ECLIPSE MV/10000™ System Functional Characteristics

DataGeneral

Notice

Ordering No. 014-000724
© Data General Corporation, 1983
All Rights Reserved
Printed in the United States of America
Rev. 00, April 1983

Preface

This book describes the functional characterisitics of the ECLIPSE MV/10000^{TM} computer system. It is aimed at assembly language programmers familiar with *Principles of Operation, 32-bit ECLIPSE® Systems* (DGC No. 014-000704) and *ECLIPSE® C/350 Principles of Operation* (DGC No. 014-000610). For ease of use, this manual maps by chapter to *Principles of Operation, 32-bit ECLIPSE® Systems.*

Manual Organization

This manual has nine chapters and seven appendices.

Chapter 1, "Technical Summary," explains the system components and functions available on the ECLIPSE MV/10000^{TM} computer.

Chapter 2, "Fixed-Point Instruction Summary," summarizes fixed-point formats and instructions.

Chapter 3, "Floating-Point Instruction Summary," summarizes floating-point formats and instructions.

Chapter 4, "Stack Management Instruction Summary," summarizes the wide stack instructions.

Chapter 5, "Program Flow Management," explains program flow, interrupt handling, and fault handling.

Chapter 6, "Queue Management Instruction Summary," summarizes the queue instructions.

Chapter 7, "Device Management," describes the ECLIPSE MV/10000 I/O devices and applicable instructions.

Chapter 8, "Memory and System Management," presents the ECLIPSE MV/10000 privileged instructions and related information for the operating system designer.

Chapter 9, "C/350 Programming," discusses ECLIPSE C/350 programming compatibility.

Appendix A lists the instructions unique to the ECLIPSE MV/10000 computer alphabetically.

Appendix B presents the typical execution time for each ECLIPSE MV/10000 instruction.

Appendix C presents tabular data for the various programmer-accessible registers.

Appendix D lists the reserved memory locations for page zero and shows the formats for the context block.

Appendix E lists standard Data General I/O device codes.

Appendix F is a tabulation of the contents of Accumulator 1 for protection and nonprotection faults.

Appendix G describes the operation of and format for the Load Control Store instruction.

Related Manuals

Other manuals useful in conjunction with the MV/10000 are as follows:

Principles of Operation, 32-Bit ECLIPSE® Systems, Programmer's Reference Series (DGC No. 014-000704)

ECLIPSE® C/350, Principles of Operation (DGC No. 014-000610)

ECLIPSE® MV/10000, Product Summary Series (DGC No. 014-000737)

ECLIPSE® MV/Family Instruction Reference Booklet (DGC No. 014-000702)

Intelligent Asynchronous Controller, Programmer's Reference Series (DGC No. 014-000703)

Data General Communications Subsystems, Product Summary Series (DGC No. 014-000635)

Programmer's Reference Manual -- Peripherals (DGC No. 015-000021)

Learning to Use AOS/VS (DGC No. 093-006931)

AOS/VS Macroassembler Reference Manual (DGC No. 093-000242)

AOS/VS Programmer's Manual (DGC No. 093-000241)

Conventions and Abbreviations

This manual uses the following conventions and abbreviations:

[]	Square brackets indicate an optional argument. Omit the square brackets when you include an optional argument with an Assembler statement.
ARGUMENT	Uppercase characters indicate a literal argument in an Assembler statement. When you include a literal argument with an Assembler statement, use the exact form.
argument and/or *argument*	Lowercase or italic characters indicate a variable argument in an Assembler statement. When you include the argument with an Assembler statement, substitute a literal value for the variable argument.
ac	Indicates a fixed-point accumulator.
acs	Indicates a source fixed-point accumulator.
acd	Indicates a destination fixed-point accumulator.
fac	Indicates a floating-point accumulator.
facs	Indicates a source floating-point accumulator.
facd	Indicates a destination floating-point accumulator.

Table of Contents

Illustrations

Tables

Chapter 1
Technical Summary

This chapter introduces the ECLIPSE MV/10000^{TM} computer system and describes initial processor conditions.

The ECLIPSE MV/10000^{TM} computer system is a general-purpose 32-bit data processing system that supports the complete 32-bit instruction set described in *Principles of Operation, 32-Bit* ECLIPSE® *Systems* (DGC No. 014-000704). In addition, the ECLIPSE MV/10000 computer system retains substantial hardware and software compatibility with 16-bit ECLIPSE® systems. (Kernel 16-bit operating system instructions such as SYC, VCT, and LMP are not supported.)

The MV/10000 system operates in the manner described in *Principles of Operation, 32-Bit* ECLIPSE® *Systems.*

System Overview

The physical MV/10000 system, shown in Figure 1.1, incorporates four main systems:

- The *central processing unit* (CPU), which consists of the instruction processor for decoding and executing instructions; the arithmetic processor for manipulation of data; and the address translator for logical to physical address translation.
- The *memory system,* which consists of a system cache that contains 1024 16-byte blocks and functions as a look-ahead/look-behind buffer; and up to eight memory modules of 2 Mbytes each.
- The *input/output system,* which consists of two I/O channels that support distributed processors for asynchronous and synchronous communications.
- The *system control processor* (SCP), a soft system console that performs diagnostic and operator-controlled functions.

