2019 south ritchey street · santa ana, california 92705 · (714) 558-8211

CAL DATA 100 ENGINE
(P/N C81080180 AND C81080190)

TECHNICAL MANUAL
C21518008-X0

DOCUMENT C21518008
Revision X0
January 1975

california data processors

$15.00

REVISIONS

Revision	Date	Approval	Description
X0	3-75		Preliminary

The revision history of each page in this document is indicated below:

Page	Revision X0		
i	✓		
ii	✓		
iii	✓		
1-1	✓		
1-2	✓		
1-3	✓		
1-4	✓		
2-1	✓		
2-2	✓		
2-3	✓		
2-4	✓		
2-5	✓		
2-6	✓		
2-7	✓		
2-8	✓		
2-9	✓		
2-10	✓		
3-1	✓		
3-2	✓		
3-3	✓		
3-4	✓		
4-1	✓		
4-2	✓		
4-3	✓		
4-4	✓		
4-5	✓		
4-6	✓		
4-7	✓		
4-8	✓		
4-9	✓		
4-10	✓		
4-11	✓		
4-12	✓		
4-13	✓		

Page	Revision X0		
4-14	✓		
4-15	✓		
5-1	✓		
5-2	✓		
5-3	✓		
5-4	✓		
5-5	✓		
5-6	✓		
5-7	✓		
5-8	✓		
5-9	✓		
5-10	✓		
5-11	✓		
5-12	✓		
5-13	✓		
5-14	✓		
5-15	✓		
5-16	✓		
5-17	✓		
5-18	✓		
5-19	✓		
5-20	✓		
5-21	✓		
5-22	✓		
5-23	✓		
5-24	✓		
5-25	✓		
5-26	✓		
5-27	✓		
5-28	✓		
5-29	✓		
5-30	✓		
5-31	✓		
5-32	✓		

Page	Revision X0		
5-33	✓		
5-34	✓		
5-35	✓		
5-36	✓		
5-37	✓		
5-38	✓		
5-39	✓		
5-40	✓		
5-41	✓		
5-42	✓		
5-43	✓		
5-44	✓		
5-45	✓		
5-46	✓		
5-47	✓		
6-1	✓		
A-1	✓		
A-2	✓		
A-3	✓		
B-1	✓		
C-1	✓		
C-2	✓		
C-3	✓		
C-4	✓		
C-5	✓		
C-6	✓		
C-7	✓		
C-8	✓		

CONTENTS

SECTION 5: MICROCOMMANDS

SECTION 6: MAINTENANCE

APPENDICES

APPENDIX A: ENGINE ARITHMETIC — Page

APPENDIX B: FIXED MEMORY ASSIGNMENTS

APPENDIX C: CONNECTOR PIN ASSIGNMENTS

TABLES

ILLUSTRATIONS

SECTION 1
INTRODUCTION

1.1 SCOPE

This manual provides the information needed to understand and maintain the Cal Data 100 Engine, part numbers C81080180 and C81080190, when used with the drawing package provided. The information in this manual is for the use of a skilled technician familiar with standard test equipment, solid-state logic theory, common maintenance practices and standard troubleshooting techniques. A basic knowledge of design principles and circuits used in small computers is assumed, hence no tutorial material of this kind is included.

As a stand-alone publication, this manual has a good functional and physical description of the Cal Data 100 Engine, providing the information needed to understand the capabilities and features of the computer and to plan a system using it. The maintenance coverage of this manual is commensurate with the prerequisite skills and knowledge of the defined user, characteristics of the product and maintainability requirements established by Cal Data.

1.2 DOCUMENTATION

This manual describes the engine of a Cal Data computer system that is equipped with a MACROBUS Channel Adapter (part number C81080300) and an Emulate Board (part number C81080210).

The following paragraphs define publications and conventions that support this manual.

1.2.1 Publications

Figure 1-1 illustrates the relationship between Cal Data system elements and technical publications. Controlled copies of publications, provided in accordance with the terms of the purchase contract, are kept current for the life of the product.

1.2.2 Engineering Drawings

For maintenance purposes, this manual is supported by a drawing package that contains schematic diagrams, assembly drawings and other required engineering drawings. The drawing package is updated with the latest revision of each drawing.

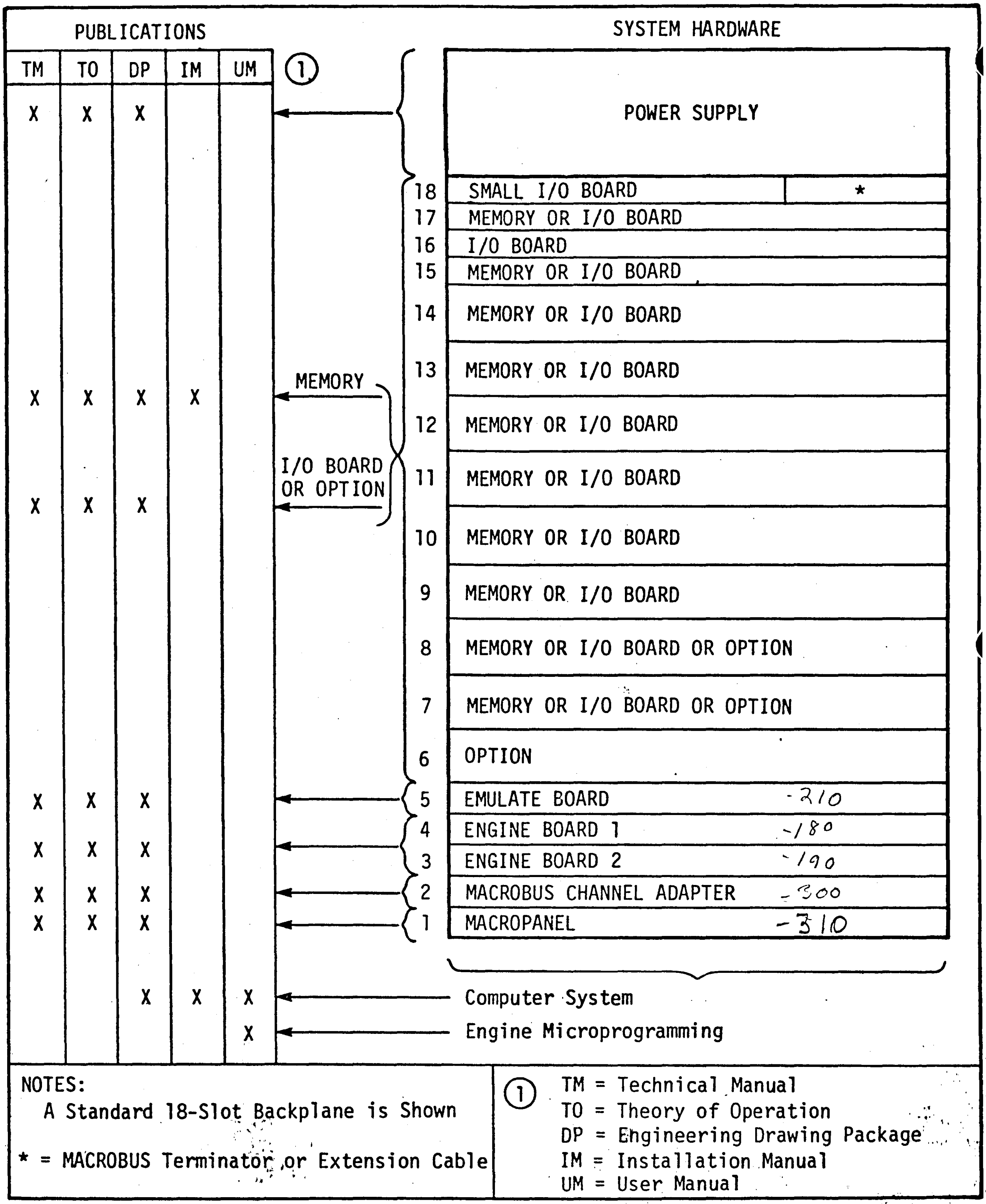


Figure 1-1. Relationship of Publications to Cal Data 1 System Elements

1.2.3 Abbreviations and Conventions

Table 1-1 lists the abbreviations found in this manual. Conventions used in the text of this manual include:

a. Equipment panel nomenclature is reproduced in all upper-case characters.
b. The proper names of instructions, microcommands and signals are capitalized.
c. ZERO and ONE are used to express binary logic "0" and "1" states, respectively.
d. Hexadecimal numbers are preceded by a dollar sign for easy identification.
e. A colon is used to indicate a range of bits. For example, the range of address bits A12 to A03 is written A12:A03.

Table 1-1. Abbreviations

Abbreviation	Meaning
Cal Data	California Data Processors
CPU	central processing unit (engine)
MCA	MACROBUS Channel Adapter
I/O	input/output
LFC	Line-Frequency Clock
RAM	random-access memory
ROM	read-only memory
PROM	programmable read-only memory
MSI	medium-scale integration
LSI	large-scale integration
MMU	Memory Management Unit
LED	light emitting diode
ACM	Alterable Control Memory
DMA	direct memory access
CM	control memory
CC	microcommand location counter
CR	microcommand register
CS	control stack
SC	stack counter
LC	loop counter
MB	M bus (data destination bus)*
FR	file register
AB	A-operand bus*
BB	B-operand bus*
AU	arithmetic/logic unit
SX	AU shift elements
PS	processor (macro)status register
LR	stack-limit register
RR	data-read register
IR	instruction register
XR	shift register
ER	emulate decode register
EIA	emulate instruction address
c	carry out microcondition code
v	overflow microcondition code
z	zero data-value (microcondition) code
n	negative data-value (microcondition) code
p	positive data-value (microcondition) code
d	odd data-value (microcondition) code

Abbreviation	Meaning
MS	microstatus register
L	MS register link bit
V	MS register overflow bit
Z	MS register zero data-value bit
N	MS register negative data-value bit
P	MS register positive data-value bit
D	MS register odd data-value bit
cps	characters per second
cpm	cards per minute
lpm	lines per minute
K	1,024(address or memory locations)
max	maximum
min	minimum
A	ampere
ac	alternating current
dc	direct current
rms	root-mean-square
V	volt
ns	nanosecond
Hz	hertz
°C	degrees celsius
cm	centimeter

* = part of the main Microbus

SECTION 2
DESCRIPTION

2.1 OVERVIEW

The Cal Data 1 Computer (Figure 2-1) is a high-speed microprogrammed digital computer designed for application in a wide variety of computing and control applications. Microprogramming, combined with a powerful and flexible hardware architecture, centering around the Cal Data 100 Engine and Microbus, permits the basic computer to be fully optimized to a specific application. The Cal Data 100 Engine is designed primarily for efficient, high-speed emulation of general-purpose computer architectures. It can also be applied as a direct function processor by implementation of problem-oriented microprograms.

2.2 SYSTEM ORGANIZATION

The overall system organization is shown in Figure 2-2. The system consists of a set of hardware and software elements that can be utilized in a wide variety of applications. A brief description of the elements of the computer system is given below. Details are given in other sections of this manual and in supporting manuals.

2.2.1 Engine

The central element of the system is the Engine (CPU), divided into control and data sections, and controlled by microprogram sequences (firmware) stored in a control memory. By changing the contents of control memory, the entire operation of the system can be altered. An emulation system is implemented by placing appropriate firmware in control memory, causing the CPU to operate like the computer being emulated.

The control and data sections contain the internal arithmetic/logic circuits, data paths, registers, control logic and timing circuitry of the machine. The CPU communicates with the rest of the system via the Microbus.

2.2.2 Microbus

The Microbus is a universal bus that is the main communication and control channel of the system. The Microbus transmits data and control information between the CPU and all elements of the system.

The Microbus can be conditioned by one or more I/O channel adapters to interface with a wide variety of I/O devices obeying specific interface rules. The primary I/O channel adapter of the Cal Data 1 system is the Cal Data 1 MACROBUS Channel Adapter.

2.2.3 MACROBUS Channel Adapter

The MACROBUS Channel Adapter (MCA) provides data, address and control circuitry for parallel I/O operations in the system. The MCA frees the

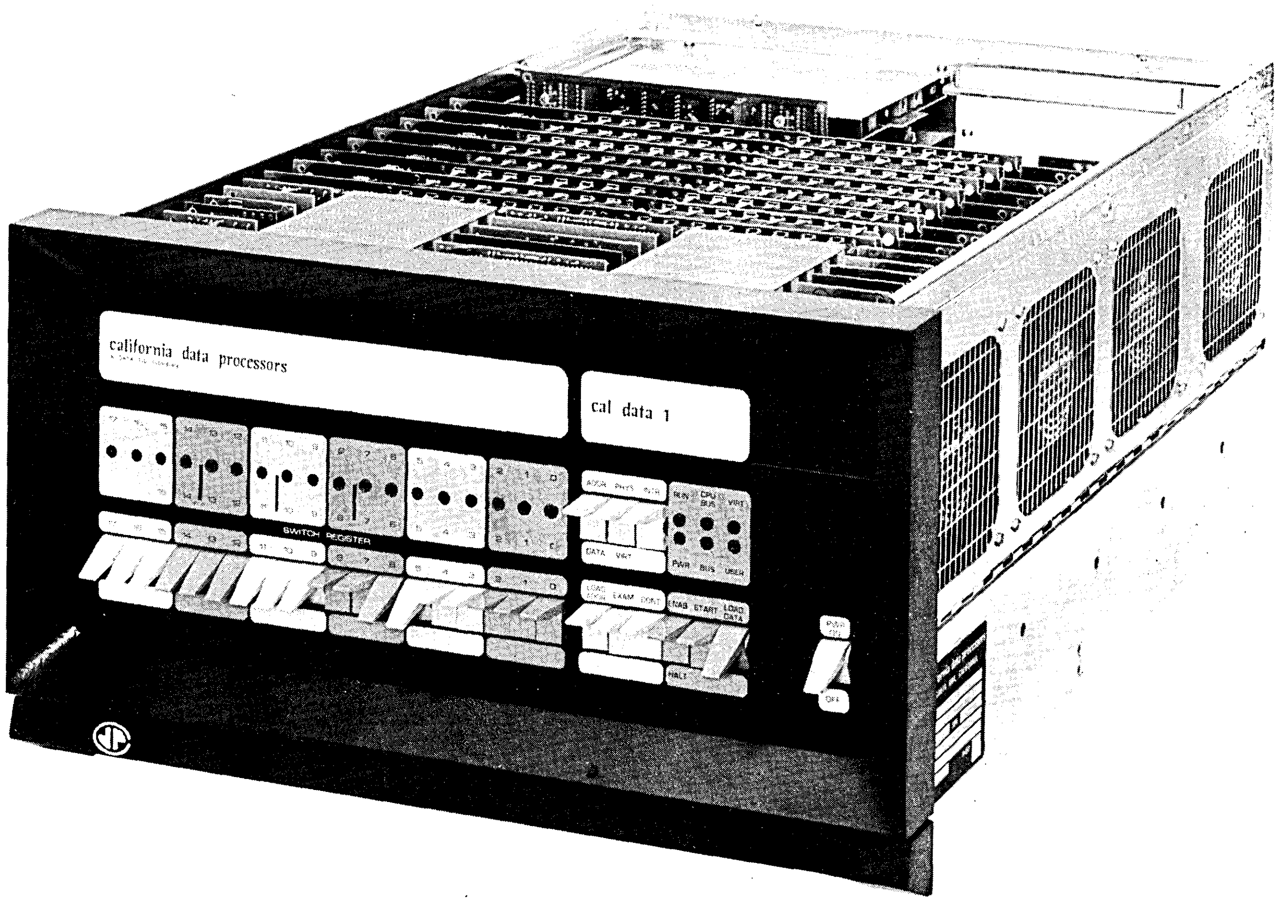


Figure 2-1. Cal Data 1 Computer System with Memory Management Unit, 128K Words of Cal Data 16KX16 (850-ns) Core Memory and Serial I/O Controller

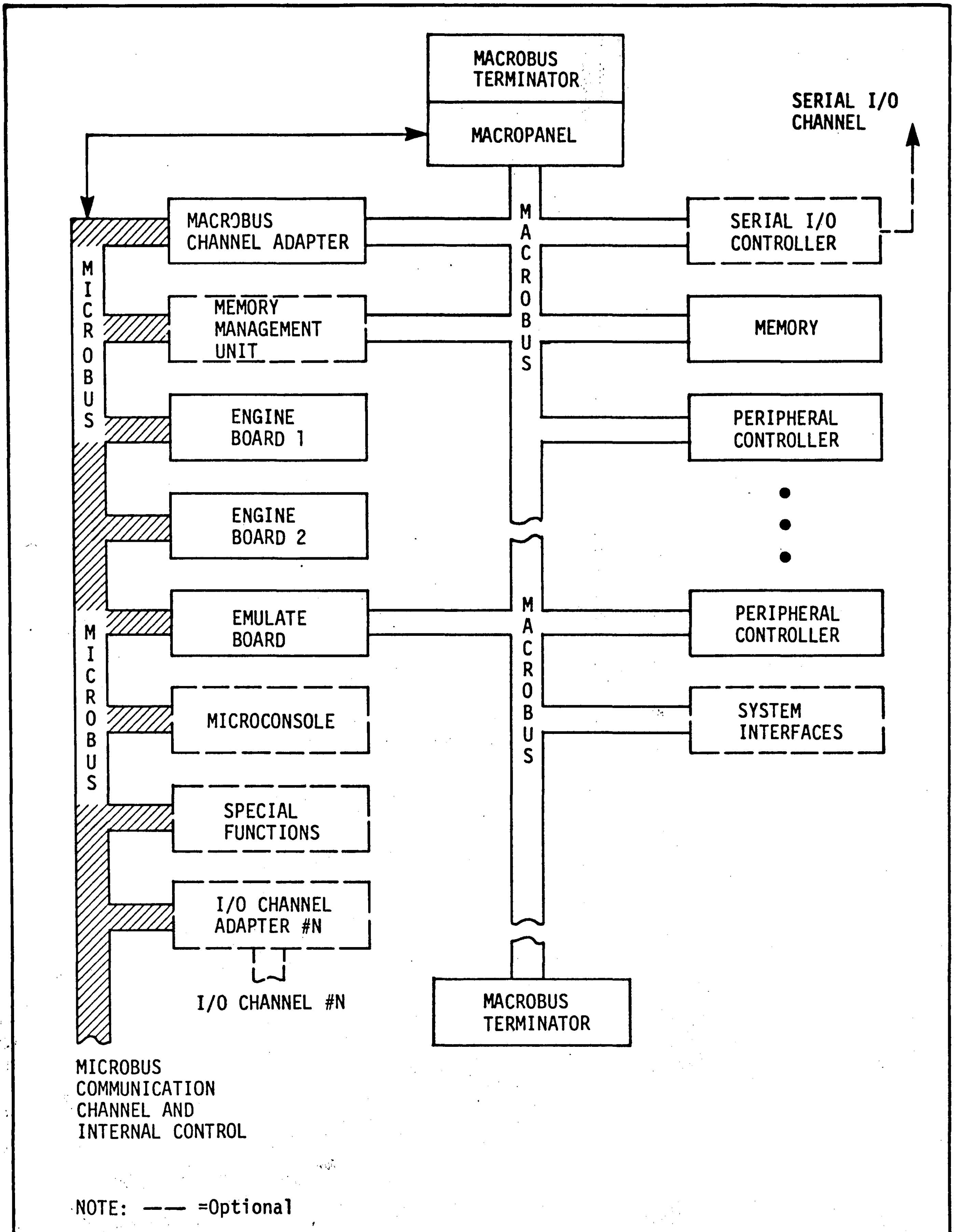


Figure 2-2. Cal Data 1 Computer System Organization

central Microbus for very-high-speed communication between the CPU and other Microbus devices, and can permit I/O channel devices to communicate directly with each other, independently of the CPU.

2.2.4 Macropanel

A Macropanel, representing the control panel of a general-purpose computer, is often provided in an emulation application. The Macropanel is serviced by the CPU as an I/O device interfacing with the MACROBUS. Special support firmware is provided for the Macropanel. The primary Macropanel for the Cal Data 1 system is the Cal Data 1 Macropanel.

2.2.5 Microconsole

A Microconsole is available to provide microlevel control and display for checking out and debugging firmware, and also for various maintenance and troubleshooting procedures. The Microconsole consists of a remotely mounted Micropanel and a plug-in Micropanel control board that permits the user to exercise direct control over the CPU. Facilities are provided to construct full microcommands, to display microcommands and to execute microcommands on a single-step or "trap-mode" basis. The Microconsole also contains 32 words of alterable control memory that can substitute for equivalent blocks of CPU control memory.

The Microconsole can be used in conjunction with the Macropanel and is useful for initial debugging of new firmware as well as for on-line troubleshooting of computer hardware, but is usually not required in an applied system configuration.

2.2.6 Magnetic Core Memory

Cal Data core memory comprises modular blocks of 8K (8,192) or 16K 16-bit words, each contained on a single circuit board. Each module plugs directly into the MACROBUS and is treated as an I/O device in the system. The maximum normal system capacity is 128K words. Two identical modules can be interleaved to achieve an increased effective throughput rate on the MACROBUS.

The MACROBUS can accommodate memory devices other than magnetic core, such as semiconductor ROM or RAM modules. The only requirement is that such units obey MACROBUS use rules. Modules of varying size and speed can be freely mixed with core memory. DMA-type MACROBUS devices may communicate directly with memory.

2.2.7 Peripheral Devices

Peripheral device controllers and system interfaces are attached to the MACROBUS as shown in Figure 2-2. The user can readily interface devices with the MACROBUS using simple design rules. Cal Data offers I/O channels such as the MACROBUS with different structures as well as several standard peripheral subsystems to enhance user applications. The subsystems offered to support normal programming and system development operations are:

a. Paper Tape Reader. High-speed photoelectric reader, 300 characters per second, fanfold tape.
b. Paper Tape Punch. High-speed punch, 75 characters per second, fanfold tape.
c. Card Reader. High-speed photoelectric card reader, 300 cards per minute with code conversion in the controller.
d. Line Printer. 80- or 132-column printer, 125 or 200 lines per minute.
e. Memory extensions.

2.3 FIRMWARE DEVELOPMENT AIDS

Cal Data offers specialized hardware and software elements to aid users in developing custom firmware. These are briefly described below.

