Technical Reference

MICROPROGRAMMING WITH THE ECLIPSE COMPUTER WCS FEATURE

014-000050-00

Ordering No. 014-000050

Printed in the United States of America
Rev. 00, November 1974

TABLE OF CONTENTS

SECTION 1

INTRODUCTION TO MICROPROGRAMMED CONTROL

Page

INTRODUCTION 1-1

SECTION 2

ARCHITECTURE AND OPERATION OF THE ECLIPSE CPU

TIMING LOGIC 2-1

MICROPROGRAMMED CONTROL LOGIC 2-1
- Control Store 2-1
- Microinstruction Register 2-2
- State Change Logic 2-2
  - Sources of Control Store Address 2-2
  - Special Control Store Addresses 2-3
  - Phantom Microinstructions 2-3

COMPUTATIONAL LOGIC 2-3
- Register File 2-3
- A Bus 2-3
- ALU 2-4
- Shifter 2-4
- LINK 2-4
- CARRY 2-4
- ALU0 SAVE and Q-BIT SAVE 2-4
- COUNT Register 2-4
- BIT decode Logic 2-4
- Program Load ROM 2-4

DATA TRANSFER LOGIC 2-5
- LA Bus 2-5
- MEM Bus 2-5
- I/O Bus 2-5
- The I/O Register 2-5
- Programmed I/O Logic 2-5
- Program Interrupt Logic 2-5
- Data Channel Logic 2-6

CPU CYCLES 2-6

MEMORY 2-6

TABLE OF CONTENTS (Continued)

SECTION 2 (Continued)

ARCHITECTURE AND OPERATION OF THE ECLIPSE CPU

	Page
CONSOLE	2-7
Console Lights	2-7
Address Compare Logic	2-8
Console Flip-flops	2-8
CONRQ Flip-flop	2-8
STOP Flip-flop	2-8
REXAM Flip-flop	2-9
MICROINST ENAB and STEP SYNC Flip-flops	2-9
Reset	2-9

SECTION 3

MICROINSTRUCTION FORMAT AND MICRO-ORDER SET

A INPUT FIELD	3-3
AREG AND BREG FIELDS	3-5
ALU FIELD	3-7
Inputs	3-7
Outputs	3-7
SHIFT FIELD	3-11
LOAD FIELD	3-15
CARRY FIELD	3-16
EXAMPLES	3-18
MA FIELD	3-20
MBUS FIELD	3-21
RAND1 FIELD	3-26
RAND2 FIELD	3-30

TABLE OF CONTENTS (Continued)

SECTION 3 (Continued)

MICROINSTRUCTION FORMAT AND MICRO-ORDER SET

	Page
STATE CHANGE FIELD	3-33
Unconditional Branches	3-33
True/False Branches	3-33
ALU Result Tests	3-34
ALU Carry-out Tests	3-34
Special Bit Tests	3-35
Tests That Modify a Register	3-36
Console Tests	3-36
I/O Bus Tests	3-37
Floating Point Processor Tests	3-38
Microsubroutining	3-38
Instruction Decoding	3-39
Microroutine Chaining	3-40
Execution of a New Instruction	3-40
Transfer of Control to a Special Microroutine	3-41
Using LDIR and NILDIR	3-42
PAGE FIELD	3-45
TRUE ADDRESS FIELD	3-45
FALSE ADDRESS FIELD	3-45

SECTION 4

HOW TO USE WCS

INTRODUCTION	4-1

This page left blank intentionally

PREFACE

The writable control store (WCS) feature of the ECLIPSE computer is an extension of the microprogrammed control logic of the computer's central processing unit. WCS gives users access to the microprogrammed control logic of the CPU and thus allows them to implement specialized instructions. WCS offers the user 256 56-bit words of high-speed random-access semiconductor memory (RAM) in the CPU's control store. Information placed in this RAM defines the execution of sixteen two-accumulator instructions (the XOP1 instructions).

This manual is intended to be a reference for the user of WCS. It describes those parts of the ECLIPSE computer's hardware that are important to the microprogrammer. There is special emphasis on the architecture and operation of the CPU, and the console and memory are also described in some detail. Other ECLIPSE computer features including memory allocation and protection (MAP), error checking and correction (ERCC), the parallel floating point processor (FPP), and I/O bus are not discussed in detail.

Before the microprogrammer continues with this manual, he should read the programmer's reference manual for the ECLIPSE computer, DGC #015-000024. It will give him an overview of the computer's architecture and a description of the standard instruction set that motivates the microprogramming capability described in this manual.

This page left blank intentionally

SECTION 1

INTRODUCTION TO MICROPROGRAMMED CONTROL

In order for a CPU to execute an instruction read from its main memory, various combinations of control signals must be asserted in sequence. A computer's control logic determines which control signals to assert and when to assert them. The instruction register (IR), a special register dedicated to storing instructions read from main memory, is a principal input to the control logic. Other inputs to the control logic are data signals from the processor's data paths.

The chief difference between a computer with microprogrammed control and one without it is the way in which control signals are generated. In a conventional machine, a complex of multiplexors, decoders and gates is fed by the IR. They assert some subset of all possible control signals in the machine. A timing generator asserts various timing signals in sequence, and one or more of these timing signals is gated with each control signal, so that a control signal may affect the flow of data in the machine only for certain instructions in the IR and even then only at certain times. Other control signals may be derived from data in the CPU and therefore depend on the contents of the IR only indirectly or not at all. These signals, too, must be gated with timing signals so that they affect CPU operations only at the right times. It is characteristic of computers without microprogrammed control that their control logic is extremely complex, and their architecture demands that a variety of timing signals be routed throughout the processor.

To the microprogrammer, a CPU is a collection of operational components, data paths, and control signals. Operational components are circuits that transform and store data (e.g., shifters and registers). Data paths carry data from one component to another. Control signals determine how and when data will move from component to component along the data paths: they must determine which function an operational component will perform if it can perform more than one; they must select the source of a data path that has more than one; and they must enable flip-flops and registers to be clocked at the right times.

In a computer with microprogrammed control, control signals are not derived directly from the IR. Instead, every combination of control signals which is necessary to the operation of the machine is stored in read-only (ROM) or random-access (RAM) semiconductor memory. The information required to generate one combination of control signals is called a microinstruction. The semiconductor memory where microinstructions are stored is called the control store.

In this manual the term memory will be used in contrast with the term control store to refer to main memory unless it is further qualified. Similarly, the term instruction will be used in contrast with the term microinstruction to refer to a higher (i.e., assembly-language) level instructions stored in main memory.

Since a computer usually has hundreds of control signals, the number of possible combinations of these is astronomical. Fortunately, only a few hundred combinations are actually necessary during the operation of the computer, so that the control store need only hold a few hundred microinstructions.

Microinstructions read from the control store are placed in the microinstruction register, which is like a conventional processor's IR in that it is dedicated to one function and control signals are derived from it. However, the logic that decodes the microinstruction register to produce control signals is much simpler than the logic that decodes the IR in a conventional machine for two reasons.

First, very few timing signals are gated with the control signals derived from a microinstruction. Control signals asserted by a microinstruction are asserted simultaneously and remain asserted for as long as the microinstruction remains in the microinstruction register. When a new combination of control signals is required, a new microinstruction must be read into the microinstruction register.

Second, the format of microinstructions is chosen specifically for the purpose of generating control signals, with the intent of keeping the microinstruction decoding logic as simple as possible. In an extremely simple processor, each bit in a microinstruction might correspond to one control signal so that there would be no need for decoding logic at all. In most processors there are too many control signals to make this practical. Instead, a group of bits in a microinstruction, called a field, is often used to select among several control signals or to determine the function to be performed by one operational component in the processor. A particular combination of bits in a field of a microinstruction is called a micro-order. The microinstruction decoding logic generates the control signal or signals corresponding to the micro-order in each field of a microinstruction. The use of multiplexors and small read-only semiconductor memories makes the decoding of fields in the microinstruction a fairly simple task.

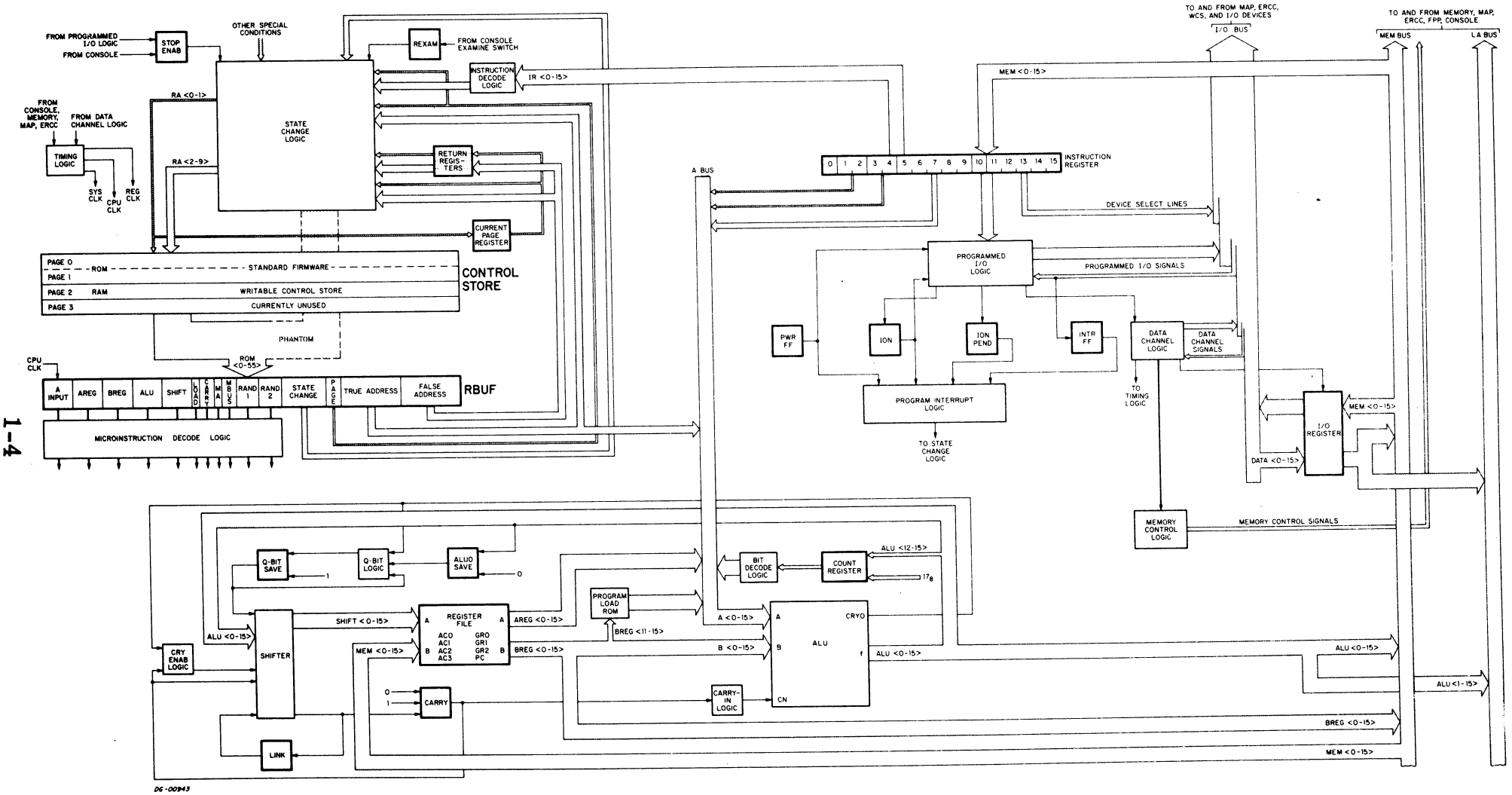
Instructions fetched from memory are stored in the IR while they are being executed. Since the execution of an instruction in the IR requires various combinations of control signals to be asserted in sequence, many microinstructions may be read from the control store to execute one instruction in the IR. A sequence of microinstructions that executes an instruction in the IR is called a microroutine. Microroutines stored in the control store ROM are called firmware.

Not all microroutines execute an instruction in the IR. A microroutine might perform some other processor function. For example, when a program interrupt occurs in a processor with microprogrammed control, a microroutine might store the contents of the program counter (PC) in a special memory location and place the address of the interrupt service routine in the PC. Or when the computer is halted, a special microroutine that might read the control switches from the console (e.g., deposit, examine) and perform the appropriate function when a switch is pressed.

In this manual, the combination of control signals asserted in a CPU at any instant is called the state of the machine. Because a microinstruction asserts one combination of control signals, each microinstruction generates one machine state. A microroutine in the control store is simply a representation of a sequence of machine states that result in the performing of a CPU operation.

The logic that determines the order in which microinstructions will be read from the control store is called the state change logic. The inputs to the state change logic are various signals that come from the data paths, the IR and the current microinstruction. The outputs are the address in the control store of the next microinstruction. To the reader who is accustomed to thinking in terms of machine states, this address represents the next state of the processor.

This page left blank intentionally


FROM PROGRAMMED I/O LOGIC
FROM CONSOLE
STOP ENAB
OTHER SPECIAL CONDITIONS
REXAM
FROM CONSOLE EXAMINE SWITCH
INSTRUCTION DECODE LOGIC
IR <0-15>
RA <0-1>
RA <2-9>
STATE CHANGE LOGIC
RETURN REGISTERS
CURRENT PAGE REGISTER
FROM CONSOLE, MEMORY, MAP, ERCC
FROM DATA CHANNEL LOGIC
TIMING LOGIC
SYS CLK
CPU CLK
REG CLK
PAGE 0
ROM
STANDARD FIRMWARE
PAGE 1
PAGE 2
RAM
WRITABLE CONTROL STORE
PAGE 3
CURRENTLY UNUSED
CONTROL STORE
PHANTOM
CPU CLK
ROM <0-55>
A INPUT
AREG
BREG
ALU
SHIFT
LOAD
CARRY
MA
MBUS
RAND 1
RAND 2
STATE CHANGE
PAGE
TRUE ADDRESS
FALSE ADDRESS
RBUF
MICROINSTRUCTION DECODE LOGIC
TO AND FROM MAP, ERCC, WCS, AND I/O DEVICES
I/O BUS
TO AND FROM MEMORY, MAP, ERCC, FPP, CONSOLE
MEM BUS
LA BUS
MEM <0-15>
A BUS
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
INSTRUCTION REGISTER
DEVICE SELECT LINES
PROGRAMMED I/O LOGIC
PROGRAMMED I/O SIGNALS
PWR FF
ION
ION PEND
INTR FF
DATA CHANNEL LOGIC
DATA CHANNEL SIGNALS
PROGRAM INTERRUPT LOGIC
TO TIMING LOGIC
TO STATE CHANGE LOGIC
I/O REGISTER
MEM <0-15>
DATA <0-15>
MEMORY CONTROL LOGIC
MEMORY CONTROL SIGNALS
Q-BIT SAVE
Q-BIT LOGIC
ALUO SAVE
BIT DECODE LOGIC
COUNT REGISTER
ALU <12-15>
17_8
PROGRAM LOAD ROM
REGISTER FILE
ACO ACI AC2 AC3
GRO GRI GR2 PC
AREG <0-15>
BREG <11-15>
A <0-15>
B <0-15>
BREG <0-15>
ALU
CRYO
CN
f
ALU <0-15>
ALU <1-15>
SHIFT <0-15>
SHIFTER
ALU <0-15>
CRY ENAB LOGIC
MEM <0-15>
CARRY
CARRY-IN LOGIC
LINK
BREG <0-15>
MEM <0-15>
DS-00943

SECTION 2

ARCHITECTURE AND OPERATION OF THE ECLIPSE CPU

The circuitry of the ECLIPSE computer central processing unit is divided into four sections: timing logic, microprogrammed control logic, computational logic, and data transfer logic. The timing logic generates timing signals used throughout the ECLIPSE computer system; the microprogrammed control generates control signals in the sequence required to perform CPU operations; the computational logic is where the CPU actually performs operations on data; and the data transfer logic is used to move data between the CPU and external media (memory, I/O devices, etc.). Although the memory and console are not, strictly speaking, a part of the CPU, an understanding of their operation is essential to the microprogrammer, so they will be discussed in this section.

The following discussion of the four sections of the CPU is complemented by a block diagram. In this diagram the timing logic and microprogrammed control logic appear in the upper left corner, the computational logic appears at the bottom, and the data transfer logic appears on the right.

The following conventions are used in the block diagram:

(1) Operational components and logic networks are represented by boxes; control and data paths are represented by arrows.

(2) The width of control and data paths corresponds roughly to the amount of information they carry in parallel.

(3) Registers (that is, data storage components that are clocked) are shown with heavy outlines.

TIMING LOGIC

The timing logic generates three timing signals that concern the microprogrammer. SYS CLK, CPU CLK, and REG CLK.

SYS CLK is the fundamental timing signal in the ECLIPSE computer system. It has a period of 100ns and is used to generate all other timing signals in the CPU, memory, and FPP.

CPU CLK is derived from SYS CLK. It ordinarily has a period of 200ns and is the principal clock signal used in the CPU. Because CPU CLK is the signal that clocks the microinstruction register, it defines the basic CPU cycle. When the memory or the data channel wants to suspend CPU operation, it does so by freezing CPU CLK.

REG CLK is a copy of CPU CLK except that it is not frozen by the data channel. It is used to clock registers that must be loaded during a data channel transfer.

MICROPROGRAMMED CONTROL LOGIC

The microprogrammed control logic has three principal parts: the control store, the microinstruction register, and the state change logic.

Control Store

The control store may contain as many as 1024_{10} 56-bit microinstructions and is addressed by the 10-bit output of the state change logic. The two high-order bits of this address select one of four pages in the control store. The remaining eight bits select one of 256 addresses within the page. Pages 0 and 1 are read-only semiconductor memory (ROM) and contain microinstructions used to implement the ECLIPSE computer's standard instruction set. Those ECLIPSE computers that have the writable control store (WCS) feature have random-access semiconductor memory (RAM) for page 2. Page 3 is reserved for future use.

Microinstruction Register

The microinstruction register, called RBUF, is a 56-bit register dedicated to storing a copy of the current microinstruction; it is clocked by CPU CLK. The microinstruction decode logic decodes the contents of fields in RBUF to produce control signals that determine CPU operation. The bits in RBUF are numbered 0-55 from left to right. They are divided into fields as follows:

Bits	Field Name
0-3	A INPUT
4-7	AREG
8-11	BREG
12-15	ALU
16-19	SHIFT
20	LOAD
21-22	CARRY
23	MA
24-25	MBUS
26-28	RAND1
29-31	RAND2
32-37	STATE CHANGE
38-39	PAGE
40-47	TRUE ADDRESS
48-55	FALSE ADDRESS

There is a 4-bit extension of RBUF in the memory allocation and protection (MAP) feature. The contents of this extension are determined by a ROM, which is, strictly speaking, an extension of the control store. The 4-bit extension of RBUF is not visible to the WCS feature and will not be discussed in this manual.

State Change Logic

The state change logic determines the next microinstruction to be loaded into RBUF. Ordinarily, it determines the next microinstruction by selecting a control store address from one of four sources and using that address to select a microinstruction in the control store. However, when a new instruction is to be loaded into the IR, the state change logic may take the next microinstruction from a special location in the control store, or it may bypass the control store and generate a microinstruction directly.

Sources of Control Store Address

The microprogrammed control logic does not include a microprogram counter. Consequently, a microinstruction in RBUF must contain information to select the next microinstruction to be read from the control store. This information resides in the STATE CHANGE, PAGE, TRUE ADDRESS, and FALSE ADDRESS fields (bits 32-55) of a microinstruction and is input to the state change logic.

Other inputs to the state change logic are the instruction decode logic, the CURRENT PAGE register, and the microsubroutine return register file.

The instruction decode logic produces an 8-bit address called a decode address according to certain bits in the IR. The address it produces is selected from a ROM if the IR contains one of the ECLIPSE computer's standard instructions or the CPU does not include WCS. However, if the IR contains an XOP1 instruction and WCS is included in the CPU, the decode address will come from the WCS decode RAM. The instruction decode logic is used by the state change logic to "find" the microroutine that will execute an instruction in the IR. When the state change logic uses the instruction decode logic to select the next microinstruction to be loaded into RBUF, the instruction in the IR is said to be "decoded".

The 2-bit CURRENT PAGE register is used to keep a record of the current page. This register is fed by the two page bits generated by the state change logic. It is clocked by CPU CLK, at the same time RBUF is clocked. Therefore, the CURRENT PAGE register always contains the page bits that were used to address the microinstruction currently in RBUF.

The microsubroutine return register file contains four 10-bit registers used to save return addresses for microcode subroutines. When a return address is saved, it is the address in the FALSE ADDRESS field combined with the contents of the CURRENT PAGE register.

The state change logic ordinarily chooses a 10-bit control store address from one of the following sources.

(1) the 2-bit contents of the PAGE field combined with the 8-bit contents of the TRUE ADDRESS field,

(2) the 2-bit contents of the PAGE field combined with the 8-bit output of the instruction decode logic,

(3) the 2-bit contents of the CURRENT PAGE register combined with the 8-bit contents of the FALSE ADDRESS field, and

(4) the four-register file used to store return addresses for microsubroutines.

The STATE CHANGE field of the microinstruction in RBUF determines how the state change logic will select among the four sources.

When the PAGE field or CURRENT PAGE register is used as a part of a source (in 1, 2, and 3 above), the two bits it provides become the high-order (page) bits of the resulting 10-bit control store address.

