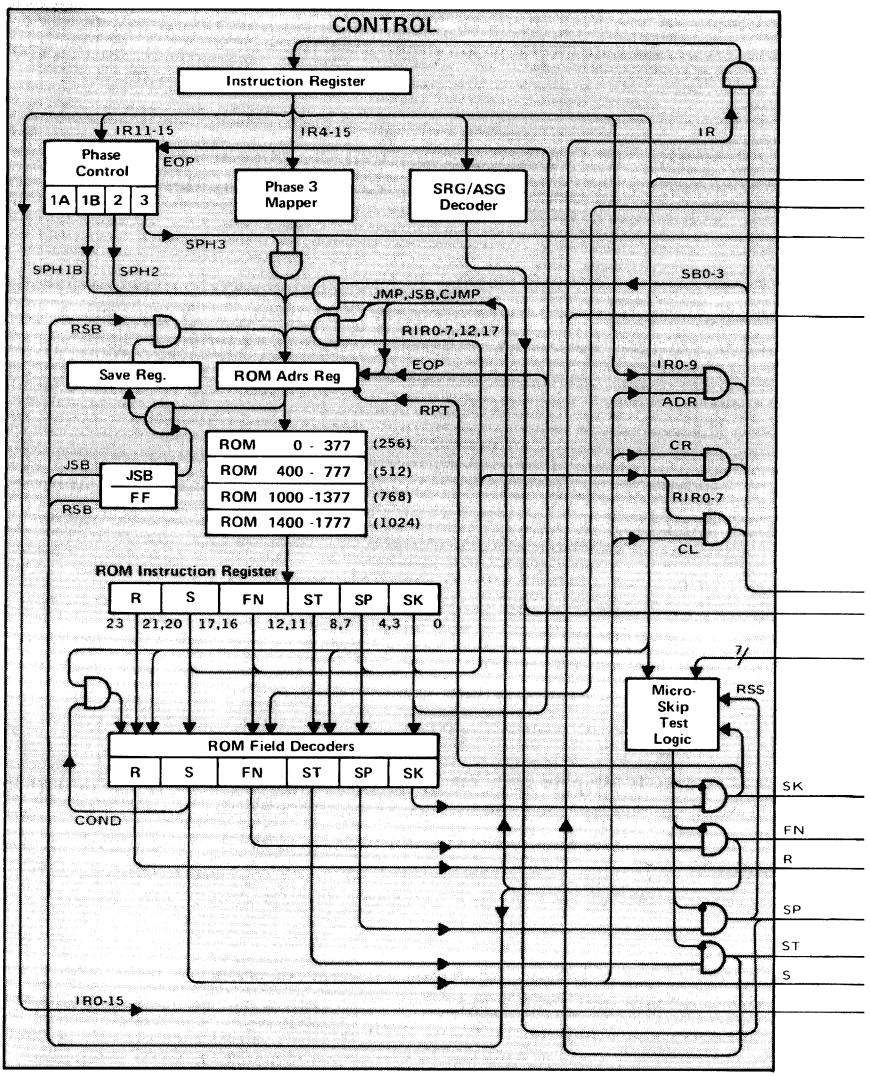


Figure 10. 2100 Block Diagram, Part A

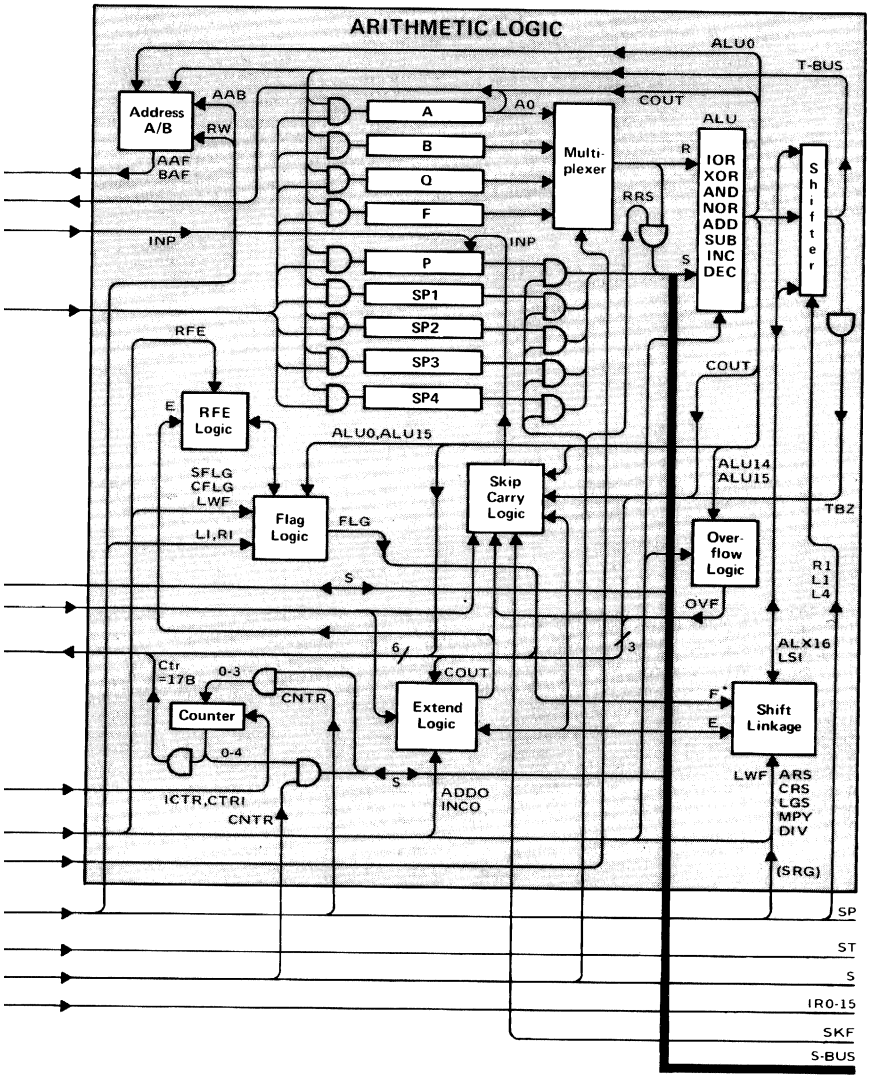


Figure 11. 2100 Block Diagram, Part B

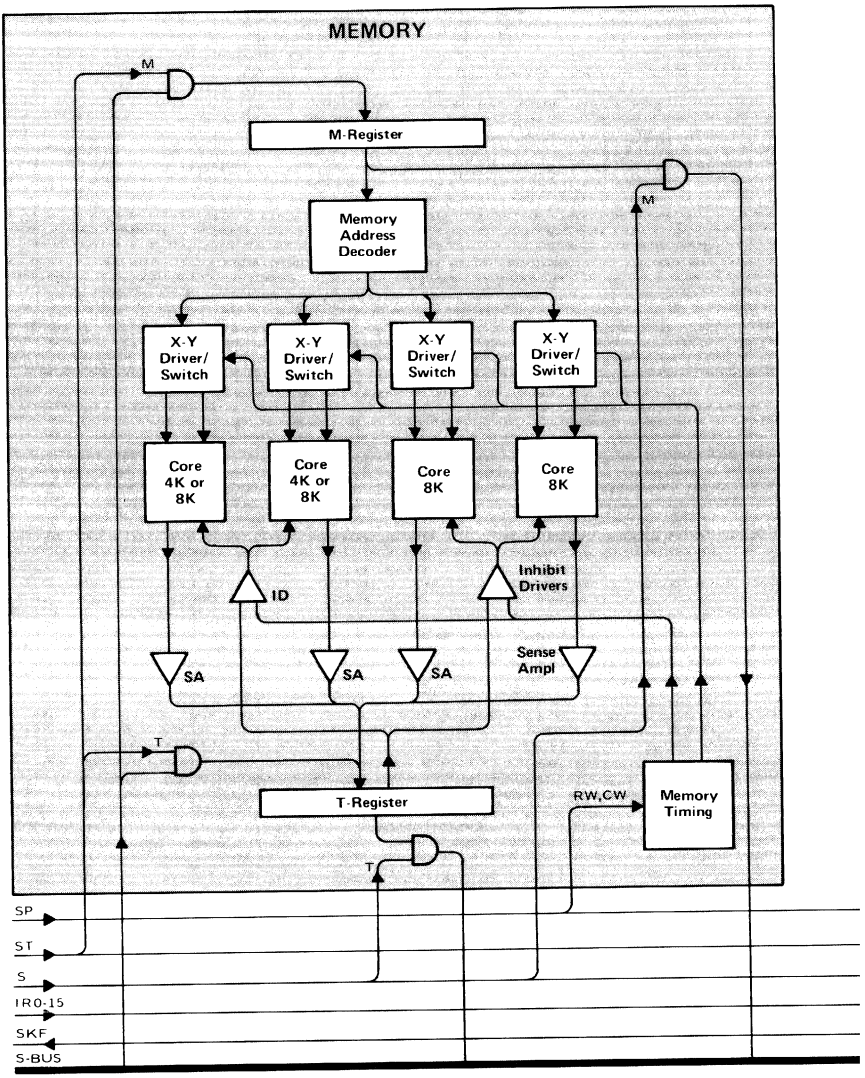


Figure 12. 2100 Block Diagram, Part C

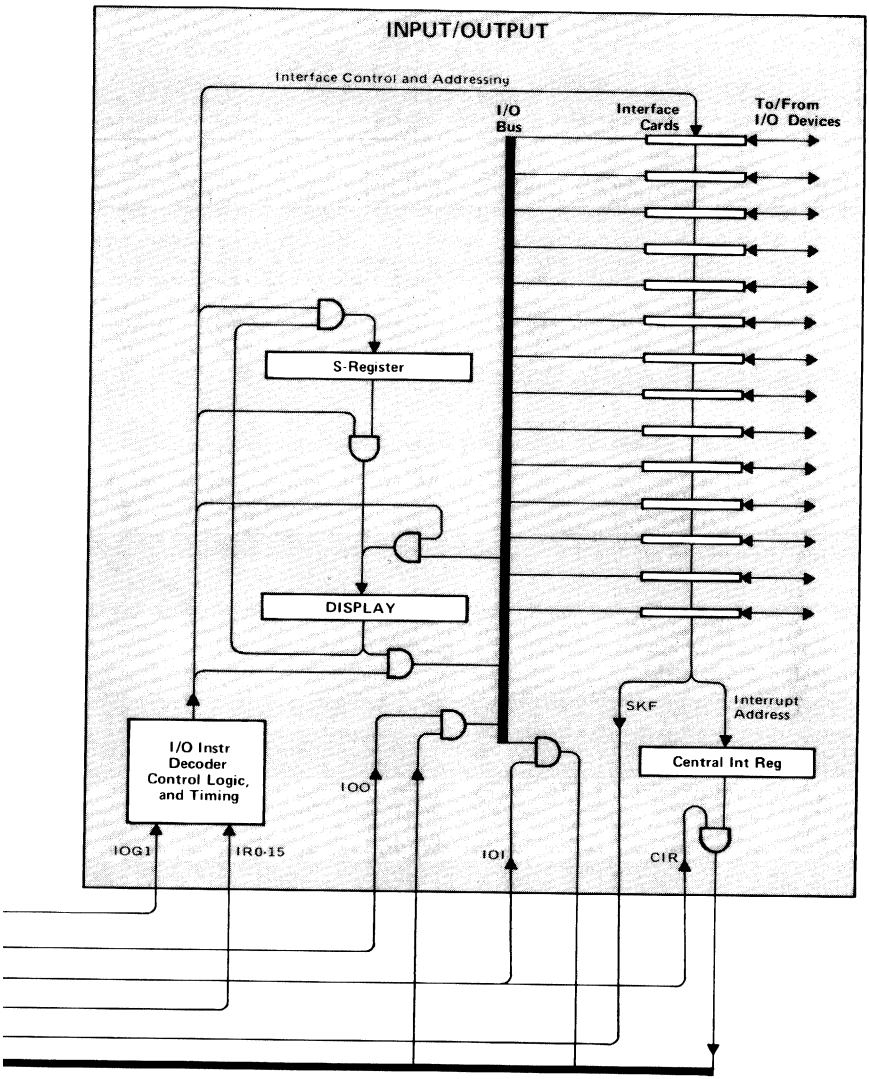


Figure 13. 2100 Block Diagram, Part D

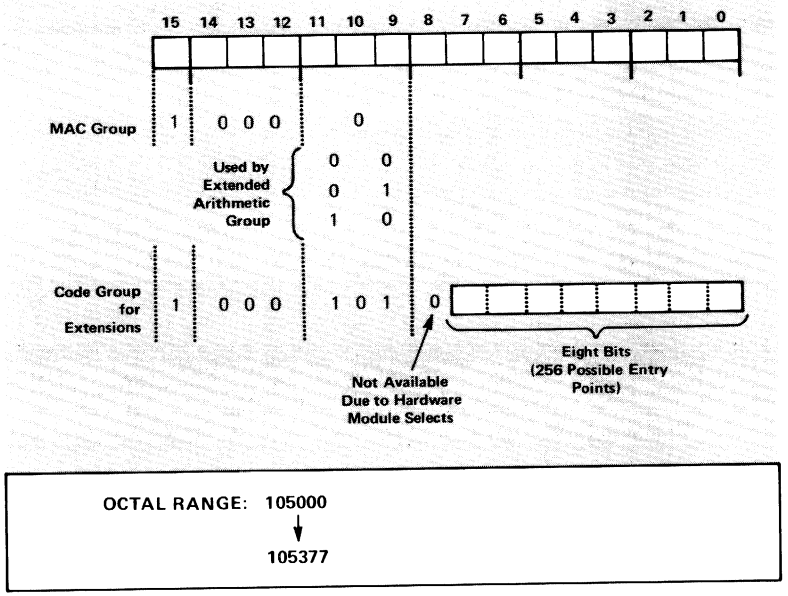
When writing microprograms, it is the responsibility of the micro-programmer to assign physical addresses for the starting points of his microprograms. This is a requirement of the micro-assembler. To assign addresses that are correct and appropriate for the machine structure requires a thorough understanding of how microprograms are accessed.