2.3.1 Alterable Control Memory

Alterable Control Memory (ACM) is a modular plug-in unit that contains increments of 256 words of electrically alterable control memory. The ACM also contains alterable elements associated with instruction emulation and decoding.

With the ACM, a programmer can load or read the contents of control memory directly and execute trial firmware code at normal processor execution speeds. The ACM is particularly useful for dynamic system tests where external real-time events must be considered to fully evaluate a firmware microprogram. The ACM is supported by a software operating system that permits the programmer to use a teleprinter to control the system.

2.3.2 Support Software

The following software is available to support firmware development:

a. Symbolic Microassembler. This program is a complete symbolic assembler that permits convenient coding and listing of microprograms. It is written in Cal Data 135 emulator language and can be run on any Cal Data 135 or compatible computer having the required memory configuration.
b. ACM Software Operating System. This program is designed to provide operational control over execution of firmware in the ACM. It requires that the Cal Data 135 emulator be resident in control memory.

2.4 FEATURES

The Cal Data computer architecture combines general microprogramming capability with specialized optional features to permit high emulation speeds with efficient control-memory space utilization. The mechanical design used provides full modularity, mounting flexibility and service convenience. Cooling, power distribution and other critical system requirements are optimized for OEM applications. Conservative electrical implementation ensures wide margins, readily available components and reliable operation over a wide environmental range. Subassemblies are designed for easy assembly and automated testing, and the overall system is structured for simple, straightforward manufacturing procedures. Basic design features of the Cal Data computer system are:

- 48-bit microcommand word length
- Parallel execution of multiple functions per microcommand
- 165-ns microcommand execution time
- 16-bit data word length
- 16 multipurpose file registers (16 bits each)
- Nine additional registers accessible by microcommand
- 16-level hardware pushdown stack
- Microcommand sequence repeat loop counter
- Optional high-speed emulation instruction decode, function generation and interrupt-response hardware.
- Bit, byte and word manipulations
- 256- to 4096-word control memory using bipolar ROM or PROM devices
- Power-failure/restart circuitry and line-frequency clock included in the computer
- Unique, control memory substitution provisions
- Optional Multiply, Divide, and single- and double-precision Shift microcommands
- Hardware microprogram interrupts

Input/Output and Memory

- Universal asynchronous I/O channel with direct-memory-access capability
- Four external priority interrupt levels
- 16-bit parallel word or byte-mode transfers
- Automatic I/O channel delay time-out protection
- Optional asynchronous serial I/O channel
- 8K-word (675-ns cycle, 275-ns access) and 16K-word (850-ns cycle, 300-ns access) core memory modules
- Interleaved data transfers between identical memory modules
- Optional extended addressing feature for addressable memory expansion to 31K without memory management
- Expansion to 124K or 127K of directly addressable memory with optional Memory Management Unit

Microprogramming Aids

- Microconsole
- Alterable Control Memory and support software
- Symbolic Microassembler

Packaging, Power and Environmental

- 10½ inch computer chassis with vertical board mounting from the top
- Printed-circuit backplane with up to 13 spare slots for memory and I/O controller boards
- Four fans for high-volume, positive-pressure air flow through the chassis with provision for air filters
- Modular power supply providing 36 A at +5 Vdc

- Low-noise internal power distribution and grounding system
- Convenient external I/O cabling
- Extension chassis available
- System designed to meet UL standards
- 0 to +50°C ambient operating temperature
- 10 to 90% relative humidity (without condensation)

Electrical and Electronic

- Bipolar TTL integrated circuits (multisourced)
- Extensive use of MSI and LSI
- Wide timing margins
- High noise immunity I/O drivers and receivers
- Single-phase clock
- Conservative component derating
- Metal can transistors and hermetically-sealed passive devices only

2.5 SPECIFICATIONS

General specifications for the Cal Data 1 Computer are given in Table 2-1.

Table 2-1. Cal Data 1 Computer Specifications

Characteristic	Specification
TYPE	High-speed microprogrammed digital computer designed for efficient emulation of general-purpose computer architectures and for direct custom applications
CONTROL	
Microcommand length	48 bits
Execution rate	165 ns, min.; 330 ns if skip or branch is made; clock rate is adjustable
Microcommand classes	8 arithmetic 16 logical 8 special
Special operations	Special microcommands include double-precision Shift, Multiply-Step and Divide-Step
Conditional skip/branch	Each microcommand with conditional skip or branch capability; Tests on either current (dynamic) conditions or on previous (static) conditions
Fixed control memory	Bipolar ROM or PROM; 4,096 words, max
Alterable control memory	Bipolar RAM; 512 words max without auxiliary power; 1,536 words max with auxiliary power.

Table 2-1. (Continued)

Characteristic	Specification
Control memory stack	16-level hardware pushdown stack
Emulation enhancement	Special emulation decode tables provide automatic addresses to control memory microroutines for high-speed program execution
Loop counter	Eight-bit counter for single or multi-instruction repeats
Interrupts	Multilevel priority-interrupt structure provides automatic addresses to control memory microroutines for internal and external conditions
PROCESSING	
Word length	16 bits
Arithmetic/logic	Both word and byte operations are provided; fixed point, one's or two's complement arithmetic; arithmetic condition codes are carry (link), overflow, negative, zero, positive, odd; arithmetic and logical shifts (multibit using loop counter for repeats are provided)
Registers	Eight or sixteen 16-bit multipurpose files (FR) Shift register (XR) Microstatus register (MS) Instruction register (IR)* Decode Register (ER)* Processor (macrolevel) status register (PS)*
INPUT/OUTPUT (TYPICAL)	
Type	Asynchronous bidirectional I/O channel derived from the Microbus; requires I/O channel adapter; handles communications between CPU, memory and peripheral elements
Data	16 bits with byte capability
Address	16 bits from Microbus (can be extended within I/O channel adapter); least-significant bit is for byte addressing

*Part of emulation enhancement circuitry

Table 2-1. (Continued)

Characteristic	Specification
I/O channel priorities and requests	Four priority-request levels with multiple requests per level; nonprocessor request (NPR) level for direct device-to-device transfers; CPU can set its own priority to any level except NPR
Serial I/O channel	Serial I/O controller (option) for rates up to 9600 baud; RS-232 or current-loop interface
Memory	Magnetic core; 8K or 16K words per module; 16 bits per word
Memory expansion	Typically, 124K words maximum; Memory Management Unit (option) is required above 32K
Memory interleave	8K-word or 16K-word Cal Data core memory pairs can be interleaved for increased throughput rate
Line-frequency clock	50/60 Hz line clock
PACKAGING	
Processor chassis	10½ inches (26.7 cm) high by 19 inches (48 cm) wide by 24 inches (43 cm) deep; rack-mounted (slides) or table-top; vertical, top-loaded boards; contains Macropanel, Engine, MCA plus slots for memory and I/O controllers; internal power supply; cooling fans; internal power distribution
Connectors	36-pin, 0.6 inch (1.5 cm) card insertion depth; mounted on printed-circuit backplane
Board size	8.9 by 15.7 inches (22.7 by 39.9 cm); six connector positions (216 pins) on long edge
POWER	
AC input	115/208/230 Vac, 50 or 60 Hz
DC outputs	Regulated: +5 Vdc, 36 A -15 Vdc, 12 A

Table 2-1. (Continued)

Characteristic	Specification
	Unregulated: -22 Vdc, 1.5 A +8 Vrms, 1.5 A
Power monitor	Power-failure/restart signals to CPU for automatic shutdown and restart operations
ENVIRONMENT	
Temperature	0° to $+50^{\circ}$C ambient temperature
Humidity	10 to 90% relative, without condensation
CIRCUITS	
Integrated circuits	Bipolar TTL; extensive MSI and LSI usage
Discrete devices	Metal-can transistors; hermetically sealed components only
Internal logic levels	ZERO = 0 Vdc; ONE = +5 Vdc, nominal
I/O logic levels	ZERO = +3.4 Vdc, nominal; ONE = 0 Vdc
MICROPROGRAMMING SUPPORT HARDWARE	
Microconsole	Provides direct control over Engine; microcommand entry and display; single-step and trap-mode microcommand execution
Alterable control memory (ACM)	Modular 256-word increments of control memory that can be loaded and read; operates CPU at full execution speed
MICROPROGRAMMING SUPPORT SOFTWARE/ FIRMWARE	
Symbolic micro-assembler	Symbolic assembler for microprogram coding and documentation
ACM software operating system	Operating system used in conjunction with ACM

SECTION 3
PHYSICAL DESCRIPTION

3.1 SYSTEM HARDWARE

All Cal Data Engine and system elements are modular and can be mounted in a standard chassis (Figure 3-1) that occupies 10.5 inches (26.7 cm) of a 19-inch (48-cm) RETMA rack. This modularity gives the user maximum flexibility in system design and configuration.

The standard computer chassis dimensions are:

10.4 inches (26.5 cm) high
19.0 inches (48.3 cm) wide
24.0 inches (61.0 cm) deep

Hardware items included with the standard computer chassis are:

a. Chassis box with backplane
b. Top and bottom covers
c. Hinged fan panel and four fans
d. Chassis slides
e. Macropanel bezel and overlay

A power supply mounts at the rear of the chassis. The ac power cord exits from a control panel accessible at the rear of the chassis. This panel also has the ac line switch, fuses, convenience outlet (115 Vac model only) and Macropanel lock switch.

The four fans provide horizontal, positive-pressure air flow across the vertical computer boards and power supply. The fan panel is hinged to permit moving the fans when boards are removed or installed.

System electronics are mounted on modular printed-circuit boards that insert vertically through the top of the chassis into connectors mounted on the backplane in the bottom of the chassis. The backplane provides printed-circuit (and wire-wrap) connections between all boards.

Device controller cables are generally connected at the top edge of I/O boards by means of flat cable. These cables are routed over the top of the boards and exit via a cutout at the top rear of the chassis. A strain-relief clamp is provided. All standard Cal Data I/O and memory boards have provision for this cable routing scheme. The backplane contains up to 18 connector rows (board slots).

The Macropanel is mounted on a printed circuit board that plugs into the first connector row of the backplane. The Macropanel is covered by an overlay held in place by the bezel. The bezel and overlay are removable from the front when the chassis is installed in a rack.

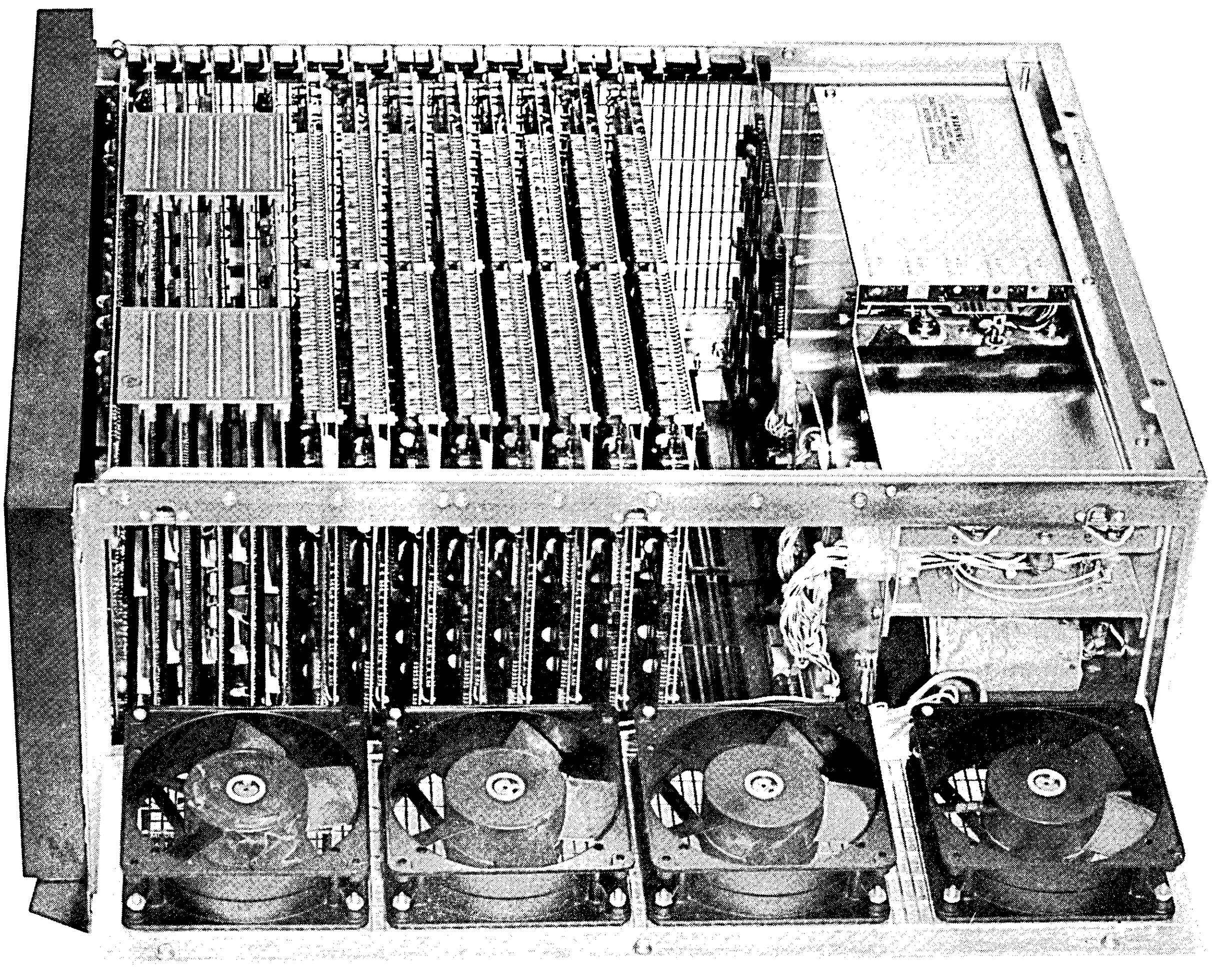


Figure 3-1. Cal Data 1 Computer with Boards Installed (Fan Panel is Shown Down)

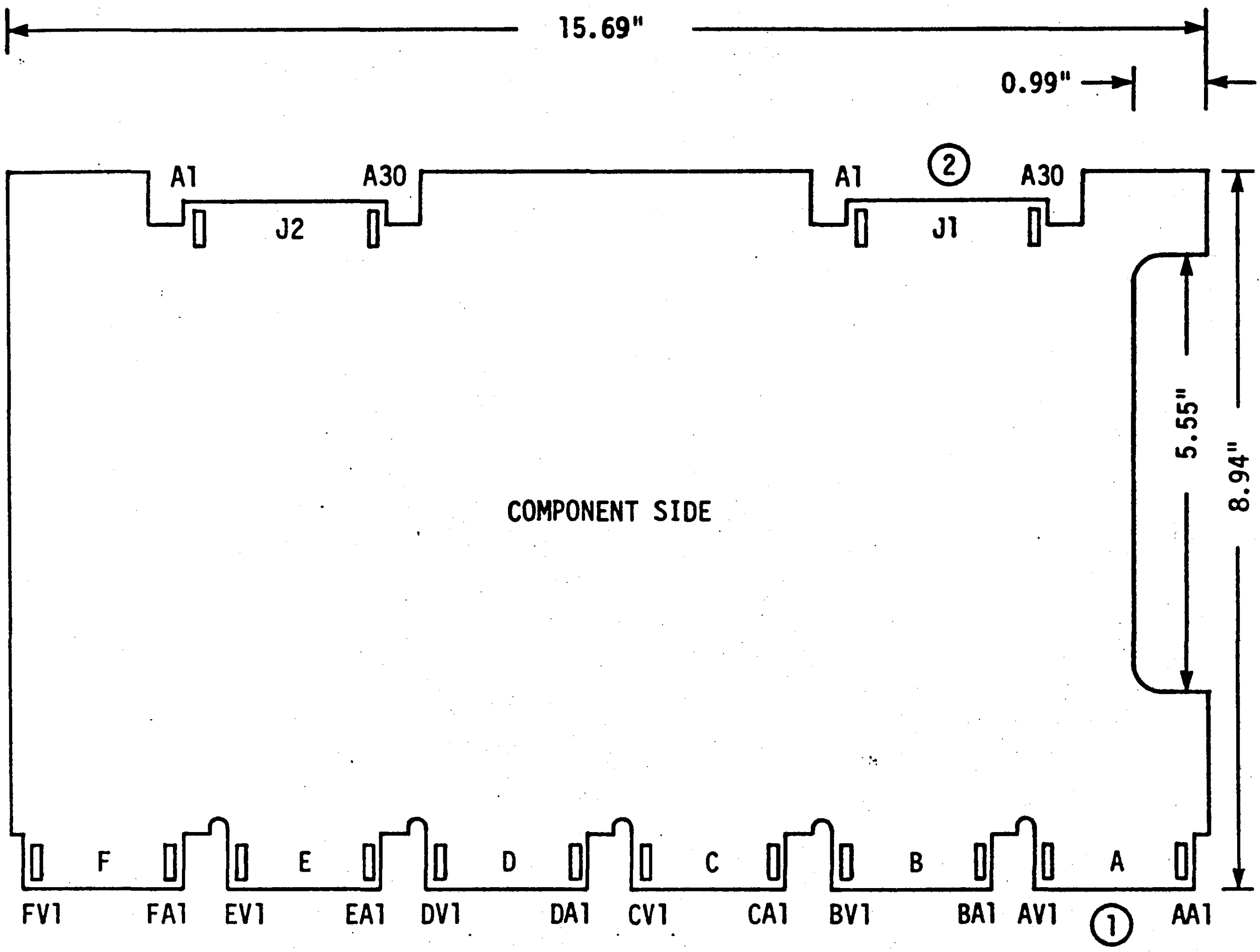
3.2 ENGINE BOARDS

The Engine comprises two boards labeled Engine 1 (part number C81080180) and Engine 2 (part number C81080190). Each Engine board (Figure 3-2) is a hex-width board 15.7 by 8.9 inches (33.9 by 22.7 cm). Engine 1 normally plugs into slot 4 of the Cal Data computer chassis.* Engine 2 normally plugs into slot 3. The right-hand edge of each board has a 1.0 by 5.5 inch (2.5 by 14.0 cm) cutout as clearance for the side-mounted cooling fans in the chassis.

There are six printed-circuit connectors (A to F) on the bottom edge of each board, and two (J1 and J2) on the top edge. Connectors A and B interface with the MACROBUS. Connectors C to F, and J1 and J2 interface with the main computer Microbus. Connectors A to F are standard backplane connectors. Connectors J1 and J2 plug into the two small processor-interconnection boards.

There are no controls or adjustable elements on the Engine.

*Because of the universal connections in the CPU area of the chassis, the Engine boards can operate in any slot from 1 to 5.



(2) Solder Side = B1 to B30. Likewise with connector J2.

(1) Solder Side = AV2 to AA2. Likewise with connectors B to F.

NOTES:

Figure 3-2. Cal Data 100 Engine Board Configuration

SECTION 4
ENGINE

4.1 FUNCTIONAL DESCRIPTION

Figure 4-1 is a block diagram of the Cal Data 100 Engine, showing three main functional sections: control, data and MCA. The control section contains the control memory, emulation enhancement circuitry (if needed) and timing circuits that control the sequence of operations performed. Emulation enhancement circuitry is provided only when a computer configuration requires the speed or special capabilities of the added circuitry. The data section contains the arithmetic/logic, gating and busing elements that perform data transfers and manipulations. The basic control and data sections together are referred to as the Engine or CPU. The main communication path in the computer is the Microbus, used for parallel transfers of information and control signals between the CPU and all functional system elements. The microbus comprises the A-operand bus (AB), the B-operand bus (BB), the M bus (MB) and other lines (Appendix A). The Engine and all external devices, including memory, Macropanel and peripherals communicate with the Microbus. The relationship of the Microbus and Engine logic is illsutrated in Figure 4-2. Certain Microbus functions can be performed by the MCA for common I/O devices, allowing the Microbus to attend to higher-speed units. Devices on the MACROBUS can communicate with the CPU and directly with other devices, depending on their design. The MCA is shown in Figure 4-1 because of its important function of conditioning the Microbus for use by the mass of common peripheral devices.

A basic MACROBUS device is the magnetic core memory, which is generally required in any system. Cal Data core memory modules are available in 8K- and 16K-word increments and can be added directly to the MACROBUS up to a typical maximum of 128K words*.

Semiconductor memory can be interchanged with core in any speed/capacity mix. The CPU addresses memory locations like any other I/O devices.

Two types of control panels are available: a Macropanel that is adapted to a particular emulation and permits the operator to control the system at the emulated level of operation, and a Microconsole that permits control and display at the microlevel and is useful for firmware development, hardware maintenance and troubleshooting. The Macropanel is treated as an I/O device. Special interpretive firmware services the functions of the Macropanel.

*Maximum memory capacity of the basic system is 32K words. A Cal Data Memory Management Unit is required for expansion beyond this capacity.