Special Control Store Addresses

If conditions in the CPU demand that control be transferred to a special microroutine, the state change logic will select the next microinstruction from one of three fixed control store locations when a new instruction is to be loaded into the IR. There are three microroutines to which control may be transferred in this way: the halt/stop microroutine, the program interrupt microroutine, and the running examine microroutine. The conditions that force a transfer of control to one of these microroutines are, respectively: stops enabled and a stop pending, interrupts enabled and an interrupt waiting, and the REXAM flip-flop in the console set (by the EXAMINE switch).

Phantom Microinstructions

If the state change logic does not transfer control to a special microroutine when the IR is to be loaded, it generates the next microinstruction to be clocked into RBUF directly. This microinstruction is called a phantom microinstruction because it is not read from the control store. The phantom microinstruction is generated while the new instruction is still on the MEM bus and enters RBUF at the same time the new instruction enters the IR. It selects the instruction decode logic as the source of the address of the succeeding microinstruction so that control will be transferred to the proper microroutine to execute the new instruction.

Although the MEM bus is not shown as an input to the state change logic in the block diagram (for simplicity), the state change logic does use the data on the MEM bus (i. e., the instruction to be loaded into the IR) to determine the phantom microinstruction. It can therefore generate a phantom microinstruction that not only uses the instruction decode logic to select the appropriate microroutine but also performs other useful operations in anticipation of the microroutine to which it will transfer control.

The ECLIPSE computer's instructions fall into three categories, according to the types of phantom microinstructions they generate: (1) The 16-bit memory-reference instructions (JMP, JSR, ISZ, DSZ, LDA, STA, and LEF) generate a phantom microinstruction that computes the effective address specified by the instruction, starts memory on that address, and decodes the instruction in the IR (unless the instruction indicates indirect addressing, in which case the indirect address is resolved before the instruction is decoded). (2) I/O instructions, two-accumulator multiple-operation instructions, and some other short arithmetic and logical instructions generate a phantom microinstruction that increments PC, starts memory to fetch the next instruction, and decodes the instruction in the IR. (3) All of the other instructions, including XOP1 instructions, generate a phantom microinstruction that increments PC and decodes the instruction in the IR.

When a phantom microinstruction causes the state change logic to select the instruction decode logic as the source of the address of the next microinstruction, the page bits of that address are specially generated so that for the ECLIPSE computer's standard instructions page 0 or 1 is selected and for XOP1 instructions page 2 is selected (provided that the CPU has WCS).

COMPUTATIONAL LOGIC

The computational logic in the ECLIPSE computer is built around a register file, an arithmetic/logical unit (ALU), and a shifter. The basic data flow in this part of the processor is from the register file into the ALU, from the ALU into the shifter, and from the shifter back into the register file.

Register File

The register file contains eight 16-bit registers. Four of these (registers 0-3) are used as the programmer-visible accumulators (AC0-AC3). Another one is the program counter (PC or GR3). The remaining three registers are general registers (GR0-GR2) available to the microprogrammer as temporary storage.

Every microinstruction selects an A register and a B register from the register file. (It may select the same register as both the A and B register.)

The contents of the selected registers (or register, if the A and B registers are the same) appear as the respective A and B outputs of the register file. The A output, in one of four forms, may be selected as the source of the A bus. The B output is the B input to the ALU. The low-order five bits of the B output also select a word in the program load ROM.

A microinstruction may specify that its A register be loaded with the output of the shifter and/or that its B register be loaded with data from the MEM bus.

A Bus

The A bus, which supplies the A input to the ALU, has twelve sources, any one of which may be selected by a microinstruction. These sources are described in detail in chapter 3. They include the contents of the A register in one of four forms, zeroes, a microcoded constant taken from the TRUE ADDRESS field of the microinstruction in RBUF, three fields from the IR, a bit mask from the bit-decode logic (see below), and a 16-bit word from the program load ROM (see below).

ALU

The ALU has two 16-bit inputs, called the A and B inputs, and one 16-bit output called ALU<0-15>. It also takes a carry-in and may produce a carry-out. The A input to the ALU is the A bus, whose source is selected by the current microinstruction. The B input to the ALU is the B output of the register file. The ALU may generate any one of nine arithmetic or seven logical functions of its A and B inputs. The ALU function is selected by the microinstruction in RBUF. The carry-in, called CN, may come from the CARRY bit (see below), from the IR when the ALC micro-order appears in the current microinstruction (see chapter 3), or from the control decoding logic when the ALU function is one that always requires a carry-in equal to one. The carry-out, called CRY0 is an input to the CRY ENAB logic, which is used to implement two-accumulator multiple operation instructions.

Shifter

The shifter modifies the output of the ALU and passes the modified result to the A input of the register file. In addition to ALU<0-15>, its inputs include a variety of one-bit registers and the CRY ENAB logic, which is used primarily by two accumulator multiple operation instructions. The shifter may shift left or right one bit or it may swap 8-bit bytes. When data is shifted left or right or passed straight through, bit 0 or bit 15 is filled with one of the one-bit inputs to the shifter. With its ability to shift, swap, and modify end bits of its 16-bit input, the shifter can perform thirteen operations in all. The current microinstruction specifies the operation to be performed.

LINK

In addition to its 16-bit output, the shifter produces a one-bit "shift-out", which is the end bit (bit 0 or 15) that is lost when a data is shifted left or right. The LINK is a one-bit register that stores the shift-out when a left or right shift is performed. the LINK is also an input to the shifter, so that its contents may be shifted into a datum and a bit that has been shifted out may be recovered.

CARRY

Another one-bit register that is an input to the shifter is the CARRY. A microinstruction may set or reset this bit directly, or it may load the CARRY with the output of the CRY ENAB logic or the shift-out. When the microinstruction does not determine the value of the CARRY directly it is determined by certain bits in the IR. This feature is used to implement two-accumulator multiple operation instructions. The CARRY may also be enabled as the source of the carry-in to the ALU by a microinstruction. When it is enabled, the CARRY is inclusive-ORed with any other sources of the carry-in by the carry-in logic.

ALU0 SAVE and Q-BIT SAVE

Three other components of the computational logic that are between the ALU and the shifter are the ALU0 SAVE register, the Q-BIT logic, and the Q-BIT SAVE register. These components are used to perform integer division by the ECLIPSE computer's standard firmware. ALU0 SAVE and Q-BIT SAVE are both enabled for loading by the same micro-order in a microinstruction. When they are loaded, ALU0 SAVE takes the value of ALU0, the high-order bit of the ALU result, and Q-BIT SAVE takes the output of the Q-BIT logic, which is the exclusive-OR of the old value of Q-BIT SAVE, the old value of ALU0 SAVE, and the carry-out from the ALU (CRY0).

COUNT Register

The COUNT register is a 4-bit register that is loaded from ALU<12-15> when it is enabled by the current microinstruction. The register serves two purposes: it may be decremented and tested by the state change logic, and it provides an integer input between 0 and 15 for the BIT decode logic.

BIT decode Logic

The BIT decode logic produces a 16-bit output and may be the source of the A bus. One and only one of the bits in the output word will be set. The bit that is set is determined by the contents of the COUNT register. For purposes of bit decoding, the bits in a word are numbered from high-order to low-order (left to right) starting with zero, so that a 3 in the COUNT register sets the fourth bit from the left end of the BIT decode logic's 16-bit output.

Program Load ROM

The program load (PL) ROM is a 32-word 16-bit read-only memory that contains a copy of a bootstrap loader. It is addressed by the five least significant bits of the B output of the register file. The PL ROM may be the source of the A bus.

DATA TRANSFER LOGIC

The data transfer logic moves data between the CPU, memory, and I/O devices. It consists of the LA bus, the MEM bus, the I/O bus, the I/O register, the programmed I/O logic, the program interrupt logic and the data channel logic.

LA Bus

The LA bus is a 15-bit bus that carries logical addresses to memory, the MAP unit and the address lights on the console. Its source is ordinarily ALU <1-15> (bit 0 of the ALU output is not used), but for data channel transfers it takes an address from the I/O register.

MEM Bus

The MEM bus is a 16-bit bus that carries data between the CPU and memory. It also carries data between the CPU and the console, between the CPU and the I/O register, and between the CPU and the floating point processor. The source and destination of data on the MEM bus is determined by the microinstruction in RBUF. During data channel transfers, data may move on the MEM bus directly between the I/O register and memory.

I/O Bus

The I/O bus can be divided into six groups of signals: the device select (DS) lines, programmed I/O signals from devices, programmed I/O signals to devices, data channel signals from devices, data channel signals to devices, and the DATA lines. A complete description of the I/O bus may be found in the "Interface Manual", DGC 015-000031. The source of the device select lines is IR<10-15>. Programmed I/O signals to and from devices are generated and received by the programmed I/O logic. Similarly, data channel signals are generated and received by the data channel logic. The DATA lines carry 16 bits of information between device controllers and the I/O register.

The I/O Register

The I/O register is a sixteen bit register used to store data being transferred between I/O devices and the CPU or memory. It is controlled by the programmed I/O logic and the data channel logic. It may be loaded with data from the MEM bus or the I/O bus and may place data on either bus.

Programmed I/O Logic

The programmed I/O logic generates signals on the I/O bus and in the CPU necessary for the execution of I/O instructions. This logic will only generate signals when it is enabled by the microinstruction in RBUF. The signals it generates depend on the IR; consequently, it should not be enabled unless the IR contains an I/O instruction. The I/O bus signals it generates are programmed I/O transfer signals (e.g., DATIA, DATOA, INTA) and I/O pulse signals (STRT, CLR, and IOPLS). Although the device select lines reflect the contents of IR<10-15> regardless of whether there is an I/O instruction in the IR, the device that happens to be selected during a CPU cycle does not perform an operation unless it receives a signal from the programmed I/O logic.

Within the CPU, the programmed I/O logic may set the ION and ION PEND flip-flops and the STOP ENAB flip-flop, it may clear the ION flip-flop, and it may select the console data switches as the source of the MEM bus. These operations, like the assertion of I/O transfer and pulse signals, only occur when the programmed I/O logic is enabled by the microinstruction in RBUF.

Even when the programmed I/O logic is not enabled by the microinstruction in RBUF it generates one signal that is asserted when an I/O skip is indicated by IR<8-9> and the state of the device selected by IR<10-15>. If the device code in IR<10-15>is 77, the state of the "device" is determined by the ION flip-flop and the power fail flag (PWR FF). The I/O skip signal is used in the standard ECLIPSE computer firmware to implement I/O SKIP instructions. However, it is a function only of IR<8-15> and may be asserted even if the IR does not contain an I/O skip instruction. The signal is an input to the state change logic and may be tested under microprogram control.

Program Interrupt Logic

The program interrupt logic indicates that an interrupt is waiting if ION is set, ION PEND is clear, and INTR FF or PWR FF is set. The ION flip-flop enables the ECLIPSE computer's interrupt system. It is set by the programmed I/O logic. However, whenever ION is set, ION PEND is also set to inhibit the effect of ION. ION PEND is cleared when the next instruction is fetched from memory and loaded into the IR. Therefore, no program interrupt occurs between an I/O instruction that sets ION and the succeeding instruction. The INTR flip-flop synchronizes device interrupt requests. It is loaded from the INTR line on the I/O bus when CPU CLK rises. PWR FF is set when power is failing.

Data Channel Logic

The ECLIPSE computer's data channel logic is implemented in hardware. It operates in parallel with the processor, asserting the request enable signal (RQENB) every 400ns until some device makes a data channel request. Then the request is acknowledged, and the address of the memory location that the device wants to reference is clocked from the DATA lines of the I/O bus (where the device has placed it) into the I/O register. At this point, the data channel freezes the I/O register so the address will not be lost and waits until the microprogrammed control logic allows data channel operation to continue.

When a microinstruction that allows the data channel to finish is loaded into RBUF, CPU CLK is frozen, and the data channel completes its cycle. This suspension of CPU operation is called a data channel break. During a data channel break, the address in the I/O register is placed on the LA bus, and control signals are generated to start memory. At the same time, if the transfer is into memory, the data from the device is clocked into the I/O register. For transfers into memory, the data in the I/O register is now placed on the MEM bus and control signals are generated to write it into memory. For transfers out of memory, control signals are generated to read data onto the MEM bus, and this data is clocked from the MEM bus into the I/O register, from which it is placed on the DATA lines and carried to the device that made the data channel request. Once a data channel break occurs, all outstanding data channel transfers are completed before CPU CLK is allowed to resume clocking microinstructions into RBUF.

Data channel memory references appear to the memory to be exactly the same as microprogrammed memory references because the data channel logic and the microprogrammed control logic generate the same memory control signals. Furthermore, data channel operation alters the data on the MEM bus, on the I/O bus, and in the I/O register. Therefore, data channel breaks may not be enabled while memory is in use or while an I/O instruction is being executed.

CPU CYCLES

To the microprogrammer, a CPU cycle is one period of CPU CLK, the signal that clocks all registers in the processor. Although the microprogrammer may usually think of the circuitry of the ECLIPSE computer as responding instantly to new signals from the microinstruction register, that is not the way it actually works. Changes in RBUF take significant time to percolate through the microinstruction decode logic, the computational and data transfer logic, the state change logic, and the control store. Consequently, the loading of RBUF is followed by a period of instability among the control of data signals that may last nearly 200ns. Not until the state of the processor has stabilized can data be clocked into registers. For this reason, CPU CLK has a period of 200ns.

Just prior to the rise of CPU CLK, the outputs of all registers in the computational logic display the old contents of those registers. The outputs enter the various logical components (e.g., the ALU), so that the components operate on the old contents of the registers. The operations performed are those specified by the current microinstruction in RBUF, and the results of those operations appear (almost) simultaneously on the output lines of each component. The only components whose outputs do not reflect their inputs are the registers; at this time their inputs are the results of the operations performed by the current microinstruction on the old contents of the registers.

When CPU CLK rises, the registers that are enabled by the current microinstruction in RBUF are clocked. Simultaneously, a new microinstruction is clocked into RBUF, which is really just another register. As the new outputs of the various registers (the new control and data signals) reach the other operational components of the processor, a new processor state will begin to emerge. When the new processor state stabilizes, new data will be waiting to be clocked into the registers and a new microinstruction will be waiting to enter RBUF. On every CPU cycle, data "flows" from the registers and is transformed by the other operational components so that new results back up behind the registers, waiting for CPU CLK.

MEMORY

The ECLIPSE computer's memory is divided into modules of 8192 16-bit words. Each module generates its own timing signals and operates asynchronously relative to other modules and to the CPU. Therefore, the operation of several modules may overlap, and the CPU may control memory with a few simple commands.

Memory operation is divided into two steps called half-cycles. When a memory module is started, the first or access half-cycle is initiated. During this half-cycle, data is read from memory into a buffer in the module. When the module is released, the second, or restore half-cycle is initiated. During this half-cycle, data may be written from the buffer into memory. The time between half-cycles is called mid-cycle; a memory module may remain at mid-cycle for an indefinitely long time.

In order to start memory, the CPU places an address on the LA bus and issues a start signal (MC1) to memory. All modules see the start signal, but only the module that contains the mem-

ory location whose address is on the LA bus actually responds to the start signal. If no module recognizes the address on the LA bus, none is started. If the selected module is busy with a prior memory operation, it sends back a "wait" signal (MSIN) to the CPU which "freezes" CPU CLK. When the module finishes its prior operation, it releases the wait signal, which allows CPU CLK to continue. The module takes the new address, and the CPU resumes executing microinstructions. Once a module is started, it proceeds to the end of the access half-cycle. When the half-cycle is complete, the buffer in the memory module contains the contents of the addressed memory location.

In order to read a memory location without releasing the module, the CPU asserts the memory control signal MC4. All modules see this signal, but only a module that has begun an access half-cycle or has reached mid-cycle will respond to it. If a module is still in the access half-cycle, it will freeze the CPU CLK until it reaches mid-cycle. When a module has reached mid-cycle, it places the contents of its data buffer on the MEM bus.

A module that is read but not released may later be written and released without being restarted. A "read without release" is used to read the contents of a memory location that will be modified.

In order to read a memory location and release the memory module, the CPU asserts the memory control signals MC3 and MC4 together. Data is read from memory as described above and then the module will begin the restore half-cycle. In core memory, data in the memory location is destroyed when it is read, so the data in a core memory module's buffer is written back into the module during this half-cycle. In semiconductor memory, data is not destroyed when it is read, so no operation is performed during the restore half-cycle.

In order to write into a memory location and release the module, the CPU asserts the memory control signal MC3 alone. If a module has been started, it loads its buffer with the data on the MEM bus. If the module has not already completed the access half-cycle, it does so, and then it begins the restore half-cycle immediately. The data that is "restored" is really the new data loaded into the module's buffer from the MEM bus.

Although different memory modules may not perform memory operations in exactly the same way, the differences are not visible to the microprogrammer. Core memories, semiconductor memories, memories with ERCC and those without it all appear the same to the CPU. If a memory module is not ready to perform an operation when the CPU requests it, CPU operation is suspended by the freezing of CPU CLK until the memory catches up. If a memory module reaches mid-cycle before the CPU is ready for it, the module waits. Therefore, the microprogrammer need not write his microroutines for one type of memory. A microroutine that will reference one kind of memory will reference all kinds.

CONSOLE

The console controls and monitors the operation of the CPU. The console lights display information that reflects the state of the CPU; the sixteen console data switches are used to enter 15-bit addresses and 16-bit data words to the CPU; the address compare logic can suspend CPU operation if a selected memory location is referenced; and the ten console function switches may initiate twenty console functions.

There are five flip-flops in the console. Nineteen of the twenty console functions control CPU operation by setting one or more of these flip-flops. Only the RESET function does not set any of them.

In the following discussion, it is assumed that the reader is already familiar with the console and has read the description of the console in the "Programmer's Reference Manual for the ECLIPSE Computer" DGC 015-000024. This discussion will concentrate on those features of the console that concern the microprogrammer.

Console Lights

The ION and CARRY lights on the console display the states of the ION and CARRY bits in the CPU.

The ADDR COMPARE light is lit when the address compare feature of the console has frozen CPU CLK.

The ROM ADDRESS lights display the output of the state change logic, which is the address of the next microinstruction to be read from the control store.

The ADDRESS lights display the address on the LA bus. Except during data channel memory references, the LA bus carries the fifteen low-order bits of the ALU output (ALU<1-15>) whether or not the CPU is actually referencing memory. During data channel memory references, the LA bus carries an address to memory from the I/O register.

The DATA lights display the contents of the console data register. Except when the address compare feature is in monitor mode, this register is loaded from the MEM bus every time CPU CLK rises. In monitor mode, the register is clocked only when the address compare logic detects an address match (see below). Since the console data

register must be clocked, data is not displayed in the data lights until the CPU CLK period after it is placed on the MEM bus. By the time it is displayed, a new microinstruction has entered RBUF and new data may be on the MEM bus.

Data Switches

The data switches may be selected as the source of the MEM bus by a microinstruction in RBUF. When data from the switches is on the MEM bus, it may be manipulated as if it is being read from memory.

Address Compare Logic

The console's address compare logic is activated by the ADDRESS COMPARE rotary switch on the console. It may operate in one of three modes: MONITOR mode, STOP/STORE mode, and STOP/ADDR mode.

In MONITOR mode, the console data register is loaded whenever the memory location whose address is set in the console data switches is read or written. Therefore, the console data lights always display the contents of that memory location after it has been referenced once by the CPU. If the EXAMINE switch is pressed while the address compare logic is in MONITOR mode and CPU operation has not been suspended, the REXAM flip-flop is set, and a special microroutine references the memory location whose address is set in the console data switches at the end of the next instruction. This memory reference causes the console data register to be loaded with the contents of that memory location so that they will be displayed in the console data lights.

In STOP/STORE mode, CPU operation is suspended if the memory location whose address is set in the data switches is written. When CPU CLK is frozen, the microinstruction in RBUF is the one that has placed data on the MEM bus and issued a write command to memory.

In STOP/ADDR mode, CPU CLK and REG CLK are frozen (so that CPU operation is suspended) if the memory location whose address is set in the console data switches is referenced. When CPU CLK is frozen, the microinstruction in RBUF is the one that has placed the memory address on the LA bus and started memory.

NOTE When the address compare logic is in STOP/ADDR mode, it will freeze the CPU if the data channel logic references the memory location whose address is set in the console data switches. Similarly, in the STOP/STORE mode, it will freeze the CPU if the data channel logic tries to write into the memory location selected by the switches. In either case, there will be no indication at the console that the data channel is responsible for the memory reference.

Console Flip-flops

There are five flip-flops in the console that are set by console switches: the CONRQ flip-flop, the STOP flip-flop, the REXAM flip-flop, the MICROINST ENAB flip-flop, and the STEP SYNC flip-flop.

CONRQ Flip flop

When any console function switch besides the RESET/STOP switch is pressed, the CONRQ flip-flop in the console is set for the duration of the next period of CPU CLK. This flip-flop may be tested by the state change logic to see if there is a console function to be performed. If there is, a 4-bit console function code may be read from the console via the MEM bus. If a console function switch is pressed when the CPU is halted, the ECLIPSE computer's standard halt/stop firmware sees that the CONRQ flip-flop is set and transfers control to the appropriate microroutine. The CONRQ flip-flop is cleared by the next CPU CLK after it is set regardless of whether the CPU has tested it.

STOP Flip-flop

The STOP flip-flop is set if the STOP, EXECUTE, or INSTRUCTION STEP switch is being pressed when an instruction is loaded into the IR. The next time CPU CLK rises, it causes the STOP ENAB flip-flop in the CPU to be set. The STOP ENAB flip-flop causes the CPU to halt the next time an instruction is to be loaded into the IR. Therefore, if the STOP, EXECUTE, or INSTRUCTION STEP switch is pressed, during one instruction, it will cause a halt at the end of the next instruction. This one-instruction delay is necessary so that the execute and instruction step console functions will allow one instruction to be executed.