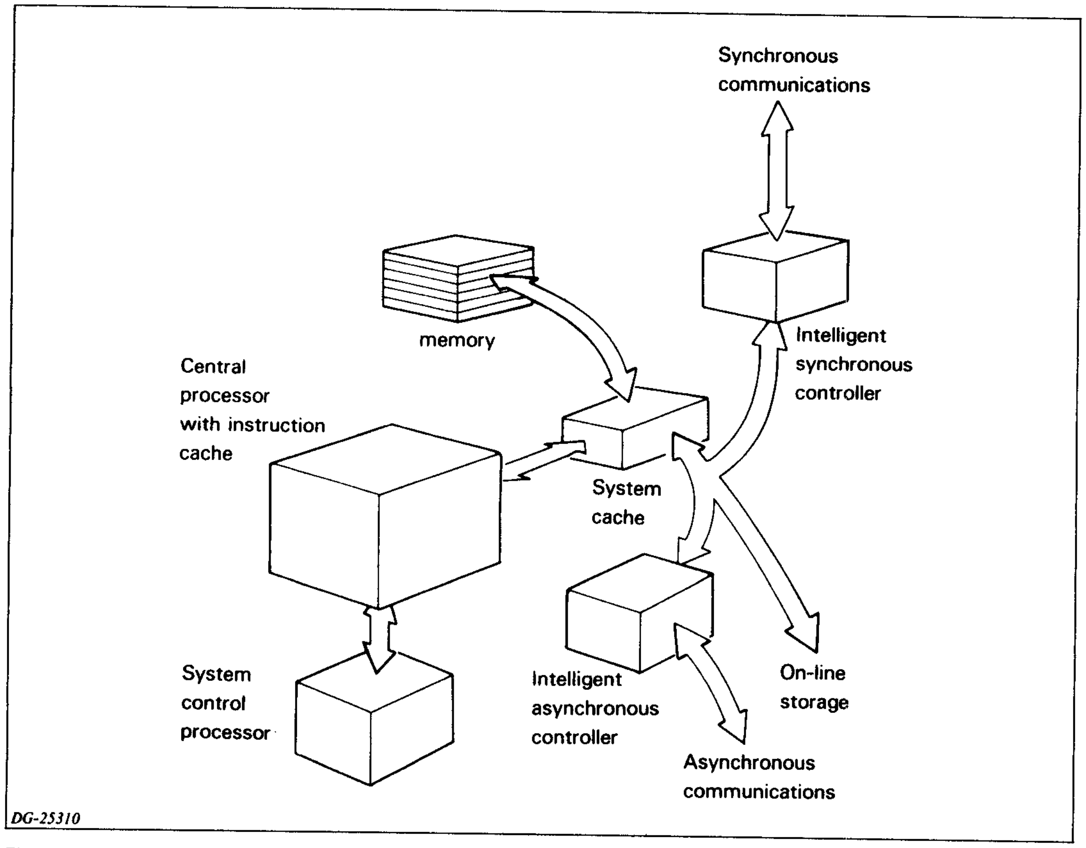

Synchronous communications
memory
Intelligent synchronous controller
Central processor with instruction cache
System cache
System control processor
Intelligent asynchronous controller
On-line storage
Asynchronous communications
DG-25310

Figure 1.1 The ECLIPSE MV/10000 system

Central Processing Unit

The CPU of the MV/10000 system executes all user programs and translates all virtual memory references into physical addresses. The CPU consists of a pipelined instruction processor, a high-speed arithmetic processor, a microsequencer, an address generator, and an address translator.

Instruction Processor

The instruction processor decodes instructions for execution. Its main component is the *instruction cache,* which provides input to the instruction decoder.

The 4-Kbyte instruction cache has 256 16-byte blocks and maps directly to the system cache. (See Figure 1.2 in the section entitled "Memory System.")

The 16 bytes of the instruction cache blocks correspond to the 16 bytes of the system cache blocks. The instruction cache block contains specific information: block 0 can contain any block 0 in the system cache, block 1 can contain any block 1 in the system cache, and so on.

Because of its look-ahead/look-behind capability, the instruction cache speeds program execution.

To execute an instruction, the instruction processor performs the following four steps:

1. Fetches an instruction from the instruction cache.
2. Parses the instruction opcode to obtain the starting address of the microcode routine and collects operand information.
3. Fetches the microinstruction and performs any effective addressing required.
4. Executes the microinstruction.

Four instructions — one instruction per step — can be in the pipeline at any one time.

Arithmetic Processor

The arithmetic processor manipulates floating-point numbers, fixed-point quantities, and addresses.

The MV/10000 system contains four 32-bit fixed-point accumulators. The ECLIPSE C/350 16-bit fixed-point accumulators correspond to bits 16 through 31 of the MV/10000 accumulators.