This section describes the access scheme. Since microprogram addresses are derived from machine language instruction codes, we must first know what codes are available. Therefore, the Macro (MAC) instruction group, which delimits the available codes, will be discussed first. Following this, it will be shown how instruction codes are mapped into addresses in the control store modules, and how software can access the microprograms.

THE MAC INSTRUCTION GROUP

The instruction set for the 2100 Computer reserves a block of binary codes for extensions beyond the basic set. This block of codes is designated as the MAC instruction group.

Figure 14 shows how the binary machine codes are allocated for the MAC group. As shown, the MAC group is specified by a "1" in bit 15 of the instruction, and a "0" in bits 14, 13, 12, and 10. Of this group, certain codes are preempted by the Extended Arithmetic Group of instructions; this subset uses bits 11 and 9 in the combinations 00, 01, and 10. This leaves only the combination 11 to specify a code group for further extensions. Thus all such extensions are designated with 105 as the first three octal digits.



2177-14

Figure 14. Binary Machine Codes for Extensions

Of the ten bits of control store addresses, the two most significant bits cannot be externally coded, since these bits are controlled internally by module-select jumpers. Thus only eight bits (0 through 7) can be used in the instruction word to specify the desired functions. Bit 8 is coded as a "0". The eight available bits allow 256 possible functions to be coded, each of which will correspond to a fixed address (i.e., micro-program entry point) in control store.

The net result is that octal codes 105000 through 105377 are available for extensions. Some of these, it should be noted, are assigned to options manufactured by Hewlett-Packard. (For example, the Floating Point option uses octal codes 105000, 105020, 105040, 105060,

105100, and 105120 for instructions FAD, FSB, FMP, FDV, FIX, and FLT, respectively.) If it is expected that such options will or may be used in your system, these codes should be avoided.

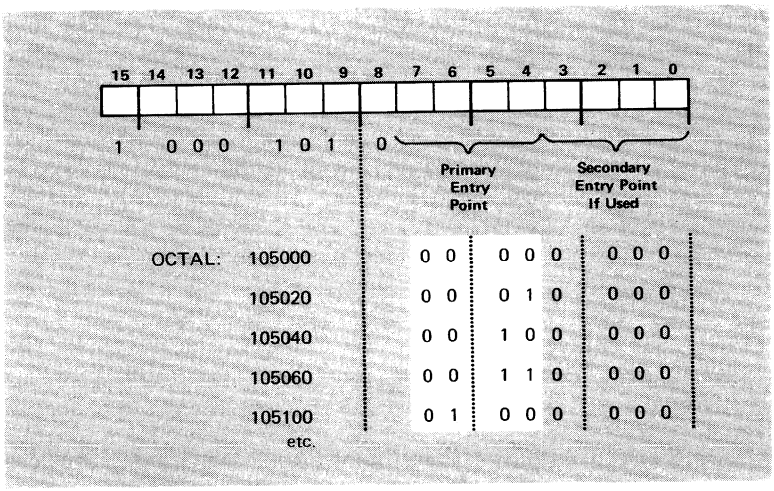
Due to the hardware mapping arrangement, which will be described next, the most convenient sequencing of codes is not necessarily according to consecutive octal numbers (105000, 105001, 105002, etc.). In fact, the six Floating Point codes listed in the preceding paragraph are actually the first six most easily implemented codes.

MAPPING

In describing the Phase 3 Mapper in Section 2, it was stated that the mapper uses bits 4 through 15 of the Instruction Register to generate a 10-bit ROM address. For our purposes (i.e., functions beyond the basic instruction set), bits 9 and 8 of the instruction will always be "1" and "0", respectively. (See preceding paragraphs.) This leaves the four bits 4 through 7 for "primary" mapping. Refer to figure 15.

Primary mapping, therefore, translates the 16 possible codes for bits 4 through 7 of an instruction into appropriate control store addresses called primary entry points. The use of secondary entry points multiplies the total number of possible entry points to 256. However, to apply the secondary entry points requires a special microprogramming technique to access bits 0 through 3 of the Instruction Register, plus a structure of secondary jump tables. This special technique will be discussed later under the heading, "Secondary Entry Points". For now, we will consider only the primary entry points.

As shown in figure 15, the series of 16 binary codes for bits 4 through 7 (i.e., 0000 through 1111) will result in a sequence of octal codes as follows: 105000, 105020, 105040, etc., through 105360. These are translated by the mapper to specific control store addresses, depending on which module has been chosen (and hard-wired) to contain the



2177-15

Figure 15. Primary Entry Point Codes

primary entry points. The rule here is that the lowest numbered module in the system, excluding module 0, must contain the primary entry points. Examples: if modules 0, 1, and 2 are present, module 1 contains the primary entry points; if modules 0 and 3 are present, module 3 contains the primary entry points. In all cases, the first 16 locations of the appropriate module are dedicated to this purpose.

Table 1 shows how the mapping is accomplished. Bits 4 through 7 of the instruction form the four least significant bits of the generated ROM address. Bits 9 and 8 of the address are fixed by hard wiring according to the rule stated in the preceding paragraph. The coding for bits 9 and 8 is 01 for module 1, 10 for module 2, and 11 for module 3. Thus the octal addresses of the primary entry points will be 400 through 417 for module 1, 1000 through 1017 for module 2, and 1400 through 1417 for module 3.

The microinstructions contained in the primary entry point locations are normally jumps. Taken together, the 16 locations are referred to as

the primary jump table. Primary jump tables may be either standard, following a convention established for Hewlett-Packard option extensions, or nonstandard. If your microprogrammed extensions are to be used in conjunction with extensions supplied by Hewlett-Packard, the standard jump table configuration must be used.

Standard and nonstandard jump tables are considered next.

Table 1. Primary Entry Point Mapping

INSTRUCTION CODE							ROM ADDRESS				
Octal	Binary						Binary				Octal
105000	1	000	101	000	000	000	0*	100	000	000	400
105020	1	000	101	000	010	000	0	100	000	001	401
105040	1	000	101	000	100	000	0	100	000	010	402
105060	1	000	101	000	110	000	0	100	000	011	403
105100	1	000	101	001	000	000	0	100	000	100	404
105120	1	000	101	001	010	000	0	100	000	101	405
105140	1	000	101	001	100	000	0	100	000	110	406
105160	1	000	101	001	110	000	0	100	000	111	407
105200	1	000	101	010	000	000	0	100	001	000	410
105220	1	000	101	010	010	000	0	100	001	001	411
105240	1	000	101	010	100	000	0	100	001	010	412
105260	1	000	101	010	110	000	0	100	001	011	413
105300	1	000	101	011	000	000	0	100	001	100	414
105320	1	000	101	011	010	000	0	100	001	101	415
105340	1	000	101	011	100	000	0	100	001	110	416
105360	1	000	101	011	110	000	0	100	001	111	417

*Coding for Module 1. Octal Addresses Start at 400
For Module 2 (10): Octal Addresses Start at 1000
For Module 3 (11): Octal Addresses Start at 1400

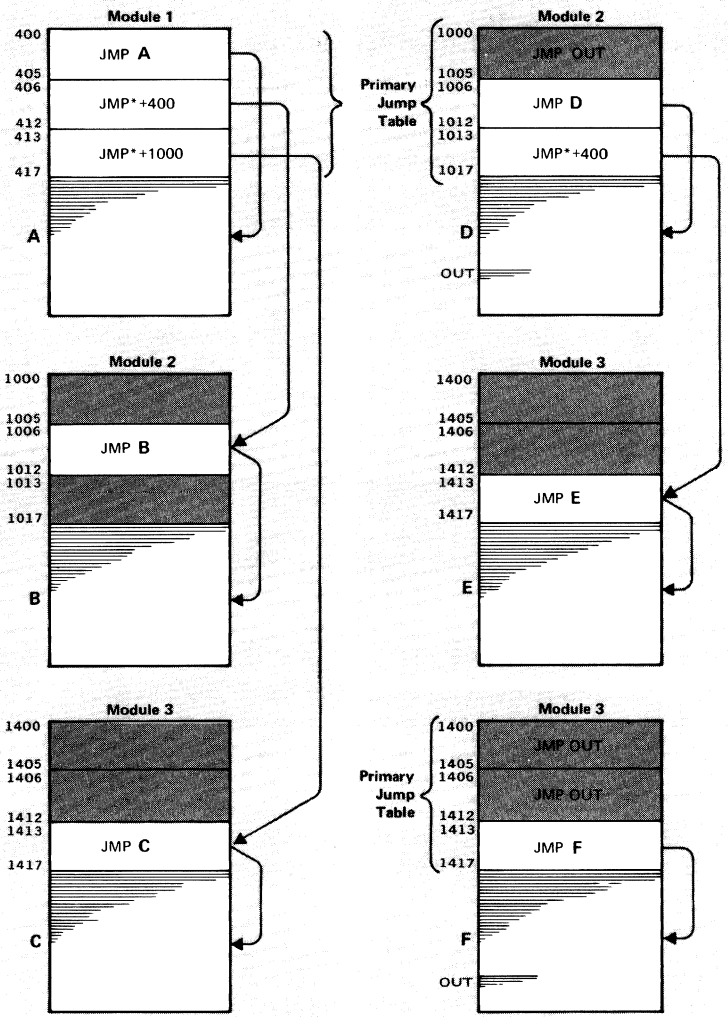
STANDARD JUMP TABLE

The jump targets of microinstructions contained in the primary jump table are defined by convention as shown in figure 16. Three different situations are shown, where: module 1 contains the primary jump table (with jump targets in modules 1, 2, and 3); module 2 contains the primary jump table (module 1 is absent); and module 3 contains the primary jump table (modules 1 and 2 are absent). In all three cases, module 0 is assumed to be present, but it does not enter into the jump table structure.

First consider the case where module 1 contains the primary jump table. As indicated, the first 16 locations of this module, octal addresses 400 through 417, are used for the table. The first six addresses, 400 through 405, are used for jumps to target addresses within module 1. The next five addresses, 406 through 412, are used for jumps to target addresses in module 2. And the last five addresses, 413 through 417, are used for jumps to target addresses in module 3.

For the jumps within module 1 (see **A**), the target is any location in the module following address 417. Since six addresses are allocated to jumps within module 1, module 1 can contain six microprograms which can be accessed by direct jumps. (This is the case for the Hewlett-Packard Floating Point option, which provides six floating point routines.) However, it is possible for any or all of these primary jumps to access secondary jump tables instead of pointing directly to a microprogram routine. As explained later, this would permit up to 96 distinct routines in module 1.

For jumps outside module 1 (see **B** and **C**), the target is the numerically corresponding location in the higher module; i.e., "this location" plus octal 400 for module 2, or "this location" plus octal 1000 for module 3. The asterisk is assembler notation for "this location". Thus, locations 406 through 412 jump to locations 1006 through 1012, and locations 413 through 417 jump to locations 1413 through 1417. In each case, the location jumped to contains another jump which points directly to one of the microprograms in that module (or accesses a secondary jump table, as explained later). Therefore, using only primary entry points,



2177-16

Figure 16. Standard Jump Tables

modules 2 and 3 could contain up to five microprograms each. Note, incidentally, that the first six and the last five locations of module 2 and the first eleven locations of module 3 (dark shaded) are not used.