REXAM Flip-flop

The REXAM flip-flop is set by the EXAMINE switch when CPU CLK rises and the CONRQ flip-flop is already set. Since the CONRQ flip-flop is only set for one period of CPU CLK, it is cleared at the same time that the REXAM flip-flop is set. The REXAM flip-flop is cleared when a console function code is read onto the MEM bus by a microinstruction in RBUF.

The REXAM flip-flop causes control to be transferred to a special microroutine included in the ECLIPSE computer's standard firmware the next time an instruction is to be loaded into the IR. This microroutine, called the running-examine microroutine, reads the memory location whose address is set in the console data switches. It is intended to be used in conjunction with the address compare logic. If the address compare logic is in MONITOR mode, the memory reference performed by the running-examine microroutine causes the contents of the referenced memory location to be clocked into the console data register and displayed in the console data lights.

MICROINST ENAB and STEP SYNC Flip-flops

When the MICROINST ENAB flip-flop is set, CPU CLK and REG CLK are frozen so that CPU operation is suspended. It is set by the MICROINSTRUCTION STEP switch when CPU CLK rises and the CONRQ flip-flop is already set (i. e., at the same time that CONRQ flip-flop is cleared) or when CPU CLK rises and the CPU is already frozen by the MICROINST ENAB flip-flop or by the address compare logic. It is cleared when CPU CLK rises and the MICROINSTRUCTION STEP switch is not being pressed.

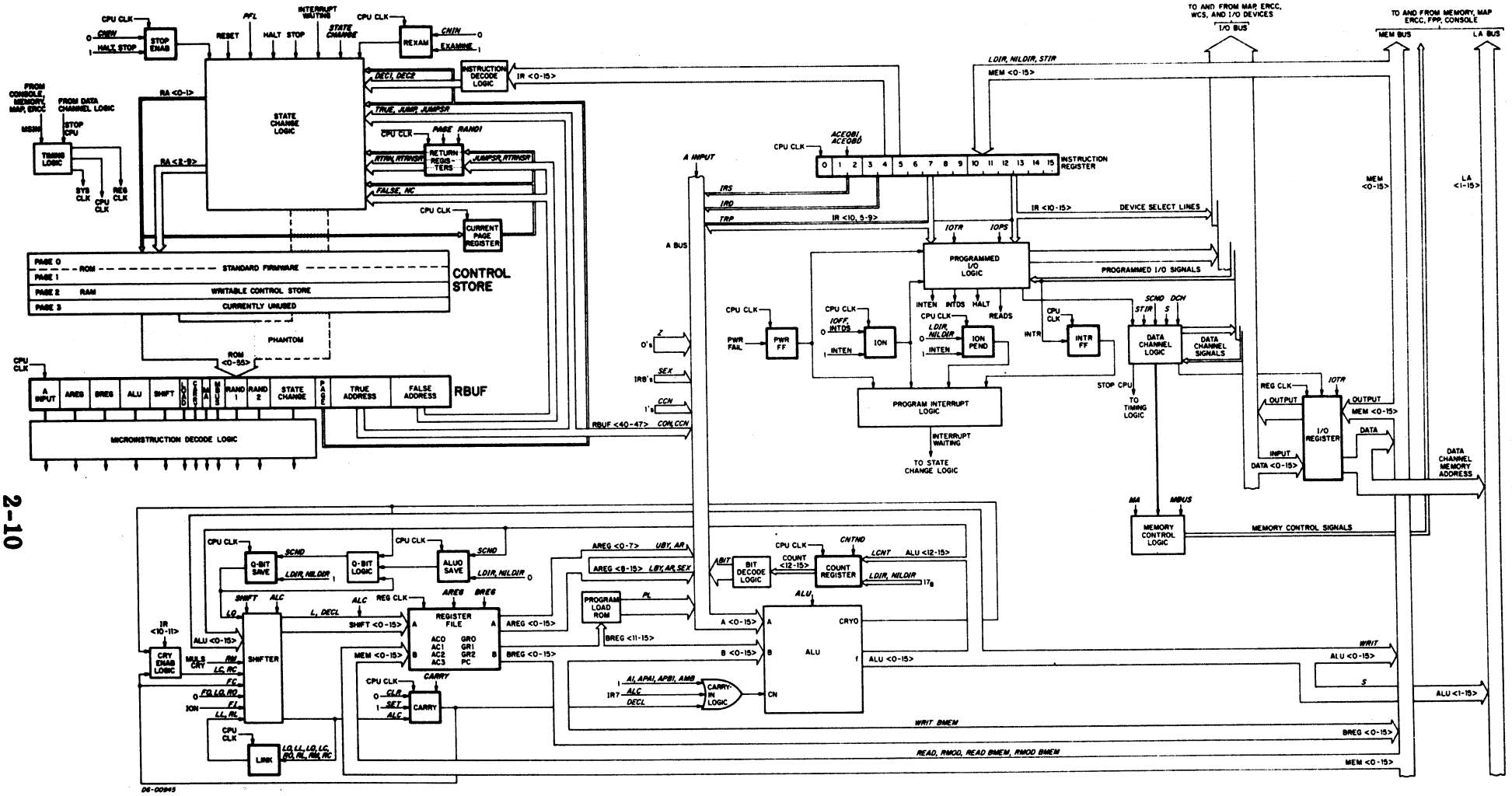
The STEP SYNC flip-flop overrides the MICROINST ENAB flip-flop to allow CPU CLK to rise once. It is set by the MICROINSTRUCTION STEP, INSTRUCTION STEP and CONTINUE switches. It is cleared when CPU CLK actually rises. If STEP SYNC is set by the INSTRUCTION STEP or CONTINUE switch, the MICROINST ENAB flip-flop will be cleared when CPU CLK rises and CPU CLK will remain unfrozen. Every time the MICROINSTRUCTION STEP switch is pressed, STEP SYNC is set, CPU CLK is allowed to rise once, and a new microinstruction is clocked into RBUF. However, the MICROINST ENAB flip-flop is not cleared, and CPU CLK will not rise again until the next time the MICROINSTRUCTION STEP, INSTRUCTION STEP, or CONTINUE switch is pressed.

Whenever CPU operation is not already suspended (whether or not the CPU is halted), CPU CLK and REG CLK may be frozen by pressing the MICROINSTRUCTION step switch. Once CPU operation has been suspended either by the MICROINSTRUCTION STEP switch or by the address compare logic, the MICROINSTRUCTION STEP switch may be pressed to single-step, or the INSTRUCTION STEP or CONTINUE switch may be pressed so that CPU operation will resume. The INSTRUCTION STEP switch will also set the STOP flip-flop (see above).

Reset

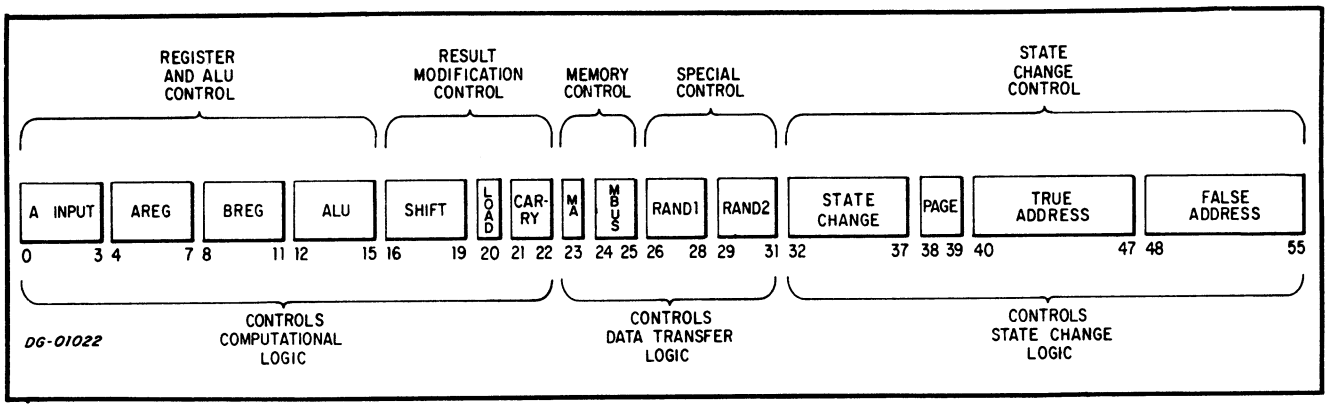
Reset is the only console function that does not set any of the console's flip-flops.

Reset causes the signals SYS RST and IORST to be asserted throughout the ECLIPSE computer system until the RESET switch is released. While these signals are asserted, the microinstruction in location 0 of page 0 of the control store is repeatedly loaded into RBUF. When the RESET switch is released, the CPU behaves as if it had just been powered up. The RESET switch is important to the microprogrammer because it is the only switch which will extricate the microprogrammed control logic from an infinite loop in a microroutine.


CPU CLK
STOP ENAB
CNIN
HALT, STOP
RESET
HALT STOP
PFL
INTERRUPT WAITING
STATE CHANGE
REXAM
EXAMINE
FROM CONSOLE, MEMORY, MAP, ERCC
FROM DATA CHANNEL LOGIC
MSYN
STOP CPU
TIMING LOGIC
SYS CLK
REG CLK
STATE CHANGE LOGIC
RA <0-1>
RA <2-9>
DECI, DEC2
TRUE, JUMP, JUMPSR
FALSE, MC
PAGE, RAND1
RTRN, RTRNSKP
JUMPSR, RTRNSR
RETURN REGISTERS
INSTRUCTION DECODE LOGIC
IR <0-15>
CURRENT PAGE REGISTER
PAGE 0
PAGE 1
PAGE 2
PAGE 3
ROM
RAM
STANDARD FIRMWARE
WRITABLE CONTROL STORE
CURRENTLY UNUSED
CONTROL STORE
PHANTOM
ROM <0-55>
A INPUT
AREG
BREG
ALU
SHIFT
STATE CHANGE
RAND 1
RAND 2
TRUE ADDRESS
FALSE ADDRESS
RBUF
MICROINSTRUCTION DECODE LOGIC
RBUF <40-47>
CON, CCN
A BUS
0's
1's
IR8's
SEX
CCN
TO AND FROM MAP, ERCC, WCS, AND I/O DEVICES
I/O BUS
TO AND FROM MEMORY, MAP, ERCC, FPP, CONSOLE
MEM BUS
LA BUS
LDIR, NILDIR, STIR
MEM <0-15>
ACEOBI, ACEOBD
INSTRUCTION REGISTER
IRS
IRD
TRP
IR <10, 5-9>
IR <10-15>
DEVICE SELECT LINES
IOTR
IOPS
PROGRAMMED I/O LOGIC
PROGRAMMED I/O SIGNALS
INTEN
INTDS
HALT
READS
PWR FAIL
PWR FF
IOFF, INTDS
ION
LDIR, NILDIR
ION PEND
INTR
INTR FF
SCND
DCH
STIR
S
DATA CHANNEL LOGIC
DATA CHANNEL SIGNALS
PROGRAM INTERRUPT LOGIC
INTERRUPT WAITING
TO STATE CHANGE LOGIC
STOP CPU
TO TIMING LOGIC
REG CLK
OUTPUT
I/O REGISTER
DATA
INPUT
DATA <0-15>
DATA CHANNEL MEMORY ADDRESS
MA
MBUS
MEMORY CONTROL LOGIC
MEMORY CONTROL SIGNALS
MEM <0-15>
LA <1-15>
Q-BIT SAVE
Q-BIT LOGIC
ALUQ SAVE
AREG <0-7>
UBY, AR
AREG <8-15>
LBY, AR, SEX
CNTND
BIT DECODE LOGIC
COUNT <12-15>
COUNT REGISTER
LCNT
ALU <12-15>
SHIFT
ALC
LQ
L, DECL
SHIFTER
ALU <0-15>
SHIFT <0-15>
REG CLK
AREG
BREG
REGISTER FILE
ACO
AC1
AC2
AC3
GRO
GR1
GR2
PC
AREG <0-15>
BREG <0-15>
PROGRAM LOAD ROM
PL
BREG <11-15>
A <0-15>
B <0-15>
ALU
CRYO
CN
IR <10-11>
CRY ENAB LOGIC
MULS
DIV
RM
LC, RC
FC
FQ, LQ, RQ
FI
LL, RL
MEM <0-15>
CARRY
CLR
SET
ALC
CARRY-IN LOGIC
AI, APAI, APBI, AMB
IR7
DECL
WRIT
WRIT BMEM
READ, RMOD, READ BMEM, RMOD BMEM
LINK
LQ, LI, LQ, LC, RQ, RI, RM, RC
DG-00945

SECTION 3

MICROINSTRUCTION FORMAT AND MICRO-ORDER SET



In the ECLIPSE computer, a 56-bit microinstruction is divided into fifteen fields. These fields can be grouped according to the purposes they serve (as shown above). The purposes of the fields are as follows:

Register and ALU control - These four fields together determine the output of the ALU by selecting a pair of registers, the ALU input data, and the ALU function to be performed. The ALU output so determined is used by the micro-orders in the four other groups of fields.

- A INPUT field: selects data to be placed on the A bus and sent to the A input of the ALU.
- AREG field: selects a register in the register file for possible input to the A bus and/or loading from the shifter.
- BREG field: selects a register in the register file for input to the ALU through its B input and for possible loading from or writing to the MEM bus.
- ALU field: selects one of 16 functions to be performed on the A and B inputs to the ALU.

Result Modification Control - These three fields control what modification of registers is performed in the computational logic.

- SHIFT field: controls the shifter and the Link bit.
- LOAD field: determines whether the ALU result is loaded into the A register.
- CARRY field: controls the Carry bit and the ALC logic.

Memory Control - These two fields control memory.

- MA field: starts a memory module.
- MBUS field: controls the use of the MEM bus.

Special Control - These two fields perform various unrelated functions around the processor.

- RAND1 field: may specify one of several special control functions or select a microsubroutine return register.
- RAND2 field: may specify one of several special control functions different from those of the RAND1 field.

State Change Control - These four fields control the selection of the next micro-instruction to be loaded into RBUF and executed.

STATE CHANGE field	specifies the state change to be performed.
PAGE field	may supply page bits for the address of the next micro-instruction or select a return register.
TRUE ADDRESS field	specifies either the address to which control is to be transferred when a state change test condition is true, or a constant for use by the micro-instruction, or both.
FALSE ADDRESS field	specifies either the address to which control is to be transferred when a state change test condition is false, or a return address to be loaded into a return register for microsubroutine calls.

Under the heading for each micro-order, its description is preceded by a mnemonic and a value in octal notation. The mnemonic is used to represent the micro-order in the text and microinstruction examples in this manual. The octal value represents the combination of bits that constitutes the micro-order when it is used in the appropriate field of a microinstruction.

A micro-order reference sheet and several block diagrams accompany the text in this section and serve to provide an overview and summary of the information presented here. The reference sheet shows the microinstruction format and lists all micro-orders with brief functional descriptions wherever appropriate. The block diagrams are portions of the single block diagram given in Section 2, with additional detail. Conventions used in these diagrams include those listed in Section 2 and also the following:

1. Data on a path ending with the upper half of an arrowhead fills the high-order byte (most significant 8 bits) of the bus it is entering. Data on a path ending with the lower half of an arrowhead occupies the low-order byte (least significant 8 bits) of the bus.
2. The names of micro-orders are italicized. A label in boldface italics represents collectively the micro-orders that may be coded in the microinstruction field of that name.
3. A label that appears within or adjacent to a control or data path and is not italicized is generally the name of the path as used in this manual and on the engineering prints.
4. A label that appears above a data path is generally the micro-order, signal, or condition which enables that path. Unlabeled paths are always enabled.
5. Certain labels which appear to be signal names have been fabricated in order to simplify the description of the ECLIPSE computer in this manual and do not actually appear on the engineering prints. They are: AREG<0-15>, BREG<0-15>, EXAMINE, INTERRUPT WAITING, and MULS CRY.

The micro-order reference sheet and a complete, fully detailed block diagram can also be found at the back of this manual.

Examples of microinstructions have been included where they clarify the explanation in text. In these examples, there are three conventions:

1. Microinstruction labels appear on the left and are separated from the labeled microinstruction by a colon; the value of a label is the address of the location in the control store occupied by the microinstruction.
2. Micro-orders are represented by the mnemonics given in this section.
3. All fifteen fields of a microinstruction are shown; the contents of fields that are not of interest (i. e., unused) are represented by dashes.

The remainder of this section discusses the various fields in turn and explains the operation of all micro-orders that may be coded in each field.

A INPUT FIELD

The A INPUT field controls the A bus, A<0-15>, and thereby determines what data is connected to the A input of the ALU. The data selected may be the A register or some variant thereof, or it may come from any of several other sources, including the IR, the Program Load ROM, the Count register, and the microinstruction itself.

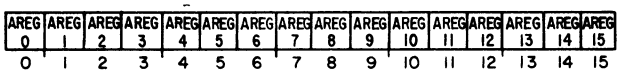
The micro-orders that may be coded in the A INPUT field are described below. The format of the data placed on the A bus by a given micro-order is pictured below the description of the micro-order.

A Register

AR

0

The contents of the register selected by the AREG field are placed on the A bus.

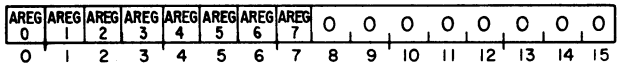


Upper Byte

UBY

17

Bits 0-7 of the register selected by the AREG field are placed on bits 0-7 of the A bus; 0's are placed on bits 8-15 of the bus.



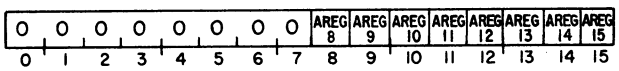
NOTE The UBY micro-order does not move the selected byte to the lower half of the A bus.

Lower Byte

LBY

10

Bits 8-15 of the register selected by the AREG field are placed on bits 8-15 of the A bus; 0's are placed on bits 0-7 of the bus.



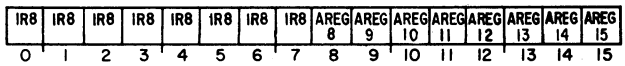
The UBY and LBY micro-orders are used to isolate the upper and lower bytes of a word in a register, most commonly for byte-oriented operations.

Sign Extend

SEX

14

Bits 8-15 of the register selected by the AREG field are placed on bits 8-15 of the A bus. A copy of IR bit 8 is placed on bits 0-7 of the bus.



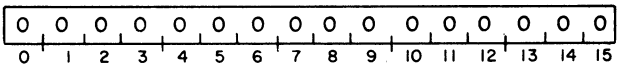
The SEX micro-order is used in "short" effective address calculations to produce a 16-bit displacement from an 8-bit displacement contained in both the IR and a general register.

Zeroes

Z

12

All 0's are placed on the A bus.

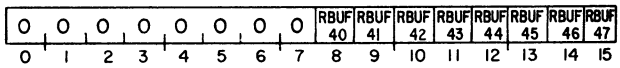


Constant

CON

13

The contents of the TRUE ADDRESS field, bits 40-47 of the microinstruction itself, are placed on bits 8-15 of the A bus; 0's are placed on bits 0-7 of the bus.



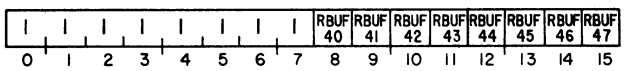
The CON micro-order enables the microprogram to generate its own constants in the range 0 to 377_8 (0 to 255_{10}).

Complement Constant

CCN

16

The contents of the TRUE ADDRESS field, bits 40-47 of the microinstruction itself, are placed on bits 8-15 of the A bus; 1's are placed on bits 0-7 of the bus.



The CCN micro-order enables the microprogram to generate its own constants in the range 177400_8 to 177777_8 (signed -256_{10} to -1).

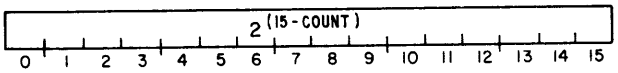
Single Bit

BIT

2

A single "1" bit is placed on the A bus in the bit position designated by the current contents of the Count register. 0's are placed in the 15 remaining positions.

For the purpose of the BIT micro-order, the contents of the Count register specify the number of the bit position of the A bus which is to receive the single "1" bit. Bit positions are labeled in ascending order from left to right, where bit 0 is the most significant bit and bit 15 is the least significant bit.



As an unsigned number, the data placed on the A bus is equal to $2^{(15-COUNT)}$, where COUNT is the current value contained in the Count register. Hence, if the Count register contains 0, the BIT micro-order will place 100000_8 on the bus. If the Count register contains 15_{10}, 1 will be placed on the bus.

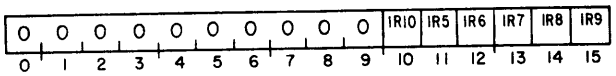
The BIT micro-order can be used to perform bit operations. If the number of the bit position to be operated on is loaded into the Count register (see the LCNT micro-order in the RAND2 field), the BIT micro-order will generate a mask for that bit.

Trap Number

TRP

3

IR bit 10 is placed on bit 10 of the A bus. IR bits 5-9 are placed on bits 11-15 of the bus. 0's are placed on bits 0-9 of the bus.



NOTE If the value of the Count Register is less than or equal to 7, the TRP micro-order will not place all 0's on the left byte of the A bus. Rather, a single "1" bit will be placed in the bit position specified by the value of the Count register (as for the BIT micro-order, above).