The program counter (PC) is 31 bits wide. Bits 1 through 3 specify the current segment of execution, and bits 4 through 31 specify an address in the segment.

Four floating-point accumulators, each 64 bits wide, contain the sign, the exponent, and the mantissa of a single- or double-precision floating-point operand. These four registers are identical to the C/350 floating-point registers. The MV/10000 floating-point status register (FPSR) is 64 bits wide.

Four 32-bit registers govern the MV/10000 wide stack: the wide stack pointer (WSP), the wide frame pointer (WFP), the wide stack limit (WSL), and the wide stack base (WSB). Maintaining the stack in hardware speeds up stack management operations.

Address Translator

The MV/10000 system has 4 Gbytes of logical memory and up to 16 Mbytes of physical memory. Because the logical address space is so much larger than the physical address space, the MV/10000 computer stores logical memory on disk in 2-Kbyte units called *pages*. When a process needs a page on disk, it moves the page to physical memory for manipulation. This page-swapping system is called a *demand-page* system. The MV/10000 system also contains an address translator that converts the logical address of a piece of data into a physical address in memory.

To avoid referring to a page table for every memory reference, the address translator maintains a table of address translations and access privileges for 1024 recently referenced pages — 128 per segment.

The address translator controls two memory management bits for each page: the *modified bit* and the *referenced bit*. The operating system uses these bits during *page faults*. (See Chapter 8 for a discussion of page faults.)

The address translator performs all the hardware checks required by the protection system. These checks include access validation, page validation, and ring crossing validation. If any of the checks fails, the address translator initiates a protection fault to the operating system. For more information on protection checks, refer to *Principles of Operation, 32-bit ECLIPSE® Systems*.

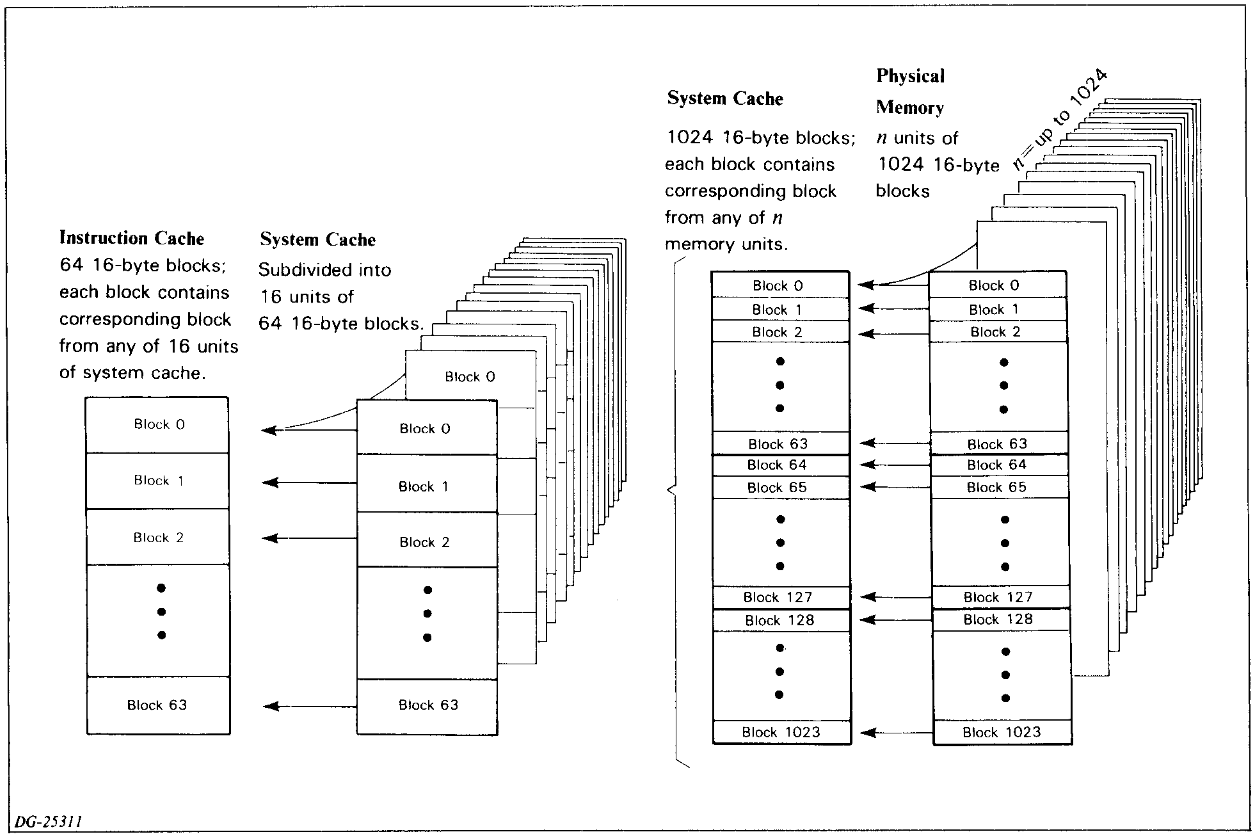

Instruction Cache
64 16-byte blocks; each block contains corresponding block from any of 16 units of system cache.
System Cache
Subdivided into 16 units of 64 16-byte blocks.
System Cache
1024 16-byte blocks; each block contains corresponding block from any of n memory units.
Physical Memory
n units of 1024 16-byte blocks
n = up to 1024
Block 0
Block 1
Block 2
Block 63
Block 64
Block 65
Block 127
Block 128
Block 1023
DG-25311

Figure 1.2 Memory, system cache, and instruction cache mapping

Memory System

The MV/10000 memory system is block-oriented. This means that the system elements expect and manipulate uniform data sizes and formats. These elements transfer data to one another in 16-byte blocks (four successive double words).