Now consider the case where module 2 contains the primary jump table. Module 1 is absent, so the module jumpers on the ROM Control card direct all references to routines beyond the basic module 0 set to primary entry points in locations 1000 through 1017. To allow for future addition of module 1, the convention here is that locations 1006 through 1012 are used for jumps within module 2 (see **D**). The next five locations are for jumps to module 3; the jump targets are "this location" plus octal 400. Each of these target locations, 1413 through 1417, contains another jump which points directly to one of the microprograms in module 3 (see **E**).

Since the decoding of machine instructions provides direct access to the primary jump table, the possibility exists that an erroneous or inappropriate code may inadvertently enter locations 1000 through 1005. To protect against unpredictable results, these locations should be filled with jumps to an exit routine (labeled OUT in figure 16). Basically, the exit routine would simply include an EOP (End of Phase) microinstruction followed by a NOP.

Note: The protective exit jumps described in the preceding paragraph could also be used in any unused locations in the primary and secondary jump tables if it is desired to protect against erroneous MAC instruction codes. Frequently, however, it may be found more important to use such locations for microprogramming, due to the restrictions of available control store space.

The final case illustrated in figure 16 shows the primary jump table located in module 3. Modules 1 and 2 are absent. The first eleven locations, 1400 through 1412, contain exit jumps. Locations 1413 through 1417 are for jumps to microprograms within module 3 (see **F**).

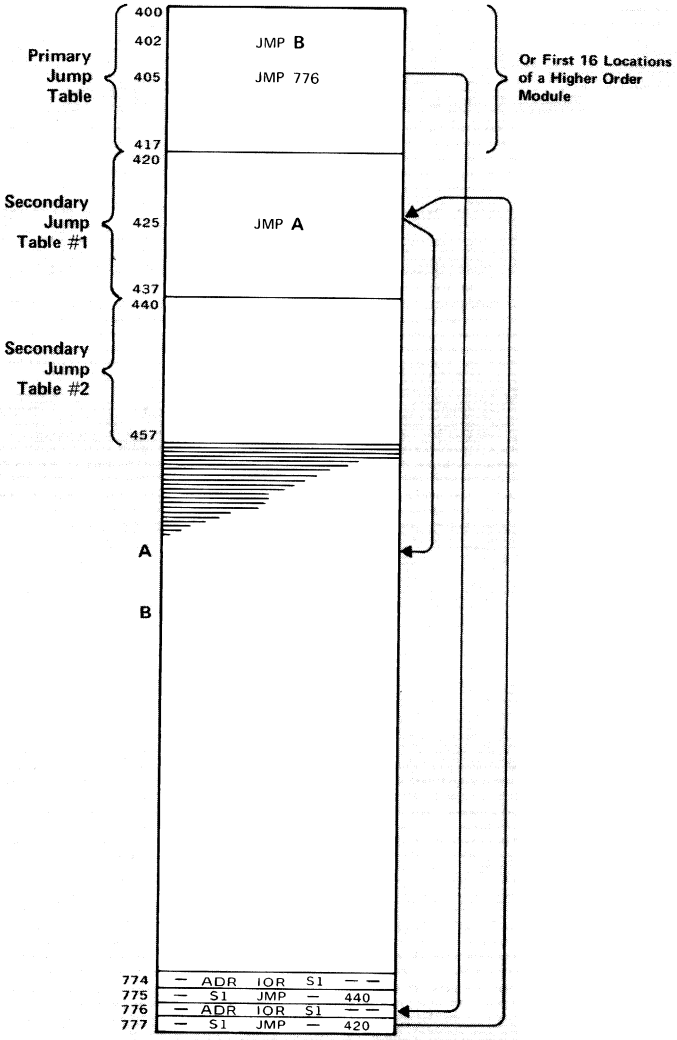
SECONDARY ENTRY POINTS

The preceding descriptions have referred several times to secondary entry points and secondary jump tables. The use of secondary entry points is discussed in the following paragraphs.

As stated earlier in discussing figure 15, the use of secondary entry points multiplies the 16 primary entry points to a total of 256 possible entry points. Refer back to figure 15. Note that for each primary entry point code, there can be 16 secondary entry point codes by varying the coding of bits 0 through 3. (Only code 0000 is shown for secondary entry points.)

Thus it is apparent that the use of these additional entry points requires special access to bits 0 through 3 of the Instruction Register. This special access is provided by hardware. Every time a microprogram jump is executed, bits 0 through 3 of the S-bus are "OR"-tied into the ROM Address Register along with the specified jump target (bits RIR0-7, 12, and 17). Normally there is no data on the S-bus when a jump is executed; however, by specifying ADR in the S-bus field, bits 0 through 9 of the Instruction Register will be read onto the S-bus. Of these 10 bits, only bits 0 through 3 are used in modifying the ROM Address Register contents. (The logic involved can be seen in the block diagram, figure 10).

Figure 17 shows how secondary jump tables can be created using the special hardware feature described in the preceding paragraph. In this figure, it is assumed that only two primary entry points are to be expanded into secondary jump tables. However, any number of the primary entry points may be expanded in this manner. If all 16 primary entry points were expanded, there would be six secondary jump tables in module 1, five in module 2, and five in module 3. The secondary jump tables need not be adjacent to each other or to the primary jump table, although they are shown this way in figure 17. Note that each set of secondary jump tables is accessed from the first 16 locations of the module, following the scheme illustrated earlier in figure 16. Module 1 is used as an example only.



2177-17A

Figure 17. Secondary Jump Tables

Each secondary jump table consists of 16 consecutive locations, and must start on a 16-word boundary (octal 420, 440, etc.). In figure 17, the two tables are assigned to octal addresses 420 through 437 and 440 through 457, and are identified as secondary jump tables number 1 and number 2. The use of secondary jump table number 1 is described in the following paragraphs.

Assume that the current instruction has an instruction code of 105125. From table 1, "Primary Entry Point Mapping", it can be seen that this code will map to location 405 in the primary jump table. Location 405 contains a jump to a two-word routine, at location 776 for this example, which generates the secondary entry point. Location 776 obtains a 15-bit operand address (see ADR definition), of which we are interested in only the four least significant bits. These four bits are from the Instruction Register, and specify the secondary entry point. The address word is stored in Scratch Pad 1 so that it may be read out in the following "jump" microinstruction.

Note: To access secondary jump tables in the odd-numbered modules (1 and 3), you must use Scratch Pad 1 or 3 in the two-word jump routine, as outlined above and in figure 17. This is because the least significant bit of the S-bus field (ROM address bit 17) is part of the jump target. Thus, when the jump is given (see next paragraph) bit 17 of the microinstruction must be a "1", which is the case when specifying S1 or S3 in the S-bus field. Conversely, to access secondary jump tables in the even-numbered modules (0 and 2) the least significant bit of the S-field must be a "0". Thus Scratch Pads 2 or 4 would have to be used. However, note that ADR also has a "0" as the least significant bit. This means that in modules 0 and 2, the two-word jump routine is not necessary. Instead, the entry in the primary jump table may specify a jump directly to the secondary jump table. For example, if the primary entry point is 1007, this location may contain "ADR, JMP, 1020", which points directly at a secondary entry point.

Continuing with the example in figure 17, note that location 777 contains a jump to location 420 — but in addition also specifies S1 in the S-bus field. Thus, according to the JMP definition, bits 0 through 3 (i.e., octal 5) will be "OR"-tied into the jump target address. As a result, the jump is to location 425, which is the sixth entry in secondary jump table number 1. Location 425 contains the secondary jump to the actual starting point of the routine (see A) corresponding to instruction code 105125.

Note that location 405 permanently refers to the jump routine, and location 777 in the jump routine permanently refers to the starting point of one secondary jump table. It is the ADR in the microinstruction which accounts for the selection of microprogram routines.

It is not necessary for all primary entry points to refer to secondary jump tables just because one entry point does so. The primary jump table may contain any mixture of direct jumps or jumps to secondary tables. Thus a primary entry point at location 402 could contain a jump directly to the starting point of a microprogram routine (see B).

Although the use of secondary jump tables greatly extends the number of specifiable functions, it should also be remembered that too many jump tables use up a significant number of valuable control storage locations.

NON-STANDARD JUMP TABLES

Non-standard jump tables are any which do not follow the conventions outlined in the preceding descriptions. For example, you may decide to use only one module and use all 16 locations in the primary jump table to reference routines or secondary jump tables within that same module. Or, jump targets for either primary or secondary tables may point backward — i.e., to lower numbered locations or modules. Any such departure from convention may be made, provided there is an adequate understanding of the principles involved and assuming the decision has been made that the modules so programmed will not be

used in conjunction with modules which use the standard jump tables (e.g., Hewlett-Packard options).

It is permissible (i.e., not a non-standard technique) for microprograms to have references outside their own module, as long as that module provides a return to the calling microprogram, or an EOP signal to allow normal software sequencing to continue. The use of external reference designators is described in the micro-assembler documentation.

ASSIGNING ADDRESSES

From the preceding, it is apparent that the instruction codes assigned to specially microprogrammed functions are dependent on the access scheme. The access scheme you establish is, in turn, dependent on several factors, including:

- the number of modules used
- the number of microprograms per module
- whether or not secondary entry points are used, and
- whether or not the resultant code has previously been assigned by Hewlett-Packard or the user.

Since these are all variable factors, hard and fast rules about assigning addresses cannot be established here. Given your specific application, and using the information given in this section, you will first allocate the necessary jump tables, configure them as necessary, and assign appropriate addresses for each table entry and each microinstruction. Gaps of two or three addresses between each microprogram are advisable to allow for future modifications.

SOFTWARE ACCESS

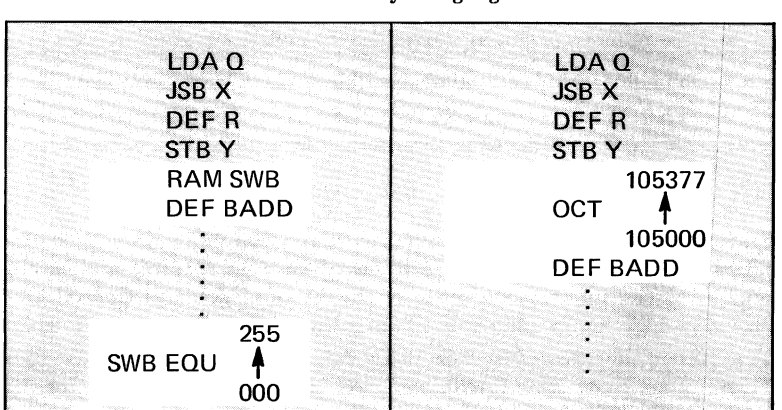
Table 2 lists two methods of accessing microprogrammed functions from assembly language. The first method (left) is for those assemblers which include the RAM (Random Access Memory) pseudo instruction. The second method (right) is for those assemblers which do not have the RAM pseudo instruction.

The RAM pseudo instruction essentially merges the MAC group code (octal 105) with a separately definable code, so that the result may be executed as a machine instruction. The Equate statement (EQU) is used to define up to 256 possible functions.

The alternative method requires an octal number to be inserted in-line with machine language instructions, so that the octal code will be executed as a machine instruction.

In both cases, parameters may be passed to the function by use of the DEF or OCT statement. This will bring the address (or the parameter itself) into the P-register where it may be accessed by the micro-

Table 2. Assembly Language Access



program. It is the responsibility of the microprogram to increment the P-register the proper number of times before exiting from the routine, so that it will point to the next instruction at the end of phase 3. Remember that the P-register is automatically incremented once upon entering phase 3. Thus if, for example, three parameters are passed into the routine via DEFs, then the P-register must be incremented three times by the microprogram to ensure proper return to the software.

USE OF MODULE 0

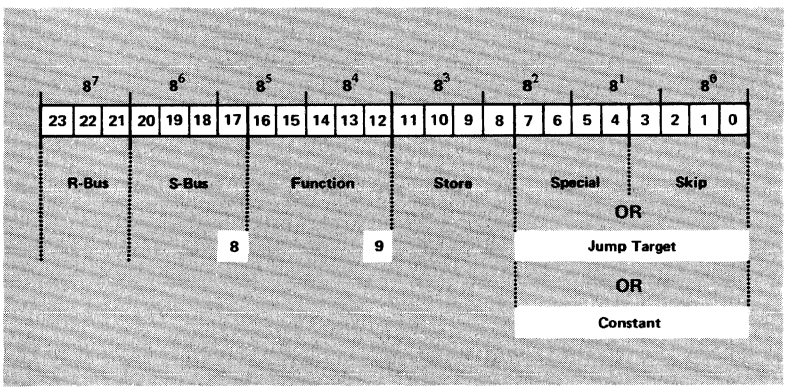
It is possible to rewrite module 0. However, as mentioned previously, this is not recommended. A very high degree of machine knowledge is required in order to do any rewriting. Furthermore, *Hewlett-Packard warranties and support guarantees are voided if module 0 is modified.* It is expected that the extension capabilities provided by the three additional control store modules will cover all needs for special microprogramming.