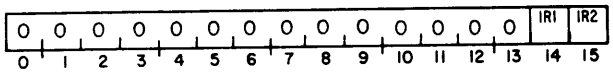
The TRP micro-order is used to access the operation number in an XOP instruction. (See the Programmer's Reference Manual for the ECLIPSE computer for a description of the XOP instruction.)

Instruction Register Source

IRS

11

IR bits 1-2 are placed on bits 14-15 of the A bus; 0's are placed on bits 0-13 of the bus.

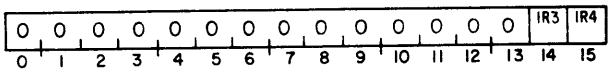


Instruction Register Destination

IRD

1

IR bits 3-4 are placed on bits 14-15 of the A bus; 0's are placed on bits 0-13 of the bus.



The IRS and IRD micro-orders are used to access the contents of the ACS and ACD fields in the current instruction.

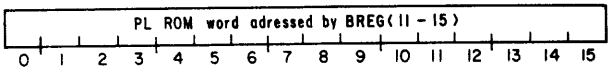
NOTE Do not confuse these micro-orders with the micro-orders ACS and ACD (which are coded in the AREG and BREG fields). IRS and IRD access the contents of the ACS and ACD fields of the IR, not the contents of the accumulators specified by those fields.

Program Load

PL

5

The word in the Program Load ROM addressed by bits 11-15 of the register selected by the BREG field is placed on bits 0-15 of the A bus.



The PL micro-order is used to implement the automatic program load feature of the ECLIPSE computer. When the program load switch on the console is pressed, the 32-word program stored in the Program Load ROM is read into memory locations $0\text{-}37_8$ by addressing and reading successive words of the ROM using the PL micro-order.

AREG AND BREG FIELDS

The AREG and BREG fields select a pair of registers in the register file. The AREG field selects one of the eight registers to be connected to the A input and A output ports of the register file. The BREG field selects one of the eight registers to be connected to the B input and B output ports of the register file. (In the hardware, the B input port is also known as the C port.) The AREG and BREG fields may select the same register or different registers. Selection can be direct, in which case one of four accumulators or one of four general registers is specified, or it can be indirect, in which case register selection is delegated to the IR. The micro-orders for both fields are identical and are listed only once below. However, the purposes served by the two fields are somewhat different.

The AREG field serves two purposes: (1) it selects a register for input, in some form, to the A bus when the A INPUT field contains AR, UBY, LBY, or SEX; and (2) it selects a register to be loaded with the output of the shifter. Loading of the register specified by the AREG field is governed normally by the LOAD field but sometimes by the CARRY or RAND2 field. The full 16-bit output of the shifter is loaded into the A register if the L micro-order is coded in the LOAD field, unless the CARRY field contains the ALC micro-order, in which case loading is controlled instead by IR bit 12 (see the description of the ALC micro-order in the description of the CARRY field). Coding of the DECL micro-order in the RAND2 field allows for loading only the low-order nibble (bits 12-15) of the shifter output into the A register when no loading would otherwise occur. Note that all or part of the shifted result can be loaded into the register specified by the AREG field whether or not that register is used for input to the ALU via the A bus.

The BREG field also serves two purposes, but they differ from those of the AREG field. They are: (1) to select the register which is to be connected to the B input of the ALU; and (2) to select a register to be connected to the memory bus when the RAND2 field contains the BMEM micro-order. This second use of the BREG field is explained in the descriptions of memory control and the BMEM micro-order.

The selection of the A and B registers is governed by the micro-orders described below.

Accumulator 0

AC0

10

Select accumulator 0.

Accumulator 1

AC1

11

Select accumulator 1.

Accumulator 2

AC2

12

Select accumulator 2.

Accumulator 3

AC3

13

Select accumulator 3.

General Register 0

GR0

14

Select general register 0.

General Register 1

GR1

15

Select general register 1.

General Register 2

GR2

16

Select general register 2.

Program Counter

PC

17

Select the program counter (general register 3).

Source Accumulator

ACS

0

Select the accumulator whose number is in IR bits 1-2.

Destination Accumulator

ACD

2

Select the accumulator whose number is in IR bits 3-4.

Destination Accumulator Plus 1

AD1

1

Select the accumulator whose number is one greater than the number in IR bits 3-4. (If IR bits 3-4 are 11, then AC0 is selected.) The IR is not altered.

The AD1 micro-order can be used to implement operations on 32-bit quantities contained in two consecutive accumulators, the first of which is addressed by the ACD field in the IR. See, for example, the DOUBLE LOGICAL SHIFT instruction (DLSH).

Source General Register

GRS

4

Select the general register whose number in in IR bits 1-2.

Destination General Register

GRD

6

Select the general register whose number is in IR bits 3-4.

Destination General Register Plus 1

GD1

5

Select the general register whose number is one greater than the number in IR bits 3-4. (If IR bits 3-4 are 11, then GR0 is selected.) The IR is not altered.

ALU FIELD

The ALU field, RBUF<12-15>, provides for the selection of one of sixteen different ALU functions. Nine of these are arithmetic functions (micro-order codes 0 through 10_8), and seven are logical functions (codes 11_8 through 17_8). The inputs to the ALU are the A bus, the B register, and a carry-in called CN. The outputs are a 16-bit function result and a carry-out of bit 0 called CRY0.

Inputs

As described under the A INPUT field, the A bus may or may not reflect the contents of the A register. In either case, however, the data on the A bus is connected to the A input of the ALU and always takes part in the ALU function performed. The B register, on the other hand, is always connected to the B input of the ALU, but not all of the ALU field micro-orders use the B register data supplied in performing their designated functions. The carry-in CN is a single bit which is generated (that is, set to 1) in several different situations. For arithmetic micro-orders only, CN is added to the result obtained from performing the designated function on the A and B input data.

A carry-in can be generated (CN can be set to 1) in three different ways:

1. Certain arithmetic micro-orders require a carry-in to perform their designated functions. These are the micro-orders A1, APA1, APB1, and AMB. For these micro-orders a carry-in is generated automatically by the hardware.
2. Certain ALC instructions similarly require a carry-in to perform their designated functions. These are the instructions INC, SUB, and NEG. For this reason the hardware automatically generates a carry-in when the ALC micro-order is coded into the CARRY field and IR bit 7 is 1. (This generates a carry-in for INC, SUB, NEG, and AND. In the case of AND, however, the carry-in does not affect the result because the function performed is a logical one.)
3. In order to perform multiple-precision additional or subtraction, a carry or borrow from the previous operation must be added to or subtracted from the result of the operation. The ECLIPSE computer provides this capability by allowing a Carry bit of 1 to generate a carry-in when the DECL micro-order is coded in the RAND2 field. For multiple-precision addition, DECL is specified, and a carry-in is generated if the Carry bit is 1. For multiple-precision subtraction, the Carry bit serves as a complemented "borrow" bit, where a Carry bit of 0 signifies a borrow from the previous subtraction, and a Carry bit of 1 signifies the "no-borrow" condition. In this case, DECL is specified, the subtraction is performed by adding the one's complement of the minuend, and the carry-in is added, thereby adding 1 only if there were no borrow from the last operation. The DECL micro-order is used to signal a multiple-precision operation because, in the standard ECLIPSE computer instruction set, only decimal arithmetic—and not 16-bit binary arithmetic—is performed on a multiple-precision basis. However, multiple-precision binary arithmetic can still be performed by the user if desired, either in microcode or at the assembly language level (see Appendix D of the Programmer's Reference Manual to the ECLIPSE Computer).

In summary, then, CN is either 0 or 1 for each microinstruction and is computed as follows:

CN = (A1 OR APA1 OR APB1 OR AMB)

OR

(ALC AND IR7)

OR

(DECL AND CARRY)

Outputs

As mentioned before, the outputs from the ALU are a 16-bit function result and a carry-out called CRY0. For arithmetic micro-orders, the result is the designated arithmetic function of the A and B inputs, plus CN. (In other words, a carry-in causes 1 to be added to the result of the operation performed on the A and B inputs.) For logical micro-orders, the result is merely the designated logical function of the A and B inputs, performed on a bit-by-bit basis; the carry-in CN has no effect at all on the result.

The other ALU output, CRY0, is a single bit which indicates when a carry-out of ALU bit 0 occurs. CRY0 may be tested by the CRY0B and SCRY micro-orders, which are coded in the STATE CHANGE field. Also, it is used in the production of CRY ENAB, a new Carry bit value used in shifting. (See the SHIFT field description for details.)

CRY0 is computed for all ALU functions, both arithmetic and logical. For arithmetic functions it is a normal carry-out determined from the data input and the function performed. For logical functions it is a normal carry-out determined from the data input and the arithmetic function corresponding to the logical function performed. The correspondences between logical and arithmetic functions depend on the structure of the ALU and are as follows:

Micro-order Code	Logical Function Micro-order	Corresponding Arithmetic Function
11	CA	AM1
12	AOB	AOB plus CN
13	AXB	APB
14	ANB	ANCB plus A plus CN
15	ANCB	ANB plus A plus CN
16	CANB	AOB plus A plus CN
17	ANBC	ANB plus CN minus 1

Thus, it is conceivable, though unlikely, that CRY0 may be meaningfully tested with a CRY0B or SCRY micro-order in a microinstruction which performs a logical function. However, CRY ENAB is never affected by CRY0 when a logical function is being performed.

The individual micro-orders available in the ALU field are described below. To simplify the description of these micro-orders, the term C_{in} is defined as follows:

$$C_{in} = (\text{ALC AND IR7}) \text{ OR } (\text{DECL AND CARRY})$$

C_{in} is like CN except that it does not include the forced carry-in of 1 which is implicit in some ALU micro-orders and over which the microprogrammer has no control.

Arithmetic Micro-orders

A Input

A

0

The ALU output is equal to the A input plus C_{in}. CRY0 is 1 only if A = 2^{16}-1 and C_{in} = 1.

A Plus 1

A1

2

The ALU output is equal to the A input plus 1. CRY0 is 1 only if A = 2^{16}-1.

A Plus A

APA

5

The ALU output is equal to C_{in} plus twice the A input. CRY0 is 1 only if A $\geq 2^{15}$.

A Plus A Plus 1

APA1

6

The ALU output is equal to 1 plus twice the A input. CRY0 is 1 only if A $\geq 2^{15}$.

A Plus B

APB

1

The ALU output is equal to the A input plus the B input plus C_{in}.
CRY0 is 1 only if A+B+$C_{in} \geq 2^{16}$.

A Plus B Plus 1

APB1

7

The ALU output is equal to the A input plus the B input plus 1. CRY0 is 1 only if A+B $\geq 2^{16}$-1.

A Minus 1

AM1

4

The ALU output is equal to the A input plus C_{in} minus 1. CRY0 is 1 if A $\neq$ 0 or if C_{in} = 1. (CRY0 is 0 only when A = 0 and C_{in} = 0.)

A Plus Complement of B

APCB

3

The ALU output is equal to the A input plus the one's complement of the B input plus C_{in}. CRY0 is 1 only if A+C_{in} > B.

A Minus B

AMB

10

The ALU output is equal to the A input minus the B input (that is, the A input plus the two's complement of the B input). CRY0 is 1 only if A $\geq$ B.

Logical Micro-orders

Complement of A

CA

11

The ALU output is equal to the one's complement of the A input. CRY0 is computed by the ALU as if the function being performed were AM1.

A OR B

AOB

12

The ALU output is equal to the logical inclusive OR (OR) of the A and B inputs. CRY0 is computed by the ALU as if the function being performed were AOB plus C_{in}.

A XOR B

AXB

13

The ALU output is equal to the logical exclusive OR (XOR) or the A and B inputs. CRY0 is computed by the ALU as if the function being performed were APB.

A AND

ANB

14

The ALU output is equal to the logical AND (AND) of the A and B inputs. CRY0 is computed by the ALU as if the function being performed were ANCB plus A plus C_{in}.

A AND Complement of B

ANCB

15

The ALU output is equal to the logical AND of the A input and the one's complement of the B input. CRY0 is computed by the ALU as if the function being performed were ANB plus A plus C_{in}.

Compliment of A AND B

CANB

16

The ALU output is equal to the logical AND of the B input and the one's complement of the A input. CRY0 is computed by the ALU as if the function being performed were AOB plus A plus C_{in}.

A AND B Complemented

ANBC

17

The ALU output is equal to the one's complement of the logical AND of the A and B inputs. CRY0 is computed by the ALU as if the function being performed were ANB plus C_{in} minus 1.

SHIFT FIELD

The SHIFT field, RBUF<16-19>, provides for the selection of one of 13 shifting operations to be performed on the ALU function result before it is passed to the A input port of the register file for possible loading into the A register. The two more significant bits of the SHIFT field, RBUF16 and RBUF17, select one of four classes of shift operations: no shift (or "straight" shift), left shift one bit, right shift one bit, and swap bytes. For each of the first three classes the two less significant bits, RBUF18 and RBUF19, select one of four specific shift operations: for straight and right shifting the microprogrammer can specify either the value 0 or the value of one of three other sources for the result bit 0, and for left shifting either 0 or one of three other sources can be chosen for the result bit 15. The various sources which supply these special values are the Carry bit, the Link bit, the ION flag, the Q bit, MULS CRY, CRY ENAB, and ALU0 itself.

One of these special sources, the Link bit, merits additional explanation here. The primary purpose of the Link is to save the bit that is shifted out and would otherwise be lost in a left or right shift operation. On a left shift, the Link is set to the value of ALU output bit 0. On a right shift, the Link is set to the value of ALU output bit 15. Note that the shifted result need not be loaded into the A register for the Link to be modified; left and right shifts always potentially alter the Link.

Once a bit has been shifted out into the Link, it may be shifted back into the same or a different bit position in a second shift operation. This allows double-word shift operations to be accomplished conveniently. The microprogram also has the capability of testing the Link as it is, with no need for shifting, by coding the LINK micro-order in the STATE CHANGE field. This gives the microprogrammer a convenient means for saving a single bit of information and branching according to that bit in a later microinstruction.

When the ALC micro-order is coded in the CARRY field, the two high-order SHIFT field bits, RBUF16 and RBUF17, are ignored and IR bits 8 and 9 instead select the class of shift operation. However, RBUF18 and RBUF19 still choose the special value to be shifted into bit 0 or 15. Therefore, to insure that two-accumulator multiple-operation instructions operate properly, the SHIFT field should be coded as FA, LC, or RC for two-accumulator multiple-operation instructions.

In two of the SHIFT field micro-orders, LC and RC, the value CRY ENAB is involved. CRY ENAB is a new Carry bit value generated for use in two main cases: in two-accumulator multiple-operation instructions, and in the UNSIGNED MULTIPLY instruction (MUL). Its value depends on the old value of the Carry bit, IR bits 10 and 11 (which are the Carry field in two-accumulator multiple-operation instructions), and any arithmetic carry-out of the ALU (that is, CRY0, provided that the ALU function selected is arithmetic).

For two-accumulator multiple-operation instructions CRY ENAB is computed as follows: IR bits 10 and 11 are examined and a base value for CRY ENAB is calculated according to the table below.

IR<10-11>	base value for CRY ENAB
00	current Carry bit (no change)
01	0
10	1
11	complement of Carry bit

For the AND instruction, since the ALU function is logical, the computed base value becomes CRY ENAB directly. For the other two-accumulator multiple-operation instructions, whose operations are all performed by arithmetic functions, the computed base value is exclusive-OR'ed with CRY0 before becoming CRY ENAB. In this way, any carry-out of ALU bit 0 in these two-accumulator multiple-operation instructions complements the base value for CRY ENAB to produce CRY ENAB. The resulting CRY ENAB is then used in all two-accumulator multiple-operation instructions to become the new value for the Carry bit (for swap and no-shift operations), the new value for result bit 0 (for right shift operations), or the new value for result bit 15 (for left shift operations).

For the MUL instruction, and, in fact, for all instructions with bit 0=1, bit 10=0, and bits 12-15=1000, CRY ENAB is equal to CRY0, provided again that an arithmetic ALU function is being performed. (If the ALU function is logical, CRY ENAB always has the value 0 in these cases.)

For instructions with bit 0=1, bit 10=1, and bits 12-15=1000 (notably all XOP1 instructions), CRY ENAB is computed according to the following table:

current Carry bit value	CRY0	CRY ENAB
0	0	1
0	1	0
1	0	0
1	1	1

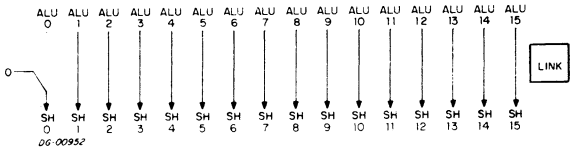
The micro-orders which may be coded in the SHIFT field are described below. Although the shifted result sent to the A input port of the register file is actually called SHIFT<0-15>, "SH<0-15>" is used for brevity in the diagrams accompanying the descriptions of the micro-orders.

Straight with 0

F0

0

Shifter bit 0 is 0. ALU output bits 1-15 are fed straight through the shifter to become shifter bits 1-15. The Link is not changed.



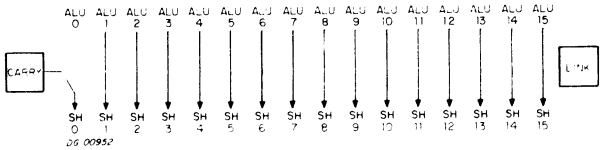
The F0 micro-order is used to force bit 0 of the ALU result to 0 before loading the result into the A register.

Straight with Carry

FC

1

The value of the Carry bit goes to shifter bit 0. ALU output bits 1-15 are fed straight through the shifter to become shifter bits 1-15. The Link is not changed.



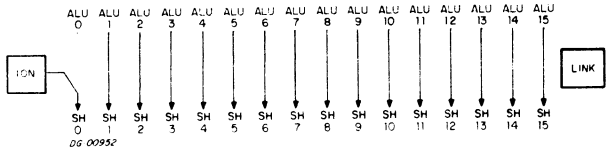
The FC micro-order is used in the standard firmware to combine the Carry bit with PC when saving the state of the machine on the stack.

Straight with ION

FI

2

The value of the ION flag goes to shifter bit 0. ALU output bits 1-15 are fed straight through the shifter to become shifter bits 1-15. The Link is not changed.

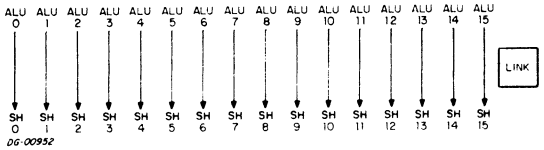


Straight with ALU0

FA

3

All sixteen ALU output bits are fed straight through the shifter to become shifter bits 0-15 The Link is not changed.

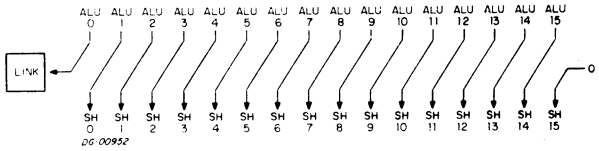


Left with 0

L0

4

The ALU output is shifted left one bit position. The value of the Q bit is brought in to become shifter bit 15. The value of ALU bit 0 is stored in the Link. The previous state of the Link is lost.

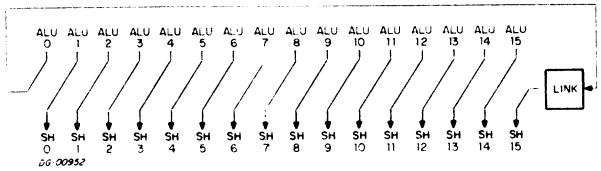


Left with Link

LL

5

The ALU output is shifted left one bit position. The value 0 is brought in to become shifter bit 15. The value of ALU bit 0 is stored in the Link.

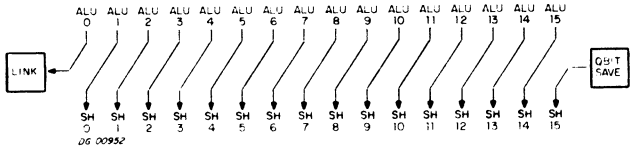


Left with Q Bit

LQ

6

The ALU output is shifted left one bit position. The value of the Q bit is brought in to become shifter bit 15. The value of ALU bit 0 is stored in the Link. The previous state of the Link is lost.

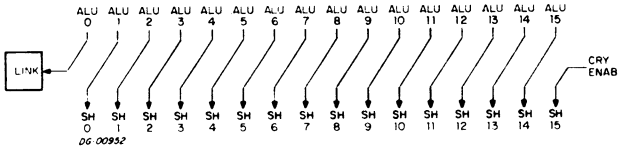


Left with CRY ENAB

LC

7

The ALU output is shifted left one bit position. The value CRY ENAB is brought in to become shifter bit 15. The value of ALU bit 0 is stored in the Link. The previous state of the Link is lost.



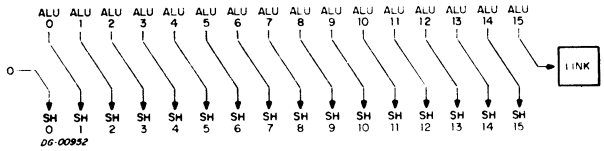
The primary use of the LC micro-order is in two-accumulator multiple-operation instructions which specify a left shift.

Right with 0

R0

10

The ALU output is shifted right one bit position. The value CRY ENAB is brought in to become shifter bit 0. The value of ALU bit 15 is stored in the Link. The previous state of the Link is lost.

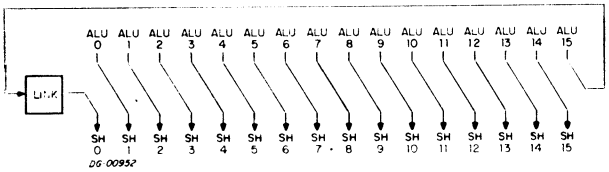


Right with Link

RL

11

The ALU output is shifted right one bit position. The value of the Link is brought in to become shifter bit 0. The value of the ALU bit 15 is stored in the Link.