The major elements of the memory system are the system cache and the memory modules.

System Cache

The system cache functions as both a look-ahead and a look-behind buffer for the system, reducing the time that both the CPU and the input/output (I/O) system need to access main memory.

The system cache contains 1024 16-byte blocks, each directly mapped to main memory locations. This means that any block in the system cache can contain 16 contiguous bytes from main memory. Note that the system cache blocks cannot contain arbitrary locations from memory.

The memory consists of up to 1024 units, each unit containing 1024 16-byte blocks. Block 0 in the system cache can contain block 0 of any unit in main memory; block 1 in the system cache can contain block 1 of any unit in main memory; and so on. Figure 1.2 illustrates memory, system cache, and instruction cache mapping.

When a process makes a memory reference to block *n* of unit *m* in main memory, the system cache loads the 16-byte memory block containing the referenced data into system cache block *n*. This memory block remains in the system cache until the process makes a new memory reference to block *n* of some other unit in main memory — for example, unit *j*.

When a process makes a reference to block *n* of unit *j*, the system cache examines the cache block modified bit of block *n* of unit *m* (the block currently in the cache). If the cache block modified bit is 1, the system cache writes block *n* of unit *m* back into main memory and then loads the new block *n* of unit *j* into system cache block *n*. If the cache block modified bit is 0, the system cache overwrites the current contents of system cache block *n* with those of block *n* of unit *j*. (See Figure 1.3.)