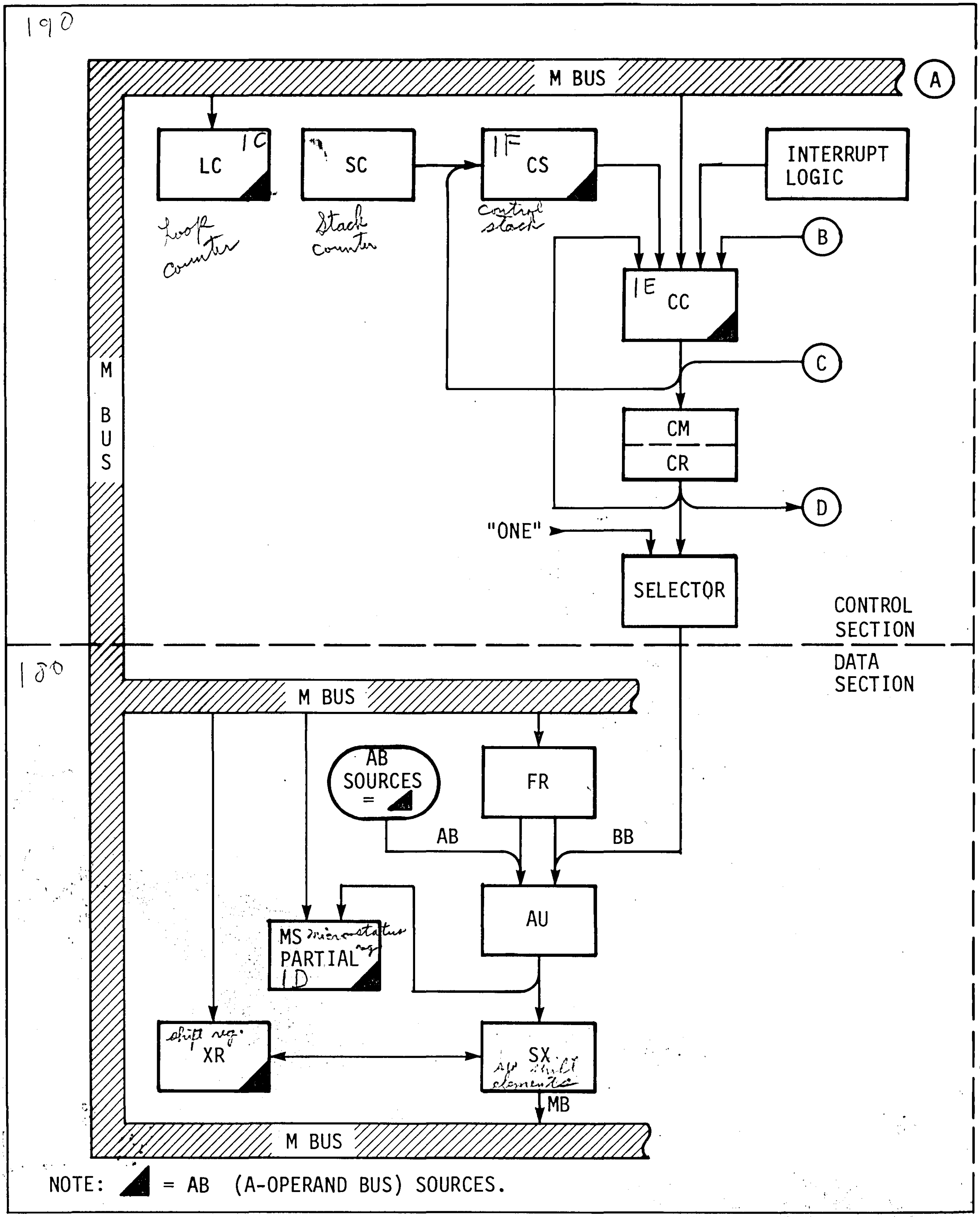


Figure 4-1. Cal Data 100 Engine Block Diagram (continued)

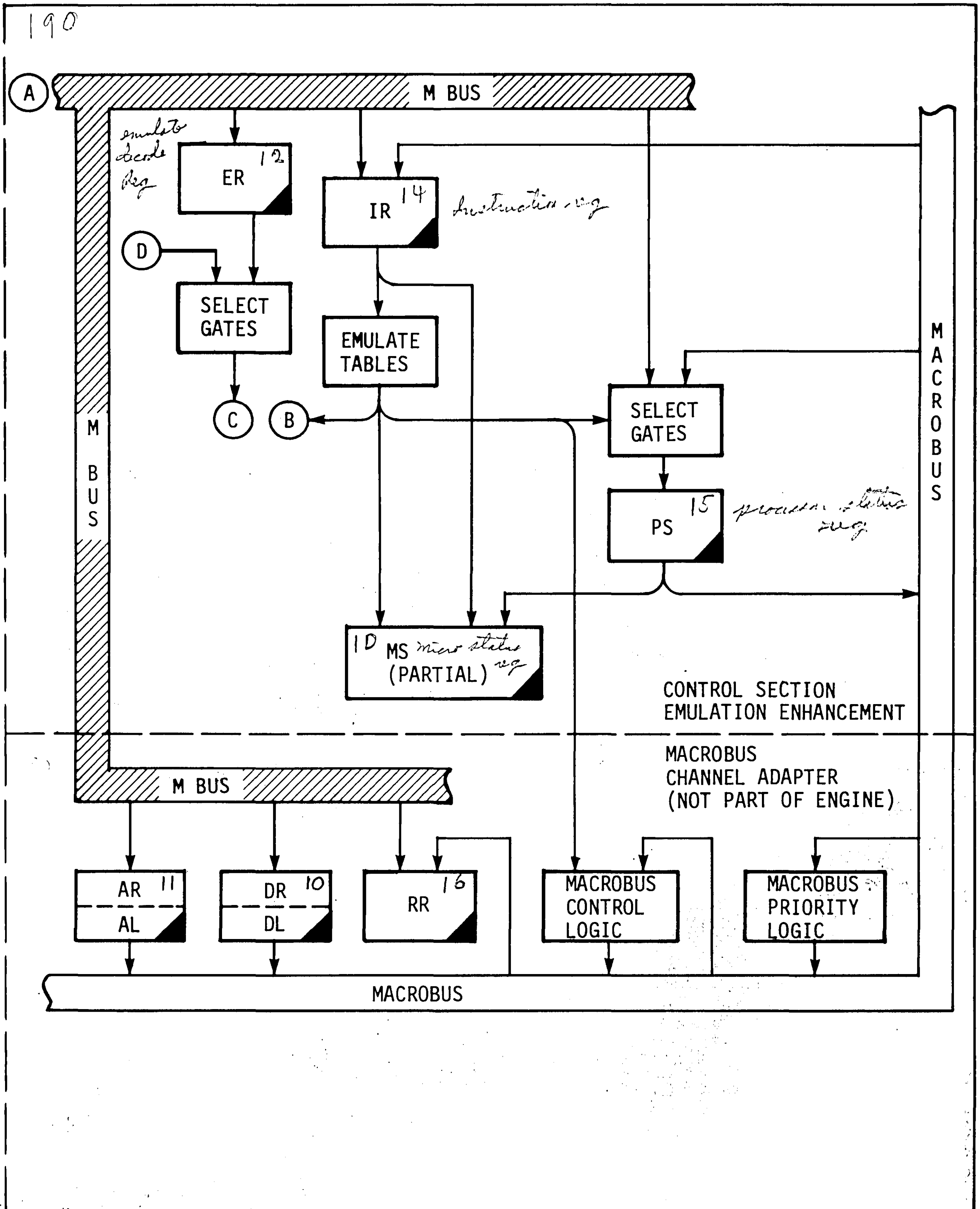


Figure 4-1. (Continued)

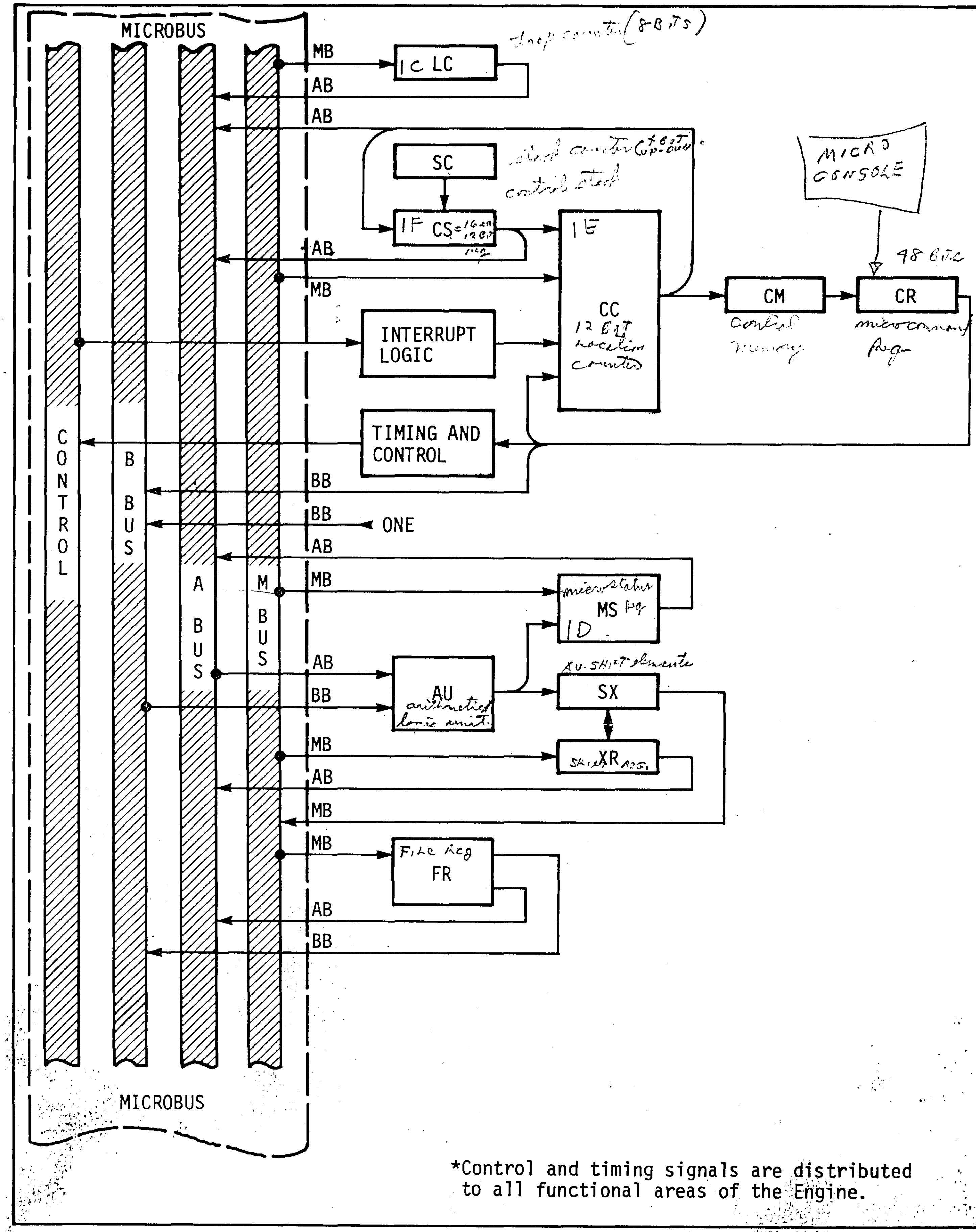


Figure 4-2. Cal Data 100 Engine Interface with the Microbus

4.2 CONTROL SECTION

A block diagram of the control section is shown in Figure 4-3. Control is organized around the control memory (CM), which stores the microprograms to be executed. Microcommands are 48 bits in length. Normal CM capacity is from 256 to 4,096 words (48 bits each).

A 12-bit location counter (CC) addresses CM and advances on each clock step unless altered by a sequence change. Microcommands read from CM are held in a microcommand register (CR) during execution. The microcommands read from CM can be modified prior to input to CR for execution. Microcommands can also be entered manually into CR and executed from the Microconsole (not shown).

A 16-level control stack (CS) is provided to permit the contents of CC to be saved and restored under microprogram control. This permits automatic nesting of microroutines and microprogram interrupts, giving increased speed and CM space efficiency. The system contains a unique facility that permits designated areas of CM to be "patched" from auxiliary CM or from the Microconsole. This is a highly useful feature, since nonalterable storage elements are generally used to implement CM.

An eight-bit loop counter (LC) is provided to permit single microcommands or entire sequences to be repeated a specified number of times. This feature enhances execution speed of iterative loops.

A special feature of the Cal Data 100 Engine is emulation enhancement circuitry, located on a separate Emulate Board. This circuitry provides:

a. Automatic table-generated addresses to CC to steer the microprogram directly to specific emulation microroutines, by-passing lengthy processing to decode instruction codes and addressing modes
b. Automatic interrupt microroutine location entry to CC
c. Automatic table-generated modifiers to microcommands read from CM
d. Automatic modification of processor status conditions for the emulated instruction
e. Direct designation of word or byte-mode operations

Emulation-related features are described in a separate emulation user manual, available for each computer model.

4.2.1 Control Memory (CM)

The control memory is a high-speed, random-access unit. Three device implementations can be used:

a. Read-only memory (ROM). These bipolar semiconductor devices are organized on chips of four by 256 (or four by 512) bits. Twelve such devices implement each 256-word (or 512-word) CM page. The code pattern in each chip is permanently inscribed during the factory manufacturing process and cannot be altered. ROM is used for high-volume production of fully debugged firmware.

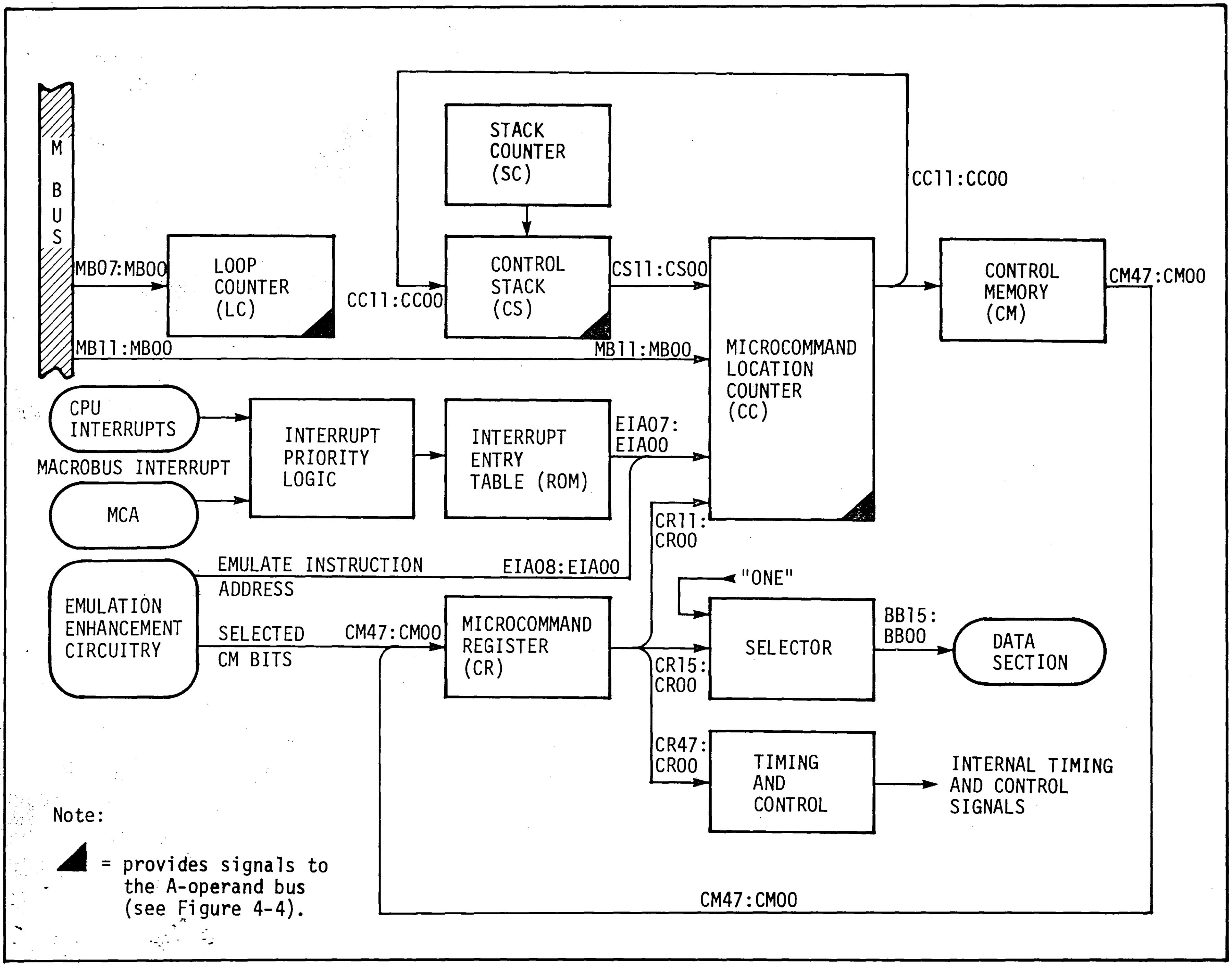


Figure 4-3. Cal Data 100 Engine Control Section Block Diagram

C21518008-X0

b. Programmable read-only memory (PROM). These bipolar semiconductor devices are organized on chips of four by 256 bits, pin- and speed-compatible with the equivalent ROM. The code pattern in each device is electrically and permanently inscribed by a portable programming device. PROM is used for development and field debugging of firmware and also for low-volume production firmware packages.

c. Alterable Control Memory (ACM). The Cal Data ACM is a complete, modular control memory that can be installed in the computer in addition to or in place of ROM and PROM devices. It is implemented with bipolar random-access memory devices that can be electrically altered (read/write). When installed in the computer, ACM can be loaded and read via the MACROBUS using I/O microcommands. The ACM can then take control of the CPU for execution of ACM firmware at real-time processor speeds. The ACM is most useful for initial and on-line checkout of new firmware prior to conversion to ROM or PROM devices.

The normal maximum capacity of CM is 4K words* when ROM or PROM devices are used. Although each microcommand is 48 bits in length, the CM addressing structure of the microcommand limits direct access to 2K words; however, a paging scheme between 2K-word blocks permits convenient access anywhere within 4K words.

Auxiliary Control Memory. It is often desirable to alter the contents of CM, either temporarily or permanently. When nonalterable devices are used, the usual requirement is replacement of the existing devices. The Cal Data 100 Engine incorporates circuitry that permits either one or two 32-word blocks of auxiliary memory in the Microconsole to functionally replace designated 32-word blocks in CM. This enables "patching" for corrections, additions or deletions from existing firmware, temporary overlay for diagnostic and troubleshooting operations, etc.

4.2.2 Location Counter (CC)

The location counter is a 12-bit binary counter/register that points to the location in CM of the next microcommand to be executed. The microprogram sequence can be altered conditionally or unconditionally as specified by the programmer and the state of the system. A sequence change is made by loading CC from one of the following sources:

a. CR for programmed branches
b. M bus for computed branches
c. The current CS register
d. A vector from the emulation enhancement circuitry
e. An interrupt vector

CC normally advances sequentially to the next location through all 4K locations in CM, including the wrap-around transition from 4,095 to 0, unless the normal sequence is altered.

*Auxiliary power is required above 512 words.

CC modifiers from CR and the emulation enhancement circuitry are 11 bits long, permitting branches to occur from these sources within only a 2K-word area. The most significant bit of CC is unaltered for such branches. To branch to a location outside a 2K-word area, the programmer must execute a microcommand that transfers a full 12-bit branch address via MB. Interrupt vectors are to only the first 256 CM locations (i.e., the four most-significant CC bits are forced to ZERO).

Certain conditions cause an automatic reset of CC to location 0 (a corresponding microstatus bit is set for each condition):

a. A catastrophic system error
b. A power-up sequence

The contents of CC can be read by microcommand via AB. For systems that do not contain an implemented CS, this provides a means of saving a return location in CM.

4.2.3 Microcommand Register (CR)

The 48-bit CR stores the current microcommand read from CM for execution. The microcommand from CM can be modified prior to entry into CR by a function specified by the special decode circuitry on the Emulate Board. CR can also be loaded from the Microconsole to permit direct operator control of internal functions. The least-significant 11 bits of CR modify CC when a branch operation is specified by the microcommand in CR.

Microcommand Sequencing and Timing. The basic clock cycle is 165 ns (adjustable) and, ordinarily, a microcommand is read from CM and executed on each cycle. There is a one-clock delay between the time CC addresses a word in CM and the time that the microcommand is transferred to CR for execution. For this reason, when the normal CC counting sequence is modified, two clock cycles are required to access the microcommand at the branch location and transfer it to CR. Furthermore, the microcommand accessed at the time CC is modified is transferred to CR even though a branch is being made. Whether or not this "extra" microcommand is executed can be specified by the programmer. The following sequence illustrates the operation:

Time	CC	CR	Operation
T-1	X	(X-1)	
T	X+1	(X)	Branch to Y specified
T+1	Y	(X+1)	Microcommand at X+1 can be executed
T+2	Y+1	(Y)	Microcommand at branch location

In addition to sequence modification, the programmer can specify that the succeeding microcommand be skipped. In this case, the succeeding microcommand is transferred to CR, but execution is inhibited. This action is not considered to be a sequence change since CC continues normal sequential counting.

The output of CR is decoded to generate the timing and control signals used throughout the computer.

Depending on the microcommand, the least-significant 16 bits of CR can be gated via BB into AU. Alternately, a literal "one" value can be placed on BB.

4.2.4 Control Stack (CS)

CS contains 16 12-bit registers that are accessed via the four-bit up/down stack counter (SC). When a CC "save" is specified by a microcommand, the contents of CC are transferred to CS. The contents of CC are always one greater than the location of the microcommand specifying the save. Likewise, a microcommand can specify a return operation that transfers the contents of the current CS location to CC. The return microcommand can simultaneously transfer the (incremented) contents of CC to the CS register that contained the return address. Incrementing and decrementing of SC can be specified independently of the save and return functions. CS permits convenient implementation of re-entrant and multilevel subroutines at the micro level. Any microcommand branch condition can specify a save operation with an automatic return to the calling sequence using a Return microcommand.

SC counts up from zero, modulo 16, and "rolls over" the boundary in either direction. There is no indication given for a stack overflow. It is the programmer's responsibility to maintain the stack within limits.