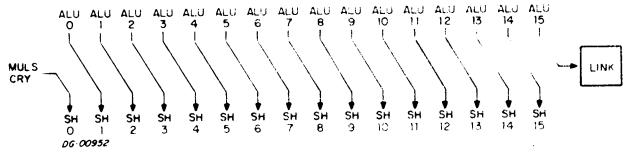


Right with with MULS CRY

RM

12

The ALU output is shifted right one bit position. The value MULS CRY (defined below) is brought in to become shifter bit 0. The value of ALU bit 15 is stored in the Link. The previous state of the Link is lost.



Provided that RBUF36 = 0 and RBUF37 = 1, MULS CRY is defined as follows:

$$\text{MULS CRY} = A0 \text{ XOR } B0 \text{ XOR } CRY0 \text{ XOR } RBUF12$$

When RBUF<36-37> ≠ 01, MULS CRY is meaningless and cannot be used for the purpose intended.

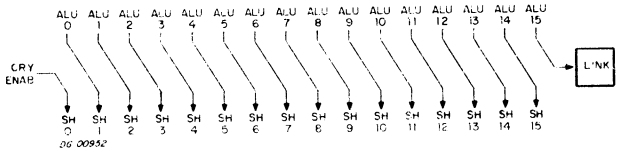
The RM micro-order is used by the standard firmware to implement the SIGNED MULTIPLY instruction (MULS).

Right with CRY ENAB

RC

13

The ALU output is shifted right one bit position. The value CRY ENAB is brought in to become shifter bit 0. The value of ALU bit 15 is stored in the Link. The previous state of the Link is lost.



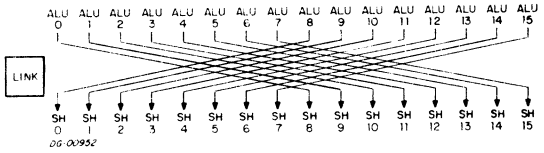
This micro-order is used by the standard firmware in two cases: two-accumulator multiple-operation instructions which specify a right shift, and the unsigned multiply instruction (MUL).

Swap Bytes

SW

14

The upper and lower bytes of the ALU output are swapped. The Link is not changed.



LOAD FIELD

The LOAD field, RBUF20, governs the loading of the A register from shifter output bits 0-15 as long as the CARRY field does not contain the ALC micro-order. When the ALC micro-order is present, the LOAD field is ignored and IR bit 12 instead governs the loading of the A register as follows:

IR12	
0	load
1	no load

The two possible micro-orders for the LOAD field are described below, along with the effects they have when the ALC micro-order is not present.

No Load

N

0

Bits 0-11 of the A register are not loaded with shifter output bits 0-11 at the end of the current instruction and consequently remain unchanged. Bits 12-15 of the A register are not loaded with shifter output bits 12-15 unless the RAND2 field contains DECL. (See the description of the DECL micro-order in the section dealing with the RAND2 field.)

Load

L

1

The entire A register is loaded with the shifter output at the end of the current microinstruction.

CARRY FIELD

The CARRY field, RBUF<21-22>, serves two main purposes: (1) to control the Carry bit, and (2) to enable the special CPU logic which implements the two-accumulator multiple-operation instructions.

No Carry Change

N

0

The Carry bit is unchanged by the current microinstruction.

Set Carry

SET

1

The Carry bit is set to 1 at the end of the current microinstruction.

Clear Carry

CLR

2

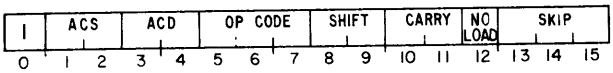
The Carry bit is set to 0 at the end of the current microinstruction.

Enable Two-Accumulator Multiple-Operation Instruction

ALC

3

The special hardware included in the CPU specifically for implementing two-accumulator multiple-operation instructions is enabled. For reference purposes, a two-accumulator multiple-operation instruction has the following general format:



where bits 12-15 are not 1000. The effects of the ALC micro-order on the various components of the computational logic are described below. (See also the description of the fields involved.)

Carry-in logic - A carry-in is generated (CN is set to 1) if IR bit 7 is 1; that is, if the two-accumulator multiple-operation instruction is NEG, INC, SUB, or AND.

Shifter - The class of shift operation to be performed is selected by IR bits 8 and 9 (the SHIFT field in the two-accumulator multiple-operation instruction) instead of by RBUF bits 16 and 17.

NOTE RBUF<18-19> still select the special value for shifting into bit 0 or 15. For a two-accumulator multiple-operation instruction to operate correctly, these bits must both be 1. Therefore, the SHIFT field of a microinstruction that contains the ALC micro-order must contain the FA, LC, or RC micro-order.

Register file - The loading of the A register with the output of the shifter is governed by IR bit 12, the "no-load" bit, instead of by RBUF20, the LOAD field. If IR12 is 0, loading is allowed. If IR12 is 1, loading is inhibited.

Caution The DECL micro-order, coded in the RAND2 field, overrides the two-accumulator multiple-operation no-load bit just as it overrides the LOAD field N micro-order. That is, if DECL and ALC are both coded in a microinstruction, and if IR bit 12 is 1, then the low nibble of the A register (bits 12-15) will be loaded with shifter output bits 12-15. However, if IR bit 12 is 0, the full 16 bits of shifter output will be loaded into the A register as usual.

Carry bit - A new value for the Carry bit is determined based on the current value of the Carry bit, IR bits 10 and 11 (the two-accumulator multiple-operation instruction Carry field), and the carry-out of the ALU. First a base value for CRY ENAB is computed, according to the table below.

IR bits 10-11	base value for CRY ENAB
00	current state of Carry bit
01	0
10	1
11	complement of current state of Carry bit

Then, for the AND instruction, the computed base value becomes CRY ENAB directly. For the other 7 two-accumulator multiple-operation instructions, the base value is complemented if there is a carry-out of the ALU; that is, if CRY0=1. (Note that, for the COM and MOV instructions, CRY0 is always 0, so no complementing of the base value can occur.) This result, complemented or not according to CRY0, becomes CRY ENAB.

CRY ENAB can be used in two ways in two-accumulator multiple-operation instructions: (1) it can be tested by the ALC skip hardware, and/or (2) it can be loaded back into the Carry bit (for swap and no shift operations), into A register bit 15 (for left shifts), or into A register bit 0 (for right shifts).

The final effect of coding ALC in the Carry field is that the ALC skip hardware is enabled, as mentioned above. This is logic which determines whether or not the next instruction should be skipped, based on the output of the shifter, on CRY ENAB (provided that RBUF bits 18 and 19 have both been properly set to 1), and on the SKIP field of the two-accumulator multiple-operation instruction, bits 13-15.

The ALC micro-order is not intended or recommended for general use.

EXAMPLES

The following example microinstructions show some of the ways in which the micro-orders that are coded in the first seven fields can be used to control the computational logic. The unused remaining fields are shown here by dashes for simplicity.

Add AC0 to AC1:

```
AR   AC1 AC0 APB    FA  L  N    -  ——   ——  ——   ——  -  ——  ——
```

Subtract AC2 from GR0.

```
AR   GR0 AC2 AMB    FA  L  N    -  ——   ——  ——   ——  -  ——  ——
```

Decrement the source accumulator:

```
AR   ACS —— AM1     FA  L  N    -  ——   ——  ——   ——  -  ——  ——
```

Increment PC:

```
AR   PC  —— A1      F0  L  N    -  ——   ——  ——   ——  -  ——  ——
```

Note that F0 is coded in the SHIFT field. Bit 0 of the PC should, in general, always remain 0.

Move the contents of the destination accumulator to GR0:

```
Z    GR0 ACD APB    FA  L  N    -  ——   ——  ——   ——  -  ——  ——
```

Note that both of the following microinstructions are equivalent to the one above:

```
Z    GR0 ACD AOB    FA  L  N    -  ——   ——  ——   ——  -  ——  ——
Z    GR0 ACD AXB    FA  L  N    -  ——   ——  ——   ——  -  ——  ——
```

Add 1 to the contents of the destination accumulator and place the result in GR0:

```
Z    GR0 ACD APB1   FA  L  N    -  ——   ——  ——   ——  -  ——  ——
```

Complement GR1:

```
AR   GR1 —— CA      FA  L  N    -  ——   ——  ——   ——  -  ——  ——
```

Complement GR1 but place the result in GR2:

```
Z    GR2 GR1 APCB   FA  L  N    -  ——   ——  ——   ——  -  ——  ——
```

Clear the upper byte of GR1:

```
LBY  GR1 —— A       FA  L  N    -  ——   ——  ——   ——  -  ——  ——
```

Select the upper byte of GR2 and leave it right-justified in GR2:

```
UBY GR2 —— A     SW L N     — ——     ——— ———     ——— — ——— ———
```

Shift AC2 right one place:

```
AR  AC2 —— A     RO L N     — ——     ——— ———     ——— — ——— ———
```

Shift GR0 left one place and put it in GR2:

```
Z   GR2 GRO AOB  LO L N     — ——     ——— ———     ——— — ——— ———
```

Shift AC3 left two places and set the Carry bit:

```
AR  AC3 —— APA   LO L SET   — ——     ——— ———     ——— — ——— ———
```

Perform the logical AND of AC0 and AC1, shift the result right with the Link, place the result in AC1, and clear the Carry bit:

```
AR  AC1 ACO ANB  RL L CLR   — ——     ——— ———     ——— — ——— ———
```

Set the Link to the logical exclusive OR of the high-order bits of GR1 and GR2, but don't change GR1, GR2, or the Carry bit:

```
AR  GR1 GR2 AXB  LO N N     — ——     ——— ———     ——— — ——— ———
```

Load the contents of the ACD field of the IR into GR1:

```
IRD GR1 —— A     FA L N     — ——     ——— ———     ——— — ——— ———
```

Add the contents of the ACS field of the IR to the destination accumulator:

```
IRS ACD ACD APB  FA L N     — ——     ——— ———     ——— — ——— ———
```

Load 213_8 into GR2:

```
CON GR2 —— A     FA L N     — ——     ——— ———     ——— — 213 ———
```

The constant (up to 8 bits) is coded in the TRUE ADDRESS field.

Add 7 to AC1:

```
CON AC1 AC1 APB  FA L N     — ——     ——— ———     ——— — 7   ———
```

Subtract 2 from the destination accumulator and place the result in the next consecutive accumulator.

```
CCN AD1 ACD APB  FA L N     — ——     ——— ———     ——— — 376 ———
```

Note that -2 is represented as octal 177776_8, the lower byte of which is 376.

MA FIELD

The MA field, RBUF23, allows the microprogram to specify when memory should be started. The two micro-orders which may be coded in the field are described below. Data channel breaks are discussed under the MBUS field, which follows.

No Effect

N

0

Memory is not started by the current microinstruction. A data channel break is not allowed during the current microinstruction unless the RAND1 field contains DCH.

Start Memory

S

1

A data channel break is allowed during the current microinstruction unless the RAND1 field contains SCND or STIR. At the end of the current microinstruction, the memory module containing the memory location selected by bits 1-15 of the ALU output is started: the access half-cycle for the selected module is initiated.

> **NOTE** If the address placed on the LA bus from the ALU output references a nonexistent memory location, the S micro-order has no effect except to allow a data channel break.

In order to access a memory location, the microinstruction must generate the desired address at the output of the ALU and specify the S micro-order. Bits 1-15 of the ALU output are always fed to the LA bus except when the data channel needs to specify a memory address. When a memory module sees the start command and realizes that it contains the memory location whose address is on the LA bus, it takes the address and initiates its access half-cycle, provided that it is not already busy. If the addressed module is busy finishing a prior memory operation, it sends back a "wait" signal which freezes CPU CLK, extending the current microinstruction. Once the previous operation is completed and the module is no longer busy, it releases the wait signal and allows the processor to proceed and finish the current microinstruction. At the end of the microinstruction, the memory module takes the new address and immediately initiates its access half-cycle.

Whether or not the memory module is or is not already busy when an S micro-order is given, once the module is free to take the new address, the memory module initiates and proceeds with its access half-cycle, while the current microinstruction finishes and the processor proceeds with the next one. Some later microinstruction (which may well be the very next one) issues a second memory control command, coded in the MBUS field, this time to tell the memory module which operation to perform.

MBUS FIELD

The MBUS field, RBUF<24-25>, serves two purposes: (1) to specify how the MEM bus is to be used during the current microinstruction; and (2) to control a memory module if one has been previously started but not yet released. The micro-orders which can be coded in this field are described below.

No Effect

N

0

The MEM bus is not used by the microprogram during the current microinstruction.

Read and Release

READ

3

If any memory module has been started but not yet released, the contents of the accessed memory location in that module are placed on the MEM bus, and the memory module is released. Whether or not any memory module has been started, the contents of the MEM bus are loaded into GR0 at the end of the current microinstruction unless the RAND2 field contains BMEM, in which case the contents of the MEM bus are loaded into the B register.

Write and Release

WRIT

2

The ALU output is placed on the MEM bus, unless the RAND2 field contains BMEM, in which case the contents of the B register are placed on the MEM bus. If any memory module has been started but not yet released, the data on the MEM bus is taken in by that memory module to be written into its accessed location and the memory module is released.

Read, No Release

RMOD

1

If any memory module has been started but not yet released, the contents of the accessed memory location in that module are placed on the MEM bus. However, unlike READ, the memory module is not released but remains at mid-cycle. Whether or not any memory module has been started, the contents of the MEM bus are loaded into GR0 at the end of the current microinstruction unless the RAND2 field contains BMEM, in which case the contents of the MEM bus are loaded into the B register.

Once the microprogram has started a memory module (see the MA field description), a second memory control command may be issued in any later microinstruction. This command is coded in the MBUS field and is one of the three micro-orders READ, WRIT, and RMOD. The READ micro-order reads data over the memory bus into the CPU and commands the memory module to initiate its restore half-cycle; the WRIT micro-order sends new data over the MEM bus to be written into the addressed location and commands the module to initiate its restore half-cycle; and the RMOD micro-order reads the data into the CPU but does not release the module. For READ and WRIT, the CPU interaction with the memory module for the current word of data is over—the memory module completes its restore half-cycle on its own, and the CPU proceeds with the next microinstruction. For RMOD, the interaction is not complete; the module will remain at mid-cycle until the CPU issues a READ or WRIT to release it. If a READ is given after an RMOD, the same data read for the RMOD is read again, and the module is released. If a WRIT is given, new data is supplied to the module for rewriting back into the addressed location and the module is released. This later case is, in fact, the primary use of RMOD: to read a memory location's contents and write back some new data based on the data just read. For example, an S...RMOD...WRIT sequence is used in implementing the ISZ and DSZ instructions in the standard instruction set for the ECLIPSE computer.

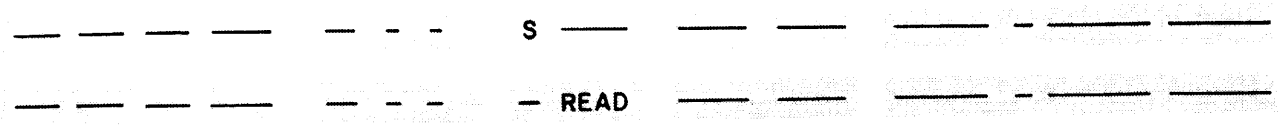
If the CPU issues an MBUS field command to a memory module before the module is prepared to accept the command, the memory module sends back a "wait" signal to the CPU. This signal freezes CPU CLK, thereby extending the current microinstruction. Once the memory module is ready to perform the designated operation, it releases the wait signal, allowing the microinstruction to finish. As the next microinstruction is loaded into RBUF, the designated operation is performed.

If, on the other hand, a memory module which has been started by the CPU completes its access half-cycle before the CPU issues a second memory command, the module pauses at mid-cycle until the second command is issued. A memory module is able to pause at mid-cycle for an indefinite length of time before performing its restore half-cycle. However, good microprogramming practice dictates that a module not be held at mid-cycle for more than a few microinstructions.

Because of the nature of memory/CPU interaction as explained above, microprogrammed control of memory is generally free of timing restrictions. A microprogram may start memory in a microinstruction and perform a READ, WRIT, or RMOD operation in any following microinstruction (subject only to the rules for overlapping memory operations as described later in this section). The microprogrammer need not concern himself with synchronizing his control of memory to a fixed memory cycle.

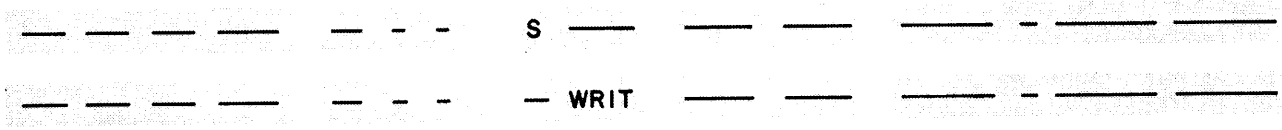
Four common sequences of MA-field and MBUS-field micro-orders are illustrated below:

1. Read operation



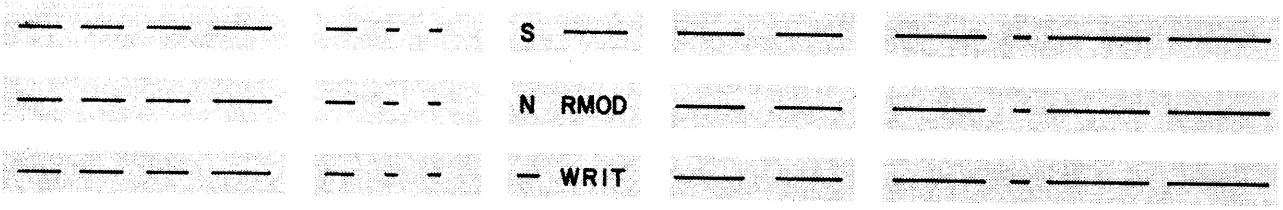
To perform a read from memory, memory is started in one microinstruction (with the desired address present on bits 1-15 of the ALU output) and read in some later microinstruction (not necessarily the next). If the microinstruction that contains the READ micro-order also contains the BMEM micro-order, data from memory will be read into the register selected by the BREG field of that microinstruction. Otherwise, data from memory will be read into GR0.

2. Write operation



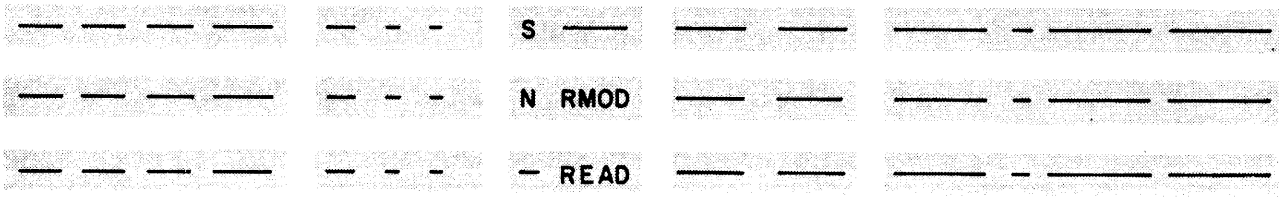
To perform a write operation, memory is started in one microinstruction (with the desired address present on bits 1-15 of the ALU output) and written in some later microinstruction (not necessarily the next). If the microinstruction that contains the WRIT micro-order also contains the BMEM micro-order, the contents of the register selected by the BREG field of that microinstruction are written into memory. Otherwise, the output of the ALU is written into memory.

3. Read-modify-write operation



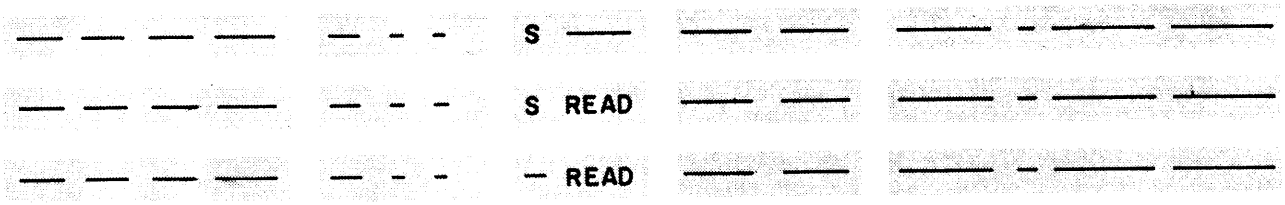
This sequence is used to read a memory location and write back new data into it which may be based on the original data. The RMOD micro-order operates exactly as the READ micro-order does (for instance, BMEM may accompany the RMOD in the same way), except that the memory module is not released. This allows the micro-program the capability to specify WRIT in a later microinstruction without having to start the module again on the same address.