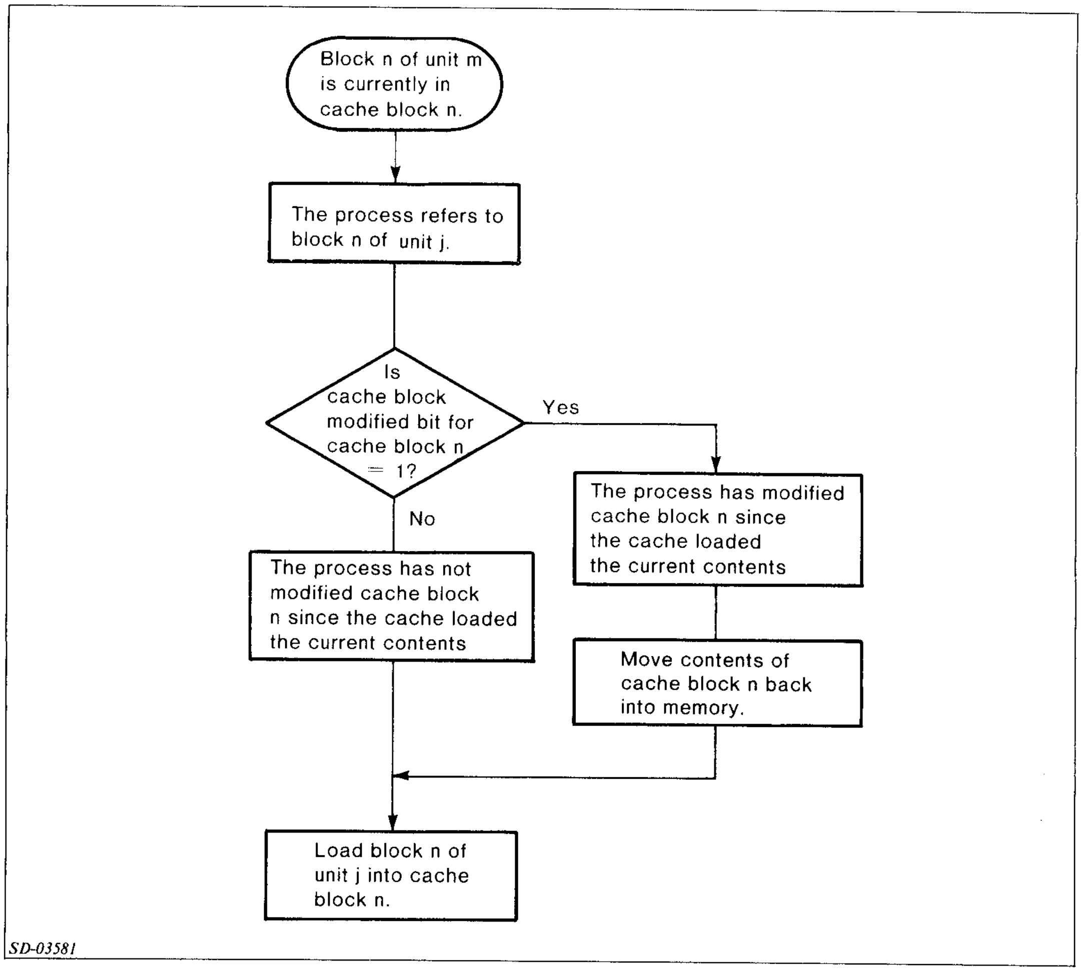

Block n of unit m is currently in cache block n.
The process refers to block n of unit j.
Is cache block modified bit for cache block n = 1?
Yes
No
The process has modified cache block n since the cache loaded the current contents
The process has not modified cache block n since the cache loaded the current contents
Move contents of cache block n back into memory.
Load block n of unit j into cache block n.
SD-03581

Figure 1.3 System cache during memory reference

The memory system contains two ports: one for the CPU and one for direct transfers between memory and the I/O system. The organization of the system cache is such that simultaneous data transfers can occur both between main memory and the CPU and between main memory and the I/O subsystem.

Memory Modules

The MV/10000 system supports up to eight dynamic random-access memory (RAM) modules of 2 Mbytes each. Each module contains 512K double words; each double word is 4 bytes long.

Every memory module consists of two independent planes, each containing 1 Mbyte. Each plane contains every other double word. With a 2-Mbyte memory module, for example, plane 0 contains the double words 0-1 and 4-5; plane 1 contains the double words 2-3 and 6-7; and so on. This arrangement allows memory operations on consecutive double words to overlap.

The MV/10000 computer transfers data at a rate of 57.1 Mbytes per second.

I/O System

The MV/10000 I/O system is electrically compatible and program compatible with the ECLIPSE C/350 and the MV/Family processors. This means that the MV/10000 computer supports the full family of standard Data General peripherals with high-speed burst multiplexor channel (BMC) I/O, data channel (DCH) I/O, and programmed I/O (PIO).

The MV/10000 I/O system supports two I/O channels and two I/O channel controllers. Each controller maintains one of the I/O channels, and each I/O channel contains its own BMC, DCH, and PIO I/O facilities. The MV/10000 I/O instructions allow you to communicate with peripherals on the I/O channels — either individually on one I/O channel or simultaneously on both I/O channels. For further information, refer to Chapter 7.

I/O Transfers

Both the BMC and the data channel transfer data to and from the system cache directly; data does not pass through the processor. The BMC transfers blocks of data to and from memory at a rate of up to 10.0 Mbytes per second on output and up to 14.2 Mbytes per second on input. The data channel operates at rates up to 1.4 Mbytes per second on output and 2.0 Mbytes per second on input.

Information can move between the system cache and the I/O channel board at a maximum rate of 28.6 Mbytes per second. Even at this rate, the CPU can continue unabated.

The programmed I/O system, working with a process, transfers words or parts of words between the processor accumulators and I/O devices. These transfers are instrumental in setting up the parameters for the higher speed channels. The MV/10000 computer executes all C/350 programmed I/O instructions exactly as the ECLIPSE C/350 does.

Communications Controllers

Two processors control the asynchronous and synchronous communications. The intelligent asynchronous controller (IAC) monitors asynchronous communications, and the intelligent synchronous controller (ISC) monitors synchronous communications.

Intelligent Asynchronous Controller

The IAC is a 16-bit processor connected to the MV/10000 computer. It features standard facilities such as accumulators, stacks, an I/O bus, an ECLIPSE C/350 instruction subset, and a priority interrupt system. Each IAC supports either 8 (IAC/8) or 16 (IAC/16) asynchronous communications lines. The MV/10000 computer can support a total of 192 asynchronous lines.

The MV/10000 central processor and the IAC must communicate to coordinate their operation. For example, the IAC must signal the host when it has completed a task or needs more information. The IAC memory allocation and protection unit and two groups of special instructions provide the MV/10000 computer and the IAC with the necessary ability to communicate.

For further information, refer to *Intelligent Asynchronous Controller, Programmer's Reference Series.*

Intelligent Synchronous Controller

The intelligent synchronous controller (ISC) — a 16-bit processor connected to the MV/10000 computer — features standard facilities such as accumulators, stacks, an I/O bus, an ECLIPSE instruction subset, and a priority interrupt system. The ISC handles two asynchronous or synchronous communications lines.

The ISC memory allocation and protection unit and two groups of special instructions provide the MV/10000 computer and the ISC with the ability to communicate.

Universal Power Supply Controller

The universal power supply controller (UPSC) is a microprocessor-controlled power system that performs diagnostic functions. The UPSC performs a power-up diagnostic self test; monitors the system power; and reports failures, problems, and status information to the MV/10000 computer. The UPSC is programmable and responds to a request for status information. If programmed to do so, it can also generate an interrupt request.

For more information on the UPSC, see Chapter 7.

System Control Processor

The system control processor (SCP) is a system within the MV/10000 computer and has its own microcomputer. That is, the SCP has its own CPU and its own operating system. The SCP is a soft system console. It performs diagnostic functions and loads microcode into the microsequencer.