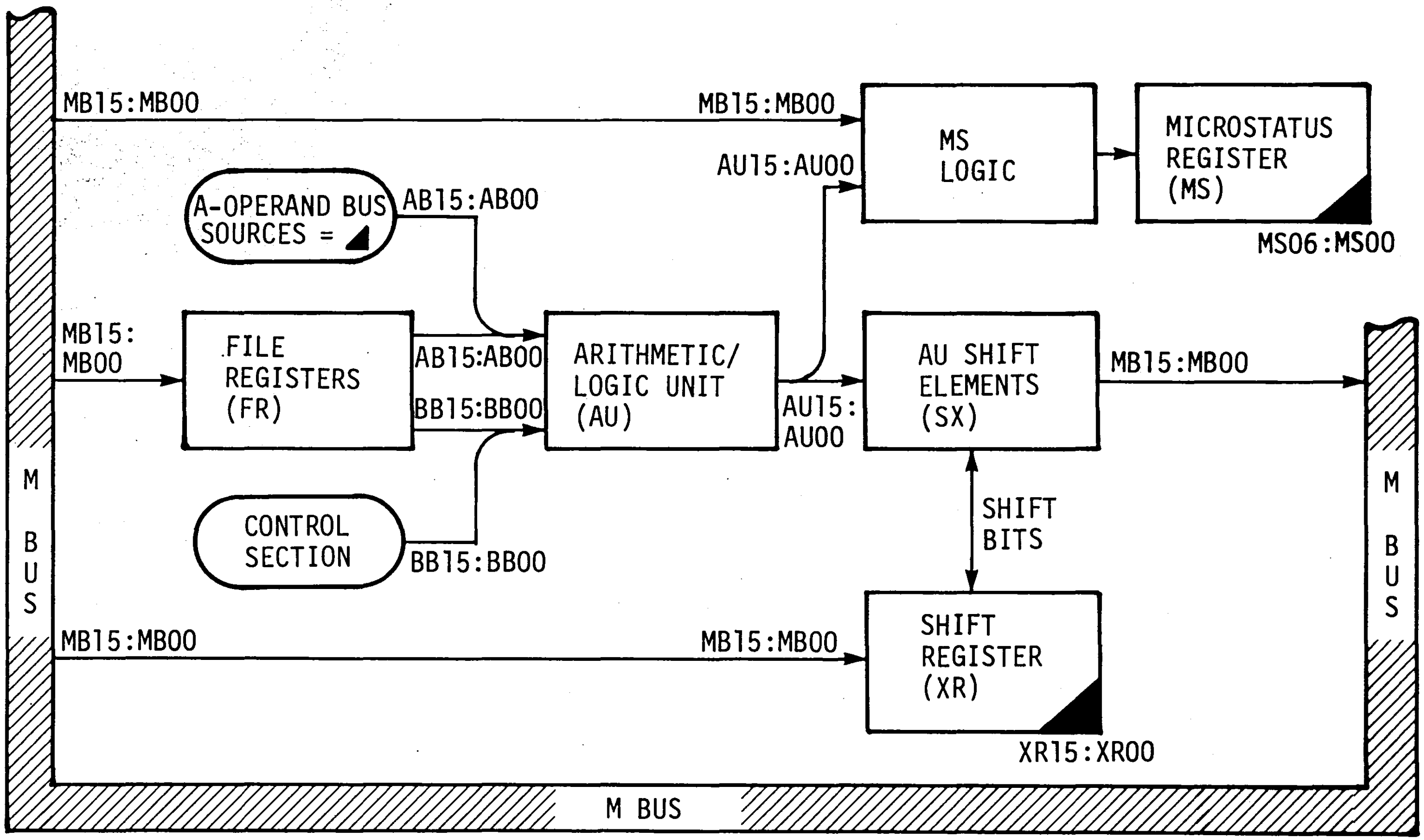
The contents of CS (current location) can be read by microcommand; however, CS cannot be directly loaded and SC is not directly accessible to the microprogram. The contents of CS, therefore, cannot be saved in the event of a power interruption. It is mandatory that provision be made to execute all returns in CS within the time available for power interruption. Since several milliseconds are available, this imposes no practical restriction on the use of the stack.

4.2.5 Loop Counter (LC)

A powerful feature of the Cal Data Engine is the eight-bit LC that permits a single microcommand or a group of microcommands to be automatically repeated up to 256 times. LC is loaded via MB and can be read with a microcommand. In a repeat sequence, LC can be tested for a zero or nonzero condition by any microcommand in the sequence, with a branch operation executed if the condition is met. LC is decremented each time it is tested. Individual microcommands can also be repeated the number of times specified by LC.

4.3 DATA SECTION

A block diagram of the data section is shown in Figure 4-4. The data section contains the basic arithmetic, logic and busing elements of the Engine required for manipulation and transfer of data throughout the computer.



*These sources are identified on all block diagrams by a shaded triangle in the lower-right corner of each block providing signals to the A-operand bus:

Figure 4-4. Cal Data 100 Engine Data Section Block Diagram

C21518008-X0

The data section utilizes 16-bit parallel data paths and operational elements. Provision is made for byte-mode operations. The general file-register (FR) structure provides either eight or 16 general-purpose registers directly addressable by each microcommand. The output of any FR can be selected as either the A- or B-operand input to the arithmetic/logic unit (AU), and the results of the operation are routed via MB to many destinations (including FR) within the Engine.

Dynamic condition codes indicating conditions of the operational results (e.g., overflow, negative, etc.) are generated for each microcommand executed. These conditions can be saved as static status bits. Either the static or the current dynamic conditions can be tested by any microcommand.

4.3.1 File Registers (FR)

FRs provide general-purpose storage within the data section. Either eight or 16 FRs (labeled FR0 to FR15) of 16 bits each can be implemented. The FRs permit the following simultaneous operations to be performed:

a. Any two FRs can be specified as the A- and B-operand sources to AU
b. The FR selected as the A-operand source can also be specified as a destination register
c. Any FR can be specified as a destination register for MB

4.3.2 Operand Buses (AB, BB)

Operands are transferred to AU via AB and BB, part of the Microbus. All microcommands executed by the CPU involve the use of information on one or both of these buses.

AB sources can be selected from any one of the FRs or from one of 11 other operational registers in the computer. There are five unused AB source addresses, of which two are reserved for user-defined functions.

The BB source can be:

a. Any one of the FRs
b. The least-significant 16 bits of the current microcommand contained in CR
c. A literal "one" value

The second BB source listed above represents a 16-bit literal value contained in the microcommand.

4.3.3 Arithmetic/Logic Unit (AU)

AU is a 16-bit parallel element that performs arithmetic (Appendix A) and logical functions on two variables input via AB and BB with the link (L) status bit from the microstatus register (MS) used conditionally as a carry input for addition and subtraction operations. A carry output (c), resulting from AU operations, can be tested as conditional skip or branch condition and can also be stored in MS (in the L bit) as a static status condition.

Each microcommand specifies, either implicitly or explicitly, the AU operation to be performed and the use of the L input. A total of 15 logical and eight arithmetic functions are implemented.

4.3.4 AU Shift Elements (SX) and Shift Register (XR)

SX is a set of gates that can be used in conjunction with shift register XR for shifting an AU operand. The following can be performed:

a. Left shift one bit
b. Right shift one bit (logical or arithmetic)
c. Swap more-significant and less-significant bytes
d. Swap more-significant and less-significant halves of the less-significant byte

For shift operations, the L bit in MS is normally used as the shift carry-in and c is the bit shifted out of SX. This carry bit can be saved as L for the next AU operation.

Provision is made for both single- and double-length shifts, either of which can be logically open, closed or arithmetic. Double-length shifts are performed in conjection with XR, which is a 16-bit shift register. In this case, the L input and c output are dependent on the direction of the shift. For left shifts, SX holds the more-significant 16-bit word. For right shifts, XR holds the more-significant word.

Shifts are performed by using shift operation codes in microcommands. Because the A operand is always used in the shift, AU performs a "copy" AB operation. Shift microcommands must specify the type of shift to be performed and the carry input function.

Multibit shifts can be performed by the use of LC by setting up a shift count and repeating the microcommand. This permits execution of shifts of all types to be performed in one clock step per bit shifted.

4.3.5 M Bus (MB)

MB, a part of the Microbus, receives the resultant output from an AU or shift operation and provides the transfer path to all internal computer destinations. Each microcommand specifies a destination address to one MB location. In addition, by setting one bit of the microcommand, the AU result can be transferred to the AB source.

4.3.6 Microcondition Codes

For each operation performed by the AU or shift gates, a set of condition codes is dynamically generated, describing the result. These are:

a. Carry-out (c). The carry-out is generated as the arithmetic carry for an add operation, the borrow for a subtract operation or the shift carry-out for a shift operation.
b. Overflow (v). Overflow is generated for add, subtract or shift operations. The conditions under which overflow occurs depends on the operation.

c. Zero (z). The zero condition exists when all bits of the result are ZERO.
d. Negative (n). The negative condition exists when the most-significant bit of the result (shifted, if applicable) is ONE.
e. Positive (p). The positive condition exists when the result is greater than zero (not zero and not negative).
f. Odd (d). The odd condition exists when the least-significant bit of the result is ONE.

The last four conditions are referred to as data value codes and are generated from the value of the AU result on MB.

A microcommand can specify dynamic conditional testing of the micro-condition codes generated as the result of an operation, and the conditional test can cause a skip of the next microcommand or a branch to a new microprogram location. This capability saves considerable time over machine designs that require conditional testing to be performed on the condition generated by a *previous* operation.

4.3.7 Microstatus Register (MS)

The six dynamic condition codes can be saved as static microstatus bits in MS. Each microcommand can specify separate storing of the carry/overflow and the four data value codes in MS. These static microstatus conditions (instead of the dynamic microcondition codes) can then be tested by microcommands for conditional skips or branches.

MS is 16 bits in length. In addition to the six microcondition codes, other status bits are stored in this register. The contents of MS can be read via AB and can be loaded as a destination via MB. The complete set of status bits contained in MS is defined in Table 4-1.

Table 4-1. Microstatus Register Bit Definitions

MS Bit	Symbol	Name	Description
00	L	Link	Stored state of dynamic carry-out (c) of AU or shift gates
01	V	Overflow	Stored state of dynamic arithmetic or shift overflow (v)
02	Z	Zero	Stored state of zero (z) data value code
03	N	Negative	Stored state of negative (n) data value code
04	P	Positive	Stored state of positive (p) data value code
05	D	Odd	Stored state of odd (d) data value code
15 to 06			Special use, depending on emulation

4.3.8 Word and Byte Operations

The AU and shift elements of the CPU handle 16 bits and, therefore, execute full word operations. The CPU is also designed to operate on bytes (half words), if so specified by a microcommand. The byte mode can be designated as unconditional or conditional. In the conditional case, a byte-mode operation is performed only if the emulation circuitry indicates that the instruction being emulated is a byte-mode instruction.

The Engine has the capability of transferring either words or bytes on the MACROBUS. For arithmetic and logical byte operations involving AU, the specified operation is performed on the full 16-bit A and B operands. Since microcondition codes are generated on only the less-significant byte (bits 07 to 00) of the result, the bytes to be manipulated must be right-justified. Carry bits propagated out of the less-significant byte can affect the results in the more-significant byte. A byte operation with a file register (FR) as a destination does not modify the most-significant byte of the specified FR. A register destination, however, reflects the full 16-bit result. For example, consider addition of the following two right-justified bytes:

```
 A : 00000000 10110110   (-74)

+B : 00000000 11101011   (-21)
     -----------------
    (00000001)10100001   (-95)
```

Microcondition codes are generated from the less-significant byte as follows:

c = 1, v = 0, z = 0, n = 1, p = 0, d = 1.

The result for byte-mode operations is interpreted for the less-significant byte only. In many cases, it is desirable to extend the sign of the less-significant byte across the entire word (e.g., where word and byte arithmetic operations are mixed). A microcommand is provided that will insert the state of the microstatus L bit into the most-significant eight bits of a word. Thus, if the state of the c bit from the previous example is saved as L, the "sign-extended" result is:

1111111110100001

This can be generated by execution of the Sign Extend microcommand.

The CPU has an extensive complement of Shift microcommands that includes arithmetic as well as logical open and closed forms, both single- and double precision (double-length shifts involve XR).

For byte-shift operations, shifting is performed on only the less-significant byte. The more-significant byte remains unchanged. The carry input and microcondition codes are associated with the less-significant byte. Examples are:

a. Byte mode, open left shift:

A = 00000011 10101101 L
R = 00000011 0101101L
(c)1

c = 1, v = 1, z = 0, n = 0, p = 1, d = L.

Note that L is the shift carry input and that the carry-out is the most-significant bit of the less-significant byte.

b. Byte mode, open right shift:

A = 00000011 L 10101111
R = 00000011 L1010111 1(c)

c = 1, v = $\overline{L}$, z = 0, n = L, p = $\overline{L}$ d = 1.

SECTION 5
MICROCOMMANDS

5.1 GENERAL

Microcommands generate the control signals that enable all internal operations of the Engine. There are no suboperations performed. All functions specified by a microcommand are executed simultaneously within a single clock step, with the following exceptions:

a. When the microprogram execution sequence is altered, one additional clock step is required to execute the branch operation.
b. A MACROBUS access delay inhibits microcommand execution until a synchronizing I/O response is received.

A CPU clock period is 165 nanoseconds and all microcommands are executed within an integer multiple of that period.

The CPU incorporates a 48-bit microcommand word to perform all operations in the machine. The microcommand structure permits simultaneous execution of many parallel functions specified in each microcommand to achieve exceptionally fast emulation of general-purpose computer operations.

The structure of the microcommands provides considerable flexibility in organizing a particular microprogram to maintain high effective execution rates with economical use of control memory space.

5.2 MICROCOMMAND CLASSES

The three classes of microcommands are:

a. Logical
b. Arithmetic
c. Special

Every microcommand, regardless of class, has the ability to specify a conditional or unconditional branch or skip operation. Since the format of the microcommands differs, depending on whether a branch or skip is specified, the microcommands in each class can be considered to be one of two types:

a. Branch type
b. Skip type

Figure 5-1 shows the formats for the classes and types of microcommands executed by the CPU. The format for the logical and arithnetic classes is identical. The general characteristics of each class and type are defined below.

5.2.1 Logical and Arithmetic Classes

As the name implies, the logical and arithmetic classes of microcommands perform logical and arithmetic functions of one or two variables, as specified by the microcommand. The specific logical or arithmetic

LOGICAL AND ARITHMETIC CLASSES

Branch Type

47 42 41 37 36 32 29 24 23 20 19 16 15 12 11 00

SB	OP	DN	NX	AO	MC	MX	BO	BF

0

Skip Type

SB	OP	DN	NX	AO	MC	MX	LL

1

SPECIAL CLASS

Branch Type

SB	OP	DN	NX	AO	MC	MX	SO	BF

0

Skip Type

SB	OP	DN	NX	AO	MC	MX	SO	FN

1

SB = branch condition code (bit 47 specifies microcommand type)

OP = basic operation performed by the microcommand

DN = destination address of result from the arithmetic/logic unit (AU)

NX = special control functions

AO = source address of A operand to AU

MC = microcondition code specification (dispositions)

MX = special control functions

BO = source address of B operand to AU

SO = special operation control functions

BF = branch address or auxiliary control functions

LL = literal value

FN = auxiliary control functions

Figure 5-1. Microcommand Formats

operation is defined by the OP field. A total of 16 logical and eight arithmetic operations are implemented. The same set of operations is performed regardless of whether a branch- or skip-type microcommand is used.

Branch Type

The logical or arithmetic branch-type microcommand permits the programmer to specify that a conditional or unconditional branch to a new program location can occur based on the results of executing the current microcommand (or on results previously stored).

In this type of microcommand, both an A and B operand to AU are specified. The destination of the resulting operation is also specified. Arithmetic condition codes resulting from the microcommand execution can be saved or ignored.

If a branch condition is specified, an 11-bit branch address is provided that alters the microprogram sequence if the branch condition is met. A control bit is also provided that can cause the next CM address to be pushed into CS before the branch is made. This permits the microprogram to later execute an automatic return to the microprogram sequence via CS.

It is not necessary to specify a branch condition, even though the microcommand is a branch type. If no branch condition is specified, an auxiliary set of control functions can be specified that are performed simultaneously with execution of the basic logical or arithmetic operation.

The remaining fields of the branch-type microcommand provide special control functions that can modify execution and content of the next microcommand in sequence. The operations performed by these fields are common to all microcommands, regardless of class and type.

Skip-Type

The logical or arithmetic skip-type microcommand performs the same basic operations as the branch type. The differences in the skip-type microcommands are:

a. Instead of a branch condition, a condition is specified under which execution of the microcommand at the next CM location can be inhibited (skipped). The CM address sequence itself is not altered.
b. The B-operand source and branch address are replaced by a 16-bit literal value. This value is used directly as the B operand for those logical and arithmetic operations that involve a B-operand input to AU.
c. Because of the space reserved for a literal value (whether or not one is required), the auxiliary control functions defined for the branch-type microcommand cannot be specified.

All other operations of a skip-type microcommand are identical to the corresponding branch-type microcommand.

5.2.2 Special Class

The special class of microcommands provides functions that affect specialized control and other operations required of the computer. Some of these microcommands involve the use of AU. The operation performed is specified by the OP field. A total of seven special operations are implemented.

Branch-Type

The special branch-type microcommand permits the programmer to specify a conditional or unconditional branch just as for the logical or arithmetic branch type. And, in the same way, either a branch address or a set of auxiliary control functions can be specified, depending on whether or not a branch condition is specified by the microcommand.

Skip-Type

The special skip-type microcommand is the same as the branch type and specifies the same operations, except that:

a. Instead of a branch condition, a condition is specified under which execution of the microcommand at the next CM location can be inhibited (skipped). The CM address sequence itself is not altered.
b. Since a branch address cannot be specified by this type of microcommand and since a B operand is never used, the space reserved for these is used to specify a set of auxiliary control functions.

5.3 LOGICAL MICROCOMMANDS

The following paragraphs present a description of each logical microcommand. A summary of all the basic microcommands executed by the CPU is given in Table 5-1.

The description of each microcommand includes the mnemonic; hexadecimal OP-field code; symbolic notation describing its operation, where applicable; a description of the function performed; and examples or other comments to clarify the description.

Table 5-1. **Cal Data 100 Engine Microcommand Summary**

Mnemonic	OP Field (Hexadecimal)	Name
		LOGICAL
EML	00	Emulate (optional)
SXA	01	Sign Extend A
MVA	02	Move A
MVB	03	Move B
OCA	04	Complement A
OCB	05	Complement B
AND	06	AND A, B
NDB	07	AND A, $\overline{B}$
NDA	08	AND $\overline{A}$, B
NOR	09	Not OR
ORI	0A	OR A, B
ORB	0B	OR A, $\overline{B}$
ORA	0C	OR $\overline{A}$, B
NAND	0D	Not AND
XOR	0E	Exclusive OR
COI	0F	Coincidence
		ARITHMETIC
ADD	10	Add, A, B
SUB	11	Subtract A, B
ADC	12	Add Carry
SBC	13	Subtract Carry
INC	14	Increase A
DEC	15	Decrease A
MSA	16	Add A Masked
-	17	(reserved)
		SPECIAL
SHF	18	Shift
MUS	19	Multiply Step
DVS	1A	Divide Step
TSB	1B	Test Bit
MMS	1C	Modify Macrostatus (optional)
CMA	1D	Conditional Memory Access, A operand (optional)
CMB	1E	Conditional Memory Access, B operand
DCD	1F	Decode (optional)

The following symbols are used (in addition to many defined in Table 1-1):

| | | = absolute value of

( ) = contents of

$(\overline{\ })$ = Boolean complement

$\cap$ = Boolean AND

$\cup$ = Boolean OR

$\oplus$ = Boolean exclusive OR

= = equal to

< = less than

$\geq$ = greater than or equal to

$\neq$ = not equal to

+ = arithmetic addition (two's complement)

- = arithmetic subtraction (two's complement)

X = arithmetic multiplication

÷ = arithmetic division

A = A operand to AU (from A-operand source specified by the microcommand)

An = nth bit of A

Am:An = A bits m to n

B = B operand to AU (from B-operand source specified by the microcommand)

R = result (word on MB)

RM = more-significant byte of R

RL = less-signficant byte of R

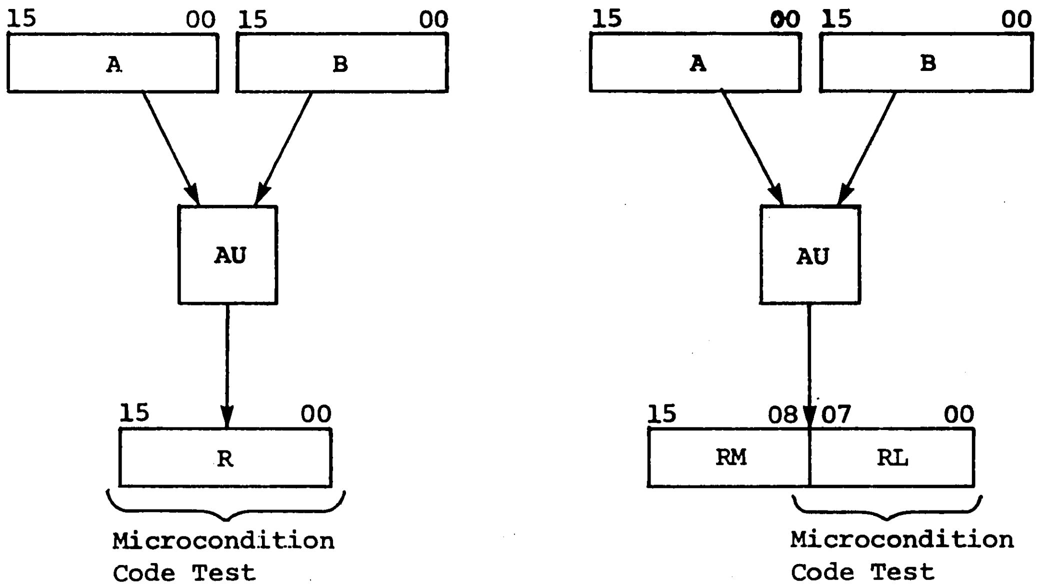
DN = destination location (specified by the microcommand)

CIN = carry input

$\rightarrow$ = replaces

The logical microcommands listed in Table 5-1 can be executed in either the word or byte mode. With one exception (SXA), the operation is performed on the full pair of operand words in AU and the 16-bit result is transferred to the destination via MB.

Microcondition codes are determined on the full word in the word mode and on the less-significant byte in the byte mode. This is illustrated below:



The microcondition codes for all logical-class microcommands are given in Table 5-2.

All logical microcommands are standard except Emulate (EML), which is optional. The main purpose of EML is for very rapid emulation decoding of instruction operation codes and control fields. The procedure is to store the instruction in IR. EML then initiates translation of the contents of IR into a CM branch address generated from a table of values. The address directs the microprogram to the proper microroutine in CM for the emulation of each instruction. The emulate table is specifically programmed for each computer to be emulated. The decoding operation performed by EML can be accomplished by other methods using only standard microcommands, but at the cost of time and CM space. For this reason, the need to implement EML depends on the specific application.