4. S, RMOD, READ operation

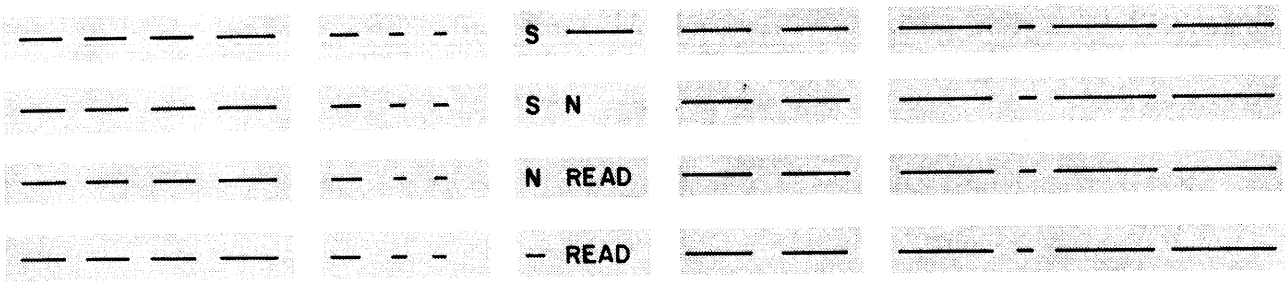


This sequence is used in two cases: (1) when the microprogram expects to perform a read-modify-write operation but discovers that the data in the memory location should not be changed; and (2) when microcode which performs a read-modify-write operation is being shared with microcode which does not want to change the data in memory. In both cases the RMOD reads the data but does not release the module. The READ is required to release the module; it also reads again the same data data that was read by the RMOD.

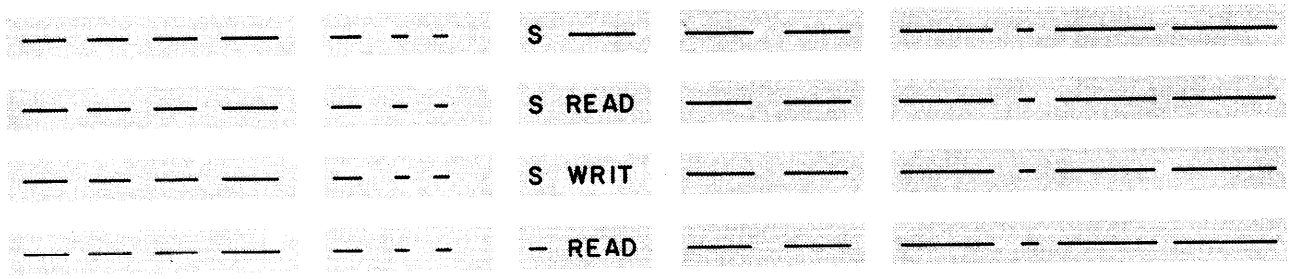
As mentioned in section 2, the asynchronous control of memory allows memory operations to be overlapped. The basic rule for overlapping memory operations is this: once a memory module has been started, no other memory module may be started prior to the microinstruction which releases the first module. Thus, this sequence is valid:



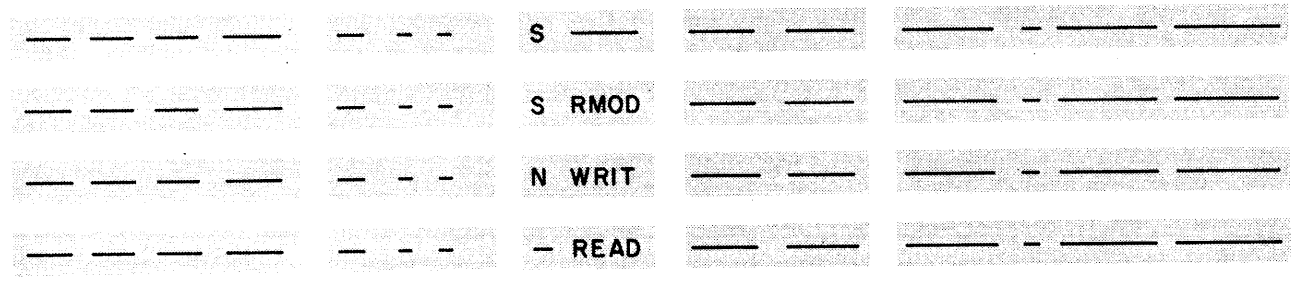
but this one is not:



Similarly, this sequence is valid:



but this one is not valid because RMOD does not release a module;



As explained in section 2, the data channel accesses memory with the same commands as the microprogram. This access is made during a data channel break which occurs within a microinstruction and causes that microinstruction to be extended. Once the data channel has acknowledged a data channel request from a device on the I/O bus, it waits for the microprogram to allow a data channel break. When a microinstruction which allows a break (see below) enters RBUF, the data channel generates the signal STOP CPU to freeze CPU CLK. First, the data channel allows any READ or WRIT specified by the frozen microinstruction to be completed, in order to free the busy module for possible access by the data channel. Then the data channel cycle is completed, including a complete write or read operation (for data channel input and output, respectively). When the data channel is done with all outstanding data channel transfer requests, it releases the STOP CPU signal and allows the frozen microinstruction to finish its execution. At this time all of the microinstruction's operations except READ or WRIT are performed (including a memory start, if the S micro-order is present). CPU CLK is then allowed to rise and the next microinstruction is loaded into RBUF.

The data channel must not be allowed to interfere with the microprogram, and, therefore, data channel breaks must not be allowed during certain microinstructions. For example, the hardware prohibits data channel breaks when the instruction in the IR is an I/O instruction. I/O instructions use the I/O bus and I/O register, so data channel cycles must not occur during I/O instructions. (Data channel cycles may occur between I/O instructions.) In other situations the control of data channel breaks is up to the microprogram. Coding DCH in the RAND1 field of a microinstruction allows a data channel break during that microinstruction. Specifying a memory start (S micro-order) also allows a data channel break unless the RAND1 field contains STIR or SCND. (This obviates the need to code DCH when S is coded. Microinstructions which start memory may allow data channel breaks since the rules for overlapping memory operations demand that any busy memory module will be released by a READ or WRIT in that microinstruction anyway.) There are several situations in which a microinstruction must not allow a data channel break. These situations are listed and explained in the description of the DCH micro-order in the RAND1 field.

The READ and WRIT micro-orders have a number of other uses dealing with control of the MEM bus but not of memory modules. READ is used with several micro-orders to read into a register the data placed on the MEM bus from a source other than memory, such as the console (CNDA and CNIN micro-orders), the I/O register (IOTR), and the Floating Point Processor (FPDA). Similarly, WRIT is used to place data on the MEM bus for transmission to a destination other than memory, including the IR (STIR micro-order), the I/O register (IOTR), and the Floating Point Processor (FPDA). Finally, if READ is given while no data is being placed on the MEM bus from any source, 0's are read into the specified register. This technique can be used to clear a register in the register file without having to feed zeroes through the ALU.

NOTE By convention, whenever a microinstruction places data on the MEM bus, that microinstruction should specify either READ, WRIT, or RMOD in the MBUS field.

RAND1 FIELD

The RAND1 field, RBUF<26-28>, serves two purposes. Its primary purpose is to allow the microinstruction to specify one of seven special control functions. The micro-orders for these control functions and their specific effects are described below. The secondary purpose of the RAND1 field is to select a return register into which the microsubroutine return address is to be loaded when the STATE CHANGE field contains JUMPSR or RTRNSR. When either of these two state changes is present, no special control signal is generated by the RAND1 field, regardless of the value of the bits in the field. Instead, RBUF<27-28> select one of four return registers, and RBUF26 is ignored.

None

N

0

No special control function is performed.

Allow Data Channel Break

DCH

1

A data channel break is allowed during the current microinstruction.

Data channel breaks should generally be allowed as frequently as possible, to insure proper operation of data channel devices. They must not be allowed, however, in the following cases:

1. When a memory module has been started but not yet released.
2. When a LDIR or NILDIR state change is being performed. The operation of these two state changes depends on the presence of the next instruction to be executed on the MEM bus when the current microinstruction terminates. A data channel break uses the MEM bus and leaves it in an indeterminate state, so such a break would keep the LDIR and NILDIR state changes from functioning correctly.
3. When the same register is designated both as the recipient of a READ operation and as an input to the ALU. Assuming that no data channel break happens, such a microinstruction does operate correctly: the current contents of the designated register are fed to the ALU for the duration of the microinstruction, and the new value is loaded into the register from the memory bus at the end of the microinstruction. If a data channel break is allowed to occur, however, the effect of the microinstruction then depends on whether or not a break actually occurs. If no break occurs, the microinstruction operates as usual. If a break occurs, however, the READ operation is allowed to finish before the break, while all other operations happen after the break. Thus, were a data channel break to occur, the data fed to the ALU would be the new (and presumably inappropriate) value for the register, instead of the value it contained before the break happened.
4. When the CNIN micro-order is present in the RAND2 field. The presence of the CNIN micro-order causes the console to place data on the MEM bus for the duration of the microinstruction. Consequently, use of the MEM bus by the data channel would result in indeterminate data being written into memory or sent out to the data channel device.
5. When the DATA lines of the I/O bus are in use. Since the data channel requires the use of the DATA lines, a data channel break must not be allowed to occur while these lines are being used; that is, during input/output transfers.

Also, it should be noted that the DCH micro-order is not necessary when the MA field contains S, since the presence of the S micro-order automatically allows a data channel break (unless either the SCND or STIR micro-order is present). Although it does no harm to code DCH along with S, it is redundant and therefore better microprogramming practice to avoid coding DCH in this case, thereby leaving the RAND1 field free to specify any of the other six special control functions or to select a return register.

Console Data

CNDA

6

Data is placed on the MEM bus according to the current setting of the 16 console data switches. A switch in the raised position causes a 1 to be placed on the corresponding line of the MEM bus; a switch in the lowered position causes a 0 to be placed on the corresponding line of the MEM bus.

NOTE A data channel break occurring in a microinstruction which specifies CNDA operates correctly, since the break keeps the console from placing its data on the MEM bus while the data channel is using the bus. After the break is over, the console is again allowed to place its data on the bus and the microinstruction terminates properly.

By convention, a microinstruction containing the CNDA micro-order should also contain a READ, WRIT, or RMOD micro-order.

Store into Instruction Register

STIR

7

At the end of the current microinstruction, the data on the MEM bus is loaded into the IR. Data channel breaks are inhibited during the current microinstruction.

The STIR micro-order can be used to load a new instruction into the IR without transferring control to a new microroutine and without producing the various other side effects of the LDIR and NILDIR state changes. However, data STIRed into the IR is, in other respects, just as much a new instruction as is data loaded into it by either LDIR or NILDIR. Instruction decoding based on the contents of the IR can be performed as always.

Caution If a microinstruction which specifies STIR also contains the ACEQBI or ACEQBD micro-order in its STATE CHANGE field the STIR micro-order does not operate as described above. IR bit 0 and IR bits 3-5 are loaded from the MEM bus as usual, but IR bits 1-2 are not. Instead, the contents of this field are decremented.

Caution STIRing a floating point instruction will activate the Floating Point Processor. Floating point instructions are instructions with bit 0 equal to 1 and bits 10-15 equal to 101000_2.

Caution STIRing an I/O instruction will freeze the CPU if the data channel logic is waiting for a microinstruction to enable data channel breaks, unless breaks are enabled by the microinstruction that immediately precedes or immediately follows the one that STIRs the I/O instruction. The CPU can be unfrozen only by the RESET switch on the console. An I/O instruction is one with bits 0-2 equal to 011_2.

Save Conditions

SCND

2

At the end of the current microinstruction, the Q bit and ALU0 SAVE are given new values. The Q bit is set to the complement of the exclusive-OR of three quantities: the current value of the Q bit, the current value of ALU0 SAVE and the value of CRY0 resulting from the current ALU function. ALU0 SAVE is set to the value of ALU result bit 0. In algebraic terms,

Q bit ← NOT (Q bit XOR ALU0 SAVE XOR CRY0)

ALU0 SAVE ← ALU0.

Data channel breaks are inhibited during the current microinstruction.

NOTE The SCND micro-order is overridden by the LDIR and NILDIR micro-orders, which initialize the Q bit to 1 and ALU0 SAVE to 0.

ALU0 SAVE and the Q bit are manipulated by the SCND, LQ, LDIR, and NILDIR micro-orders and are used primarily in the implementation of integer division.

In the following descriptions of the IOTR and IOPS micro-orders, it is assumed that the reader is familiar with the structure and signals of the I/O bus.

I/O Transfer

IOTR

3

The IOTR micro-order is used to implement in microcode the input/output transfer functions of I/O instructions. The IOTR micro-order generally has two effects: (1) it enables data contained in the I/O register to be placed on either the DATA lines of the I/O bus or on the MEM bus, depending on the direction of the transfer; and (2) it generates a data strobe signal which is sent out on the I/O bus.

The specific effects of the IOTR micro-order depend on the direction of the data transfer specified by the instruction contained in the IR and on whether or not the instruction specifies device code 77_8, indicating a special CPU I/O instruction. If IR bit 7 is 1, an input transfer is indicated. The data present on the DATA bus during the microinstruction which specifies IOTR is loaded into the I/O register at the end of the microinstruction and appears on the MEM bus during the following microinstruction. In addition, the IOTR micro-order causes a data strobe signal to be generated and sent out on the I/O bus during the following microinstruction. If IR bits 10-15 are not all 1, the signal generated is determined as follows:

IR5	IR6	Signal
0	0	DATIA
0	1	DATIB
1	0	DATIC
1	1	none

If IR bits 10-15 are all 1, a CPU I/O instruction is indicated, and the signal generated is determined as follows:

IR5	IR6	Signal
0	0	READS
0	1	INTA
1	0	IORST
1	1	none

NOTES The READS signal is not an I/O bus signal. Instead, like the CNDA micro-order described earlier in this section, it causes the current setting of the console data switches to be placed on the MEM bus.

The generation of the IORST signal prevents the READ and RMOD micro-orders from causing any register to be loaded from the MEM bus.

If IR bit 7 is 0, an output transfer is indicated. The data contained in the I/O register during the microinstruction which specifies IOTR is placed on the DATA lines of the I/O bus during that microinstruction. At the end of the microinstruction, the data present on the MEM bus is loaded into the I/O register. In addition, the IOTR micro-order causes a data strobe signal to be generated and sent out on the I/O bus during the following microinstruction if ALU bit 14 is 1 in the current microinstruction. If IR bits 10-15 are not all 1, the signal generated is determined as follows:

IR5	IR6	Signal
0	0	none
0	1	DATOA
1	0	DATOB
1	1	DATOC

If IR bits 10-15 are all 1, a CPU I/O instruction is indicated, and the signal generated is determined as follows:

IR5	IR6	Signal
0	0	none
0	1	none
1	0	MSKO
1	1	HALT

NOTE The HALT signal is not an I/O bus signal. Instead, it sets the STOP ENAB flip-flop to indicate a "stop pending" condition which may cause the processor to halt the next time a LDIR or NILDIR state change is performed.

I/O Pulse

IOPS

4

The IOPS micro-order is used to implement in microcode the "pulse" functions of I/O instructions. The IOPS micro-order generates one of a number of signals, based on the instruction contained in the IR and on whether or not the instruction specifies device code 77_8, indicating a special CPU I/O instruction. If IR bits 10-15 are not all 1, a "pulse" signal is generated and sent out on the I/O bus during the following microinstruction. The specific signal generated is determined by IR bits 8-9 as follows:

IR8	IR9	Signal
0	0	none
0	1	STRT
1	0	CLR
1	1	IOPLS

If IR bits 10-15 are all 1, a CPU I/O instruction is indicated, and a signal internal to the CPU is generated during the following microinstruction. The signals generated and their effects on the CPU are as follows:

IR8	IR9	Signal	Effect
0	0	-	none
0	1	SET ION	Set ION to 1 at the end of the current microinstruction unless the RAND2 field contains IOFF. Set ION PEND to 1 at the end of the current microinstruction unless the current microinstruction performs a successful LDIR or NILDIR state change.
1	0	CLR ION	Set ION to 0 at the end of the current microinstruction.
1	1	(unnamed)	Enable special CPU logic which causes the VECTOR instruction (VCT) to "read" a device code of 0 when power is failing (PWR FF = 1).

Floating Point Data

FPDA

5

A signal is sent to the Floating Point Processor to indicate either that it should place data on the MEM bus or take data from the bus.

The interaction between standard firmware and the Floating Point Processor is complicated and is not explained in this manual, and the FPDA micro-order is not intended or recommended for general use.

RAND2 FIELD

The RAND2 field, RBUF<29-31>, allows the microinstruction to specify one of seven special control functions, different from those of the RAND1 field. The micro-orders for these control functions and their specific effects are described below.

None

N

0

No special control function is performed.

Decimal Load

DECL

2

If the Carry bit is 1, a carry-in to the ALU is generated (CN is set to 1). At the end of the current microinstruction, shifter output bits 12-15 are loaded into bits 12-15 of the register selected by the AREG field. (The DECL micro-order assures that the low-order four bits of shifter output are loaded into the A register. Bits 0-11 of the A register may remain unchanged or may be loaded from shifter bits 0-11, depending on the contents of the LOAD and CARRY fields. Refer to these two fields for specifics.)

The DECL micro-order can be used to perform 4-bit and 16-bit multiple-precision arithmetic. When used with the DCRY and CRY12B micro-orders, DECL allows decimal arithmetic to be performed. When used with the L micro-order, full 16-bit multiple-precision arithmetic can be performed. (See the description of the ALU field.)

If the DECL micro-order is used in conjunction with any of the micro-orders A1, APA1, APB1, or AMB, remember that a carry-in is forced automatically by these micro-orders, so that the state of the Carry bit has no affect on the ALU result.

Load Count Register

LCNT

3

The 4-bit Count register is loaded with ALU output bits 12-15 at the end of the current microinstruction, unless a LDIR or NILDIR state change is performed, in which case the Count register is set to 17_8.

If the CNTND micro-order is coded in the STATE CHANGE field of a microinstruction which specifies LCNT, the state change will be performed properly, but the Count register will be loaded from the ALU output instead of being decremented. In summary then, with respect to determining the new value for the Count register, LDIR and NILDIR have priority over LCNT, which has priority over CNTND.

The LCNT micro-order is generally used to load the Count register either with an initial value to be decremented and tested in a loop with the CNTND micro-order (coded in the STATE CHANGE field) or with the number of a selected bit position, in preparation for use of the BIT micro-order (coded in the A INPUT field).

B Register to/from Memory

BMEM

1

The register specified by the BREG field is selected as the source or destination of the MEM bus.

A microinstruction containing the BMEM micro-order and the READ or RMOD micro-order causes the data on the MEM bus to be loaded into the B register rather than GR0 (unless the B register is GR0). The microinstruction may simultaneously load the output of the shifter into the A register (which may be GR0). If a microinstruction enables the loading of a register with both the output of the shifter and data from the MEM bus, the register will receive the bit-wise logical-OR of the two data.

A microinstruction containing the BMEM micro-order and the WRIT micro-order causes the contents of the B register, rather than the output of the ALU, to be placed on the MEM bus. This leaves the ALU free to compute something besides MEM bus data.

When the B register is used as the source or destination of the MEM bus, it may also be used as an operand by the ALU. A microinstruction that uses the B register as an ALU operand and also reads data into the B register from the MEM bus must not enable data channel breaks.

Interrupts

IOFF

5

ION is set to 0 at the end of the current microinstruction, thereby disabling the interrupt system.

The effect of the IOFF micro-order is the same as that of the INTDS (NIOC CPU) instruction. However, the IOFF function is performed directly by the microcode and is much quicker than executing the INTDS instruction.

Console Instruction

CNIN

6

The 4-bit function code corresponding to the highest-priority console function switch currently being pressed is placed on bits 1-4 of the MEM bus. The STOP ENAB and REXAM flip-flops are cleared.

The console functions, excluding Reset and Stop, are assigned function codes as shown in the table below. The priority runs from highest for AC0 Examine (code 0000) down to lowest for Instruction Step, Microinstruction Step, and Continue (code 1111), all three of which share the same instruction code and priority.

MEM1	MEM2	MEM3	MEM4	Console Function
0	0	0	0	AC0 Examine
0	0	0	1	AC1 Examine
0	0	1	0	AC2 Examine
0	0	1	1	AC3 Examine
0	1	0	0	AC0 Deposit
0	1	0	1	AC1 Deposit
0	1	1	0	AC2 Deposit
0	1	1	1	AC3 Deposit
1	0	0	0	Deposit
1	0	0	1	Deposit Next
1	0	1	0	Examine
1	0	1	1	Examine Next
1	1	0	0	Start
1	1	0	1	Execute
1	1	1	0	Program Load
1	1	1	1	Instruction Step, Microinstruction Step, and Continue

NOTES Do not allow a data channel break in a microinstruction which contains the CNIN micro-order. (See the description of the DCH micro-order in the RAND1 field.)

If no switch is being pressed when the CNIN micro-order is given, 1's are placed on bits 1-4 of the MEM bus, as if the Instruction Step, Microinstruction Step, or Continue switch were being pressed.

Power Fail

PFL

4

If PWR FF=1, a system reset is performed. The effect is the same as if the Reset console switch is pressed when the power switch is in the "On" position. However, PFL results in a system reset even when the power switch is in the "Lock" position.

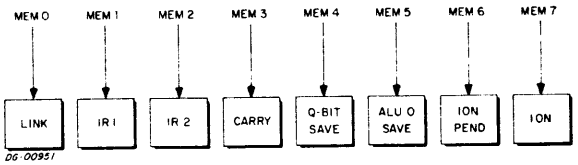
The PFL micro-order has a special use in the standard firmware and is not for general use.

Load Processor State

LPST

7

At the end of the current microinstruction, eight 1-bit registers in the CPU are loaded from MEM bus bits 0-7 as diagrammed below.



By convention, a microinstruction containing the LPST micro-order will also contain the READ, WRIT, or RMOD micro-order.