As a soft console, the SCP performs system control functions under operator control. It permits the operator to load or examine and modify main memory and to single-step through a program, instruction by instruction.

As a diagnostic tool, the SCP runs programs designed to help isolate hardware problems. It also maintains an error log. When an error occurs, the SCP records the type of error, its location, and the time it occurred.

The SCP provides all the system timing for the MV/10000 system. It also connects to other components via several buses to allow examination and modification of internal registers.

The operator terminal of the SCP gives the operator control over the MV/10000 system by transmitting commands to the system and providing direct responses and reports.

The SCP also contains the real-time clock, the programmable interval timer, and the primary asynchronous line, all of which appear to the main processor to be I/O devices.

For further information on the SCP, see Chapter 7.

C/350 Compatibility

The MV/10000 computer supports the instruction mnemonics and binary opcodes of most instructions implemented on the ECLIPSE C/350. This means that most programs that execute on the C/350 computer will also execute on the MV/10000 computer without recompiling or reassembling.

Several C/350 instructions manipulate data between accumulators without referring to memory. You do not have to modify these instructions to use them in MV/10000-system-specific programs.

Principles of Operation, 32-Bit ECLIPSE® *Systems* describes the compatibility of C/350 and 32-bit instructions, data types, and formats.

Appendix A lists MV/10000-specific instructions.

Registers

The MV/10000 system implements the following registers.

- four 64-bit floating-point accumulators
- four 32-bit fixed-point accumulators
- one 32-bit processor status register
- one 64-bit floating-point status register
- four 32-bit stack management registers
- one 31-bit program counter
- eight 32-bit segment base registers

For more information on these registers, see *Principles of Operation, 32-Bit* ECLIPSE® *Systems*.

Initialization

The processor assumes the physical mode upon power-up, a system reset, or the execution of the *Reset* instruction (IORST).

The processor performs the following functions when it first powers up (and the system microcode loads) or after a system reset:

- Disables logical address translation, which means that logical and physical addresses are equal.
- Keeps the logical address translation protection system in force. That is, the protection system functions as if ring 0 were the current ring of execution.
- Does not initialize the values of the referenced and modified bits. (The values are indeterminate.)
- Sets the processor status register (PSR) and bits 0 through 9 of the floating-point status register (FPSR) to 0.
- Disables error reporting.
- Does not initialize data channel maps. (Data channel maps are undefined.)
- Sets the I/O channel mask bit for channel 1 to a 1.

- Sets bits 3, 4, 7, 8, 9, and 14 of the I/O channel definition register.
- Halts.

After the execution of the *Reset* instruction, the processor performs the following actions:

- Disables logical address translation.
- Keeps the logical address translation protection system in force.
- Sets the PSR and bits 0 through 9 of the FPSR to 0.
- Disables error reporting.
- Disables data channel maps and places itself in physical mode.

When in physical mode, effective address translation works the same way it does when logical address translation is enabled. However, because the logical address space exceeds the physical address space in size, the processor truncates a number of the 31 most significant bits of the logical address before referring to memory. The number of bits truncated depends on the amount of physical memory available. The maximum length of the word address formed from this procedure will be 25 bits for 16 Mbytes of physical memory.

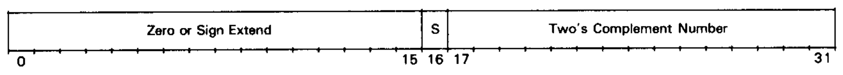
Chapter 2
Fixed-Point Instruction Summary

This chapter summarizes the data formats and instructions for fixed-point and decimal/byte operations and lists the instructions used to manipulate the processor status register (PSR). For further information, refer to *Principles of Operation, 32-Bit* ECLIPSE® *Systems.*

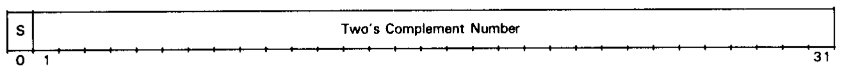
Fixed-Point Data Formats

This section presents the fixed-point accumulator formats for the 16- and 32-bit two's complement numbers and the 16- and 32-bit logical numbers.

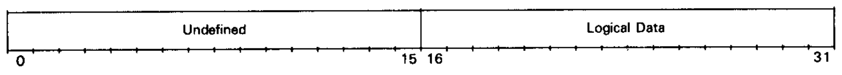
16-Bit Fixed-Point Two's Complement Format



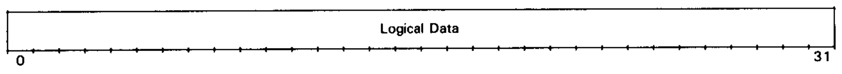
32-Bit Fixed-Point Two's Complement Format



16-Bit Fixed-Point Logical Format



32-Bit Fixed-Point Logical Format



Fixed-Point Instructions

Tables 2.1 through 2.12 list the fixed-point instructions.