Table 5-2. Microcondition Codes for Logical Microcommands.

Microcondition	Definition	
Code	Word Mode	Byte Mode
c	controlled by the MC	controlled by the MC
v	field of the microcommand	field of the microcommand
z	1 if R=0; 0 otherwise	1 if RL=0; 0 otherwise
n	1 if R15=1; 0 otherwise	1 if R07=1; 0 otherwise
p	1 if R>0; 0 otherwise	1 if RL>0; 0 otherwise
d	1 if R00=1; 0 otherwise	1 if R00=1; 0 otherwise

5.3.1 Emulate (Optional)

Mnemonic: EML $00

Operation:

R = A
R →(DN)
(emulation table)→(CC), unless higher-priority CC modification occurs

Description: The A operand is transferred to the destination. The contents of IR are translated into a branch address (emulate instruction address, EIA) to CC using an emulate table on the Emulate Board. If a higher-priority CC modification occurs concurrent with the microcommand, the EIA is ignored. All microcommand fields are effective as defined, except that the BO field is ignored, since no B operand is used.

5.3.2 Sign Extend A

Mnemonic: SXA $01

Operation:

Word mode	Byte mode
R = A	RM =(N) ∪ A15:A08
R →(DN)	RL = A07:A00
	RM, RL→(DN)

Description: In the word mode, the A operand is transferred to the destination.

In the byte mode, the state of the negative microstatus bit, N, is extended to the more-significant byte of the A operand. The contents of the less-significant byte of the A operand are unmodified. The result is transferred to the destination.

Example: Perform a byte mode add on A and B and store the result in the A-operand location, then extend the sign of the byte result.

ADD:

```
   A   00000000 10110100   (-76)
  +B   00000000 11101100   (-20)
   R   00000001 10100000   (-96)
```

The microcondition codes generated are:
l = 1, v = 0, z = 0, n = 1, p = 0, d = 0

SXA:

```
   A   00000001 10100000
 ∪(N)  11111111
   R   11111111 10100000
       \______/ \______/
          RM       RL
```

5.3.3 Move A

Mnemonic: MVA $02

Operation: $R = A$
$R \longrightarrow (DN)$

Description: The A operand is transferred unmodified to the destination.

5.3.4 Move B

Mnemonic: MVB $03

Operation: $R = B$
$R \longrightarrow (DN)$

Description: The B operand is transferred unmodified to the destination.

5.3.5 Complement A

Mnemonic: OCA $04

Operation: $R = \overline{A}$
$R \longrightarrow (DN)$

Description: The logical or one's complement of the A operand is transferred to the destination.

Example:

	Binary	Octal	Hexadecimal
A	0110110100101100	066454	6D2C
R	1001001011010011	111323	92D3

5.3.6 Complement B

Mnemonic: OCB $05

Operation: $R = \overline{B}$
$R \longrightarrow (DN)$

Description: The logical or one's complement of the B operand is transferred to the destination.

5.3.7 AND A, B

Mnemonic: AND \$06

Operation: $R = A \cap B$
$R \longrightarrow (DN)$

Description: The logical AND of the A and B operands is transferred to the destination.

5.3.8 AND A, $\overline{B}$

Mnemonic: NDB \$07

Operation: $R = A \cap \overline{B}$
$R \longrightarrow (DN)$

Description: The logical complement of the B operand is ANDed with the A operand and the result is transferred to the destination.

5.3.9 AND $\overline{A}$, B

Mnemonic: NDA \$08

Operation: $R = \overline{A} \cap B$
$R \longrightarrow (DN)$

Description: The logical complement of the A operand is ANDed with the B operand and the result is transferred to the destination.

5.3.10 Not OR

Mnemonic: NOR \$09

Operation: $R = \overline{A \cup B}$
$R \longrightarrow (DN)$

Description: The logical NOR of the A and B operands is transferred to the destination.

5.3.11 OR A, B

Mnemonic: ORI \$0A

Operation: $R = A \cup B$
$R \longrightarrow (DN)$

Description: The logical OR of the A and B operands is transferred to the destination.

5.3.12 OR A, $\overline{B}$

Mnemonic: ORB \$0B

Operation: $R = A \cup \overline{B}$
$R \longrightarrow (DN)$

Description: The logical complement of the B operand is ORed with the A operand and the result is transferred to the destination.

5.3.13 OR $\overline{A}$, B

Mnemonic: ORA \$0C

Operation: $R = \overline{A} \cup B$
$R \longrightarrow (DN)$

Description: The logical complement of the A operand is ORed with the B operand and the result is transferred to the destination.

5.3.14 Not AND

Mnemonic: NAND \$0D

Operation: $R = \overline{A \cap B}$
$R \longrightarrow (DN)$

Description: The logical NAND of the A and B operands is transferred to the destination.

5.3.15 Exclusive OR

Mnemonic: XOR $0E

Operation: $R = A \oplus B$
$R \rightarrow (DN)$

Description: The logical exclusive OR of the A and B operands is transferred to the destination. The exclusive OR by definition is:

$$A \oplus B = [A \cap \bar{B}] \cup [\bar{A} \cap B]$$

5.3.16 Coincidence

Mnemonic: COT $0F

Operation: $R = \overline{A \oplus B}$
$R \rightarrow (DN)$

Description: The complement of the logical exclusive OR of the A and B operands is transferred to the destination. This is the coincidence function:

$$\overline{A \oplus B} = [A \cap B] \cup [\bar{A} \cap \bar{B}]$$

5.4 ARITHMETIC MICROCOMMANDS

The arithmetic microcommands are listed in Table 5-1. There are eight microcommands in this class.

The CPU performs both binary addition and subtraction (as opposed to complementary addition). Negative numbers are assumed to be represented as two's complements of positive numbers (although one's complement arithmetic can be performed, since the programmer has independent control of the carry and borrow inputs to AU). A complete description of binary arithmetic operations in the CPU is given in Appendix A. The carry and overflow microcondition codes differ for the addition and subtraction operations, as does the use of the carry-in term. The data value microcondition codes (z, n, p and d) are the same for addition and subtraction and depend only on the value of the arithmetic result.

Arithmetic operations can be executed in either the word or byte mode. In either mode, the specified operation is performed on the full pair of operand words in AU. The 16-bit result is transferred to the destination via MB. The microcondition codes are determined on the full word in the word mode and on the less-significant byte in the byte mode (see illustration in subsection 5.3). The microcondition codes for addition and subtraction operations are defined in Table 5-3.

Table 5-3. Microcondition Codes for Arithmetic Microcommands

Micro Condition Code	Arithmetic Operation	Definition	
		Word Mode	Byte Mode
c	Addition	$[A15 \cap \overline{R15}] \cup [B15 \cap \overline{R15}]$ $\cup [A15 \cap B15]$	$[A07 \cap \overline{R07}] \cup [B07 \cap \overline{R07}]$ $\cup [A07 \cap B07]$
	Subtraction	$[\overline{A15} \cap R15] \cup [\overline{A15} \cap B15]$ $\cup [B15 \cap R15]$	$[\overline{A07} \cap R07] \cup [\overline{A07} \cap B07]$ $\cup [B07 \cap R07]$
v	Addition	$[A15 \cap B15 \cap \overline{R15}]$ $\cap [\overline{A15} \cap \overline{B15} \cap R15]$	$[A07 \cap B07 \cap \overline{R07}]$ $\cup [\overline{A07} \cap \overline{B07} \cap R07]$
	Subtraction	$[\overline{A15} \cap B15 \cap R15]$ $\cup [A15 \cap \overline{B15} \cap \overline{R15}]$	$[\overline{A07} \cap B07 \cap R07]$ $\cup [A07 \cap \overline{B07} \cap \overline{R07}]$
z	Addition or Subtraction	1 if R = 0; 0 otherwise	1 if RL = 0; 0 otherwise
n	Addition or Subtraction	1 if R15 = 1; 0 otherwise	1 if R07 = 1; 0 otherwise
p	Addition or Subtraction	1 if R > 0; 0 otherwise	1 if RL > 0; 0 otherwise
d	Addition or Subtraction	1 if R00 = 1; 0 otherwise	1 if R00 = 1; 0 otherwise

5.4.1 Add A, B

Mnemonic: ADD $10

Operation: R=A+B+CIN
R ⟶ (DN)

Description: The A and B operands and the value of CIN designated by the MC field are added arithmetically and the result is transferred to the destination.

Micro-condition Codes: Addition (Table 5-3).

Example: Add A and B and increment the result:

```
    A = +27,435 =  0110101100101011
   +B = - 1,747 =  1111100100101101
 +CIN =      +1 =  0000000000000001
    R = +25,689 =1 0110010001011001
                 |
                 └──► c
```

The microcondition codes generated are:
c = 1, v = 0, z = 0, n = 0, p = 1, d = 1

5.4.2 Subtract A, B

Mnemonic: SUB $11

Operation: R=A-B-CIN
R ⟶ (DN)

Description: The B operand and the value of CIN designated by the MC field are subtracted from the A operand and the result is transferred to the destination.

Micro-condition Codes: Subtraction (Table 5-3).

Example: Subtract B from A and decrement the result:

```
    A =       -444 =       1111111001000100
   -B = -(-1,747) =     - 1111100100101101
 -CIN =     -(+1) =     - 0000000000000001
    R =     +1,302 =     0 0000010100010110
                        |
                        └──► c
```

This operation produces a one's complement result when the result is negative, since CIN is specified as a ONE.

The microcondition codes generated are:
c = 0, v = 0, z = 0, n = 0, p = 1, d = 0

5.4.3 Add Carry

Mnemonic: ADC $12

Operation: R=A+CIN
R ⟶ (DN)

Description: The value of CIN designated by the MC field is added to the A operand and the result is transferred to the destination.

Micro-condition Codes: Addition (Table 5-3).

5.4.4 Subtract Carry

Mnemonic: SBC $13

Operation: R=A-CIN
R ⟶ (DN)

Destination: The value of CIN designated by the MC field is subtracted from the A operand and the result is transferred to the destination.

Micro-condition Codes: Subtraction (Table 5-3).

5.4.5 Increase A

Mnemonic: INC $14

Operation: R=A+1+CIN
R ⟶ (DN)

Description: The value one and the value of CIN designated by the MC field are added to the A operand and the result is transferred to the destination. If CIN is ONE, the A operand is increased by two; otherwise, it is increased by one.

Micro-condition Codes: Addition (Table 5-3),

Examples: Increase the A operand by two if MC designates CIN as ONE; increase by one otherwise:

```
   A = +7817 =  0001111010001001
  +1 =    +1 =  0000000000000001
+CIN =     0 =  0000000000000000
   R = +7818 =  0001111010001010
```

The microcondition codes generated are:
c = 0, v = 0, z = 0, n = 0, p = 1, d = 0

Another example, where overflow is affected:

```
   A = +32,766 = 0111111111111110
  +1 =      +1 = 0000000000000001
+CIN =      +1 = 0000000000000001
   R = +32,768 = 1000000000000000
```

The microcondition codes generated are:
c = 0, v = 1, z = 0, n = 1, p = 0, d = 0

5.4.6 Decrease A

Mnemonic: DEC $15

Operation: R=A-1-CIN
R ——► (DN)

Description: The quantity one and the value of CIN designated by the MC field are subtracted from the A operand and the result is transferred to the destination. If CIN is ONE, the A operand is decreased by two; otherwise, it is decreased by one.

Micro-condition Codes: Subtraction (Table 5-3).

Example: Decrease the A operand by two if the L microstatus bit is set; decrease by one otherwise:

a. If (L) = 1:

```
      A =    +1 =   0000000000000001
     -1 = -(+1) = - 0000000000000001
    CIN = -(+1) = - 0000000000000001
      R = -(+1) = 1 1111111111111111
                  |
                  └──► c
```

The microcondition codes generated are:
c = 1 (borrow), v = 0, z = 0, n = 1,
p = 0, d = 1

b. If (L) = 0:

```
      A =    +1 =   0000000000000001
     -1 = -(+1) = - 0000000000000001
    CIN =    -0 = - 0000000000000000
      R =     0 =   0000000000000000
```

The microcondition codes generated are:
c = 0, v = 0, z = 1, n = 0, p = 0, d = 0

5.4.7 Add A Masked

Mnemonic: MSA $16

Operation: $R = A + [A \cap B] + CIN$
$R \longrightarrow (DN)$

Description: The logical AND of the A and B operands is added to the A operand and to the value of CIN designated by the MC field, and the result is transferred to the destination.

Microcondition Codes: Addition (Table 5-3).

Example: Add the absolute value of the less-significant byte of A to the A operand:

A =	-110	=	1111111110010010
B =	mask	=	0000000001111111
$A \cap B$ =	+18	=	0000000000010010
+A =	+(-110)	=	1111111110010010
+CIN =	+0	=	0000000000000000
R =	-92	=	1111111110100100

The microcondition codes generated are:
c = 0, v = 0, z = 0, n = 1, p = 0, d = 0

5.5 SPECIAL MICROCOMMANDS

The seven special microcommands listed in Table 5-1 provide a powerful extension of the basic logical and arithmetic microcommands. Four of these are standard and have general application in all emulation microprograms. Three microcommands are defined as optional, since they must be tailored to a particular emulation system. The hardware elements that implement the optional microcommands are modularized to permit them to be either omitted or redefined without affecting the basic hardware of the Engine.

Microcondition codes generated for the special microcommands are generally identical to those defined for the arithmetic microcommands, with major exceptions. The overall uses and limitations of the special microcommands are described in the following paragraphs.

5.5.1 Shift

The Shift microcommand provides complete flexibility for single-length shifts involving only the A operand and AU, and double-length shifts involving AU and XR. Shifts can be left or right, logical or arithmetic, open or closed. While the basic microcommand shifts only a single bit, multibit shifts can be performed by repeating the microcommand using LC.

Mnemonic: SHF $18

Microcommand Type: Special branch or special skip.

Description: The SO field specifies the type and direction of shift as shown in Table 5-4. For a single-precision (16-bit) shift, the A operand is shifted one place left or right and the result is transferred to the destination. For a double-precision (32-bit) shift, the A operand and the contents of XR are shifted one place left or right with a linked carry between the two words. The shifted AU result is transferred to the destination and the shifted XR result remains in XR. The shift operation is performed in the word mode unless a byte operation is specified by the FN field.

Using the special skip-type format for the shift can lead to possible conflict between a double-length shift specification in the SO field and an XR shift specification in the FN field. If such a conflicting specification is made, the SO field control is effective and the FN field control is ignored.

Table 5-4. SO-Field Shift Specification

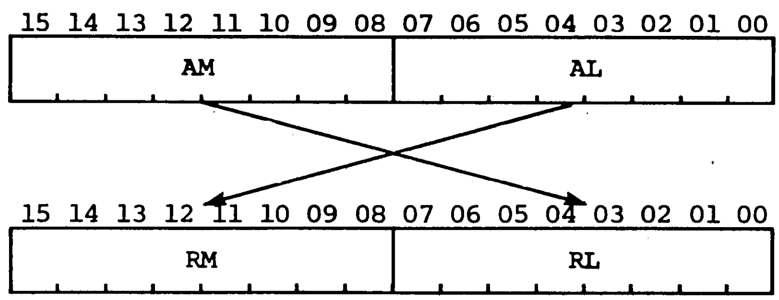
SO Field Bits 15	14	13	12	Shift Operation
0	0	x	x	swap halves
0	1	x	x	shift left, logical
1	0	x	x	shift right, logical
1	1	x	x	shift right, arithmetic
x	x	0	x	single precision
x	x	1	x	double precision
x	x	x	0	open shift
x	x	x	1	closed shift

5.5.1.1 Single-Precision Shifts.

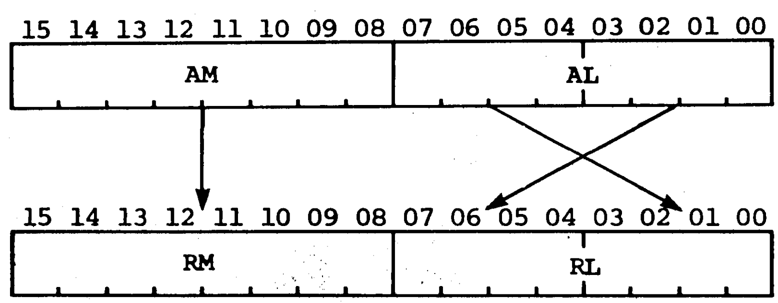
Single-precision shifts involve only AU shift elements operating on the A operand.

Swap Halves (Word or Byte).

Operation: a. Word mode:
R15:R08=A07:A00
R07:R00=A15:A08
R ⟶ (DN)



b. Byte mode:
R15:R08=A15:A08
R07:R04=A03:A00
R03:R00=A07:A04
R ⟶ (DN)



Description: For word swaps, the more-significant and less-significant bytes of the A operand are swapped and the result is transferred to the destination.

For byte swaps, the more-significant and less-significant halves of the less-significant byte of the A operand are swapped and the result is transferred to the less-significant byte of the destination. The more-significant byte of the A operand is transferred, unchanged, to the more-significant byte of the destination.

Micro-condition Codes:

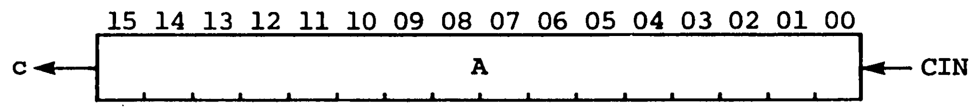
a. Word mode:
- c = A15
- v = A15 $\oplus$ A14
- z = 1 if R=0; 0 otherwise
- n = 1 if R15=1; 0 otherwise
- p = 1 if R>0; 0 otherwise
- d = 1 if R00=1; 0 otherwise

b. Byte Mode:
- c = A07
- v = A07 $\oplus$ A06
- z = 1 if RL=0 0 otherwise
- n = 1 if R07=1; 0 otherwise
- p = 1 if RL>0; 0 otherwise
- d = 1 if R00=1; 0 otherwise

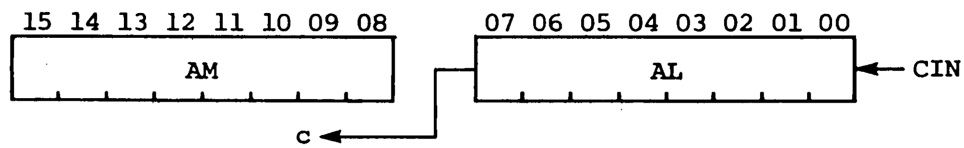
Logical Open Left Shift (Word or Byte).

Operation:

a. Word mode:



b. Byte mode:



Description: For word shifts, the A operand is shifted left one bit. The value of CIN designated by the MC field is shifted into bit 00. The bit shifted out of bit 15 is the shift carry out. The result is transferred to the destination.

Byte shifts are the same as word shifts, except that the shift is on the less-significant byte only. The carry bit is shifted out of bit 07. The more-significant byte is unmodified. The resulting word is transferred to the destination.

Micro-condition Codes:

a. Word Mode:

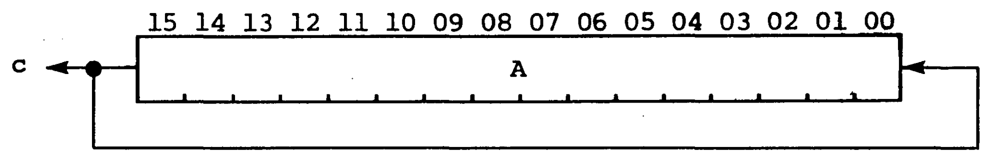
c = A15
v = A15 $\oplus$ A14
z = 1 if R=0;
    0 otherwise
n = 1 if R15=1;
    0 otherwise
p = 1 if R>0;
    0 otherwise
d = 1 if R00=1;
    0 otherwise

b. Byte mode:

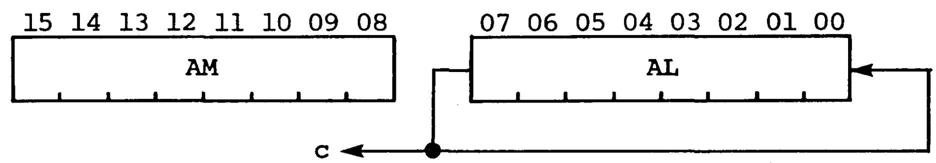
c = A07
v = A07 $\oplus$ A06
z = 1 if RL=0;
    0 otherwise
n = 1 if R07=1;
    0 otherwise
p = 1 if RL>0;
    0 otherwise
d = 1 if R00=1;
    0 otherwise

Logical Closed Left Shift (Word or Byte).

Operation: a. Word mode:



b. Byte mode:



Description: For word shifts, the A operand is shifted left one bit. Bit 15 is the shift carry out and is also shifted into bit 00. The result is transferred to the destination.

Byte shifts are the same as word shifts, except that the shift is on the less-significant byte only. The carry bit is shifted out of bit 07. The more-significant byte is unmodified. The resulting word is transferred to the destination.

Micro-condition Codes:

a. Word mode:

- c = A15
- v = A15 $\oplus$ A14
- z = 1 if R=0; 0 otherwise
- n = 1 if R15=1; 0 otherwise
- p = 1 if R>0; 0 otherwise
- d = 1 if R00=1; 0 otherwise

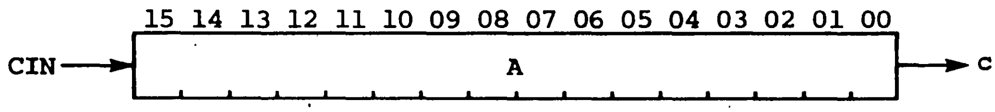
b. Byte mode:

- c = A07
- v = A07 $\oplus$ A06
- z = 1 if RL=0; 0 otherwise
- n = 1 if R07=1; 0 otherwise
- p = 1 if RL>0; 0 otherwise
- d = 1 if R00=1; 0 otherwise

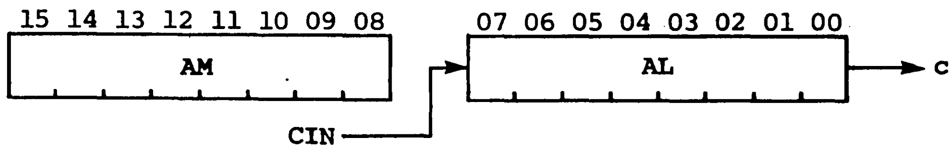
Logical Open Right Shift (Word or Byte).