If, however, you do intend to use module 0, machine control can be switched from the basic ROM module to Writable Control Store by a switch located on the Writable Control Store card. In this way, the new module 0 microprograms may be debugged without making any hardware modifications. The next step, if desired, would be to install a permanent module 0 ROM in place of the original ROM module. This requires unsoldering the six ROM packs and installing the six new packs in their place.

MICROINSTRUCTION WORD FORMAT

Figure 18 illustrates the basic formats for microinstruction words. As shown, the 24-bit word is divided into six fields, and can be represented by an eight-digit octal number.

When a jump (JMP, JSB, or CJMP) is specified in the Function field, bits 0 through 7 of the microinstruction are used for part of the jump target address instead of for Special and Skip operations. The remaining part of the jump target address consists of bits 12 and 17 of the microinstruction. Bit 12 is the least significant bit of the Function field (which accounts for the fact that JMP, JSB, and CJMP each have two valid Function codes), and forms the most significant bit of the jump target (i.e., bit 9). Bit 17 is the second most significant bit (i.e., bit 8)



2177-18

Figure 18. Microinstruction Formats

of the jump target address; note the position reversal with bit 12. When a jump code is present in the Function field, the micro-assembler examines the jump address and automatically codes the proper values for bits 12 and 17. Therefore, care must be taken if an S-bus function is to be coded concurrently with the jump micro-order. (See detailed note under the heading "Secondary Entry Points" in section 3.)

When a constant is to be read onto the S-bus (CL or CR in the S-bus field), bits 0 through 7 of the microinstruction are used for the constant, instead of for Special and Skip operations.

Coding a CL or CR in the S-bus field or JMP, JSB, or CJMP in the Function field automatically inhibits execution of the Special and Skip fields.

Table 3 is a consolidated coding table showing the binary coding of each of the micro-orders. For the R-bus field, only the three least significant code bits are used. For the Function field, all five bits are used. For all remaining fields, only the four least significant bits are used.

ASSEMBLY FORMAT

Normally, the microprogrammer will write his microprograms for assembly by a micro-assembler, and thus generally he does not have to be concerned with binary coding. The micro-assembler has its own set of requirements and rules, which are described in the micro-assembler documentation. It is assumed that the microprogrammer has this documentation at his disposal.

For completeness of this text, however, an approximate representation of the assembly format is shown in figure 19. For assembly, a fixed-field format is used. Typically, cards would be punched according to this format, and read by the micro-assembler using a card reader. The assembly format allows for comments following each microinstruction.

Table 3. Microinstruction Coding

CODE					R-Bus Field	S-Bus Field	Function Field	Store Field	Special Field	Skip Field
5-Bit Field										
	4-Bit Field									
		3-Bit Field			3 Bits	4 Bits	5 Bits	4 Bits	4 Bits	4 Bits
1	1	1	1	1	NOP	NOP	IOR	NOP	NOP	NOP
1	1	1	1	0	*CQ	P	SOV	A	RW	UNC
1	1	1	0	1	AAB	CL	CLO	B	IOG1	EOP
1	1	1	0	0	CAB	CR	SFLG	AAB	CW	NAAB
1	1	0	1	1	F	S1	CFLG	CAB	ASG2	AAB
1	1	0	1	0	Q	S2	LWF	Q	ASG1	NMPV
1	1	0	0	1	B	S3	CJMP	F	ECYN	CTR
1	1	0	0	0	A	S4	ARS	P	ECYZ	CTRI
1	0	1	1	1		COND	CRS	S1	*LEP	TBZ
1	0	1	1	0		ADR	LGS	S2	AAB	FLG
1	0	1	0	1		CNTR	RSB	S3	SRG2	OVF
1	0	1	0	0		RRS	CJMP	S4	SRG1	COUT
1	0	0	1	1		M	JMP	IR	CNTR	NEG
1	0	0	1	0		T	JMP	T	R1	ODD
1	0	0	0	1		IOI	JSB	M	L1	RPT
1	0	0	0	0		CIR	JSB	IOO	RSS	ICTR
0	1	1	1	1			**			
0	1	1	1	0			XOR			
0	1	1	0	1			NOR			
0	1	1	0	0			AND			
0	1	0	1	1			ADD			
0	1	0	1	0			ADDO			
0	1	0	0	1			INC			
0	1	0	0	0			INCO			
0	0	1	1	1			**			
0	0	1	1	0			DEC			
0	0	1	0	1			SUB			
0	0	1	0	0			DIV			
0	0	0	1	1			MPY			
0	0	0	1	0			P1A			
0	0	0	0	1			RFE			
0	0	0	0	0			*RFI			

*See CAUTION note in definitions

**Undefined codes

Refer to the assembled listing in the appendix of this handbook for examples of the assembly format.

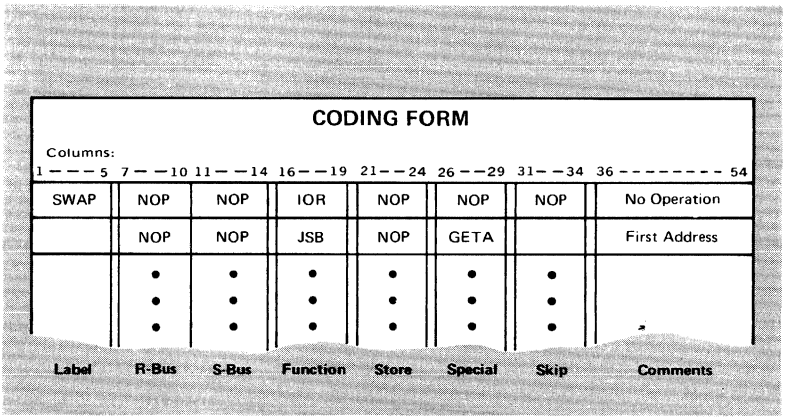


Figure 19. Sample Assembly Coding

MICRO-ORDER INSTRUCTION SET

R-BUS FIELD

A — Reads the A-register onto the R-bus.

AAB — Reads the A-register or B-register onto the R-bus, depending on whether the A Addressable FF or B Addressable FF is set. (Both flip-flops cannot be set at the same time.) If neither AAF nor BAF is set, the A-register will be read onto the R-bus (unless COND is present in the S-bus field, in which case the T-register is read onto the S-bus).

B — Reads the B-register onto the R-bus.

CAB Reads the A-register or B-register onto the R-bus, depending on whether Instruction Register bit 11 is "0" (A) or "1" (B).

CQ Reads the Q-register onto the R-bus if Instruction Register bit 9 is a "1" and the Index Mode flip-flop is set. Used only for diagnostics.

CAUTION

The CQ and RFI codes are not intended for use in special microprogramming. The use of these codes will affect the operation of module 0 and consequently will cause incorrect operation of HP software. To allow continued use of existing software, it would be necessary to rewrite those instruction routines in module 0 which use the Q-register. As noted under the heading "Use of Module 0", *such changes will void Hewlett-Packard warranties and support guarantees.*

F Reads the F-register onto the R-bus.

NOP No operation; results in all "0"s on the R-bus.

Q Reads the Q-register onto the R-bus.

S-BUS FIELD

ADR Reads Instruction Register bits 0 through 9 and (if Instruction Register bit 10 is a "1") P-register bits 10 through 15, or (if Instruction Register bit 10 is a "0") six "0"s onto the S-bus. ADR is normally used to obtain an operand address.

CIR Reads the Central Interrupt Register onto S-bus bits 0 through 5.

CL Reads a constant onto the left half (bits 8 through 15) of the S-bus; the constant is obtained from bits 0 through 7 of the ROM Instruction Register. Execution of the Special and Skip fields is inhibited. Reads "0"s onto the right half (bits 0 through 7) of the S-bus.

CNTR Reads the counter contents onto S-bus bits 0 through 4. (Bit 0 is the least significant bit.)

COND Normally used with AAB coded in the R-bus field. If so, and AAF is set, the A-register is read onto both the R- and S-buses; if BAF is set, the B-register is read onto both the R- and S-buses. If neither is set, the T-register is read onto the S-bus. If some function other than AAB is coded in the R-bus field when COND is used in the S-bus field, one of the following will occur:

a. if neither AAF nor BAF is set, the T-register is read onto the S-bus;

b. if either AAF or BAF is set, the register selected in the R-bus field (including F and Q) is read onto both the R- and S-buses.

If the "Data Ready" signal from memory is false, the CPU will freeze until this signal becomes true.

CR Reads a constant onto the right half (bits 0 through 7) of the S-bus; the constant is obtained from bits 0 through 7 of the ROM Instruction Register. Execution of the Special and Skip fields is inhibited. Reads "0"s onto the left half (bits 8 through 15) of the S-bus.

IOI Reads the I/O bus onto the S-bus.

M Reads the M-register onto the S-bus. Note that the M-register contains only 15 bits.

NOP No operation; results in all "0"s on the S-bus.

P Reads the P-register onto the S-bus.

RRS Reads the R-bus onto the S-bus.

S1 Reads Scratch Pad 1 onto the S-bus.

S2 Reads Scratch Pad 2 onto the S-bus.

S3 Reads Scratch Pad 3 onto the S-bus.

S4 Reads Scratch Pad 4 onto the S-bus.

T Reads the T-register onto the S-bus. If the "Data Ready" signal from memory is false, the CPU will freeze until this signal becomes true.

FUNCTION FIELD

ADD Adds the R-bus and S-bus.

ADDO Adds the R-bus and S-bus and enables the Overflow logic. If the instruction ADA or ADB is detected, the Extend logic is also enabled.

AND Logical AND of the R-bus and S-bus.

ARS 32-bit arithmetic shift. The direction of shift is specified in the Special field (L1 or R1). If L1, a "0" is shifted into bit 0 of the low-order register; bit 14 of the high-order register is lost,

and the sign bit is unchanged; the Overflow flip-flop is set if ALU bits 14 and 15 differ. If R1, the sign is copied into bit 14 of the high-order register, and bit 0 of the low-order register is lost. ARS also enables IOR.

Note: On 32-bit right shifts, the B- and A-registers must be used; the B-register contains the high-order bits and the A-register contains the low-order bits. On 32-bit left shifts, the F- and Q-registers must be used; the F-register contains the high-order bits and the Q-register contains the low-order bits.

CFLG Clears the CPU Flag flip-flop. Also enables IOR.

CJMP Conditional jump. (See JMP note.) Executes a jump if, in the run mode, an interrupt or an operator panel halt command is detected. Otherwise, the microinstruction is treated as an IOR. (The IOR function is enabled regardless of whether or not the jump condition is detected.) In the single-cycle mode, the detection of CJMP will cause the computer to halt unconditionally. The Special and Skip fields are inhibited, as is the "read-P" micro-order in the S-bus field. See JMP definition for derivation of the jump address.

CLO Clears the Overflow flip-flop. Also enables IOR.

CRS 32-bit circular shift (rotate). (See ARS note.) The direction of shift is specified in the Special field (L1 or R1). If L1, bit 15 of the high-order register is transferred to bit 0 of the low-order register. If R1, bit 0 of the low-order register is transferred to bit 15 of the high-order register. CRS also enables IOR.

DEC Subtracts the S-bus from the R-bus in one's complement form. If the S-bus contains all "0"s, the R-bus is decremented.

DIV Divide step. Normally used in a repeat loop as part of a divide algorithm. DIV subtracts the S-bus from the R-bus (two's complement) and checks the COUT (Carry Out) signal for a store decision. If COUT is "1", the result of the subtraction is left-shifted one place and stored in a register (normally the F-register). If COUT is "0", the existing contents of the F-register are shifted left one place internally in the F-register; the subtraction result is not stored. In either case, the Q-register also shifts left one place, COUT is shifted into bit 0 of the Q-register, bit 15 of the Q-register shifts into bit 0 of the F-register, and bit 15 of the F-register is lost. A valid divide step requires L1 in the Special field, F in the R-bus and Store fields, and an S-bus register (normally a Scratch Pad) specified in the S-bus field. DIV requires two CPU clock cycles to execute.

INC Increments the sum of the R-bus and S-bus.

INCO Increments the sum of the R-bus and S-bus, and enables the Overflow logic. If the memory reference instruction ADA or ADB is detected, the Extend logic is also enabled.

IOR Logical "inclusive OR" of the R-bus and S-bus.

JMP Jump. Transfers bits 0 through 7 of the ROM Instruction Register (RIR0-7) to the corresponding bits of the ROM Address Register, RIR17 to bit 8 of the ROM Address Register, and RIR12 to bit 9. Bits 0 through 3 of the S-bus are "OR"-tied with RIR0-3 in the forming the jump address. The Special and Skip fields are inhibited, as is the "read-P" micro-order in the S-bus field. Also enables IOR.