STATE CHANGE FIELD

The STATE CHANGE field, RBUF<32-37>, governs the selection of the next microinstruction to be loaded into RBUF. Most STATE CHANGE micro-orders specify a test condition which, when true, delegates selection to the PAGE and TRUE ADDRESS fields and, when false, delegates selection to the FALSE ADDRESS field. Other micro-orders determine the address of the next microinstruction from sources other than the TRUE ADDRESS and FALSE ADDRESS fields, including the instruction decode logic and the four microsubroutine return registers. Regardless of the source of such an address, it is used to address a microinstruction in the control store. Two special micro-orders, LDIR and NILDIR, which are used to link the end of one microroutine with the beginning of the next, either force one of several fixed addresses, depending on specific conditions within the CPU, or actually generate the next microinstruction directly, instead of causing a microinstruction to be read from the control store. Whether the next microinstruction is read from the control store or generated by the hardware, it is clocked into RBUF<0-55> on the next CPU CLK and subsequently assumes control of the processor for the following clock period.

The micro-orders which may be coded in the STATE CHANGE field are described below. They are grouped primarily according to the manner in which they choose the next microinstruction to be executed. Within these larger groups, micro-orders with similar effects are described together.

At present, only the first 36 of the 64 possible STATE CHANGE micro-orders may be used. The remaining micro-orders are reserved for future use.

In the block diagram that accompanies the description of the STATE CHANGE micro-orders, the labels "TRUE" and "FALSE" are used to denote collectively all "true" and all "false" instances of those micro-orders which may branch to either the true or false address. The paths labeled "TRUE" are activated by any one of these micro-orders when the true address is selected; the "FALSE" paths are activated when the false address is selected.

Unconditional Branches

The first two state changes described below, NC and JUMP, cause unconditional transfers of control. Control may be transferred either to the false address in the current page or to the true address in any page.

No Change

NC

27

Control is transferred to the address in the current page specified by the FALSE ADDRESS field.

The NC micro-order is used when control is to be transferred unconditionally to an address in the current page.

Jump

JUMP

5

Control is transferred to the address specified by the PAGE and TRUE ADDRESS fields.

The JUMP micro-order is most often used to transfer control to an address in a page different from the current one. It can, of course, be used to transfer control within the current page. However, it is generally better microprogramming practice to use the NC micro-order in this case in order to leave the TRUE ADDRESS field free whenever possible for use as a constant field with the CON and CCN micro-orders, which are coded in the A INPUT field.

True/False Branches

The next 25 state changes all perform a specific test and branch to either the true or false address according to the result of the test. Micro-orders are available to test the ALU result, various ALU carry-outs, special bits and registers in the processor, and a number of special non-processor conditions dealing with the console, the I/O bus, and the Floating Point Processor.

NOTE The letter "B" is often used in the mnemonics for the micro-orders in this group to indicate that the false address is selected when the condition named by the characters preceding the "B" is true, and vice versa. For example, when the CRY0B state change is performed, the false address is selected if CRY0 is 1 ("true"), and the true address is selected if CRY0 is 0 ("false").

ALU Result Tests

The following micro-orders perform various tests on the ALU function result. The test is performed on the unshifted result.

ALU Bit 0

ALU0

13

If ALU result bit 0 is 1, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

ALU Bit 12

ALU12

12

If ALU result bit 12 is 1, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

ALU Bit 14

ALU14

11

If ALU result bit 14 is 1, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

ALU Bit 15

ALU15

10

If ALU result bit 15 is 1, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

ALU Output Zero

ALUZ

37

If the ALU function result is 0, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

Auto Index

AUTIX

20

If the value of ALU result bits 1-15 is between 20_8 and 37_8, inclusive, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

The main purpose of the AUTIX micro-order is to recognize an auto-incrementing or auto-decrementing memory location when it is encountered in the calculation of a short effective address.

ALU Carry-out Tests

The following four micro-orders provide for the testing of various types of carries and borrows which may be generated when the ALU function selected in the microinstruction is applied to its inputs. With these micro-orders the microprogram can perform signed and unsigned integer comparisons and decimal arithmetic.

Complement of CRY0

CRY0B

17

If the carry-out of ALU bit 0 is 1, control is transferred to the address in the current page specified by the FALSE ADDRESS field. Otherwise, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields.

Signed Carry-out

SCRY

15

If CRY0 XOR A0 XOR B0=1, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address specified by the FALSE ADDRESS field.

The SCRY micro-order is used for comparison of signed integers. It is an extension of the normal unsigned integer comparison methods. To compare two unsigned integers, one takes their difference (using either one's or two's complement arithmetic) and checks the carry-out. If S is subtracted from D, a carry-out will occur if S is less than or equal to D. When the complement of S is added to D, a carry-out will occur if S is strictly less than D. To compare two signed integers, the same operations are performed and the carry-out is again tested. However, when the two numbers are of different sign, the meaning of the carry-out is reversed. Therefore, the signs of the two numbers are exclusive-ORed together and then with the actual carry-out, CRY0, to produce a signed carry-out which can be tested with the SCRY micro-order.

Complement of CRY12

CRY12B

14

If the carry-out of bit 12 of the ALU is 1, control is transferred to the address in the current page specified by the FALSE ADDRESS field. Otherwise, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields.

The CRY12B micro-order is used to detect carries when performing 4-bit arithmetic. In particular, it is used to detect a borrow generated by a decimal subtraction. When no borrow is generated, CRY12 is 1, and control is transferred via the FALSE ADDRESS field. When a borrow occurs, CRY12 is 0, and control is transferred via the PAGE and TRUE ADDRESS fields.

Decimal Carry-out

DCRY

16

If CRY12 is 1 or if ALU output bits 12-15 represent a number greater than 9, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

The DCRY micro-order is used to detect a carry condition when performing decimal addition. Such a condition is indicated when the two decimal numbers added sum to greater than 9.

Special Bit Tests

The following four micro-orders test single-bit quantities in the CPU.

Carry Bit

CARRY

35

If the Carry bit is 1, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

Link Bit

LINK

26

If the Link bit is 1, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

Q Bit

QBIT

34

If the Q bit is 1, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

A Input Bit 0

A0

36

If bit 0 of the A bus is 1, control is transferred to the address specified by the PAGE and TRUE

ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

Note the difference between the A0 and ALU0 micro-orders. The A0 micro-order tests bit 0 of the A input to the ALU. The ALU0 micro-order tests bit 0 of the output of the ALU.

Tests That Modify a Register

The following three state changes each test a register in the processor to select either the true or false address, then alter that register at the end of the current microinstruction. It is important to remember that the contents of the register are tested before they are altered.

Count Register Nonzero, Decrement

CNTND

25

If the current value contained in the Count register is nonzero, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field. At the end of the current microinstruction (after the test is performed), the Count register is decremented. (Decrementing 0 gives 17_8.)

NOTE The LCNT micro-order coded in the RAND2 field, overrides the decrementing function of the CNTND micro-order. That is, when the LCNT and CNTND micro-orders are both present in a microinstruction, the true or false address will be selected properly based on the current value of the Count register, but at the end of the current microinstruction the Count register will be loaded from the ALU output (as for LCNT) instead of being decremented.

The CNTND micro-order allows for convenient looping up to 16 times within microcode.

Accumulator Specifiers Not Equal, Increment

ACEQBI

23

If IR1 = IR3 and IR2 = IR4, control is transferred to the address in the current page specified by the FALSE ADDRESS field. Otherwise, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. At the end of the current microinstruction (after the test is performed), the contents of IR bits 1-2 are incremented. (Incrementing 11 gives 00.)

Caution The presence of the STIR micro-order in the RAND1 field of a microinstruction which specifies an ACEQBI state change will cause IR bits 1-2 to be decremented instead of incremented. The test will be performed properly.

Accumulator Specifiers Not Equal, Decrement

ACEQBD

24

If IR1 = IR3 and IR2 = IR4, control is transferred to the address in the current page specified by the FALSE ADDRESS field. Otherwise, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. At the end of the current microinstruction (after the test is performed), the contents of IR bits 1-2 are decremented. (Decrementing 00 gives 11.)

The ACEQBI and ACEQBD micro-orders can be used in a loop to perform the same operation on consecutive accumulators or general registers. The PUSH MULTIPLE ACCUMULATORS (PSH) and POP MULTIPLE ACCUMULATORS (POP) instructions use the ACEQBI and ACEQBD micro-orders, respectively, in this way.

Caution Use of either the ACEQBI or ACEQBD micro-order with STIR causes IR bits 1-2 to be decremented instead of being loaded with new values from the MEM bus.

Console Tests

The following two micro-orders allow a microinstruction to test two conditions which reflect the state of the console.

Console Request

CONRQ

32

If any console function switch except Reset/Stop is pressed, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

When any console function switch except Reset/Stop is pressed, a console request signal is generated for one period of CPU CLK. A microinstruction can test this signal with the CONRQ micro-order. Since the request lasts for only one period of CPU CLK, a microroutine must test for it continually (that is, every microinstruction) when it wants to recognize that a switch has been pressed. One way to do this is with a one-microinstruction loop which contains a CONRQ state change and branches to itself when no request is detected. This is what the standard firmware does when the CPU is "halted" and waiting for a console function switch to be pressed.

The CONRQ micro-order allows the following switch functions to be implemented at least partially in microcode: Deposit/Examine for all accumulators, Examine/Examine Next, Instruction Step/Microinstruction Step, Program Load/Execute, Start/Continue, and Deposit/Deposit Next. The two remaining console functions, Reset and Stop, are implemented in hardware alone.

Power Switch Position Test

LOCKB

33

If the ECLIPSE computer's power switch is in the "Lock" position, control is transferred to the address in the current page specified by the FALSE ADDRESS field. Otherwise, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields.

The LOCKB micro-order is used by the standard firmware to determine what to do at auto-restart time. If the power switch is in the "Lock" position, instruction execution begins at memory location 0. If not, the computer merely halts.

I/O Bus Tests

The following two micro-orders allow a microinstruction to test two conditions dealing with the state of peripherals on the I/O bus.

Interrupt Waiting

INTR

30

If an interrupt is waiting, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

An interrupt is "waiting" if the following three conditions are true:

1. ION is 1;
2. ION PEND is 0;

   and
3. either: (a) PWR FF is 1;

   or

   (b) INTR FF is 1 and the microinstruction does not contain the CNDA micro-order.

(See the description of LDIR and NILDIR later in this section for additional explanation of the above.)

The INTR micro-order allows a lengthy microroutine to be interruptable. The microroutine can check explicitly for a waiting interrupt at various points during its execution, rather than having to perform repeated LDIR state changes or to wait for the end of the microroutine to allow an interrupt to occur.

I/O SKIP Test

IOSKPB

31

If the I/O SKIP test condition specified by IR bits 8-9 is true for the peripheral specified by IR bits 10-15, then control is transferred to the address in the current page specified by the FALSE ADDRESS field. Otherwise, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields.

The IOSKPB micro-order is used to implement the I/O SKIP instruction on the ECLIPSE computer. The test condition checked by the IOSKPB micro-order and hence by the I/O SKIP instruction is determined by the instruction contained in the IR. If IR bits 10-15 are not all 1, a test is made on the Busy or Done flag in the peripheral selected by IR<10-15> according to IR bits 8-9 as follows:

IR8	IR9	I/O SKIP test condition
0	0	Busy = 1
0	1	Busy = 0
1	0	Done = 1
1	1	Done = 0

If IR bits 10-15 are all 1, a test is made on ION or PWR FF according to IR bits 8-9 as follows:

IR8	IR9	I/O SKIP test condition
0	0	ION = 1
0	1	ION = 0
1	0	PWR FF = 1
1	1	PWR FF = 0

Floating Point Processor Tests

The following four micro-orders allow the microprogram to test various conditions in the Floating Point Processor.

Floating Point Busy

FPB

41

If the Floating Point Processor is busy executing a previously initiated floating point instruction, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

Floating Point Trap

FPT

42

If a floating point trap is waiting, that is, if traps are enabled and a trap condition has occurred in the Floating Point Processor as a result of the previous floating point instruction, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

Floating Point Trap or Busy

FPTB

40

If the Floating Point Processor is busy or if a trap is waiting, control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

Floating Point Skip

FPSK

43

If the floating point skip test condition specified in the Floating Point Instruction Register is true, then control is transferred to the address specified by the PAGE and TRUE ADDRESS fields. Otherwise, control is transferred to the address in the current page specified by the FALSE ADDRESS field.

Microsubroutining

The following three micro-orders allow the microprogrammer to implement subroutines and coroutines in microcode.

The JUMPSR and RTRN micro-orders are used together to implement "microsubroutines". JUMPSR "calls" a microsubroutine and RTRN returns from it. RTRNSR is used, along with JUMPSR and RTRN, to implement "microcoroutines". Note that a microinstruction may call a microsubroutine or microcoroutine in any page and that the return from that microsubroutine or microcoroutine will restore control to the original page.

Jump and Save Return

JUMPSR

3

The current page and the address contained in the FALSE ADDRESS field are loaded into the return register selected by the RAND1 field. (RBUF<27-28> select one of four return registers; RBUF26 is ignored.) Control is then transferred to the address specified by the PAGE and TRUE ADDRESS fields.

Return

RTRN

4

Control is transferred to the 10-bit control store address contained in the return register selected by the PAGE field, RBUF<38-39>.

Return and Save Return

RTRNSR

6

The current page and the address contained in the FALSE ADDRESS field are loaded into the return register selected by the RAND1 field. (RBUF<27-28> select one of four return registers; RBUF26 is ignored.) Control is then transferred to the 10-bit microinstruction address contained in the return register selected by the PAGE field, RBUF<38-39>.

NOTE Do not allow the RAND1 and PAGE fields to select the same return register when the RTRNSR micro-order is used. RTRNSR does not work properly in this situation because the new return address is loaded into the selected return register before control is transferred to the old return address originally contained in that register.

Instruction Decoding

The instruction decode logic provides multi-way branching based on the instruction currently contained in the IR. This logic can be enabled by any of the four micro-orders described below. The first two micro-orders, DEC1 and DEC2, cause an unconditional branch to one of two decode addresses corresponding to the instruction contained in the IR. The other two, IR5BD1 and A0BD1, perform a test and choose between the false address and a decode address. Whenever a decode address is selected, the page bits are supplied by the PAGE field. (See the description of the LDIR micro-order later in this section for the one qualification to this rule.)

Decode 1

DEC1

1

Control is transferred to the page selected by the PAGE field and to the decode 1 address in that page for the instruction currently contained in the IR.

When a new instruction has been fetched and read into the IR, the DEC1 state change can be used to transfer control to the proper microroutine for that instruction.

NOTE When a phantom microinstruction contains the DEC1 micro-order, the instruction decode logic produces new page bits as well as an 8-bit address within the page. If the CPU includes the WCS feature, page 2 is selected for XOP1 instructions, and page 0 or 1 is selected for all other instructions. Otherwise, page 0 or 1 is always selected.

Decode 2

DEC2

2

Control is transferred to the page selected by the PAGE field and to the decode 2 address in that page for the instruction currently contained in the IR.

The DEC2 micro-order extends the instruction decoding capability. Like DEC1, it effects a multi-way branch from microcode common to several instructions to different sections of microcode specific to individual instructions. The DEC2 micro-order provides a very convenient method for "microcode sharing". Several instructions which execute the same first steps can all branch to the same microinstruction under control of DEC1, execute the same sequence of microinstructions as long as this execution remains identical, and then branch off to their own sections of microcode with a DEC2 state change in order to continue their individual executions.

Initial Defer Test

IR5BD1

21

If IR bit 5 is 1, control is transferred to the address in the current page specified by the FALSE ADDRESS field. Otherwise, a DEC1 state change is performed.

Multi-level Defer Test

A0BD1

22

If bit 0 of the A input to the ALU is 1, control is transferred to the address in the current page specified by the FALSE ADDRESS field. Otherwise, a DEC1 state change is performed.

The IR5BD1 and A0BD1 micro-orders are used to check for indirect addresses in effective address calculations.

Microroutine Chaining

The last two micro-orders described in this section, LDIR and NILDIR, are used to chain microroutines together; that is, they perform the functions required to transfer control from the end of one microroutine to the beginning of the next. Most frequently, control must be transferred to a new microroutine because a new instruction has been fetched from memory and must be executed. Occasionally, however, some condition in the processor will demand that control be transferred to a special microroutine. The special microroutines are the halt/stop microroutine, the program interrupt microroutine, and the running examine microroutine.

Execution of a New Instruction.

Unless some condition in the CPU causes a transfer of control to a special microroutine, the LDIR and NILDIR micro-orders cause the loading of a new instruction into the IR from the MEM bus, the initialization of certain registers in the CPU, and the generation of a phantom microinstruction.

Since the IR is loaded from the MEM bus, it is the responsibility of each microroutine to fetch the next instruction and insure that it is placed on the MEM bus during a microinstruction that contains LDIR or NILDIR. Most commonly the next instruction is located in memory. In this case, the microroutine need only start memory on the correct address (which is most often contained in PC) in one microinstruction, then perform a READ LDIR combination in some later microinstruction (usually the next).

The next instruction to be executed need not always reside in memory. It may be contained in an accumulator, as in the case of the EXECUTE instruction (XCT), or in a general register, or it may even be computed by a microinstruction and therefore exist only as the ALU result. In any of these situations the instruction must be placed on the MEM bus with a WRIT micro-order. (As explained later, the microprogrammer must insure that the new instruction is not only placed on the MEM bus but also loaded into GR0 if it is not already contained in GR0.) Typical terminating sequences for the various cases described in the above two paragraphs are discussed in detail later.

In order for the standard firmware to work properly, certain registers in the CPU must be initialized by the hardware at the beginning of each instruction. When a new instruction is loaded into the IR, the LDIR and NILDIR state changes enable this initialization: The Q bit is set to 1 and ALU0 SAVE is set to 0, primarily for the benefit of divide instructions. Unless the new instruction sets ION and ION PEND to 1, ION PEND is set to zero so that interrupts may occur at the end of the new instruction. The COUNT register is set to 17_8 (all 1's) so that microroutines can conveniently loop 16 times without having to load the COUNT register explicitly.

There are several types of phantom microinstructions; however, the one that concerns the user of WCS is the one that is generated for XOP1 instructions. It increments PC and decodes the XOP1 instruction in the IR:

Transfer of Control to a Special Microroutine

When a microinstruction containing the LDIR or NILDIR micro-order is clocked into RBUF, there are three conditions that may inhibit the execution of the instruction to be loaded into the IR. In order of decreasing priority, these conditions are (1) a stop pending, (2) a program interrupt waiting and, (3) the REXAM flip-flop set. When one of these conditions is met, the next microinstruction is taken from a special location in the control store. (See the description of forced control store addresses in section 2.)

A stop is pending when the STOP ENAB flip-flop is set or is about to be set the next time CPU CLK rises. The STOP ENAB flip-flop is set if (1) the STOP flip-flop in the console is set or (2) the IR contains a HALT instruction and the previous microinstruction contained an IOTR micro-order. Once STOP ENAB is set, it remains set until a microinstruction containing a CNIN micro-order is loaded into RBUF.

If a stop is pending when the microinstruction in RBUF contains an LDIR or NILDIR micro-order and RBUF55 is 0, control is transferred to location 6 in the page of the control store specified by the PAGE field of the microinstruction. If the PAGE field is 0, control is transferred to the standard halt/stop microroutine, which STIRs zeroes into the IR (so that a leftover I/O instruction will not inhibit data channel breaks while the CPU is halted) and enters a one-microinstruction loop. This microinstruction displays the contents of the PC in the address lights and the contents of the accumulator selected by bits 3-4 of the most recently executed instruction in the data lights, and waits for a console switch to be pressed.

If a microinstruction in RBUF contains a LDIR or NILDIR micro-order and RBUF55 is 1 (that is, there is an odd address in the FALSE ADDRESS field), stops will be inhibited.

A program interrupt is waiting if the following conditions are met:

1. ION is 1 (that is, the interrupt system is enabled),
2. ION PEND is 0 (that is, the instruction which has just been executed did not set ION), and
3. either (a) PWR FF is 1 (that is, power is failing) or
   (b) INTR FF is 1 and the microinstruction in RBUF does not contain a CNDA micro-order (that is, an interrupt request is on the I/O bus and the microinstruction is not placing data from the console switches on the MEM bus).

A program interrupt will occur when

1. an interrupt is waiting; and
2. either (a) no stop is pending or
   (b) stops are inhibited by a 1 in RBUF55; and
3. the microinstruction in RBUF contains a LDIR micro-order.

When a program interrupt occurs, control is transferred to location 4 in the page of the control store specified by the PAGE field of the microinstruction. If the PAGE field is 0, control is transferred to the standard program interrupt microroutine which disables the interrupt system (with the IOFF micro-order), STIRs zeroes into the IR (as for stops), stores the contents of PC in memory location 0, and resumes instruction execution beginning with the instruction addressed (possibly indirectly) by memory location 1.

Program interrupts are inhibited when the NILDIR micro-order is used instead of the LDIR micro-order. In other respects, the two micro-orders are identical. Unless there is a specific reason why interrupts should be inhibited, the LDIR micro-order should be used to initiate execution of an instruction so that program interrupt latency is kept to a minimum.

If the REXAM flip-flop is set when the microinstruction in RBUF contains a LDIR or NILDIR micro-order, control is transferred to location 5 in the page of the control store specified by the PAGE field of the microinstruction, unless control is preempted by a stop or program interrupt. If the PAGE field is 0, control will be transferred to the standard running examine microroutine, which reads the memory location whose address is set in the console data switches and then loads the next instruction into the IR and resumes instruction execution. This microroutine is intended for use in conjunction with the address compare logic in the console. (See section 2.)

The REXAM flip-flop is set by the console EXAMINE switch and is cleared by the CNIN micro-order. If it is set, there is no way for the microprogrammer to inhibit special transfers of control to the running examine microroutine caused by the LDIR and NILDIR micro-orders.