Instruction	Function
CVWN	Convert from 32-bit to 16-bit
SEX	Sign extend 16 bits to 32 bits
ZEX	Zero extend 16 bits to 32 bits

Table 2.1 Fixed-point precision conversion

Instruction	Function
LDATS	Load accumulator with double word addressed by WSP
LNLDA	Narrow load accumulator
LNSTA	Narrow store accumulator
LWLDA	Wide load accumulator
LWSTA	Wide store accumulator
MOV *	Move and skip
NLDAI	Narrow load immediate
STATS	Store accumulator into double word addressed by WSP
WBLM	Wide block move
WLDAI	Wide load with wide immediate
WMOV	Wide move
WPOP	Wide pop accumulators
WPSH	Wide push accumulators
WXCH	Wide exchange accumulators
XCH *	Exchange accumulators
XNLDA	Narrow load accumulator
XNSTA	Narrow store accumulator
XWLDA	Wide load accumulator
XWSTA	Wide store accumulator

Table 2.2 Fixed-point data movement instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
ADC *	Add complement and skip
ADD *	Add and skip
ADDI *	Extended add immediate
ADI *	Add immediate
INC *	Increment and skip
LNADD	Narrow add memory word to accumulator
LNADI	Narrow add immediate
LWADD	Wide add memory word to accumulator
LWADI	Wide add immediate
NADD	Narrow add
NADDI	Narrow extended add immediate
NADI	Narrow add immediate
WADC	Wide add complement
WADD	Wide add
WADDI	Wide add with wide immediate
WADI	Wide add immediate
WINC	Wide increment (no skip)
WNADI	Wide add with narrow immediate
XNADD	Narrow add accumulator to memory word
XNADI	Narrow add immediate
XWADD	Wide add memory word to accumulator
XWADI	Wide add immediate

Table 2.3 Fixed-point addition instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
LNSBI	Narrow subtract immediate
LNSUB	Narrow subtract memory word
LWSBI	Wide subtract immediate
LWSUB	Wide subtract memory word
NSBI	Narrow subtract immediate
NSUB	Narrow subtract
SBI *	Subtract immediate
SUB *	Subtract and skip
WSBI	Wide subtract immediate
WSUB	Wide subtract
XNSBI	Narrow subtract immediate
XNSUB	Narrow subtract memory word
XWSBI	Wide subtract immediate
XWSUB	Wide subtract memory word

Table 2.4 Fixed-point subtraction instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
LNMUL	Wide multiply memory word
LWMUL	Wide multiply memory word
MUL *	Unsigned multiply
MULS *	Signed multiply
NMUL	Narrow sign extend multiply
WMUL	Wide multiply
WMULS	Wide signed multiply
XNMUL	Narrow multiply memory word
XWMUL	Wide multiply memory word

Table 2.5 Fixed-point multiplication instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
DIV *	Unsigned divide
DIVS *	Signed divide
DIVX *	Sign extend and divide
HLV *	Halve (AC/2)
LNDIV	Narrow divide memory word
LWDIV	Wide divide memory word
NDIV	Narrow sign extend divide
WDIV	Wide divide
WDIVS	Wide signed divide
WHLV	Wide halve
XNDIV	Narrow divide memory word
XWDIV	Wide divide memory word

Table 2.6 Fixed-point division instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
ADC*	Add complement with optional Carry initialization
ADD*	Add with optional Carry initialization
AND*	AND with optional Carry initialization
COM*	One's complement with optional Carry initialization
CRYTC	Complement Carry
CRYTO	Set Carry bit to 1
CRYTZ	Set Carry bit to 0
INC*	Increment with optional Carry initialization
MOV*	Move with optional Carry initialization
NEG*	Negate with optional Carry initialization
SUB*	Subtract with optional Carry initialization

Table 2.7 Initializing Carry instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
ADC*	Add complement with optional skip
ADD*	Add with optional skip
INC *	Increment with optional skip
MOV *	Move with optional skip
NSALA	Narrow skip on all bits set in accumulator
NSALM	Narrow skip on all bits set in memory location
NSANA	Narrow skip on any bit set in accumulator
NSANM	Narrow skip on any bit set in memory location
SGE *	Skip if ACS greater than or equal to ACD
SGT *	Skip if ACS greater than ACD
SNOVR	Skip on OVR reset
SUB *	Subtract with optional skip
WCLM	Wide compare to limits and skip
WSALA	Wide skip on all bits set in accumulator
WSALM	Wide skip on all bits set in double word memory location
WSANA	Wide skip on any bit set in accumulator
WSANM	Wide skip on any bit set in double word memory location
WSEQ	Wide skip if ACS equal to ACD
WSEQI	Wide skip if equal to immediate
WSGE	Wide signed skip if ACS greater than or equal to ACD
WSGT	Wide signed skip if ACS greater than ACD
WSGTI	Wide skip if AC greater than immediate
WSKBO	Wide skip on AC bit set to 1
WSKBZ	Wide skip on AC bit set to 0
WSLE	Wide signed skip if ACS less than or equal to ACD
WSLEI	Wide skip if AC less than or equal to immediate
WSLT	Wide signed skip if ACS less than ACD
WSNB	Wide skip on addressed bit set to 1
WSNE	Wide skip if ACS not equal to ACD
WSNEI	Wide skip if AC not equal to immediate
WSZB	Wide skip on addressed bit set to 0
WSZBO	Wide skip on addressed bit set to 0 and set bit to 1
WUGTI	Wide unsigned skip if AC greater than immediate
WULEI	Wide unsigned skip if AC less than or equal to immediate
WUSGE	Wide unsigned skip if ACS greater than or equal to ACD
WUSGT	Wide unsigned skip if ACS greater than ACD