Note: JMP, JSB, RSB, and CJMP each require two machine cycles to execute. The microinstruction containing the jump is executed during the first cycle, and a NOP is executed in the second cycle. The second cycle is used

to allow data to be accessed from control store at the new address.

JSB Jump to microprogram subroutine. Same as JMP, except also sets the JSB flip-flop, thus locking the return address in the Save register.

LGS 32-bit logical shift. (See ARS note.) The direction of shift is specified in the Special field (L1 or R1). If L1, a "0" is shifted into bit 0 of the low-order register and bit 15 of the high-order register is lost. If R1, a "0" is shifted into bit 15 of the high-order register and bit 0 of the low-order register is lost. LGS also enables IOR.

LWF Link with Flag. If L1 is coded in the Special field, the content of the Flag flip-flop is transferred to the left-shift input (LSI) of the shifter, thereby transferring its content to T-bus bit 0, and ALU bit 15 is transferred to the Flag flip-flop. If R1 is coded in the Special field, the content of the Flag flip-flop is transferred to the right-shift input (ALX16) of the shifter, thereby transferring its content to T-bus bit 15, and ALU bit 0 is transferred to the Flag flip-flop. Also enables IOR.

MPY Multiply step. Normally used in a repeat loop as part of a multiply algorithm. MPY first checks bit 0 of the A-register for an add decision. If this bit is a "1", the R- and S-bus inputs to the ALU are added; if a "0", the R-bus only is routed through the ALU. In either case, the output of the ALU is shifted right one place and stored back into the R-bus register (normally the B-register, assumed to be specified in the Store field), with COUT forming bit 15. The A-register is shifted right, and ALU bit 0 fills vacated bit position 15. Bit 0 of the A-register is lost. A valid multiply step requires R1 in the Special field; also, normally, B in the R-bus and Store fields, and S1/2/3/4 in the S-bus field.

NOR Logical NOR of the R-bus and S-bus. If a NOP is specified in either the R-bus or S-bus field, the complement of the other is

obtained. If both the R-bus and S-bus fields contain a NOP, the ALU output (shifter input) consists of all "1"s.

P1A Sets phase 1A and clears the current phase. Used mainly by diagnostics.

RFE Rotates the contents of the Flag and Extend flip-flops.

RFI Rotates the contents of the Flag and Index Mode flip-flops. See CAUTION note under CQ definition.

RSB Return from microprogram subroutine. Transfers the contents of the Save register into the ROM Address Register. Clears the JSB flip-flop. Also enables IOR. (See JMP note.)

SFLG Sets the CPU Flag flip-flop. Also enables IOR.

SOV Sets the Overflow flip-flop. Also enables IOR.

SUB Subtracts the S-bus from the R-bus in two's complement form.

XOR Logical "exclusive OR" of the R-bus and S-bus.

STORE FIELD

A Stores the T-bus into the A-register.

AAB Stores the T-bus into the A- or B-register, depending on whether the A Addressable FF or B Addressable FF is set. If neither flip-flop is set, no store will occur.

B Stores the T-bus into the B-register.

CAB Stores the T-bus into the A- or B-register, depending on whether Instruction Register bit 11 is "0" (A) or "1" (B).

F Stores the T-bus into the F-register.

IOO Reads the S-bus onto the I/O bus.

IR Stores the S-bus into the Instruction Register.

M Stores the S-bus into the M-register, and also into the Violation register if the computer is in phase 1A, memory protect mode is set and no memory protect violation has been detected, and no parity error exists.

NOP No store.

P Stores the T-bus into the P-register.

Q Stores the T-bus into the Q-register.

S1 Stores the T-bus into Scratch Pad 1.

S2 Stores the T-bus into Scratch Pad 2.

S3 Stores the T-bus into Scratch Pad 3.

S4 Stores the T-bus into Scratch Pad 4.

T Stores the S-bus into the T-register.

SPECIAL FIELD

Note: The special functions IOG1, ASG1, ASG2, SRG1, and SRG2 are used as hardware enables to allow the 2100 microprocessor to more easily emulate the 2116-family instruction set. These functions are available for use by the microprogrammer; however, proper use of these functions requires a deeper

understanding of the machine hardware than is given here. Refer to the Theory of Operation in the 2100 maintenance documentation for a more complete explanation of the operation of these functions.

AAB Enables the setting of A Addressable FF or B Addressable FF, depending on whether ALU bit 0 is a "0" (A) or a "1" (B), with T-bus bits 1 through 14 all "0".

ASG1 Enables skip and Extend logic specified by Instruction Register bits 0 and 3 through 7. (Register handling is done by microprogram; see appendix listing.)

ASG2 Enables skip, Extend, and increment logic specified by Instruction Register bits 0, 1, and 2. (Register handling is done by microprogram; see appendix listing.)

CNTR Stores S-bus bits 0 through 3 into the counter, and clears bit 4 of the counter.

CW Clear/write memory cycle. CPU freezes until I/O time T6, then commands memory to begin a clear/write cycle. (The address into which data is being stored is also normally sent to the M-register at this time.) Data stored in the T-register, normally loaded during the following T3, is stored into memory at the end of the memory cycle. The Memory Busy FF is then cleared. CW is enabled by a "set skip" signal, normally as a result of NMPV in the Skip field (or UNC if memory-protect testing is not desired).

ECYN Enables the Skip Carry logic for software skips. Sets the Carry flip-flop if the T-bus does not contain all "0"s.

Note: If the Carry flip-flop is set, the P-register will be incremented upon exiting from phase 3.

ECYZ Enables the Skip Carry logic for software skips. Sets the Carry flip-flop if the T-bus contains all "0"s. (See note above.)

IOG1 Synchronizes CPU with I/O timing and enables I/O group decoder.

L1 Enables left-shift-one logic. Shifts ALU bits 0 through 14 to T-bus lines 1 through 15.

LEP Legal entry point. Prevents illegal entry into an Extended Arithmetic Group instruction microprogram through an incorrect MAC code. Causes the microprocessor to execute NOPs until LEP is detected, or until EOP is detected in the Skip field. LEP cannot be used for anything other than enabling entry points to the 2100 Extended Arithmetic Group instructions, coded only in module 0. Included here for completeness only.

NOP No operation.

R1 Enables right-shift-one logic. Shifts ALU bits 1 through 15 to T-bus lines 0 through 14.

RSS Reverses (complements) the skip sense of the Skip field functions.

RW Read/write memory cycle. CPU freezes until I/O time T6, then commands memory to begin a read cycle and sets the Memory Busy flip-flop. Memory data output is read into the T-register prior to the following time T4. The Data Ready flip-flop is then set to indicate to the CPU that data is available (valid only through T5 following the above T6). The ALU shifter output is tested for address 0 or 1, and the A Addressable FF or B Addressable FF is set accordingly.

SRG1 Enables shift-rotate group functions specified by Instruction Register bits 6 through 9. Sets the SRG flip-flop, which enables CLE and SL* instruction logic during the next cycle.

SRG2 Enables shift-rotate group functions specified by Instruction Register bits 4 and 0 through 2.

SKIP FIELD

Note: The term "skip", as used in the 2100 microprocessor, is unconventional in that, if a skip condition is detected, the following instruction is not "jumped over". Instead, the "skipped" instruction is actually forced to be a NOP (except see EOP).

AAB Skips the next microinstruction if either the A Addressable FF or B Addressable FF is set.

COUT Skips the next microinstruction if there is a carry-out (COUT) signal from the ALU.

CTR Skips the next microinstruction if counter bits 0 through 3 are all "1"s (octal 17). Ignores bit 4.

CTRI Skips the next microinstruction if counter bits 0 through 3 are all "1"s (octal 17). Ignores bit 4. Increments counter after testing.

EOP End of phase. Used to terminate the current phase. Sets the correct next phase flip-flop and executes a hardware jump through the mapper to the address which begins the next phase. As opposed to a normal micro-jump, the instruction in sequence following the instruction containing the EOP is executed before the jump is taken. EOP also clears the JSB flip-flop. The microinstruction containing an EOP cannot be skipped (although it can be jumped over). For example, if the microinstruction preceding the EOP microinstruction contains UNC, the microinstruction containing EOP will be treated as a NOP except for the EOP micro-order, which will be executed.

FLG Skips the next microinstruction if the CPU Flag flip-flop is set.

ICTR Increments the counter.

NAAB Skips the next microinstruction if T-bus bits 1 through 14 are not all-zero. Normally used to detect addressable A/B addresses on the T-bus.

NEG Skips the next microinstruction if the ALU output is negative (bit 15 is a "1").

NMPV Skips the next microinstruction if memory protect is disabled and AAF and BAF are both clear, or if memory protect is enabled, AAF and BAF are both clear, and no violation is detected. If either AAF or BAF is set, no skip will occur.

NOP No operation.

ODD Skips the next microinstruction if the ALU output is odd (bit 0 is a "1").

OVF Skips the next microinstruction if the Overflow flip-flop is set.

RPT Causes the next microinstruction to be repeated until a skip condition is met. The next microinstruction cannot contain TBZ or RSS, TBZ; also, it cannot contain an add-type function (ADD, INC, etc.) if the Skip field contains NEG or ODD (with or without RSS in the Special field).

TBZ Skips the next microinstruction if the T-bus contains all "0"s.

UNC Skips the next microinstruction unconditionally.

INTRODUCTION

The most expedient method of creating new microprograms is to examine the module 0 listing in the appendix of this handbook, see how similar operations were implemented, and adapt the appropriate segments. In all cases, however, it is necessary for the microprogrammer to know the effect of each micro-order he writes into his microprograms. As given in the appendix, the "comments" do not necessarily list all the effects of a given line of microcode. It is conceivable that nonmentioned side effects could be detrimental to other microprograms. The preceding sections of this handbook must be studied and understood before undertaking a microprogramming project for the 2100.

The next major heading, Example Microprogram, takes the reader through the fundamental operations required to originate a new microprogram. The example chosen is deliberately designed to use as many diverse functions as possible (rather than for maximum efficiency). Even so, the example does not cover all special cases. A list of programming aids and restrictions is given at the end of this section.

EXAMPLE MICROPROGRAM

Suppose we wish to microcode a routine to swap two words in core memory, located at addresses W and W+1. The routine will be called SWP, and will be entered by the first available primary entry point in module 2. Thus the octal instruction code for SWP is 105140, and the entry point (containing the jump to the routine's starting address) is

location 1006. (See table 1 and figure 16 in section 2.) The routine is entered from assembly language by:

```
        OCT 105140
        DEF W
```

where W contains the address of the first word to be swapped (may be indirect). Or, for assemblers having the RAM pseudoinstruction, the routine is entered by:

```
        RAM SWP
        DEF W
          :
SWP EQU 140B
```

Table 4 lists the complete microprogram. The following paragraphs describe how it is generated.

Table 4. SWP Microprogram

R	S	FN	ST	SP	SK	Comments
		JSB		GETAD		1 Put address of 1st word in S1
		JSB		OPGET		2 Put first word in S2
	S1	INC	S3			3 Put address of 2nd word in S3
	S3	IOR	M	RW		4 Fetch 2nd word or set AAF/BAF
AAB	COND	IOR	S4			5 Put second word in S4
	P	INC	P			6 Increment P past DEF
	S3	IOR	M	CW	NMPV	7 Start CW cycle if AAF/BAF not set
	S2	IOR	AAB		UNC	8 Load 1st word in A/B if AAF/BAF set
	S2	IOR	T			9 Or send 1st word to memory
	S1	IOR		AAB		10 Reset AAF/BAF according to 1st address
	S1	IOR	M	CW	NMPV	11 Start CW cycle if AAF/BAF not set
	S4	IOR	AAB		UNC	12 Load 2nd word in A/B if AAF/BAF set
	S4	IOR	T		EOP	13 Or send 2nd word to memory
		IOR				14 End of routine

The first requirement is to fetch the address defined as W. A routine to fetch this address already exists in module 0 (see address 362 in the appendix listing), labeled GETAD. Similarly, a routine to fetch the contents of this address also exists in module 0, labeled OPGET. However, to use these subroutines, we must supply their addresses to the swap routine. This is done by inserting two "external reference designator" statements (cards) for the desired routines. (Refer to micro-assembler documentation.)

These labels may then be used as targets for subroutine calls, as shown in lines 1 and 2 of the microprogram. The GETAD subroutine stores the address of W (i.e., the current contents of the address in the P-register) into Scratch Pad 1. The OPGET subroutine stores the contents of W (i.e., the contents of the location pointed to by the contents of Scratch Pad 1) into Scratch Pad 2.