Whenever control is transferred to one of the special microroutines by LDIR or NILDIR, the loading of the IR and the initialization of the Q bit, ALU0 SAVE, and ION PEND are inhibited; however, the COUNT register is still set to 17_8. The IR will be loaded and all four registers will be initialized at the conclusion of the special microroutine, when the execution of a new instruction is initiated by a LDIR or NILDIR micro-order.

Using LDIR and NILDIR

The usual manner in which a microroutine terminates and transfers control to a new microroutine is shown in the example below.

```
FIN: AR  PC  —— A      FO  L  -     S ——     ——  ——     NC      -  ———  ———

ISH: ——  ——  ——  ——    —  -  -     N READ   N    ——    LDIR    0  ———  0
```

The following points should be noted about such a terminating sequence:

1. It is not usually necessary to reload the PC, as it is done above. However, it is imperative that bit 0 of the PC be 0 at LDIR time, since the program interrupt microroutine stores all 16 bits in memory location 0.
2. It is not necessary, of course, that the NC micro-order be used to transfer control from the first microinstruction to the second, or even that the second microinstruction follow the first immediately.
3. Memory must not be started by the microinstruction which does the LDIR or NILDIR or the standard firmware will not operate correctly.
4. The READ which accompanies the LDIR or NILDIR must read the memory data into GR0. The phantom microinstructions expect this and will not work properly otherwise.
5. No data channel break may be allowed in a microinstruction which does LDIR or NILDIR. If a break occurred, the data channel would use the MEM bus and leave it in an indeterminate state at the end of the microinstruction, causing the phantom to be generated incorrectly.
6. The PAGE field must be set to 0 when a LDIR or NILDIR is done or the phantom will not operate correctly.
7. RBUF55 must be 0 if stops are to be allowed.

If stops are to be suppressed, RBUF55 is set to 1 in the microinstruction which does the LDIR or NILDIR:

```
ISH: ——  ——  ——  ——    —  -  -     N READ   N    ——    LDIR    0  ———  1
```

If program interrupts are to be suppressed, the NILDIR micro-order is used in place of LDIR:

```
ISH: ——  ——  ——  ——    —  -  -     N READ   N    ——    NILDIR  0  ———  0
```

If both stops and program interrupts are to be suppressed, the final microinstruction should be coded as follows:

```
ISH: —— —— —— ——   — - -   N READ   N   ——   NILDIR  0 ——— 1
```

If the new instruction is contained in a register, say GR0, no memory fetch is required, and either of the following single microinstructions will suffice:

```
DONE: AR  GRO —— A    — - -   N WRIT   N   ——   LDIR   0 ——— 0
```

or

```
DONE: —— —— GRO ——    — - -   N WRIT   N   BMEM   LDIR   0 ——— 0
```

If the instruction is contained in any register other than GR0, or if the instruction is generated by an ALU function, a copy of the instruction must be placed in GR0 as well as on the MEM bus. This is required because the standard firmware and the phantom microinstructions expect GR0 to contain a copy of the instruction at the beginning of each new microroutine. For example, if the instruction is contained in ACD, the following microinstruction could be used:

```
DONE: Z   GRO ACD AOB   FA  L -   N WRIT   N   ——   LDIR   0 ——— 0
```

In all three of the above examples, as always, the LDIR micro-order may be replaced by NILDIR if program interrupts are to be suppressed, and RBUF55 may be changed to 1 if stops are to be suppressed.

One final observation on the use of LDIR and NILDIR: it is useless to code any of the micro-orders SCND, STIR, and LCNT along with either LDIR or NILDIR, since the effects of LDIR and NILDIR override the effects of SCND, STIR, and LCNT.

The effects of LDIR and NILDIR, along with the restrictions of their use, are summarized below.

Load Instruction Register

LDIR

0

The Count register is initialized to 17_8 at the end of the current microinstruction. Then, if a stop is pending and RBUF55 is 0, or if a program interrupt is waiting, or if the REXAM flip-flop is set, control is transferred to the fixed address corresponding to the highest-priority condition present. (Stops have priority over program interrupts, which have priority over running examines.) Otherwise, at the end of the current microinstruction, the Q bit is set to 1, ALU0 SAVE is set to 0, ION PEND is set to 0, the IR is loaded from the MEM bus, and a phantom microinstruction, generated as a function of the instruction on the MEM bus, is loaded into RBUF.

No Interrupt, Load Instruction Register

NILDIR

7

The Count register is initialized to 17_8 at the end of the current microinstruction. Then, if a stop is pending and RBUF55 is 0, or if the REXAM flip-flop is set, control is transferred to the fixed address corresponding to the condition present. (If both are present, control is transferred to the stop microroutine.) Otherwise, at the end of the current microinstruction, the Q bit is set to 1, ALU0 SAVE is set to 0, ION PEND is set to 0, the IR is loaded from the MEM bus, and a phantom microinstruction, generated as a function of the MEM bus, is loaded into RBUF and begins executing.

There are five restrictions on the use of LDIR and NILDIR:

1. A data channel break must not be allowed in a microinstruction that does a LDIR or NILDIR.
2. Memory must not be started by a microinstruction which does an LDIR or NILDIR.
3. The PAGE field in a microinstruction which does an LDIR or NILDIR must select page 0.
4. At the conclusion of a microinstruction containing LDIR, bit 0 of the PC must be 0.
5. At the conclusion of a microinstruction containing LDIR or NILDIR, GR0 must contain a copy of the IR.

PAGE FIELD

The PAGE field, RBUF<38-39>, serves two purposes:

1. To supply the page bits for a microinstruction address when the state change logic selects either the true address or a decode address; and
2. to select one of four return registers when the state change is RTRN or RTRNSR.

The two restrictions on the use of the PAGE field are summarized below:

1. when coding the LDIR and NILDIR state changes, make sure that the PAGE field is 0;
2. when coding the RTRNSR state change, avoid having the PAGE and RAND1 fields select the same return register.

TRUE ADDRESS FIELD

The TRUE ADDRESS field, RBUF<40-47> serves two purposes:

1. to supply the low-order 8 bits of a control store address when the state change logic selects the true address; and
2. to supply an 8-bit constant which is placed on bits 8-15 of the A bus when the CON or CCN micro-order is coded in the A INPUT field.

A single microinstruction may use the contents of the TRUE ADDRESS field both as an 8-bit constant and as the low-order bits of a control store address.

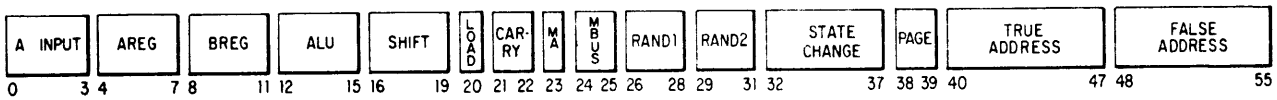
FALSE ADDRESS FIELD

The FALSE ADDRESS field, <RBUF 48-55> serves three purposes:

1. to supply the low-order 8 bits of a control store address when the state change logic selects the false address;
2. to supply the low-order 8 bits of a control store address which is to be saved in a return register when the state change is JUMPSR or RTRNSR; and
3. to allow stops to be suppressed when the state change is LDIR or NILDIR. (To suppress stops, RBUF55, the least significant bit of the FALSE ADDRESS field is set to 1.)

Whenever the FALSE ADDRESS field supplies the low-order 8 bits of a control store address--in cases (1) and (2) above--the contents of the CURRENT PAGE register are prefixed as the page bits. This means that transfers via the FALSE ADDRESS field can only be made within the current page, and that return from a microsubroutine is made to the page from which it was called.

SUMMARY OF MICRO-ORDERS



A INPUT

0	AR	AREG⟨0-15⟩ → A ⟨0-15⟩
1	IRD	IR⟨3-4⟩ → A⟨14-15⟩, 0's → A ⟨0-13⟩
2	BIT	$2^{(15-COUNT)}$ → A ⟨0-15⟩
3	TRP	IR10 → A10, IR ⟨5-9⟩ → A ⟨11-15⟩, 0's → A⟨0-9⟩ (1)
4	-	
5	PL	PL ROM word addressed by BREG ⟨11-15⟩ → A ⟨0-15⟩
6	-	
7	-	
10	LBY	AREG⟨8-15⟩ → A⟨8-15⟩, 0's → A⟨0-7⟩
11	IRS	IR⟨1-2⟩ → A⟨14-15⟩, 0's → A⟨0-13⟩
12	Z	0's → A⟨0-15⟩
13	CON	RBUF⟨40-47⟩ → A⟨8-15⟩, 0's → A⟨0-7⟩
14	SEX	AREG⟨8-15⟩ → A⟨8-15⟩, IR8's → A⟨0-7⟩
15	-	
16	CCN	RBUF⟨40-47⟩ → A⟨8-15⟩, 1's → A⟨0-7⟩
17	UBY	AREG⟨0-7⟩ → A ⟨0-7⟩, 0's → A⟨8-15⟩

AREG/BREG

0	ACS
1	AD1
2	ACD
3	-
4	GRS
5	GD1
6	GRD
7	-
10	AC0
11	AC1
12	AC2
13	AC3
14	GR0
15	GR1
16	GR2
17	PC

ALU (2)

0	A	A + C_{in}
1	APB	A + B + C_{in}
2	A1	A + 1
3	APCB	A + $\bar{B}$ + C_{in}
4	AM1	A - 1 + C_{in}
5	APA	A + A + C_{in}
6	APA1	A + A + 1
7	APB1	A + B + 1
10	AMB	A - B
11	CA	$\bar{A}$
12	AOB	A OR B
13	AXB	A XOR B
14	ANB	A AND B
15	ANCB	A AND $\bar{B}$
16	CANB	$\bar{A}$ AND B
17	ANBC	$\bar{A}$ AND $\bar{B}$

NOTES

1) COUNT must be ⟩ 7
2) C_{in} = (DECL AND CARRY) OR (ALC AND IR7)
3) LINK modified by left and right shifts
4) Unless ALC with IR12=1
5) Allows data channel break unless STIR or SCND
6) Do not allow data channel break
7) Disable data channel break
8) Do not code with ACEQBI or ACEQBD
9) False address (F) is in current page, true address (T) may change current page
10) Inhibited by HALT⁄STOP (if RBUF55=0), INTERRUPT WAITING, or REXAM
11) Current page saved in return register (RREG) with false address
12) Do not code RBUF⟨27-28⟩ = RBUF⟨38-39⟩
13) Inhibited by HALT⁄STOP (if RBUF55=0) or REXAM
14) Do not code with STIR
15) A bar (e.g., $\bar{A}$) represents logical complement

SHIFT (3)

0	F0	Straight, 0 → SHIFT0
1	FC	Straight, CARRY → SHIFT0
2	FI	Straight, ION → SHIFT0
3	FA	Straight, all 16 bits
4	L0	Left, 0 → SHIFT15
5	LL	Left, LINK → SHIFT15
6	LQ	Left, QBIT → SHIFT15
7	LC	Left, CRY ENAB → SHIFT15
10	R0	Right, 0 → SHIFT0
11	RL	Right, LINK → SHIFT0
12	RM	Right, MULS CRY → SHIFT0
13	RC	Right, CRY ENAB → SHIFT0
14	SW	Swap bytes
15	-	
16	-	
17	-	

LOAD

0	N	No effect
1	L	SHIFT⟨0-15⟩ → AREG⟨0-15⟩ (4)

MA

0	N	No effect
1	S	ALU⟨1-15⟩ → LA⟨1-15⟩, start memory (5)

CARRY

0	N	No effect
1	SET	1 → CARRY
2	CLR	0 → CARRY
3	ALC	Enable ALC logic

MBUS

0	N	MEM bus unused
1	RMOD	Read, no release (6)
2	WRIT	Write and release
3	READ	Read and release

RAND1

0	N	No effect
1	DCH	Allow data channel break
2	SCND	$\overline{QBIT}$ XOR ALU0 SAVE XOR CRY0 → QBIT, ALU0 → ALU0 SAVE (7)
3	IOTR	I/O transfer (6)
4	IOPS	I/O pulse (6)
5	FPDA	Floating point data
6	CNDA	Console data switches → MEM⟨0-15⟩
7	STIR	MEM⟨0-15⟩ → IR⟨0-15⟩ (7), (8)

RAND2

0	N	No effect
1	BMEM	BREG⟨0-15⟩ ⟷ MEM⟨0-15⟩
2	DECL	CARRY → C_{in}; SHIFT⟨12-15⟩ → AREG⟨12-15⟩
3	LCNT	ALU⟨12-15⟩ → COUNT⟨12-15⟩
4	PFL	SYS RST if PWR FF = 1
5	IOFF	0 → ION
6	CNIN	Console function code → MEM⟨1-4⟩ (6)
7	LPST	Load processor state

STATE CHANGE (9)

0	LDIR	Phantom; MEM → IR, 17 → COUNT, 1 → QBIT, 0 → ALU0 SAVE, 0 → ION PEND (6), (10)
1	DEC1	Decode 1
2	DEC2	Decode 2
3	JUMPSR	T; F → RREG selected by RBUF⟨27-28⟩ (11)
4	RTRN	RREG selected by RBUF⟨38-39⟩
5	JUMP	T
6	RTRNSR	RREG selected by RBUF⟨38-39⟩; F → RREG selected by RBUF⟨27-28⟩ (11), (12)
7	NILDIR	Phantom; MEM → IR, 17 → COUNT, 1 → QBIT, 0 → ALU0 SAVE, 0 → ION PEND (6), (13)
10	ALU15	T if ALU15 = 1, else F
11	ALU14	T if ALU14 = 1, else F
12	ALU12	T if ALU12 = 1, else F
13	ALU0	T if ALU0 = 1, else F
14	CRY12B	F if CRY12 = 1, else T
15	SCRY	T if A0 XOR B0 XOR CRY0 = 1, else F
16	DCRY	T if CRY12 = 1 or ALU⟨12-15⟩ > 9, else F
17	CRY0B	F if CRY0 = 1, else T
20	AUTIX	T if $20_8 \leq$ ALU⟨1-15⟩ $\leq 37_8$, else F
21	IR5BD1	F if IR5 = 1, else decode 1
22	A0BD1	F if A0 = 1, else decode 1
23	ACEQBI	F if IR⟨1-2⟩ = IR⟨3-4⟩, else T; increment IR⟨1-2⟩ (14)
24	ACEQBD	F if IR⟨1-2⟩ = IR⟨3-4⟩, else T; decrement IR⟨1-2⟩ (14)
25	CNTND	T if COUNT ≠ 0, else F; decrement COUNT
26	LINK	T if LINK = 1, else F
27	NC	F
30	INTR	T if INTERRUPT WAITING, else F
31	IOSKPB	F if I/O SKIP test true, else T
32	CONRQ	T if console switch pressed, else F
33	LOCKB	F if power switch in LOCK position, else T
34	QBIT	T if QBIT = 1, else F
35	CARRY	T if CARRY = 1, else F
36	A0	T if A0 = 1, else F
37	ALUZ	T if ALU⟨0-15⟩ = 0, else F
40	FPTB	T if FPP trap or busy, else F
41	FPB	T if FPP busy, else F
42	FPT	T if FPP trap, else F
43	FPSK	T if FPP skip, else F

SECTION 4

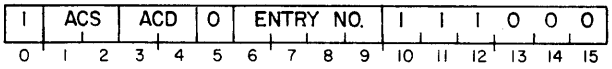
HOW TO USE WCS

The principal parts of WCS are a 32-word 8-bit decode RAM that stores decode addresses; a 256-word 56-bit control store RAM that stores microinstructions; and a 10-bit RAM address register used for loading the two RAM's.

The decode RAM provides the state change logic with 8-bit addresses when the microinstruction in RBUF has a DEC1 or DEC2 micro-order in the STATE CHANGE field and the IR contains an XOP1 instruction. The RAM can store 32 addresses, one for each of two decode micro-orders for each of sixteen entry numbers that may appear in bits 6-9 of an XOP1 instruction.

The control store RAM is page 2 of the ECLIPSE computer's control store. Microinstructions stored in this RAM control the operation of the CPU when an XOP1 instruction is executed. The format of the XOP1 instruction is as follows:

XOP1 acs, acd, entry number

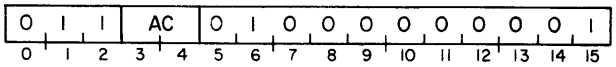


When an XOP1 instruction is loaded into the IR by a LDIR or NILDIR micro-order, the subsequent phantom microinstruction has a DEC1 micro-order in its STATE CHANGE field, and special hardware forces the succeeding microinstruction to be read from page 2 (the control store RAM). Since DEC1 may yield a unique address for each of the sixteen potential entry numbers in an XOP1 instruction, each entry number may select the beginning of a different microroutine in the control store RAM.

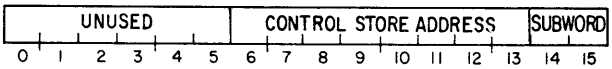
Information is loaded into WCS by three I/O instructions. (For discussion of I/O instructions, see the I/O section of the "Programmer's Reference Manual for the ECLIPSE Computer" DGC 015-000024.) These I/O instructions must be executed in pairs. The first specifies where information is to be stored in WCS. The second sends the information (a decode address or a part of a microinstruction) to WCS.

SPECIFY ADDRESS

DOA ac, WCS

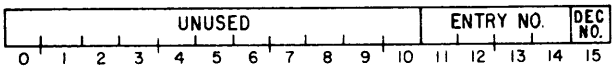


The contents of the specified AC are transferred to the WCS address register. The format of the information in the specified AC is dependent upon whether the user is transferring decode addresses or microinstructions into WCS. If this SPECIFY ADDRESS instruction is to be followed by a LOAD MICROCODE instruction, the contents of the specified AC are interpreted as follows:



Bit Number	Contents
0-5	Unused
6-13	Eight-bit address specifying a location in page 2 of the control store to be loaded by the following LOAD MICROCODE instruction.
14-15	Two-bit subword selector specifying which of the 56 bits in the specified location will be loaded by the following LOAD MICROCODE instruction. Subwords in a microinstruction are numbered as follows: subword 0 is bits 0-15; subword 1 is bits 16-31; subword 2 is bits 32-47; subword 3 is bits 48-55.

If the SPECIFY ADDRESS instruction is to be followed by a LOAD DECODE ADDRESS instruction, the contents of the specified AC are interpreted as follows:

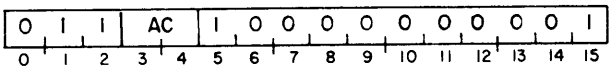


Bit Number	Contents
0-10	Unused
11-14	Entry number - from bits 6-9 of the corresponding XOP1 instruction.
15	Decode number - 0 if the following LOAD DECODE ADDRESS instruction specifies a decode 1 address, 1 if the following LOAD DECODE ADDRESS instruction specifies a decode 2 address.

A SPECIFY ADDRESS instruction stores the contents of the specified AC in the WCS address register until a subsequent LOAD MICROCODE or LOAD DECODE ADDRESS is executed. The contents of the AC remain unchanged.

LOAD MICROCODE

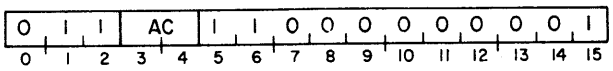
DOB ac,WCS



The contents of the specified AC are placed in the subword of the WCS control store RAM location selected by the previous SPECIFY ADDRESS instruction. If the SPECIFY ADDRESS instruction has selected subword 3 (bits 48-55) of the control store location, only bits 0-7 of the AC are stored. The contents of the AC remain unchanged.

LOAD DECODE ADDRESS

DOC ac,WCS



Bits 8-15 of the specified AC are placed in the decode RAM in the word specified by the previous SPECIFY ADDRESS instruction. The contents of the AC remain unchanged.

It is important to remember that WCS sometimes functions as an I/O device and sometimes functions as a part of the CPU. For purposes of loading decode addresses and microinstructions into WCS, it is an I/O device (device code 01). When CPU operation is determined by decode addresses and microinstructions already stored in WCS, it is an integral part of the CPU's control logic.

CUT ALONG DOTTED LINE

READERS COMMENT FORM

DOCUMENT TITLE: .....

Your comments, accompanied by answers to the following questions, help us improve the quality and usefulness of our publications. If your answer to a question is "no" or requires qualification, please explain.

How did you use this publication?

( ) As an introduction to the subject.
( ) As an aid for advanced knowledge.
( ) For information about operating procedures.
( ) To instruct in a class.
( ) As a student in a class.
( ) As a reference manual.
( ) Other.....

Did you find the material:

- Useful..... YES ( ) NO ( )
- Complete..... YES ( ) NO ( )
- Accurate..... YES ( ) NO ( )
- Well organized..... YES ( ) NO ( )
- Well written..... YES ( ) NO ( )
- Well illustrated..... YES ( ) NO ( )
- Well indexed..... YES ( ) NO ( )
- Easy to read..... YES ( ) NO ( )
- Easy to understand..... YES ( ) NO ( )

We would appreciate any other comments; please label each comment as an addition, deletion, change, or error and reference page numbers where applicable.

COMMENTS

PAGE	COL	PARA	LINE	FROM	TO

From:

NAME..... TITLE.....
FIRM..... DIV.....
ADDRESS.....
CITY..... STATE..... ZIP.....
TELEPHONE..... DATE.....

Data General Corporation

ENGINEERING PUBLICATIONS
COMMENT FORM
DG-00935

FOLD DOWN FIRST FOLD DOWN

FIRST CLASS
PERMIT NO. 26
SOUTHBORO
MASS. 01772

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

Postage will be paid by:

DataGeneral
Southboro, Massachusetts 01772

ATTENTION: Engineering Publications

FOLD UP SECOND FOLD UP

STAPLE