Operation:

a. Word mode:



b. Byte mode:



Description:

For word shifts, the A operand is shifted one bit to the right. The value of CIN designated by the MC field is shifted into bit 15. The bit shifted out of bit 00 is the shift carry out. The result is transferred to the destination.

Byte shifts are the same as word shifts, except that the shift is on the less-significant byte only. The value of CIN is shifted into bit 07. The more-significant byte is unmodified. The resulting word is transferred to the destination.

Micro-condition Codes:

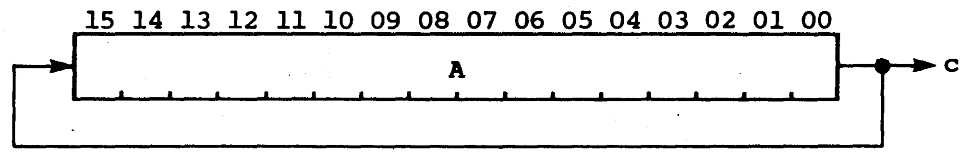
a. Word mode:
- c = A00
- v = A15 $\oplus$ CIN
- z = 1 if R=0; 0 otherwise
- n = 1 if R15=1; 0 otherwise
- p = 1 if R>0; 0 otherwise
- d = 1 if R00=1; 0 otherwise

b. Byte mode:
- c = A00
- v = A07 $\oplus$ CIN
- z = 1 if RL=0; 0 otherwise
- n = 1 if R07=1; 0 otherwise
- p = 1 if RL>0; 0 otherwise
- d = 1 if R00=1; 0 otherwise

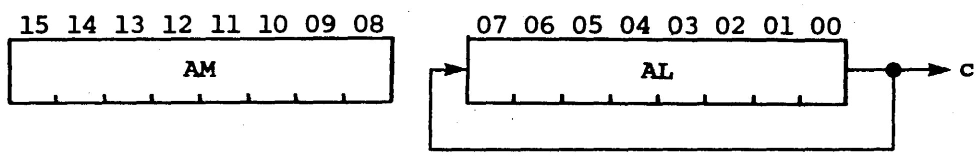
Logical Closed Right Shift (Word or Byte):

Operation:

a. Word mode:



b. Byte mode:



Description:

For word shifts, the A operand is shifted right one bit. Bit 00 is the shift carry out and is also shifted into bit 15. The result is transferred to the destination.

Byte shifts are the same as word shifts, except that the shift is on the less-significant byte only. The carry bit is shifted into bit 07. The more-significant byte is unmodified. The resulting word is transferred to the destination.

Micro-condition Codes:

a. Word mode:

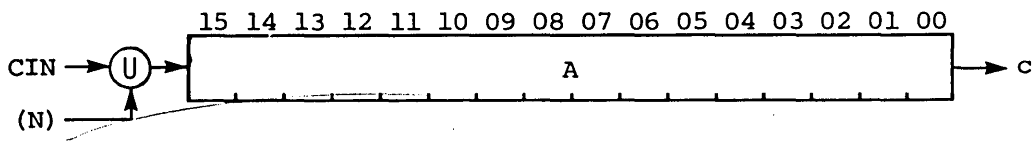
c = A00
v = A15 $\oplus$ A00
z = 1 if R=0;
0 otherwise
n = 1 if R15=1;
0 otherwise
p = 1 if R>0;
0 otherwise
d = 1 if R00=1;
0 otherwise

b. Byte mode:

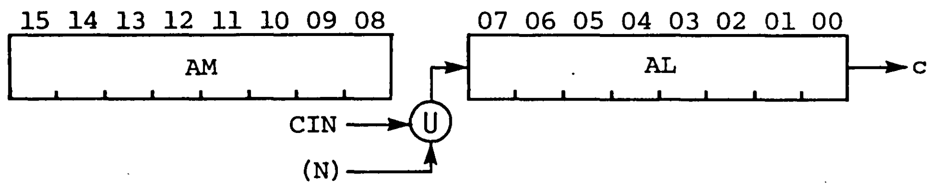
c = A00
v = A07 $\oplus$ A00
z = 1 if RL=0;
0 otherwise
n = 1 if R07=1;
0 otherwise
p = 1 if RL>0;
0 otherwise
d = 1 if R00=1;
0 otherwise

Arithmetic Open Right Shift (Word or Byte):

Operation: a. Word mode:



b. Byte mode:



Description: For word shifts, the A operand is shifted right one bit. The state of the N microstatus bit, ORed with the value of CIN designated by the MC field, is shifted into bit 15. Bit 00 is the shift carry out. The result is transferred to the destination.

Byte shifts are the same as word shifts, except that the shift is on the less-significant byte only. The value of N ORed with CIN is shifted into bit 07. The more-significant byte is unmodified. The resulting word is transferred to the destination.

Micro-condition Codes:

a. Word mode:
- c = A00
- v = A15 $\oplus$ CIN $\cup$ (N)
- z = 1 if R=0; 0 otherwise
- n = 1 if R15=1; 0 otherwise
- p = 1 if R>0; 0 otherwise
- d = 1 if R00 =1; 0 otherwise

b. Byte mode:
- c = A00
- v = A07 $\oplus$ CIN $\cup$ (N)
- z = 1 if RL=0; 0 otherwise
- n = 1 if R07=1; 0 otherwise
- p = 1 if RL>0; 0 otherwise
- d = 1 if R00=1; 0 otherwise

Arithmetic Closed Right Shift (Word or Byte).

Description: Same as logical closed right shift.

5.5.1.2 Double-Precision Shifts.

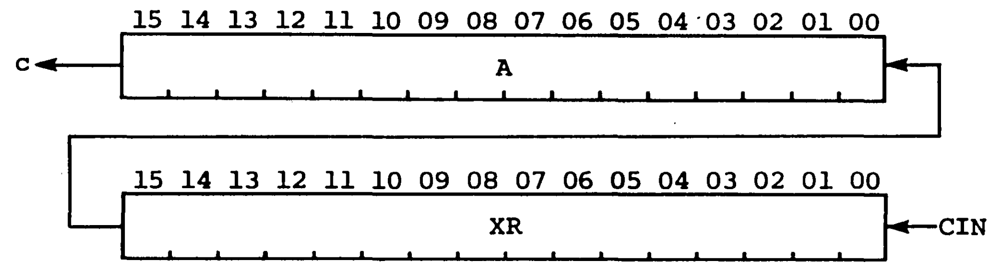
Double-precision shifts involve AU shift elements and XR. When a double-precision shift is specified in the SO field, an XR shift operation specified by the FN field is ignored.

Swap Halves (Word or Byte).

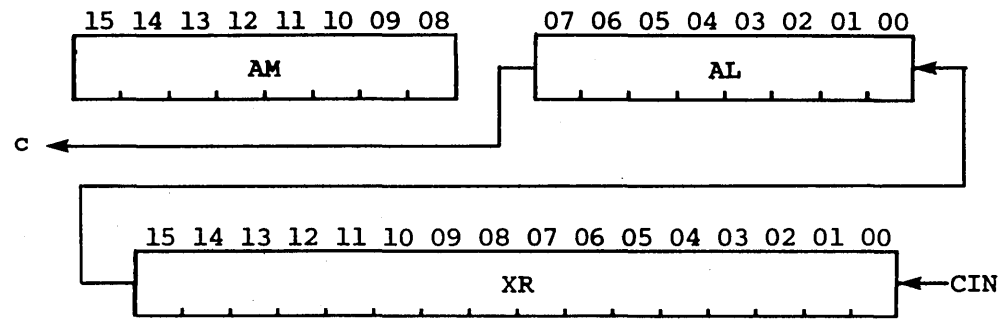
Description: Operation on the A operand and the microcondition codes generated are the same as for single-precision swap. The contents of XR are unmodified.

Logical Open Left Shift (Word or Byte).

Operation: a. Word mode:



b. Byte mode:



Description: For word shifts, the A operand and the contents of XR are shifted left one bit. The value of CIN designated by the MC field is shifted into XR bit 00. The XR bit 15 is shifted into the A-operand bit 00. A-operand bit 15 is the shift carry out. The shifted A-operand result is transferred to the destination. The shifted XR result remains in XR.

Byte shifts are the same as word shifts, except that the A-operand shift is on the less-significant byte only. A-operand bit 07 is the shift carry out. The more-significant byte is unmodified.

Micro-condition Codes:

a. Word modes:

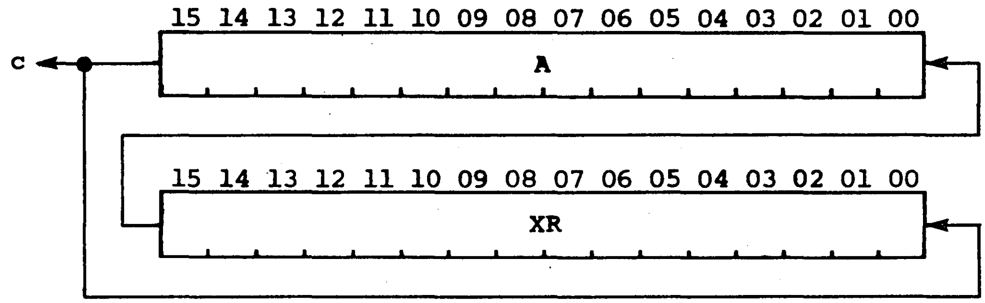
c = A15
v = A15 ⊕ A14
z = 1 if A=0;
    0 otherwise
n = 1 if A15=1;
    0 otherwise
p = 1 if A>0;
    0 otherwise
d = 1 if XR15=1;
    0 otherwise

b. Byte modes:

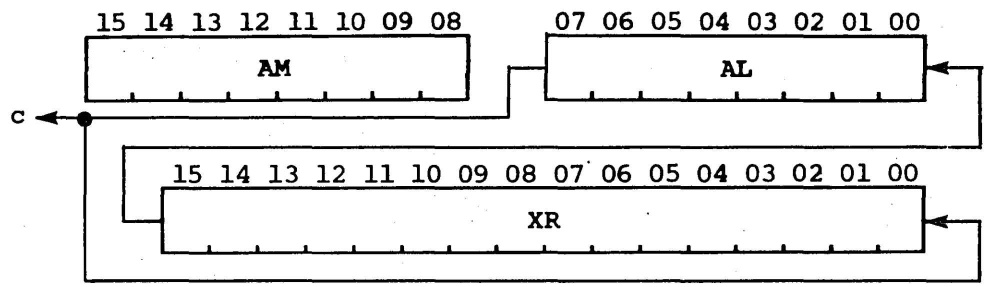
c = A07
v = A07 ⊕ A06
z = 1 if AL=0;
    0 otherwise
n = 1 if A07=1;
    0 otherwise
p = 1 if AL>0;
    0 otherwise
d = 1 if XR15=1;
    0 otherwise

Logical Closed Left Shift (Word or Byte).

Operation: a. Word mode:



b. Byte mode:



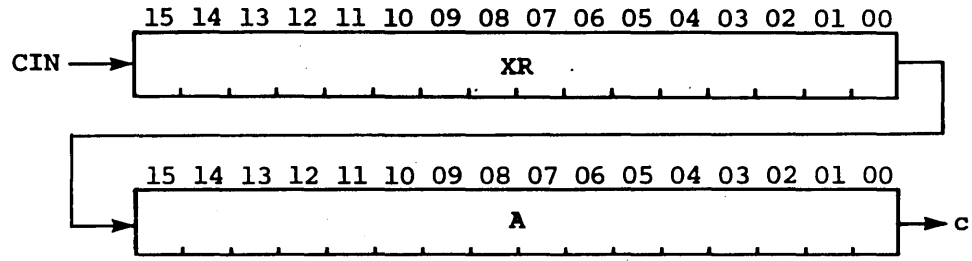
Description: For word shifts, the A operand and the contents of XR are shifted left one bit. XR bit 15 is shifted into A-operand bit 00. A-operand bit 15 is the shift carry out and is also shifted into XR bit 00. The shifted A-operand result is transferred to the destination. The shifted XR result remains in XR.

Byte shifts are the same as word shifts, except that the A-operand shift is on the less-significant byte only. A-operand bit 07 is the shift carry-out and is also shifted into XR bit 00. The more-significant byte is unmodified.

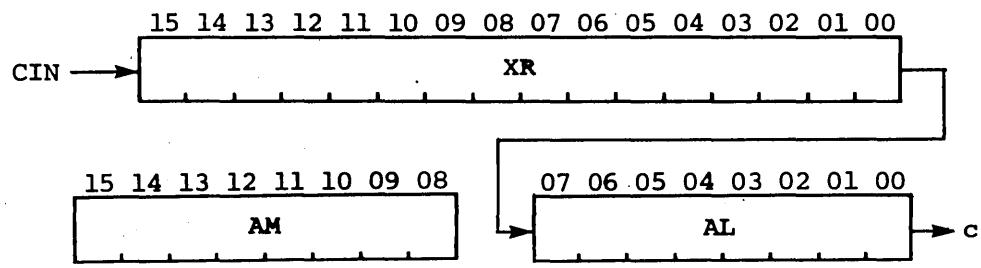
Micro-condition Codes: Same as logical open left shift.

Logical Open Right Shift (Word or Byte).

Operation: a. Word mode:



b. Byte mode:



Description: For word shifts, the A operand and the contents of XR are shifted right one bit. The state of CIN designated by the MC field is shifted into XR bit 15. XR bit 00 is shifted into A-operand bit 15. A-operand bit 00 is the shift carry out. The shifted A-operand result is transferred to the destination. The shifted XR result remains in XR.

Byte shifts are the same as word shifts, except that the A-operand shift is on the less-significant byte only. XR bit 00 is shifted into A-operand bit 07. The more-significant byte is unmodified.

Micro-condition Codes:

a. Word mode:

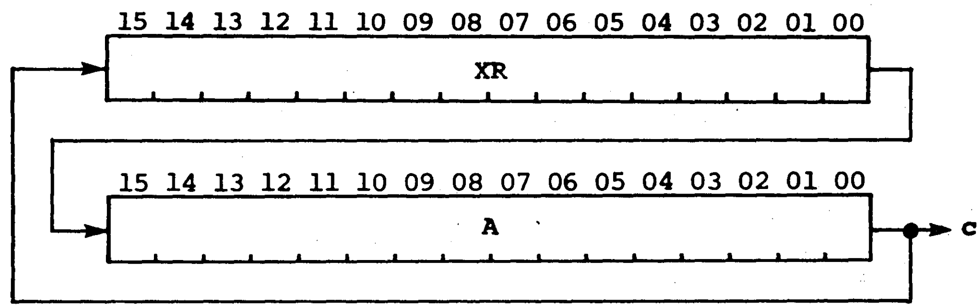
c = A00
v = XR00 ⊕ A15
z = 1 if A=0;
0 otherwise
n = 1 if XR15=1;
0 otherwise
p = 1 if A>0;
= 0 otherwise
d = 1 if A00=1;
0 otherwise

b. Byte mode:

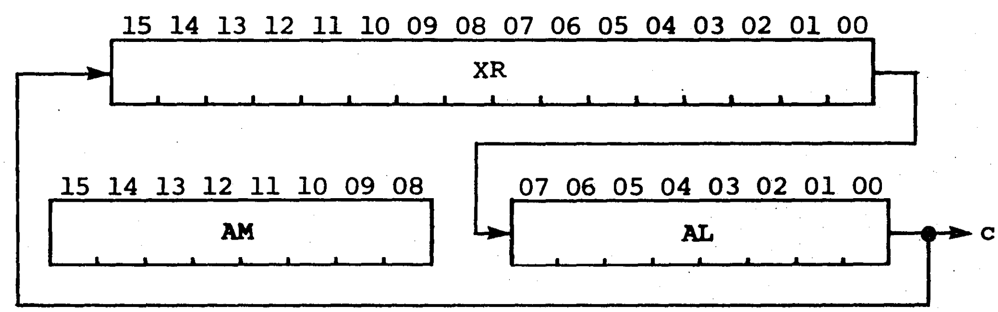
c = A00
v = XR00 ⊕ A07
z = 1 if AL=0;
0 otherwise
n = 1 if XR15=1;
0 otherwise
p = 1 if AL>0;
0 otherwise
d = 1 if A00=1;
0 otherwise

Logical Closed Right Shift (Word or Byte).

Operation: a. Word mode:



b. Byte mode:



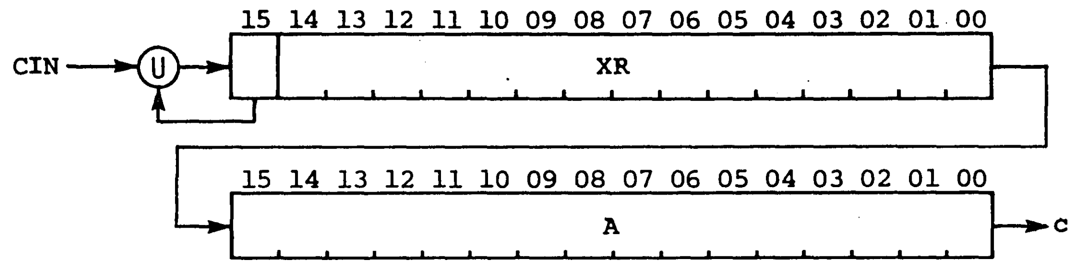
Description: For word shifts, the A operand and the contents of XR are shifted right one bit. XR bit 00 is shifted into A-operand bit 15. A-operand bit 00 is the shift carry out. The shifted A-operand is transferred to the destination. The shifted XR result remains in XR.

Byte shifts are the same as word shifts, except that the A-operand shift is on the less-significant byte only. XR bit 00 is shifted into A-operand bit 07. The more-significant byte is unmodified.

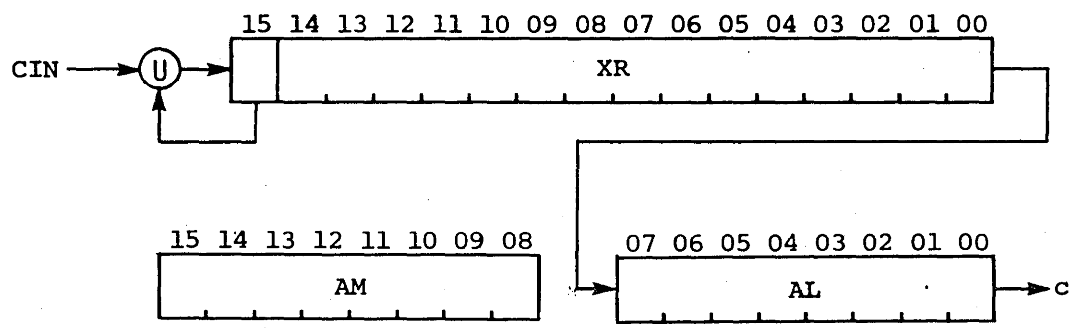
Micro-condition Codes: Same as logical open right shift.

Arithmetic Open Right Shift (Word or Byte).

Operation: a. Word mode:



b. Byte mode:



Description: For word shifts, the A operand and the contents of XR are shifted right one bit. XR bit 15, ORed with the state of CIN designated by the MC field, is shifted into XR bit 15. XR bit 00 is shifted into A-operand bit 15. A-operand result is transferred to the destination. The shifted XR result remains in XR.

Byte shifts are the same as word shifts, except that the A-operand shift is on the less-significant byte only. XR bit 00 is shifted into A-operand bit 07. The more-significant byte is unmodified.

Micro-condition Codes: Same as logical open right shift.

Arithmetic Closed Right Shift (Word or Byte).

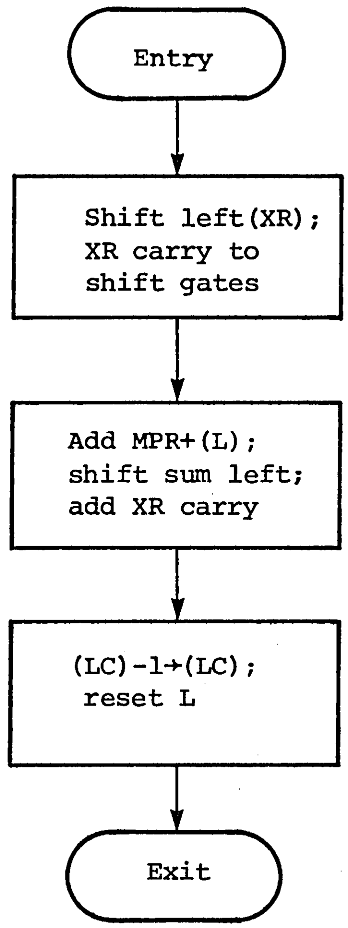
Description: Same as logical closed right shift.

5.5.2 Multiply Step

The MUS microcommand is a specialized version of the Shift microcommand with an automatic iterative repeat that permits high-speed implementation of a Multiply instruction. The average execution time is 300 ns per bit plus the additional time required to preformat the multiplier and multiplicand, determine the sign of the product and format the final result. No additional hardware is required for the high-speed multiply function, since all operations are implemented in control memory.

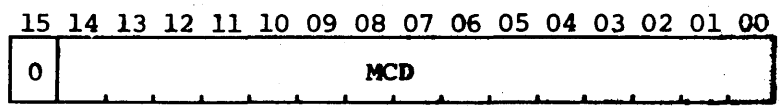
Mnemonic:	MUS $19
Microcommand Type:	Special branch
Description:	The MUS microcommand provides a set of simultaneous add, shift and test operations involving a register containing the multiplier (MPR) plus XR, LC and the state of the next MPR digit. The microcommand is automatically repeated until (LC)=0. For each ONE in MPR, a branch is made to a microcommand that adds the multiplicand (MPD) to XR. This permits complete execution of multiply steps in one clock cycle for a ZERO MPR digit and three clock cycles for a ONE MPR digit. The MUS microcommand is used for multiplication of two 16-bit operands with a resulting 32-bit product.
Registers Used:	a. MPR in a register designated by the AO field of the MUS microcommand. This register contains the more-significant half of the product at the end of the complete multiplication. b. MCD in a register designated by a separate microcommand that adds MCD to the partial product. c. XR, which accumulates MCD additions to the partial product and contains the less-significant half of the product at the end of the complete multiplication. d. LC, which counts the number of MUS iterations performed. e. The L microstatus bit, used to propagate carries from the less-significant half to the more-significant half of the partial product.
SO Field:	The SO field must be programmed for a double-precision, logical open lift shift (bits 15:12 = $6), as specified in the Shift microcommand description (Table 5-4).