Since we still have two unused Scratch Pad registers, these may be used for the address and contents of the next word, in location W+1. Lines 3, 4, and 5 of the microprogram accomplish this purpose. Line 3 reads out the address of the first word (in Scratch Pad 1), increments this value, and stores the resultant address in Scratch Pad 3. Line 4 sends the address in Scratch Pad 3 to memory with a read/write command; this will cause memory to load the contents of the addressed memory location into the T-register. If the addressed location is the A- or B-register, the A Addressable FF or B Addressable FF will be set. Line 5 transfers the contents of the T-register (or A- or B-register) into Scratch Pad 4.

At this point, the two words and their addresses are in Scratch Pad registers 1 through 4. We may now proceed to swap the words. First, however, since memory is still busy from the preceding RW (line 4), this is an appropriate time to increment the P-register past the DEF statement. (On entering phase 3, the P-register is automatically incremented, and thus up until now has been pointing at the address of W — i.e., the DEF statement.) Line 6 increments the P-register.

Line 7 reads out the address of the second word and sends it to the M-register in memory; simultaneously (unless inhibited), the CW micro-

order tells memory to begin a clear/write memory cycle for the location addressed by the M-register. The CW micro-order is inhibited if AAF or BAF is set. As noted in the CW definition, CW is enabled only if there is a true "set skip" signal. Line 8 loads the first word into the A- or B-register if the second word originally was in either of these registers (AAF or BAF would still be set from line 4); this line would be skipped if neither AAF nor BAF were set. An unconditional skip then omits line 9. Line 9 is executed instead of line 8 if the second word was not in the A- or B-register. In this case, the first word is sent to the T-register in memory; the clear/write cycle, already in progress, will store the T-register contents into the location currently addressed (i.e., the original location of the second word).

Note: The routine as shown is not concerned with memory protect, and in fact assumes that memory protect is not enabled. The NMPV in the Skip field is used only to check for addressable A-/B-register references. If we wished to check for memory protect violation, the coding in lines 7 and 11 would be:

```
F  S3  DEC  M  CW  NMPV  (7)
F  S1  DEC  M  CW  NMPV  (11)
```

Line 10 reads out the address of the first word (in S1) and, depending on the contents, either sets the A Addressable or B Addressable flip-flops or clears both. Line 11 reads out the address of the first word and sends it to the M-register; simultaneously, unless inhibited, the CW micro-order tells memory to begin a clear/write cycle. The CW micro-order is inhibited if the address is 0 or 1. Line 12 loads the second word into the A- or B-register if the first word originally was in either of these registers (note AAF or BAF would have been set in line 10); otherwise this line is skipped. The UNC micro-order inhibits storing S4 in the T-register; however, since EOP cannot be inhibited, the routine ends here and the next phase is set. Line 13 is executed instead of line 12 if the first word was not in the A- or B-register. In this case, the second word is sent to the T-register; the clear/write cycle, already in progress, will store the T-register contents into the location currently addressed (i.e., the original location of the first word).

Line 13 also specifies End of Phase (EOP). However, since this micro-order takes effect after the succeeding microinstruction, one additional line (no operation in line 14) is necessary to complete the microprogram.

This completes the example given in table 4. The example has shown typical coding for: the use of external references (lines 1 and 2), reading from memory (lines 4 and 5), writing into memory (lines 7, 8, 9, and 11, 12, 13), passing parameters (lines 1, 2, and 6), and ending a microprogram (lines 13 and 14).

PROGRAMMING AIDS AND RESTRICTIONS

Most of the special situations that must be considered in writing a microprogram have been explained in the preceding sections of this handbook. There are, however, a few unique cases which do not fit into the general context of the preceding discussions, plus a few which bear repeating. The following paragraphs define some of these cases.

BLANK FIELD. Leaving a field blank results in a NOP for that field. The exception is the Function field, which will be given an IOR function.

AAF AND BAF DEFINED. The A Addressable flip-flop (AAF) and B Addressable flip-flop (BAF) are hardware flip-flops which, when set, indicate that the desired data is in, respectively, the A-register or B-register. When location 00000 is referenced, AAF will be set, thus selecting the A-register. When location 00001 is referenced, BAF will be set, thus selecting the B-register. (Note that this makes core locations 00000 and 00001 inaccessible to the software programmer.) See following two paragraphs for application.

AAB IN R-BUS FIELD. With AAB programmed in the R-bus field and AAF and BAF both clear, the A-register will be read onto the R-bus (except see next paragraph).

COND IN S-BUS FIELD. Programming COND in the S-bus field normally requires that AAB be specified in the R-bus field. If a NOP is coded instead of AAB, then COND will result in reading the T-register onto the S-bus or "0"s onto both the R- and S-buses. If a specific register is selected in the R-bus field with COND in the S-bus field, and AAF or BAF is set, then the R-bus register is read onto both the R- and S-buses. In the normal case (coded AAB, COND, ...), if AAF or BAF is set, the A-register or B-register will be read onto both the R-bus and S-bus simultaneously. Be careful of what is specified in the Function field here.

JUMPS AND S-BUS FIELD. On jump micro-orders (JMP, JSB, CJMP), bits 0 through 3 of the S-bus are "OR"-tied into the least significant bits of the jump address. Therefore, do not use the S-bus field with jumps unless this effect is specifically desired.

CJMP TARGET. Early 2100 models may have a version of the Microinstruction Decoder 2 card (A4), part no. 02100-60022, which restricts CJMP targets to the lower 512 locations of control store.

CW EXECUTION. In order to execute the CW micro-order, the Skip field must contain a skip micro-order (normally NMPV or UNC), and the following microinstruction must be skipped as a result of the true "set skip" signal.

MEMORY TIMING. When reading from memory, the T-register must be read within four time periods after giving the RW command. Normally this means four microinstructions; however, DIV, JMP, JSB, RSB, and CJMP take two time periods each, and this would have to be taken into account. When writing into memory, the data must be sent to memory within two time periods after giving the CW command. Since, for a valid CW, the following time period is dedicated to executing a skip, the data is always sent in the microinstruction following the skipped microinstruction.

EOP AND SKIP. If a microinstruction containing EOP (End of Phase) is skipped, due to a true "set skip" signal in the previous microinstruction, the EOP micro-order will still be executed.

USE OF T- AND M-REGISTERS. Do not use the T- or M-registers as temporary storage locations. A DMA transfer could alter their contents at any time.

USE OF SCRATCH PADS. Do not attempt to read from and store into the same Scratch Pad in the same cycle (microinstruction). These registers use latches for storage elements, and this type of usage would result in a race condition.

USE OF F-REGISTER. The F-register is used by memory protect as a fence register. If it is necessary to use the F-register, the microprogram must first save the contents of this register (see next paragraph for a suggestion). On exit from the routine, the microprogram must restore the contents of the F-register.

USE OF CORE LOCATIONS 0 AND 1. Since software cannot access core locations 0 and 1, these locations may be used by firmware for temporary storage locations. For example the F- and P-register contents may be stored in these locations and thus free these two registers for use by the microprogram; the contents, of course, must be restored on exit from the routine. Table 5 shows the access technique. Line 1 stores a value of 0 into the M-register and issues a clear/write command, along with the required skip (UNC). Line 2 is a NOP, and line 3 stores the contents of the F-register into location 0. Line 4 creates a value of 1, and lines 5, 6, and 7 store the contents of the P-register into location 1. To restore the F-register, begin by storing 0 into the M-register and issuing RW (line 8). While we are waiting for the read/write memory cycle, this is a good time to create the value of 1 (line 9). Line 10 stores the fetched contents of the T-register into the F-register. (Note that this differs from the usual method of reading, which would specify AAB and COND in the R- and S-bus fields.) Similarly, lines 11 and 12 read location 1 and store the contents into the P-register.

Table 5. Storing/Reading Locations 0 and 1

R	S	FN	ST	SP	SK	Comments
		IOR	M	CW	UNC	1 Address location 0
		IOR				2 Skip
F	RRS	IOR	T			3 Store F in location 0
		INC	S4			4 Put 1 in S4
	S4	IOR	M	CW	UNC	5 Address location 1
		IOR				6 Skip
	P	IOR	T			7 Store P in location 1
		IOR	M	RW		8 Read location 0
		INC	S1			9 Put 1 in S1
	T	IOR	F			10 Restore F
	S1	IOR	M	RW		11 Read location 1
	T	IOR	P			12 Restore P

INFORMATION CARRYOVER. Do not expect information to be carried over in any registers (except A- and B-registers) from one microprogram to the next. A power failure can alter their contents.

LONG SHIFTS. When doing left shifts on 32-bit quantities, the F- and Q-registers must be used, with the F-register containing the high-order bits. For right shifts on 32-bit quantities, the B- and A-registers must be used, with the B-register containing the high-order bits.

SHIFTING THE F-REGISTER. The F-register may be right-shifted by itself, but not left-shifted. See detailed note in section 2.

MICROPROGRAM LISTING

for

BASIC INSTRUCTION SET

ROM Adrs	ROM Word (octal)	Entry Point Label	Field Contents					
			R	S	FN	ST	SP	SK
0000	77330757	PH1A	–	P	CFLG	M	RW	–
0001	53771775		AAB	COND	IOR	IR	–	EOP
0002	73373557		–	ADR	IOR	S1	AAB	–
0004	70330757	PH1B	–	CIR	CFLG	M	RW	–
0005	77054377		–	P	SUB	P	–	–
0006	77154377		–	P	NOR	P	–	–
0007	53771775		AAB	COND	IOR	IR	–	EOP
0010	73373557		–	ADR	IOR	S1	AAB	–
0011	77054375	PDEC	–	P	SUB	P	–	EOP
0012	77154377		–	P	NOR	P	–	–
0014	75770757	PH2	–	S1	IOR	M	RW	–
0015	53773775		AAB	COND	IOR	S1	–	EOP
0016	75777557		–	S1	IOR	–	AAB	–
0017	45167635	XX	CAB	S2	XOR	–	ECYN	EOP
0020	77777777		–	–	IOR	–	–	–
0021	47777657	ASGD	CAB	–	IOR	–	ASG1	–
0022	47525675		CAB	–	ADDO	CAB	ASG2	EOP
0023	77777777		–	–	IOR	–	–	–
0025	77775657	ASGA	–	–	IOR	CAB	ASG1	–
0026	47525675		CAB	–	ADDO	CAB	ASG2	EOP
0027	77777777		–	–	IOR	–	–	–
0031	47555657	ASGB	CAB	–	NOR	CAB	ASG1	–
0032	47525675		CAB	–	ADDO	CAB	ASG2	EOP
0033	77777777		–	–	IOR	–	–	–
0035	77555657	ASGC	–	–	NOR	CAB	ASG1	–
0036	47525675		CAB	–	ADDO	CAB	ASG2	EOP
0037	77777777		–	–	IOR	–	–	–

ROM Adrs	Comments
0000	Send current instr adrs to memory, start read cycle.
0001	Put instr from memory or A/B into I-reg, set next phase.
0002	If instr is MRG, put operand adrs in S1, else NOP.
0004	Send interrupt adrs to memory and start read cycle.
0005	Decrement the
0006	P-register.
0007	Put trap cell instr into I-reg and set next phase.
0010	If MRG, put operand adrs in S1, else NOP.
0011	Decrement the P-register and
0012	set next phase.
0014	Send indirect adrs to memory, start read cycle.
0015	Put operand adrs in S1 (also maybe ind), set next phase.
0016	Test operand adrs for A- or B-reg.
0017	Exclusive-OR the contents of A/B-reg with S2 and
0020	set Carry if result is non-zero. Set next phase.
0021	Executes the ASG instructions where I-reg bits 8:9 = 0.
0022	
0023	
0025	Executes the ASG instructions where I-reg bits 8:9
0026	indicate CLA/B.
0027	
0031	Executes the ASG instructions where I-reg bits 8:9
0032	indicate CMA/B.
0033	
0035	Executes the ASG instructions where I-reg bits 8:9
0036	indicate CCA/B.
0037	