Operation: The following operations are executed simultaneously by the MUS microcommand.

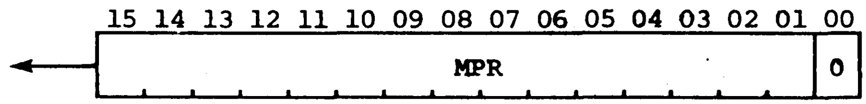


Procedure:

1. Convert MCD to a positive number:



2. Convert MPR to a positive number, shift left and test MPR relative to zero:

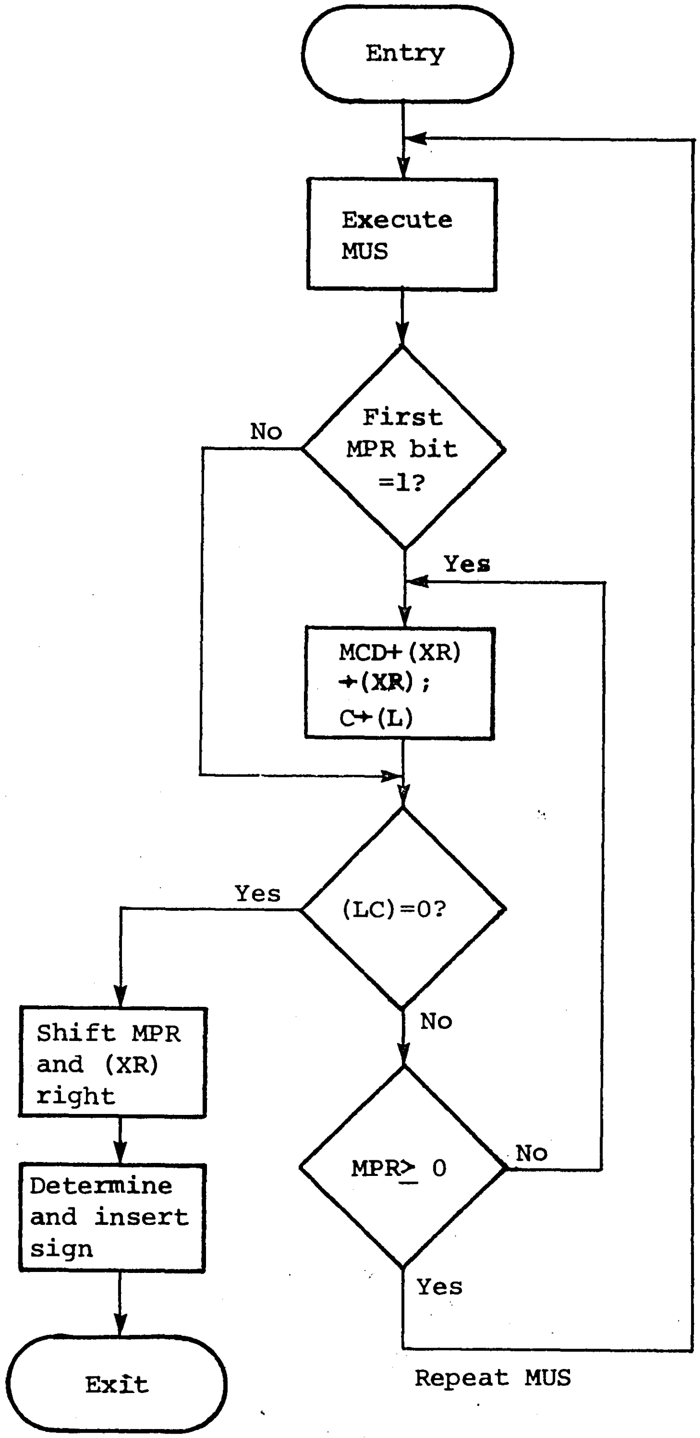


3. Initial conditions:
   (XR)=0
   (L) =0
   (LC)=15 (for a 16-bit multiplication)
4. Program MUS control fields as follows:
   SB = $10 (dynamic branch < 0)
   OP = $19 (MUS)
   DN = MPR address
   NX = $3 (inhibit next microcommand, if branch)
   AO = MPR address
   MC = $6 (add and update L)
   MX = $0 (no operation)
   SO = $6 (double-precision logical open left shift)
   BF = MUS location minus one

   The symbolic microassembler automatically sets up all fields except DN and AO.

5. For the final double-length result, the MPR register and XR must be shifted right one bit after the last iteration. This can be performed using the standard SHF microcommand programmed for a double-precision logical open right shift. The sign of the product must also be determined and inserted.

6. The basic microcommand sequence is illustrated below:



Micro-Condition Codes:

c = AU shift carry
v = shift overflow
z = 1 if AU shift result =0; 0 otherwise
n = 1 if AU shift result most-significant bit = 0; 0 otherwise
p = 1 if AU shift result > 0; 0 otherwise
d = 1 if AU shift result least-significant bit = 0; 0 otherwise

Example:

Multiply the following four-bit numbers (all registers assumed to be four bits):

MPR = 0101
MCD = 0111

MPR	L	XR	LC	Explanation	
0101	0	0000	3	Initial condition	
1010		――――		Shift MPR left; test MPR<0	
	0	0111		Add MCD to (XR)	
0				Add (L) to MPR	
1010			2		MUS
0100	0	1110		Shift MPR and (XR) left	
0					
0100			1		
1001	0	1100			MUS
	1	0011		Add MCD to (XR)	
1					MUS
1010			0	Exit, (LC)=0	
0100		0110			
0010		0011		Shift MPR and (XR) right for final product	

product

5.5.3 Divide Step

The DVS microcommand is a specialized version of the subtract operation (conditional) with an automatic iterative repeat that permits high-speed implementation of a Divide instruction. The fixed execution time is two clock cycles per bit plus the time required to preformat the divisor and dividend, check for overflow, determine the sign of the quotient and format the final result. No additional hardware is required for the high-speed division, since all operations are implemented in control memory.

Mnemonic: DVS $1A

Microcommand Type: Special branch.

Operation: $R = A + B + 1$
If c=1, $R \rightarrow A$

Description: The DVS microcommand executes a two's complement addition of the divisor (DVR) to the more-significant word of a double-precision dividend (DVD). If a carry out is generated by the addition, the result replaces the more-significant word of DVD; otherwise DVD is unchanged.

The carry out must be saved in the L microstatus bit. the microcommand is used in conjunction with a double-length left shift of the A operand and XR on each iteration, with the carry out saved in L shifted into XR. DVS can be automatically repeated using LC. The result is a single-length quotient with a single-length remainder.

Registers Used:

a. DVD more-significant word (DVDM) in a file register designated by the AO field of the DVS microcommand. This register contains the remainder at the end of the complete division operation.
b. DVD less-significant word (DVDL) in XR.
c. DVR in a register designated by the BO field of the DVS microcommand. This register contains the quotient at the end of the complete division operation.
d. LC, which counts the number of iterations performed.
e. The L microstatus bit used to propagate quotient bits into XR.

Procedure:

1. Convert DVD to a 31-bit positive number:

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	DVDM														

XR

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DVDL															

A Operand

2. Convert DVR to a positive number:

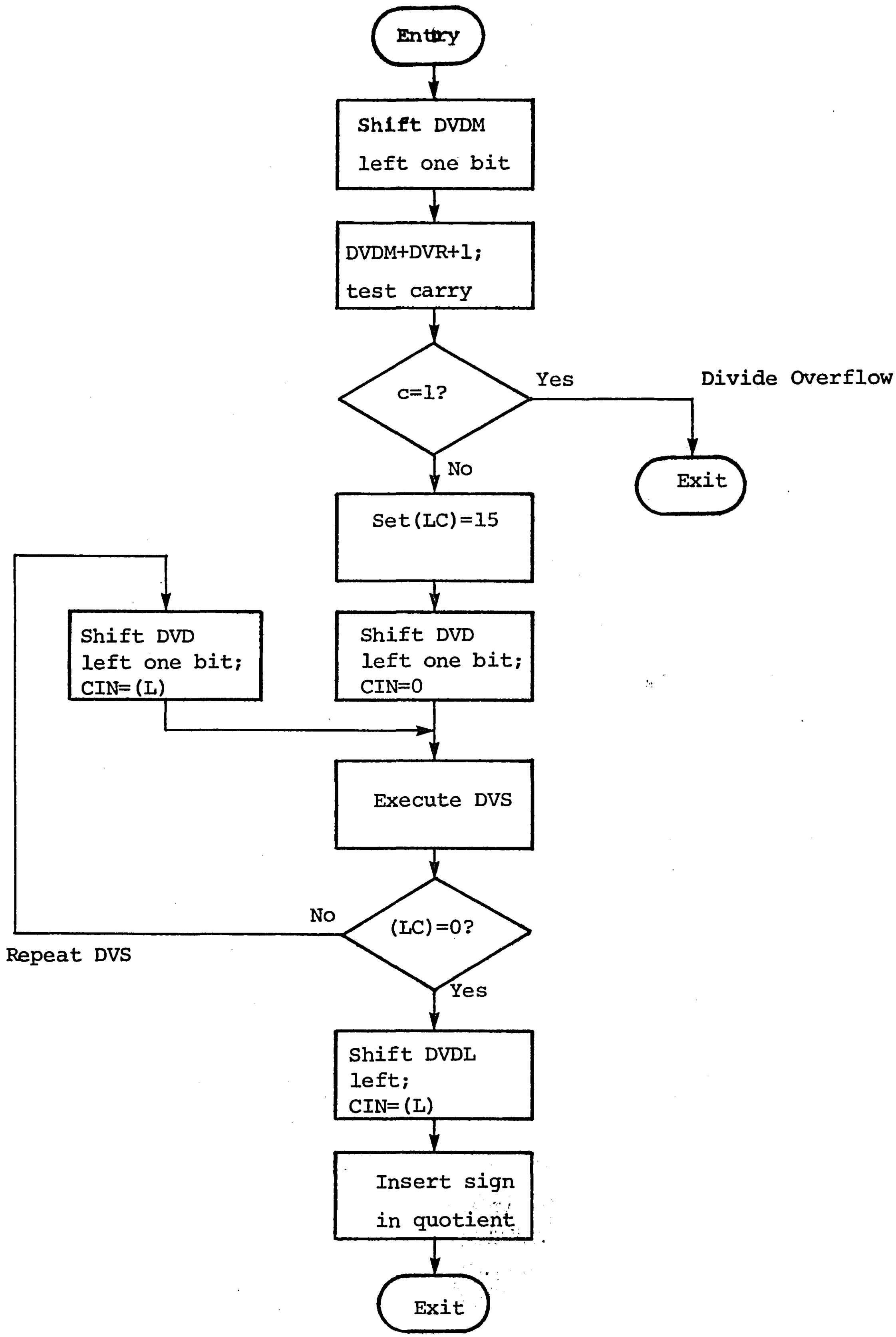
15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	DVR														

B operand

3. Test for DVR ≤2 X DVD. If true, set overflow and exit.
4. If no overflow, shift DVD left and set initial conditions:
   (L)=0
   (LC)=15
5. Program DVS control fields as follows:
   SB = \$03 (branch if (LC)=0 before decrementing)
   OP = \$1A (DVS)
   DN = DVDL
   NX = \$1 (execute next microcommand, if branch)
   AO = DVDM
   MC = \$5 (modify link status; CIN=1)
   MX = \$0 (no operation)
   BO = DVR
   BF = location of DVS

   The symbolic microassembler automatically sets up all fields except DN, AO and BO.
6. DVS is used in conjunction with a double-length left shift on the A operand and XR, with L added to XR.
7. The final double-length result must be left shifted one bit after the final iteration. The quotient is in XR and the remainder in the A-operand source. The sign of the quotient is determined and set separately.

8. The basic microcommand sequence is illustrated below:



Example: Divide the following numbers (all registers are assumed to be four bits):

DVD = 0011 0110

DVR = 0111

L	DVDM	DVDL(XR)	LC	Explanation	
0	0011	0110	3	Initial condition	
0	0110	1100	3	Shift DVD left	
	1001			Add DVR two's complement (no DVD modify)	
	1111			No overflow, continue	DVS
	1101	1000	3	Shift DVD left	
	1001			Add DVR two's complement	
1	0110	1000	2	Modify DVDM and test (LC)	DVS
	1101	0001	2	Shift DVD left and add (L)	
	1001				
1	0110	0001	1		DVS
	1100	0011	1	Shift DVD left and add (L)	
	1001				
1	0101	0011	0	Exit, (LC)=0	DVS
	0101	0111		Shift DVDL left and add (L)	
	Remainder	Quotient			

5.5.4 Test Bit

The TSB microcommand provides the ability to test and conditionally branch on the state of a specified bit in the A operand. The microcommand cannot be a skip type (the K bit, 47, is ignored) and the BF field is always treated as a branch address. The SB field can also specify a separate branch condition. If either the bit test or the SB-field condition is met, the branch occurs. This provides considerable flexibility in performing multiple test operations at high speed.

Mnemonic: TSB $1B

Microcommand Type: Special branch.

Description: The A operand is transferred, unmodified, to the destination. The state of the A-operand bit specified by the SO field is tested. If the bit test condition is met (as specified by the T bit, 43) a branch is made to the location given in the BF field. The SB field can specify an additional branch condition. If either the bit test or the SB condition is met, the branch occurs.

SB Field:

a. The K bit (47) is ignored.
b. The T bit (43) specifies the ONE or ZERO state of both the bit test and the SB-field test (i.e., both must test the same state).
c. The SB-field test conditions are given below. The normal unconditional branch condition is treated as a no-branch. This no-branch must be programmed if only the bit test condition is to be tested.

SB Field Code	Test Condition
$0	loop count equals zero
$1	carry
$2	overflow
$3	zero
$4	negative
$5	positive
$6	odd
$7	unconditional branch (treated as no-branch in TSB)

Microcondition Codes: During the execution of TSB, AU is set to copy AB onto MB, leading to unpredictable (generally meaningless) microcondition codes.

5.5.5 Modify Macrostatus (Optional)

When emulation enhancement circuitry is included, the CPU contains, in addition to the microlevel status in MS, a processor status register (PS) that stores macrolevel conditions, including link, overflow, negative and zero as well as other information on the states of the emulated computer. These status conditions are generated at intermediate times by the emulation microroutines and must be transferred to PS by microcommand. Since PS update can differ for many types of instructions being emulated, the update function can add an excessive number of microcommands to the emulation microroutines.

To provide fast PS update, the emulate table can be programmed to generate a set of PS-update control bits that are specific to the instruction contained in IR. When the MMS microcommand is executed, the PS-update control bits steer the contents of MB directly to the proper PS location. In this way, the microprogram generates proper values for each emulated instruction using only one clock step.

Since each emulated computer requires a different treatment of PS values, the emulate table associated with MMS is unique. In some cases, MMS is not needed at all to meet overall emulation speed objectives. For this reason, MMS is considered an optional microcommand that can be omitted or tailored for a specific emulation task.

Mnemonic: MMS $1C

Microcommand Type: Special branch or special skip.

Description: The A operand is transferred to the destination. Any or all of the least-significant four bits of the A operand can also be transferred to the corresponding least-significant four bits of PS, if specified by the microcommand. The contents of IR are translated into a set of update functions that specify a modification of the least-significant four bits of PS. The update functions are contained in the emulate table. The update functions that can be specified individually for PS bits 03:00 are:

PS Update Function	PS Bit			
	PS03	PS02	PS01	PS00
No change	(PS03)→(PS03)	(PS02)→(PS02)	(PS01)→(PS01)	(PS00)→(PS00)
Reset	0→(PS03)	0→(PS02)	0→(PS01)	0→(PS00)
Set	1→(PS03)	1→(PS02)	1→(PS01)	1→(PS00)
Conditional	A03→(PS03)	A02→(PS02)	A01→(PS01)	A00→(PS00)

The update functions permit each PS bit to be left unaltered, unconditionally reset, unconditionally set or modified by the contents of the corresponding four bits of the A operand, which is routed via MB. If MS is selected as the A-operand source, the MMS microcommand can transfer the L, V, N and Z microstatus bits directly to PS.

Micro-condition Codes:	During the execution of MMS, AU is set to copy AB onto MB, leading to unpredictable (generally meaningless) microcondition codes.

5.5.6 Conditional Memory Access (Optional)

For emulation of a set of instructions involving one or more operands, it is usually desirable to read some operands from memory in a read/restore mode and others in a read/modify/write mode. The read/restore mode is associated with operands that are not modified by the instruction. Examples are:

a. Load (memory to hardware register)
b. Add (memory to hardware register)
c. Compare (memory with hardware register)

For high emulation speed, the address mode and operand fetch operations are generally executed before the specific operation is determined, so the memory access mode is not known at the time the operand fetch cycle is initiated. If a read/restore mode is used in all cases, an extra memory cycle is required to write the modified operand. Use of a read/modify/write operation saves both memory and CPU time.

<u>CMA</u>

The CMA microcommand uses the emulate table to generate a control signal that specifies whether the memory access to the A-operand is to be read/restore or read/modify/write. This is determined by the contents of IR, which holds the current instruction being emulated. Since the table is unique for each emulation, and in some cases may not be required, the CMA microcommand is considered optional.

Mnemonic:	CMA $1D
Microcommand Type:	Special skip or special branch
Description:	The A operand is transferred to the destination. The CMA microcommand automatically generates either a memory read/restore or a memory read/modify/write operation on the MACROBUS. The type of operation is determined by the state of the conditional memory access control bit from the emulate table. The location of the memory word is specified by the contents of the A-operand source. The operation is performed in the word or byte mode, depending on the state of the I/O byte control bit from the emulate table. The word (or byte) read from memory is stored in RR when received.
Micro-condition Codes:	During the execution of CMA, AU is set to copy AB onto MB, leading to unpredictable (generally meaningless) microcondition codes.

Programming: In a special skip-type microcommand, the FN field can generally designate a memory access operation; however, an FN field memory access operation is overridden by the conditional memory access operation specified by the OP field.

CMB

The CMB microcommand performs the same functions for the B operand as CMA performs for the A operand.

Mnemonic: CMB $1E

5.5.7 Decode (Optional)

In emulation microroutines, it is desirable to have a means to modify specific bit fields in a given microcommand, based on the particular instruction being emulated. For example, an add and a subtract microroutine may differ only in that the operands are added or subtracted. By modifying the OP field of the arithmetic microcommand, a common routine can be used. Another example is accessing a particular FR based on a field in the microcommand. The DCD microcommand permits this type of operation to be accomplished directly through use of a decode table that modifies specified bits in the microcommand following the DCD microcommand. The table is set up for the specific emulation and, in some cases, may not be needed. For this reason, the DCD is considered an optional microcommand.

Mnemonic: DCD $1F

Microcommand Type: Special skip or special branch

Description: A zero word is placed on MB. The DCD microcommand selects a 16-bit modifier from the decode table. This word modifies a specified set of bits in the least-significant 16 bits of the next microcommand read from control memory (prior to execution). The modifier and bit fields to be modified are selected using the AO and BO fields of DCD and the contents of the emulate decode register (ER).

The AO and BO fields of DCD are used as follows:

a. Bits 28 and 27 of the AO field select one of four groups of four bits each in ER. The ER bit group selected is taken as the least-significant four bits of an eight-bit address to the decode table.
b. Bits 26 to 24 of the AO field select one of eight possible field modification patterns for the next microcommand read from CM.
c. The BO field is taken as the most-significant four bits of the eight-bit address to the decode table. The 16-bit modifier word fetched from the decode table is ANDed with the least-significant 16 bits in the next microcommand read from CM before it is transferred to CR for execution.

Micro- condition Codes:	During the execution of DCD, AU is set to copy AB onto MB, leading to unpredictable (generally meaningless) microcondition codes.

SECTION 6
MAINTENANCE

6.1 GENERAL

This section describes preventive and corrective maintenance procedures that apply to the Engine. In general, corrective maintenance is limited to isolation of a fault to a specific Engine board, followed by replacement of the board. Troubleshooting may then be used to verify that the suspected board is malfunctioning and to help diagnose the specific problem. Repair should be conducted at the factory or by an authorized Cal Data representative.

6.2 PREVENTIVE MAINTENANCE

The Engine is a reliable solid-state device designed to perform continuously for many years without degredation. Preventive maintenance consists of performing the following tasks every six months:

a. Inspect the boards for damaged wires or components, or other obvious defects.
b. Using a low-pressure source of air (75 psi one foot from the board or 5 kg/cm^2 30 cm from the board), blow off accumulated dust and foreign matter.
c. Check the +5 Vdc input to the Engine. It should be within ±5 percent.

Another aspect of preventive maintenance is proper handling of the unit. The following points should be observed:

a. Always be sure that system power is OFF before installing or removing any board.
b. Install each board with the component side toward the front of the chassis. Check each board for proper orientation before attempting to install it. Because the connectors are keyed, excessive force applied to a reversed board can result in connector damage. Make sure that the board is completely and evenly seated.
c. Insert and remove each board slowly and carefully so that it does not make contact with adjacent boards.
d. Never use components as finger grips; use the grip areas at the corners of the board.
e. To prevent oxides from forming on the gold plating, do not touch connector pins.

6.3 CORRECTIVE MAINTENANCE

Repair of the Engine in the field is not recommended. If a malfunction is detected, replace the board with a spare known to be operating properly and return the malfunctioning board for repair to California Data Processors or an authorized representative.