ROM Adrs	ROM Word (octal)	Entry Point Label	Field Contents					
			R	S	FN	ST	SP	SK
0040	77777737	FLAG	—	—	IOR	—	IOG1	—
0041	77777777		—	—	IOR	—	—	—
0042	77777775		—	—	IOR	—	—	EOP
0043	77777777		—	—	IOR	—	—	—
0060	77777737	MI*	—	—	IOR	—	IOG1	—
0061	77777777		—	—	IOR	—	—	—
0062	70777775		—	IOI	IOR	—	—	EOP
0063	40775777		CAB	IOI	IOR	CAB	—	—
0064	77777737	LI*	—	—	IOR	—	IOG1	—
0065	77777777		—	—	IOR	—	—	—
0066	70777775		—	IOI	IOR	—	—	EOP
0067	70775777		—	IOI	IOR	CAB	—	—
0070	77777737	OT*	—	—	IOR	—	IOG1	—
0071	42377777		CAB	RRS	IOR	—	—	—
0072	42370375		CAB	RRS	IOR	IOO	—	EOP
0073	42370377		CAB	RRS	IOR	IOO	—	—
0074	77777737	CTRL	—	—	IOR	—	IOG1	—
0075	77777777		—	—	IOR	—	—	—
0076	77777775		—	—	IOR	—	—	EOP
0077	77777777		—	—	IOR	—	—	—
0100	07777117	SRGA	A	—	IOR	A	SRG1	—
0101	07777777		A	—	IOR	—	—	—
0102	07777135		A	—	IOR	A	SRG2	EOP
0103	77777777		—	—	IOR	—	—	—
0104	17776517	SRGB	B	—	IOR	B	SRG1	—
0105	17777777		B	—	IOR	—	—	—
0106	17776535		B	—	IOR	B	SRG2	EOP
0107	77777777		—	—	IOR	—	—	—
0110	75770757	AND	—	S1	IOR	M	RW	—
0111	53773375		AAB	COND	IOR	S2	—	EOP
0112	05147377		A	S2	AND	A	—	—

ROM Adrs	Comments
0040 0041 0042 0043	Synchronize CPU to I/O time T2. Executes the IOG instructions HLT, STF, CLF, SFS, SFC, SOS, SOC.
0060 0061 0062 0063	Synchronize CPU to I/O time T2. Executes the IOG instructions MIA and MIB.
0064 0065 0066 0067	Synchronize CPU to I/O time T2. Executes the IOG instructions LIA and LIB.
0070 0071 0072 0073	Synchronize CPU to I/O time T2. Executes the IOG instructions OTA and OTB.
0074 0075 0076 0077	Synchronize CPU to I/O time T2. Executes the IOG instructions STC, CLC, SOV, CLO.
0100 0101 0102 0103	Executes the SRG instructions involving the A-register.
0104 0105 0106 0107	Executes the SRG instructions involving the B-register.
0110 0111 0112	Send operand adrs to memory and start read cycle. Put data from operand adrs into S2. Set next phase. AND operand data with A-reg, put result back in A-reg.

ROM Adrs	ROM Word (octal)	Entry Point Label	Field Contents					
			R	S	FN	ST	SP	SK
0114	75770754	CP*	–	S1	IOR	M	RW	NAAB
0115	57223017		AAB	–	JMP	S2	XX	–
0116	41167635		CAB	T	XOR	–	ECYN	EOP
0117	77777777		–	–	IOR	–	–	–
0120	75770757	XOR	–	S1	IOR	M	RW	–
0121	53773375		AAB	COND	IOR	S2	–	EOP
0122	05167377		A	S2	XOR	A	–	–
0124	77557557	JMP	–	–	NOR	–	AAB	–
0125	35467412		F	S1	DEC	–	RSS	NMPV
0126	75774375		–	S1	IOR	P	–	EOP
0127	77777777		–	–	IOR	–	–	–
0130	75770757	IOR	–	S1	IOR	M	RW	–
0131	53773375		AAB	COND	IOR	S2	–	EOP
0132	05377377		A	S2	IOR	A	–	–
0134	35460712	ST*	F	S1	DEC	M	CW	NMPV
0135	42376376		CAB	RRS	IOR	AAB	–	UNC
0136	42371375		CAB	RRS	IOR	T	–	EOP
0137	77777777		–	–	IOR	–	–	–
0140	75377461	RRRA	–	S2	IOR	–	CNTR	RPT
0141	17676450		B	–	CRS	B	R1	CTRI
0142	77777775		–	–	IOR	–	–	EOP
0143	77777777		–	–	IOR	–	–	–

ROM Adrs	Comments
0114	Send operand adrs to memory, start read cycle. Skip next line if operand adrs in S1 is not A/B-reg.
0115	Put A/B-reg contents into S2 and jump to 0017.
0116	Exclusive-OR operand with A/B-reg (depending on IR11),
0117	set Carry if result is non-zero. Set next phase.
0120	Send operand adrs to memory, start read cycle.
0121	Put operand data into S2. Set next phase.
0122	Exclusive-OR S2 with A-reg, put result in A-reg.
0124	Clear AAF and BAF.
0125	Test jump adrs in S1, skip next line if violation is detected and if memory protect is enabled.
0126	Put jump address in P-register and set next phase.
0127	
0130	Send operand adrs to memory and start read cycle.
0131	Put operand data into S2. Set next phase.
0132	OR operand in S2 with A-reg, put result in A-reg.
0134	Test operand adrs. If no violation is detected, start memory clear/write cycle and skip next line.
0135	Put A/B-reg data (per IR11) into A/B-reg (per AAF/BAF) and skip next line.
0136	Put A/B-reg data (per IR11) into T-reg for storage into
0137	operand location. Set next phase.
	Continued from 0247:
0140	Put shift count into counter and set repeat mode.
0141	Rotate B- and A-reg right until counter = 17B. Increment counter each shift. Exits with counter = 20B.
0142	Set next phase.
0143	NOP.

ROM Adrs	ROM Word (octal)	Entry Point Label	Field Contents					
			R	S	FN	ST	SP	SK
0144	75770757	ADD	–	S1	IOR	M	RW	–
0145	53773375		AAB	COND	IOR	S2	–	EOP
0146	45125777		CAB	S2	ADDO	CAB	–	–
0150	35460712	JSB	F	S1	DEC	M	CW	NMPV
0151	77346373		–	P	SFLG	AAB	–	AAB
0152	77371366		–	P	IOR	T	–	FLG
0153	75514375		–	S1	INC	P	–	EOP
0154	77777777		–	–	IOR	–	–	–
0155	77756461	MPY	–	–	CLO	B	CNTR	RPT
0156	15036450		B	S2	MPY	B	R1	CTRI
0157	14377403		–	S4	IOR	–	RSS	NEG
0160	15056777		B	S2	SUB	B	–	–
0161	75377403		–	S2	IOR	–	RSS	NEG
0162	14056775		B	S4	SUB	B	–	EOP
0163	77114377		–	P	INC	P	–	–
0164	75770757	LD*	–	S1	IOR	M	RW	–
0165	53773375		AAB	COND	IOR	S2	–	EOP
0166	75375777		–	S2	IOR	CAB	–	–
0170	75770757	ISZ	–	S1	IOR	M	RW	–
0171	57513373		AAB	–	INC	S2	–	AAB
0172	71113376		–	T	INC	S2	–	UNC
0173	75377616		–	S2	IOR	–	ECYZ	UNC
0174	35460712		F	S1	DEC	M	CW	NMPV
0175	75376376		–	S2	IOR	AAB	–	UNC
0176	75371215		–	S2	IOR	T	ECYZ	EOP
0177	77777777		–	–	IOR	–	–	–

ROM Adrs	Comments
0144	Send operand adrs to memory and start read cycle.
0145	Put operand data into S2. Set next phase.
0146	Add A/B-reg data (per IR11) to operand in S2, store result in A/B-reg (per IR11). Enable Overflow.
0150	Test jump address. If no violation is detected, start memory clear/write cycle and skip next line.
0151	If AAF/BAF is set, store P-reg into A/B-reg and skip next line. Set CPU Flag FF.
0152	Store P-reg into T-reg. Skip next line if CPU Flag set.
0153	Increment jump adrs and store in P-reg. Set next phase.
0154	
	Continued from 0213:
0155	Clear counter, B-reg, and Overflow. Set repeat mode.
0156	Execute MPY on B and S2 16 times. Result in B, A-regs.
0157	Skip next line if multiplicand (was A-reg) is positive.
0160	Subtract multiplier from high order word of result.
0161	Skip next line if multiplier (from memory) is positive.
0162	Subtract multiplicand from high word. Set next phase.
0163	Increment P-reg past the DEF software instruction.
0164	Send operand adrs to memory and start read cycle.
0165	Put operand data into S2. Set next phase.
0166	Put S2 contents into A/B-reg, depending on IR11.
0170	Send operand adrs to memory and start read cycle.
0171	Increment A- or B-reg (per AAF/BAF) and save in S2. Skip next line if AAF/BAF is set.
0172	Increment T-reg contents, save in S2. Skip next line.
0173	Set Carry if S2 content is 0. Skip next line.
0174	Test operand adrs. If no violation is detected, start memory clear/write cycle and skip next line.
0175	Put S2 contents into A/B (per AAF/BAF). Skip next line.
0176	Put S2 contents into T-reg. Set carry if S2 content is 0.
0177	Set next phase.

ROM Adrs	ROM Word (octal)	Entry Point Label	Field Contents					
			R	S	FN	ST	SP	SK
0200	77777775		–	–	IOR	–	–	EOP
0201	77777577	ASL	–	–	IOR	–	LEP	–
0202	37222622	LSL	F	–	JMP	S3	ASLA	–
0203	77777577		–	–	IOR	–	LEP	–
0204	37222742	RRL	F	–	JMP	S3	LSLA	–
0205	77777577		–	–	IOR	–	LEP	–
0206	37222752		F	–	JMP	S3	RRLA	–
0207	77777777		–	–	IOR	–	–	–
0210	77777577	MULT	–	–	IOR	–	LEP	–
0211	77207762		–	–	JSB	–	GETAD	–
0212	77207632		–	–	JSB	–	OPGET	–
0213	07222155		A	–	JMP	S4	MPY	–
0214	77777777		–	–	IOR	–	–	–
0215	77777777		–	–	IOR	–	–	–
0216	77777775		–	–	IOR	–	–	EOP
0217	77777777		–	–	IOR	–	–	–
0220	77777577	DIVID	–	–	IOR	–	LEP	–
0221	77227651		–	–	JMP	–	DIV	–
0222	73053377	ASLA	–	ADR	SUB	S2	–	–
0223	17754777		B	–	CLO	F	–	–
0224	07775377		A	–	IOR	Q	–	–
0225	75377461		–	S2	IOR	–	CNTR	RPT
0226	37704430		F	–	ARS	F	L1	CTRI
0227	27777377		Q	–	IOR	A	–	–
0230	37776775		F	–	IOR	B	–	EOP
0231	74774777		–	S3	IOR	F	–	–

ROM Adrs	Comments
0200	Unused.
0201	Legal entry point for ASL. Execute next line.
0202	Save Fence reg in S3, jump to 0222.
0203	Legal entry point for LSL. Execute next line.
0204	Save Fence reg in S3, jump to 0342.
0205	Legal entry point for RRL. Execute next line.
0206	Save Fence reg in S3, jump to 0352.
0207	NOP.
0210	Legal entry point for MPY. Execute next line.
0211	Execute GETAD subroutine. Puts multiplier adrs in S1.
0212	Execute OPGET subroutine. Puts multiplier in S2.
0213	Save multiplicand in S4. Jump to 0155.
0214	Unused.
0215	Unused.
0216	Unused.
0217	Unused.
0220	Legal entry point for DIV. Execute next line.
0221	Jump to 0251.
	Continued from 0202:
0222	Put 2's complement of shift count (IR0:3) into S2.
0223	Put B-reg contents into F-reg (high word). Clear OVF.
0224	Put A-reg contents into Q-reg (low order word).
0225	Put shift count into counter and set repeat mode.
0226	Arith left shift F, Q-regs until counter = 17B. Increment counter each shift. Set OVF if F15, F14 = 10 or 01 at any time. Exits with counter = 20B.
0227	Replace low order word in A-reg.
0230	Replace high order word in B-reg. Set next phase.
0231	Restore fence value to F-register.

ROM Adrs	ROM Word (octal)	Entry Point Label	Field Contents					
			R	S	FN	ST	SP	SK
0232	75770757	OPGET	–	S1	IOR	M	RW	–
0233	53773377		AAB	COND	IOR	S2	–	–
0234	77247411		–	–	CJMP	–	PDEC	–
0235	77657777		–	–	RSB	–	–	–
0236	77777777		–	–	IOR	–	–	–
0237	77777775		–	–	IOR	–	–	EOP
0240	77777777		–	–	IOR	–	–	–
0241	77777577	ASR	–	–	IOR	–	LEP	–
0242	77227732	LSR	–	–	JMP	–	ASRA	–
0243	77777577		–	–	IOR	–	LEP	–
0244	77227736	RRR	–	–	JMP	–	LSRA	–
0245	77777577		–	–	IOR	–	LEP	–
0246	73053377		–	ADR	SUB	S2	–	–
0247	77227540		–	–	JMP	–	RRRA	–
0250	77777777		–	–	IOR	–	–	–
0251	77207762	DIV	–	–	JSB	–	GETAD	–
0252	37202232		F	–	JSB	S4	OPGET	–
0253	17764763		B	–	SOV	F	–	NEG
0254	07225262		A	–	JMP	Q	DVS	–
0255	07773777		A	–	IOR	S1	–	–
0256	17772777		B	–	IOR	S3	–	–
0257	75455364		–	S1	SUB	Q	–	COUT
0260	74554776		–	S3	NOR	F	–	UNC
0261	74454777		–	S3	SUB	F	–	–
0262	75373403	DVS	–	S2	IOR	S1	RSS	NEG
0263	75453377		–	S1	SUB	S2	–	–
0264	35057763		F	S2	SUB	–	–	NEG
0265	37226702		F	–	JMP	B	DONE	–

ROM Adrs	Comments
0232	Send operand adrs to memory and start read cycle.
0233	Put operand data into S2.
0234	Jump to 0011 if interrupt or panel halt. Else continue.
0235	Return to calling routine.
0236	Unused.
0237	Unused.
0240	Unused.
0241	Legal entry point for ASR. Execute next line.
0242	Jump to 0332.
0243	Legal entry point for LSR. Execute next line.
0244	Jump to 0336.
0245	Legal entry point for RRR. Execute next line.
0246	Put 2's complement of shift count (IR0:3) into S2.
0247	Jump to 0140.
0250	Unused.
	Continued from 0221:
0251	Execute GETAD subroutine. Puts divisor address in S1.
0252	Execute OPGET subroutine. Puts divisor in S2.
0253	Put high order word of dividend in F-reg. Set Overflow and skip next line if dividend is negative.
0254	Put low order word of dividend in Q-reg, jump to 0262.
0255	Dividend is negative. Two's
0256	complement it to make
0257	it positive, and put it
0260	into the F- and Q-
0261	registers.
0262	Save original divisor in S1. Skip next line if positive.
0263	Convert negative divisor to positive.
0264	First overflow check. If dividend high word $\geqslant$ divisor, set Overflow and exit with dividend unaltered.
0265	Exit: put dividend high word back in B-reg, jump to 0302.

ROM Adrs	ROM Word (octal)	Entry Point Label	Field Contents					
			R	S	FN	ST	SP	SK
0266	37664437		F	–	LGS	F	L1	–
0267	77752461		–	–	CLO	S3	CNTR	RPT
0270	35044430		F	S2	DIV	F	L1	CTRI
0271	27773367		Q	–	IOR	S2	–	TBZ
0272	15562403		B	S1	XOR	S3	RSS	NEG
0273	75055377		–	S2	SUB	Q	–	–
0274	24567403		Q	S3	XOR	–	RSS	NEG
0275	77767777		–	–	SOV	–	–	–
0276	37773057		F	–	IOR	S2	R1	–
0277	17777403		B	–	IOR	–	RSS	NEG
0300	75056776		–	S2	SUB	B	–	UNC
0301	75376777		–	S2	IOR	B	–	–
0302	27777377	DONE	Q	–	IOR	A	–	–
0303	74374775		–	S4	IOR	F	–	EOP
0304	77114377		–	P	INC	P	–	–
0310	77777577	DLD	–	–	IOR	–	LEP	–
0311	77207762		–	–	JSB	–	GETAD	–
0312	75513377		–	S1	INC	S2	–	–
0313	75770757		–	S1	IOR	M	RW	–
0314	53777377		AAB	COND	IOR	A	–	–
0315	75370757		–	S2	IOR	M	RW	–
0316	53776775		AAB	COND	IOR	B	–	EOP
0317	77114377		–	P	INC	P	–	–
0320	77777577	DST	–	–	IOR	–	LEP	–
0321	77207762		–	–	JSB	–	GETAD	–
0322	35460712		F	S1	DEC	M	CW	NMPV

ROM Adrs	Comments
0266	Logical left shift the dividend (F, Q-regs) one place.
0267	Clear OVF, S3 reg, and counter. Set repeat mode.
0270	Execute DIV on F-reg and S2 16 times. Positive quotient is left in Q-reg, and 2X remainder in F-reg.
0271	Save quotient in S2 for negation test. Skip next line if contents of Q-reg = 0.
0272	Compare signs of dividend and divisor, save result in S3. Skip next line if signs are alike.
0273	2's complement quotient and put back in Q-reg.
0274	Compare quotient sign with expected sign. Skip next line if the same. This tests for most neg integer = 100. . .00.
0275	Set OVF.
0276	Divide remainder by 2 (shift right) and save in S2.
0277	Skip next line if dividend is positive.
0300	2's complement remainder and put in B-reg. Skip next line.
0301	Put remainder in B-register.
0302	Put quotient into A-reg (or dividend low order word if entered from 0265).
0303	Restore fence value to F-reg. Set next phase.
0304	Increment P-reg past the software DEF instruction.
0310	Legal entry point for DLD.
0311	Use GETAD subroutine (0362) to fetch adrs of first word.
0312	Increment first word address, and put in S2.
0313	Send first word address to memory and start read cycle.
0314	Put first operand in A-register.
0315	Send 2nd word address to memory and start read cycle.
0316	Put second operand in B-register. Set next phase.
0317	Increment P-reg past the software DEF instruction.
0320	Legal entry point for DST.
0321	Use GETAD subroutine (0362) to fetch adrs of first word.
0322	Test 1st adrs for memory protect violation. If none, send it to memory (S1 stored in M regardless), start a clear/write cycle, and skip next line.

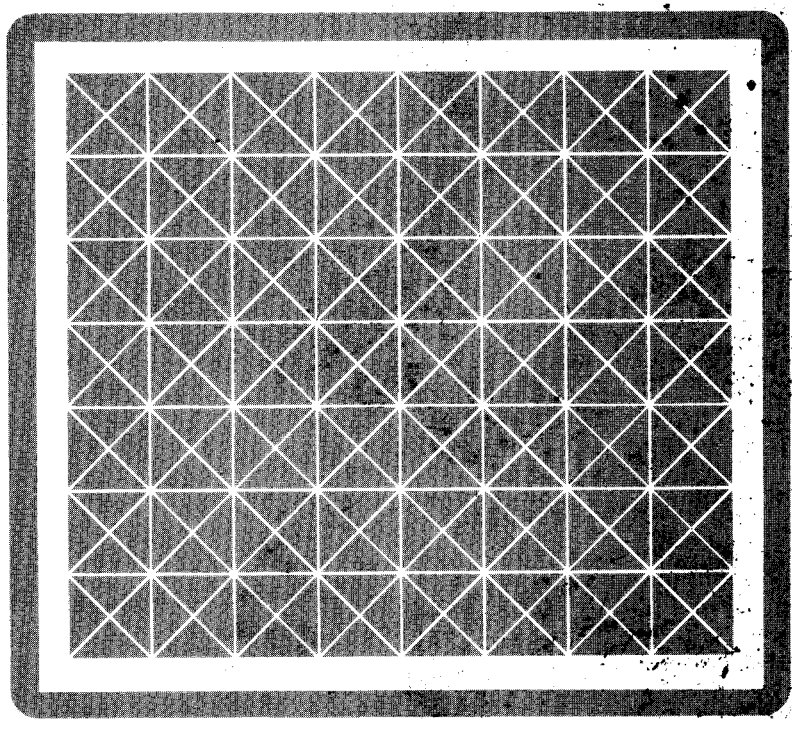
ROM Adrs	ROM Word (octal)	Entry Point Label	Field Contents					
			R	S	FN	ST	SP	SK
0323	02376376		A	RRS	IOR	AAB	–	UNC
0324	02371377		A	RRS	IOR	T	–	–
0325	75512557		–	S1	INC	S3	AAB	–
0326	34460712		F	S3	DEC	M	CW	NMPV
0327	12376376		B	RRS	IOR	AAB	–	UNC
0330	12371375		B	RRS	IOR	T	–	EOP
0331	77114377		–	P	INC	P	–	–
0332	73053377	ASRA	–	ADR	SUB	S2	–	–
0333	75357461		–	S2	CLO	–	CNTR	RPT
0334	17706450		B	–	ARS	B	R1	CTRI
0335	77777775		–	–	IOR	–	–	EOP
0336	73053377	LSRA	–	ADR	SUB	S2	–	–
0337	75377461		–	S2	IOR	–	CNTR	RPT
0340	17666450		B	–	LGS	B	R1	CTRI
0341	77777775		–	–	IOR	–	–	EOP
0342	73053377	LSLA	–	ADR	SUB	S2	–	–
0343	17774777		B	–	IOR	F	–	–
0344	07775377		A	–	IOR	Q	–	–
0345	75377461		–	S2	IOR	–	CNTR	RPT
0346	37664430		F	–	LGS	F	L1	CTRI
0347	37776777		F	–	IOR	B	–	–
0350	27777375		Q	–	IOR	A	–	EOP
0351	74774777		–	S3	IOR	F	–	–

ROM Adrs	Comments
0323	Put 1st word in A/B-reg if AAF/BAF set. Skip next line.
0324	Send 1st word to memory (T-reg) for storing.
0325	Incr 1st word adrs, put in S3. Set AAF/BAF if 0 or 1.
0326	Test 2nd adrs for memory protect violation. If none, send it to memory (S3 stored in M regardless), start clear/write cycle, and skip next line.
0327	Put 2nd word in A/B-reg if AAF/BAF set. Skip next line.
0330	Send 2nd word to memory for storing. Set next phase.
0331	Increment P-reg past the software DEF instruction.
	Continued from 0242:
0332	Put 2's complement of shift count (IR0:3) into S2.
0333	Put shift count into counter, set repeat mode, clear OVF.
0334	Arith right shift B, A-regs until counter = 17B. Increment counter each shift. Exits with counter = 20B.
0335	Set next phase.
	Continued from 0244:
0336	Put 2's complement of shift count (IR0:3) into S2.
0337	Put shift count into counter and set repeat mode.
0340	Logical right shift B, A-regs until counter = 17B. Increment counter each shift. Exits with counter = 20B.
0341	Set next phase.
	Continued from 0204:
0342	Put 2's complement of shift count (IR0:3) into S2.
0343	Put high order word into F-register.
0344	Put low order word into Q-register.
0345	Put shift count into counter and set repeat mode.
0346	Logical left shift F, Q-regs until counter = 17B. Increment counter each shift. Exits with counter = 20B.
0347	Put high order word into B-register.
0350	Put low order word into A-register. Set next phase.
0351	Restore fence value to F-register.

ROM Adrs	ROM Word (octal)	Entry Point Label	Field Contents					
			R	S	FN	ST	SP	SK
0352	73053377	RRLA	–	ADR	SUB	S2	–	–
0353	17774777		B	–	IOR	F	–	–
0354	07775377		A	–	IOR	Q	–	–
0355	75377461		–	S2	IOR	–	CNTR	RPT
0356	37674430		F	–	CRS	F	L1	CTRI
0357	37776777		F	–	IOR	B	–	–
0360	27777375		Q	–	IOR	A	–	EOP
0361	74774777		–	S3	IOR	F	–	–
0362	77370757	GETAD	–	P	IOR	M	RW	–
0363	53773403	ONEMO	AAB	COND	IOR	S1	RSS	NEG
0364	77227766		–	–	JMP	–	IND	–
0365	75657557		–	S1	RSB	–	AAB	–
0366	77247411	IND	–	–	CJMP	–	PDEC	–
0367	75770757		–	S1	IOR	M	RW	–
0370	77227763		–	–	JMP	–	ONEMO	–
0371	77777777		–	–	IOR	–	–	–
0372	77777777		–	–	IOR	–	–	–
0373	77777775		–	–	IOR	–	–	EOP
0374	77777777		–	–	IOR	–	–	–
0375	77777777		–	–	IOR	–	–	–
0376	77777775		–	–	IOR	–	–	EOP
0377	77777777		–	–	IOR	–	–	–

ROM Adrs	Comments
	Continued from 0206:
0352	Put 2's complement of shift count (IR0:3) into S2.
0353	Put high order word into F-register.
0354	Put low order word into Q-register.
0355	Put shift count into counter and set repeat mode.
0356	Rotate F, Q-registers left until counter = 17B. Increment counter each shift. Exits with counter = 20B.
0357	Put high order word into B-register.
0360	Put low order word into A-register. Set next phase.
0361	Restore fence value to F-register.
0362	Send P-reg adrs to memory, start read cycle. Normally fetches contents of a software DEF instruction.
0363	Put fetched word into S1. Skip next line if not indirect (i.e., if bit 15 = 0).
0364	Jump to 0366.
0365	Set AAF/BAF if S1 contents = 0 or 1. Return to calling subroutine.
0366	Jump to 0011 if interrupt or panel halt. Else continue.
0367	Send operand adrs to memory and start read cycle.
0370	Jump to 0363.
0371	Unused.
0372	Unused.
0373	Unused.
0374	Unused.
0375	Unused.
0376	Unused.
0377	Unused.

HEWLETT hp PACKARD



5951-3028

Printed in U.S.A. 2/72