APPENDIX A
ENGINE ARITHMETIC

A.1 NUMBER REPRESENTATION

In the Cal Data Engine, the AU is implemented to perform both addition and subtraction internally (as opposed to complement addition for the subtraction function). Hence, the dynamic arithmetic condition codes generated (carry out and overflow) and the function of the carry in (CIN) to the AU depend on whether addition or subtraction is performed. Arithmetic operations assume the use of the two's complement representation for negative numbers in the computer, with the state of the most-significant bit representing the sign of the number. The 16-bit single-precision number range of the computer is therefore:

Binary	Hex	Decimal
0111111111111111	7FFF	$2^{15} - 1 = 32{,}767$
.	.	.
.	.	.
.	.	.
0000000000000001	0001	1
0000000000000000	0000	0
1111111111111111	FFFF	-1
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
1000000000000000	8000	$2^{-15} = -32{,}768$

To form the two's complement of a binary number, perform:

$$-|X| = \overline{X} + 1$$

where X is the logical (or one's) complement of the binary number. For example:

$X = 5 = 0101$

$\overline{X} \quad = 1010$ (one's complement)

$+1 \quad = +\underline{0001}$

$-X \quad = 1011$ (two's complement)

A.2 ADDITION

If all negative numbers are represented in two's complement form, then the result of any addition generates the proper result, regardless of the sign of the two operands.

Examples:

(+4) =	0100	(−4) =	1100
+(+2) =	+0010	+(−2) =	+1110
=(+6) =	0110	=(−6) = c	1010

(+4) =	0100	(−4) =	1100
+(−2) =	+1110	+(+2) =	+0010
=(+2) = c	0010	=(−2) =	1110

The notation "c" indicates that a carry output is generated by AU. This carry out is generally of no significance in addition unless the two operands represent something other than the most-significant bits of a multiple-precision set of numbers. In such a case, the carry out bit can be saved as the link (L) bit and added to the next most-significant set of bits when the next step of the multiple-precision addition is performed. For example, suppose that the following two eight-bit numbers are added using a four-bit adder:

(+44) =	0100		1100
+(−23) =	+1110		1001
	c 0000	c	0101
Add link =	+0001 ←		
=(+21) =	0001		0101

In the previous example, none of the additions resulted in an arithmetic overflow (i.e., all results are within the maximum number range possible, which for the four-bit numbers is 2^7-1 < range < 2^{-7}). An overflow occurs if two positive numbers are added with a sum greater than seven:

(+5) =	0101	
+(+4) =	0100	
=(−7) =	1001	(overflow)

The negative seven is an incorrect result, and the overflow is determined by a change of sign to negative when the two positive operands are added. A carry out is not generated.

The carry and overflow condition orders for addition are determined in the CPU by:

$$c = [A15 \cap \overline{R15}] \cup [B15 \cap \overline{R15}] \cup [A15 \cap B15]$$

$$v = [A15 \cap B15 \cap \overline{R15}] \cup [\overline{A15} \cap \overline{B15} \cap R15]$$

where A15, B15 and R15 are the most-significant bits of the A operand, B operand and result, respectively. The c and v microcondition codes can be stored in the microstatus register (MS) L and V bits, respectively, using the MC field of the microcommand.

In the CPU, the carry input (CIN) to AU can also be specified by the MC field of the microcommand. The states can be programmed as:
CIN = 1, CIN = 0 or CIN = (L).

The ADD microcommand is A+B+CIN, where CIN is the carry in under MC field control. Thus, it is possible to add the fixed constants ONE or ZERO to the result, or to add the state of the L bit (which can contain the carry propagation for multiple-precision addition, for example).

A.3 SUBTRACTION

The CPU performs true binary subtraction as well as addition. This provides considerably greater flexibility in implementing the arithmetic microcommands than would the usual use of complement addition.

Examples:

$$
\begin{array}{rlrrlr}
(+4) & = & 0100 & (-4) & = & 1100 \\
-(+2) & = & \underline{-0010} & -(-2) & = & \underline{-1110} \\
=(+2) & = & 0010 & =(-2) & = c & 1110
\end{array}
$$

$$
\begin{array}{rlrrlr}
(+4) & = & 0100 & (-4) & = & 1100 \\
-(-2) & = & \underline{-1110} & -(+2) & = & \underline{-0010} \\
=(+6) & = c & 0110 & =(-6) & = & 1010
\end{array}
$$

The notation "c" in this case indicates that a borrow output is generated by the subtractor. The borrow out is of significance only if the two operands represent something other than the most-significant bits of a multiple-precision set of numbers. In such a case, the borrow-out bit can be saved as the link (L) bit and then subtracted from the result of subtracting the next-most-significant bits. For example, suppose that the following two eight-bit numbers are subtracted using a four-bit subtractor:

$$
\begin{array}{rlrcr}
(+60) & = & 0011 & & 1100 \\
-(+30) & = & \underline{-0001} & c & \underline{1110} \\
 & & 0010 & & 1110 \\
\text{Subtraction Link} & = & \underline{-0001} & \leftarrow & \\
=(+30) & = & 0001 & & 1110
\end{array}
$$

Overflow results from subtraction, as from addition, when the result is outside the range of the number system $(2^{7}-1 < \text{result} < 2^{-7}$ for a four-bit range). The borrow and overflow condition codes for subtraction are determined in the CPU by:

$$c = [\overline{A15} \cap B15] \cup [\overline{A15} \cap R15] \cup [B15 \cap R15]$$

$$v = [\overline{A15} \cap B15 \cap R15] \cup [A15 \cap \overline{B15} \cap \overline{R15}]$$

The borrow is designated as c in the computer. The microcondition codes can be stored in the microstatus register L and V bits, respectively, using the MC field of the microcommand.

In the CPU, the borrow input (also CIN) to AU can also be specified by the MC field of the microcommand. The states can be programmed as:
CIN = 1, CIN = 0 or CIN = (L).

The SUB microcommand is A-B-CIN, where CIN is the borrow in under MC field control.

APPENDIX B
FIXED MEMORY ASSIGNMENTS

System interrupt vectors (two words per vector) are given in Table B-1. Only those vectors used by the Cal Data computer and standard options are given. Other vector locations are reserved. Users should observe these assignments if full software compatibility is to be retained.

Table B-1. Interrupt Vectors

Octal Address	Use
000	Reserved
004	I/O channel time-out error
010	Reserved instruction vector
014	Debug trap vector
024	Power-failure trap vector
034	"Trap" trap vector
060	Serial channel in (BR4)
064	Serial channel out (BR4)
070	High-speed reader (BR4)
074	High-speed punch (BR4)
100	Line-Frequency Clock (BR6)
200	Line printer (BR4)
244	Floating-point error
250	Memory-management abort
254	Macropanel interrupt
300	Start of floating vectors

APPENDIX C
CONNECTOR PIN ASSIGNMENTS

Table C-1. Connector A Pin Assignments, MACROBUS

Name	Signal	Pin	Pin	Signal	Name
Initialize	* BUS INIT-L	A1	A2	+5V	+5 Vdc
Interrupt	* BUS INTR-L	B1	B2	GND	Ground
Data 00	* BUS D00-L	C1	C2	GND	Ground
Data 02	* BUS D02-L	D1	D2	*BUS D01-L	Data 01
Data 04	* BUS D04-L	E1	E2	*BUS D03-L	Data 03
Data 06	* BUS D06-L	F1	F2	*BUS D05-L	Data 05
Data 08	* BUS D08-L	G1	H2	*BUS D07-L	Data 07
Data 10	* BUS D10-L	J1	J2	*BUS D09-L	Data 09
Data 12	* BUS D12-L	K1	K2	*BUS D11-L	Data 11
Data 14	* BUS D14-L	L1	L2	*BUS D13-L	Data 13
Parity Bit Low	* BUS PA-L	M1	M2	*BUS D15-L	Data 15
Ground	GND	N1	N2	*BUS PB-L	Parity Bit High
Ground	GND	P1	P2	*BUS BBSY-L	Bus Busy
Ground	GND	R1	R2	*BUS SACK-L	Selection Acknowledgement
Ground	GND	S1	S2	*BUS NPR-L	Nonprocessor Request
Ground	GND	T1	T2	*BUS BR7-L	Bus Request 7
Nonprocessor Grant	* BUS NPG-H	U1	U2	*BUS BR6-L	Bus Request 6
Bus Grant 7	* BUS BG7-H	V1	V2	GND	Ground

* These signals are assigned on the backplane but are not used on this assembly.

Table C-2. Connector B Pin Assignments, MACROBUS

Name	Signal	Pin	Pin	Signal	Name
Bus Grant 6	* BUS BG6-H	A1	A2	+5V	+5 Vdc
Bus Grant 5	* BUS BG5-H	B1	B2	GND	Ground
Bus Request 5	* BUS BR5-L	C1	C2	GND	Ground
Ground	GND	D1	D2	* BUS BR4-L	Bus Request 4
Ground	GND	E1	E2	* BUS BG4-H	Bus Grant 4
AC Low	* BUS ACLO-L	F1	F2	* BUS DCLO-L	DC Low
Address 01	* BUS A01-L	H1	H2	* BUS A00-L	Address 00
Address 03	* BUS A03-L	J1	J2	* BUS A02-L	Address 02
Address 05	* BUS A05-L	K1	K2	* BUS A04-L	Address 04
Address 07	* BUS A07-L	L1	L2	* BUS A06-L	Address 06
Address 09	* BUS A09-L	M1	M2	* BUS A08-L	Address 08
Address 11	* BUS A11-L	N1	N2	* BUS A10-L	Address 10
Address 13	* BUS A13-L	P1	P2	* BUS A12-L	Address 12
Address 15	* BUS A15-L	R1	R2	* BUS A14-L	Address 14
Address 17	* BUS A17-L	S1	S2	* BUS A16-L	Address 16
Ground	GND	T1	T2	* BUS C1-L	Control 1
Slave Synchronization	* BUS SSYN-L	U1	U2	* BUS C0-L	Control 0
Master Synchronization	* BUS MSYN-L	V1	V2	GND	Ground

* These signals are assigned on the backplane but are not used on this assembly.

Table C-3. Connector C Pin Assignments

Name	Signal	Pin	Pin	Signal	Name
M Bus 00	MB000-L	A1	A2	+5V	+5 Vdc
M Bus 01	MB001-L	B2	B2	* -15V	-15 Vdc
M Bus 02	MB002-L	C1	C2	GND	Ground
M Bus 03	MB003-L	D1	D2	MB004-L	M Bus 04
M Bus 05	MB005-L	E1	E2	MB006-L	M Bus 06
M Bus 07	MB007-L	F1	F2	MB008-L	M Bus 08
M Bus 09	MB009-L	H1	H2	MB010-L	M Bus 10
M Bus 11	MB011-L	J1	J2	MB012-L	M Bus 12
M Bus 13	MB013-L	K1	K2	MB014-L	M Bus 14
M Bus 15	MB015-L	L1	L2	AB000-H	A Bus 00
A Bus 01	AB001-H	M1	M2	AB002-H	A Bus 02
A Bus 03	AB003-H	N1	N2	AB004-H	A Bus 04
A Bus 05	AB005-H	P1	P2	AB006-H	A Bus 06
A Bus 07	AB007-H	R1	R2	AB008-H	A Bus 08
A Bus 09	AB009-H	S1	S2	AB010-H	A Bus 10
Ground	GND	T1	T2	AB011-H	A Bus 11
A Bus 13	AB013-H	U1	U2	AB012-H	A Bus 12
A Bus 15	AB015-H	V1	V2	AB014-H	A Bus 14

* These signals are assigned on the backplane but are not used on this assembly.

Table C-4. Connector D Pin Assignments

Name	Signal	Pin	Pin	Signal	Name
Power Failure Interrupt	2 PFINT-H	A1	A2	+5V	+5 Vdc
Halt Interrupt	2 HLINT-H	B1	B2	* -15V	-15 Vdc
Data Switch 16	* DS16-H	C1	C2	GND	Ground
Data Switch 17	* DS17-H	D1	D2	* LTCL-L	Line-Frequency Clock
Virtual Address	* VIRTAD-H	E1	E2	* PBBSY-L	Processor Bus Busy
Control Count 00	2 CC000-L	F1	F2	* HALTP-L	Panel Halt
Control Count 01	2 CC001-L	H1	H2	* MSR15-L	Microstatus Register 15
Control Count 02	2 CC002-L	J1	J2	RESET-L	Reset
Control Count 03	2 CC003-L	K1	K2	* BUS BG7-IN	Bus Grant 7 In
Control Count 04	2 CC004-L	L1	L2	* BUS BG7-OUT	Bus Grant 7 Out
Control Count 05	2 CC005-L	M1	M2	* BUS BG6-IN	Bus Grant 6 In
Control Count 06	2 CC006-L	N1	N2	* BUS BG6-OUT	Bus Grant 6 Out
Control Count 07	2 CC007-L	P1	P2	* BUS BG5-IN	Bus Grant 5 In
Control Count 08	2 CC008-L	R1	R2	* BUS BG5-OUT	Bus Grant 5 Out
Control Count 09	2 CC009-L	S1	S2	* BUS BG4-IN	Bus Grant 4 In
Ground	GND	T1	T2	* BUS BG4-OUT	Bus Grant 4 Out
Control Count 10	2 CC010-L	U1	U2	* BUS NPG-IN	Nonprocessor Grant In
Control Count 11	2 CC011-L	V1	V2	* BUS NPG-OUT	Nonprocessor Grant Out

* These signals are assigned on the backplane but are not used on this assembly.
2 = Signal used only on Engine 2.

Table C-5. Connector E Pin Assignments

Name	Signal	Pin	Pin	Signal	Name
Control Memory 00	CM000-H	A1	A2	+5V	+5 Vdc
Control Memory 01	CM001-H	B1	B2	* -15V	-15 Vdc
Control Memory 02	CM002-H	C1	C2	GND	Ground
Control Memory 03	CM003-H	D1	D2	CM004-H	Control Memory 04
Control Memory 05	CM005-H	E1	E2	CM006-H	Control Memory 06
Control Memory 07	CM007-H	F1	F2	1 EMINH-L	Emulate Inhibit
Control Memory 09	CM009-H	H1	H2	CM008-H	Control Memory 08
Control Memory 11	CM011-H	J1	J2	CM010-H	Control Memory 10
Decode Address 00	2 DAD00-H	K1	K2	CM012-H	Control Memory 12
Control Memory 13	CM013-H	L1	L2	CM014-H	Control Memory 14
Control Memory 15	CM015-H	M1	M2	2 DAD01-H	Decode Address 01
Control Memory 17	2 CM017-H	N1	N2	2 CM016-H	Control Memory 16
Control Memory 19	2 CM019-H	P1	P2	2 CM018-H	Control Memory 18
Switch Register 0	* SR0-L	R1	R2	CM020-H	Control Memory 20
Control Memory 21	CM021-H	S1	S2	CM022-H	Control Memory 22
Ground	GND	T1	T2	CM024-H	Control Memory 24
Control Memory 23	CM023-H	U1	U2	CM026-H	Control Memory 26
Control Memory 25	CM025-H	V1	V2	CM027-H	Control Memory 27

* These signals are assigned on the backplane but are not used on this assembly.

1 = Signal used only on Engine 1.

2 = Signal used only on Engine 2.

Table C-6. Connector F Pin Assignments

Name	Signal	Pin	Pin	Signal	Name
Control Memory 28	CM028-H	A1	A2	+5V	+5 Vdc
Control Memory 29	CM029-H	B1	B2	* -15V	-15 Vdc
Control Memory 31	2 CM031-H	C1	C2	GND	Ground
Control Memory 31	2 CM030-H	D1	D2	CM032-H	Control Memory 32
Control Memory 33	CM033-H	E1	E2	CM034-H	Control Memory 34
Control Memory 35	CM035-H	F1	F2	2 DAD02-H	Decode Address 02
Control Memory 37	CM037-H	H1	H2	CM036-H	Control Memory 36
Control Memory 39	CM039-H	J1	J2	CM038-H	Control Memory 38
Instruction Repeat	IRPTE-L	K1	K2	CM040-H	Control Memory 40
Control Memory 41	CM041-H	L1	L2	CM042-H	Control Memory 42
Control Memory 43	CM043-H	M1	M2	2 CPEN-L	Control Panel Enable
Control Memory 45	CM045-H	N1	N2	CM044-H	Control Memory 44
Control Memory 47	CM047-H	P1	P2	CM046-H	Control Memory 46
Decode Address 03	2 DAD03-H	R1	R2	2 ACMSL-L	Alterable Control Memory Select
	Reserved	S1	S2	2 AUXRM-L	Auxiliary ROM Select
Ground	GND	T1	T2	IRINH-L	Instruction Inhibit
	Reserved	U1	U2	IWAIT-L	Instruction Wait
System Clock	SYSCK-L	V1	V2	GND	Ground

*These signals are assigned on the backplane but are not used on this assembly

2 = Signal used only on Engine 2.

Table C-7. Connector J1 Pin Assignments

Name	Signal	Pin	Pin	Signal	Name
Skip	SKIPP-L	1A	1B	* EMA00-H	Emulate Address 00
AR Write Enable	2 ARWEN-L	2A	2B	* EMA01-H	Emulate Address 01
Stack Limit Write Enable	2 SLWEN-L	3A	3B	* EMA02-H	Emulate Address 02
Slave Synchronization Error	* SSYER-H	4A	4B	* EMA03-H	Emulate Address 03
Double Slave Synchronization Error	* DSYER-H	5A	5B	* EMA04-H	Emulate Address 04
Load Special Function	* LDSPF-H	6A	6B	* EMA05-H	Emulate Address 05
Fatal Interrupt	FINTP-L	7A	7B	* EMA06-H	Emulate Address 06
Special Function	SPFNC-H	8A	8B	* EMA07-H	Emulate Address 07
Panel Halt	* HALTP-L	9A	9B	Reserved	
	Reserved	10A	10B	PSSEL-L	Program Status Select
Carry	1 CARRY-H	11A	11B	Reserved	
	Reserved	12A	12B	Reserved	
Address Error	* ADERR-H	13A	13B	Reserved	
Program Status 03	* PS003-L	14A	14B	Reserved	
	Reserved	15A	15B	2 XD007-L	Inhibit Destination File 0 to 7
	Reserved	16A	16B	2 XD815-L	Inhibit Destination File 8 to 15
	Reserved	17A	17B	2 XB815-L	Inhibit B-Field File 8 to 15
Control Count Write Enable	CCWEN-H	18A	18B	2 XB007-L	Inhibit B-Field File 0 to 7
Static Condition	STATIC-L	19A	19B	LITRL-L	Literal
Master Synchronization	* MSYN-H	20A	20B	PLUS1-L	Plus 1
Special Function 04	1 SPF04-L	21A	21B	2 PSWEN-L	Processor Status Write Enable
B Bus Inhibit	1 BBINH-L	22A	22B	2 IRWEN-L	IR Write Enable
B Bus 01	BB001-H	23A	23B	BB000-H	B Bus 00
B Bus 03	BB003-H	24A	24B	BB002-H	B Bus 02
B Bus 05	BB005-H	25A	25B	BB004-H	B Bus 04
B Bus 07	BB007-H	26A	26B	BB006-H	B Bus 06
B Bus 09	BB009-H	27A	27B	BB008-H	B Bus 08
B Bus 11	BB011-H	28A	28B	BB010-H	B Bus 10
B Bus 13	BB013-H	29A	29B	BB012-H	B Bus 12
B Bus 15	BB015-H	30A	30B	BB014-H	B Bus 14

*These signals are assigned on the small processor interconnection board but are not used on this assembly.

1 = Signal used only on Engine 1.

2 = Signal used only on Engine 2.

Table C-8. Connector J2 Pin Assignments

Name	Signal	Pin	Pin	Signal	Name
Load CC Register	LOADC-L	1A	1B	Reserved	
Bus Request	* BREQ-H	2A	2B	MINTP-L	Microinterrupt
Bus Grant	* BGRNT-L	3A	3B	BYTDA-L	Byte Data
Bus Grant Enable	* BGEN-H	4A	4B	Reserved	
Memory Management Inhibit	* MMINH-L	5A	5B	* MARLD-H	Management Address Load
Data Inhibit	* DAINH-L	6A	6B	2 CCCEN-H	CC Count Enable
Special Function 7	1 SPF07-L	7A	7B	1 SPR1A-L	Special Register 1A
Special Function 5	1 SPF05-L	8A	8B	1 SPR19-L	Special Register 19
Special Function 6	1 SPF06-L	9A	9B	1 SPR1B-L	Special Register 1B
Special Function Decode	* SPFNC-H	10A	10B	1 MLTPY-L	Multiply
Inhibit B Field	* INHBF-L	11A	11B	1 ENSPF-H	Enable Special Function
Emulate	EMLAT-H	12A	12B	1 CR008-H	Microcommand Register 08
Power Failure	* PFAIL-L	13A	13B	Reserved	
AU Carry In	1 AUCIN-L	14A	14B	Reserved	
Write	* WRITE-L	15A	15B	2 FILE6-H	File 6
IR Read	* IRERD-H	16A	16B	2 XA815-L	Inhibit A-Field File 8 to 15
Interrupt	* INTR-H	17A	17B	2 XA007-L	Inhibit A-Field File 0 to 7
Memory Management C0	* MMC0-L	18A	18B	2 RSTRA-L	Restore A
Memory Management C1	* MMC1-L	19A	19B	* YELLW-L	Yellow
Microcommand Register 07	2 CR007-H	20A	20B	1 BYTMD-L	Byte Mode
Stack Limit Interrupt	1 SLINT-H	21A	21B	1 MS006-H	Microstatus Register 06
DR Write Enable	2 DRWEN-L	22A	22B	2 RRWEN-L	RR Write Enable
Emulate Instruction Address 01	2 EIA001-H	23A	23B	2 EIA000-H	Emulate Instruction Address 00
Emulate Instruction Address 03	2 EIA003-H	24A	24B	2 EIA002-H	Emulate Instruction Address 02
Emulate Instruction Address 05	2 EIA005-H	25A	25B	2 EIA004-H	Emulate Instruction Address 04
Emulate Instruction Address 07	2 EIA007-H	26A	26B	2 EIA006-H	Emulate Instruction Address 06
Emulate Instruction Address 09	* EIA009-H	27A	27B	2 EIA008-H	Emulate Instruction Address 08
Emulate Instruction Address 11	* EIA011-H	28A	28B	* EIA010-H	Emulate Instruction Address 10
Emulate Instruction Address 13	* EIA013-H	29A	29B	* EIA012-H	Emulate Instruction Address 12
Emulate Instruction Address 15	* EIA015-H	30A	30B	* EIA014-H	Emulate Instruction Address 14

*These signals are assigned on the small processor interconnection board but are not used on this assembly.

1 = Signal used only on Engine 1.

2 = Signal used only on Engine 2.