Table 2.8 Fixed-point skip on condition instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
DSZTS	Decrement the double word addressed by WSP (skip if 0)
INC *	Increment and skip
ISZTS	Increment the double word addressed by WSP (skip if 0)
LNDSZ	Narrow decrement and skip if 0
LNISZ	Narrow increment and skip if 0
LWDSZ	Wide decrement and skip if 0
LWISZ	Wide increment and skip if 0
XNDSZ	Narrow decrement and skip if 0
XNISZ	Narrow increment and skip if 0
XWDSZ	Wide decrement and skip if 0
XWISZ	Wide increment and skip if 0

Table 2.9 Fixed-point increment or decrement word and skip instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
ANC *	AND with complemented source
AND *	AND
ANDI *	AND immediate
COM *	Complement
IOR *	Inclusive OR
IORI *	Inclusive OR immediate
LOB *	Locate lead bit
LRB *	Locate and reset lead bit
NEG *	Negate
NNEG	Narrow negate
WANC	Wide AND with complemented source
WAND	Wide AND
WANDI	Wide AND immediate
WBTO	Wide set bit to 1
WBTZ	Wide set bit to 0
WCOB	Wide count bits
WCOM	Wide complement (one's complement)
WIOR	Wide inclusive OR
WIORI	Wide inclusive OR immediate
WLOB	Wide locate lead bit
WLRB	Wide locate and reset lead bit
WLSN	Wide load sign
WNEG	Wide negate
WXOR	Wide exclusive OR
WXORI	Wide exclusive OR immediate
XOR *	Exclusive OR
XORI *	Exclusive OR immediate

Table 2.10 Logical instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
AND *	Logical AND with optional shift
COM *	Logical one's complement with optional shift
DLSH *	Double logical shift
LSH *	Logical shift
NEG *	Logical negate with optional shift
WLSH	Wide logical shift
WLSHI	Wide logical shift immediate
WLSI	Wide logical shift left immediate

Table 2.11 Logical shift instructions

*ECLIPSE C/350 compatible instruction

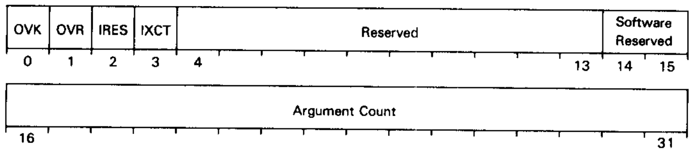
Instruction	Function
AND *	AND with optional skip
COM *	One's complement with optional skip
NEG *	Negate with optional skip
WSNB	Wide skip on nonzero bit
WSZB	Wide skip on 0 bit
WSZBO	Wide skip on 0 bit and set bit to 1

Table 2.12 Fixed-point logical skip instructions

*ECLIPSE C/350 compatible instruction

Processor Status Register

Table 2.13 lists the PSR manipulation instructions. The format for the PSR is:



Bits 4 through 15 of the PSR are set to zero in a return block. When the PSR is loaded or restored, bits 4 through 15 are ignored.

The argument count appears as the second word of the PSR in a wide return block.

Instruction	Function
FXTD	Disable fixed-point trap (resets OVK and disables trap)
FXTE	Enable fixed-point trap (sets OVK and enables trap)
LCALL	Call subroutine
LPSR	Load PSR into AC0
SPSR	Store PSR from AC0
WPOPB	Wide pop block
WRSTR	Wide restore
WDPOP	Wide pop context block
WRTN	Wide return
WSAVR	Wide save and set OVK to 0
WSAVS	Wide save and set OVK to 1
WSSVR	Wide special save and set OVK to 0
WSSVS	Wide special save and set OVK to 1
XCALL	Call subroutine
XVCT	I/O vector interrupt

Table 2.13 PSR manipulation instructions

*ECLIPSE C/350 compatible instruction

Decimal/Byte Operations

Tables 2.14 through 2.19 list the decimal/byte instructions.

Instruction	Function
LLDB	Load byte
LSTB	Store byte
WCMT	Wide character move until true
WCMV	Wide character move
WCTR	Wide character translate and compare
WEDIT	Convert and insert string of decimal or ASCII characters
WLDB	Wide load byte
WSTB	Wide store byte
XLDB	Load byte
XSTB	Store byte

Table 2.14 Fixed-point byte movement instructions

Instruction	Function
WLDI	Convert a decimal and load into FPAC
WLDIX	Convert a decimal, extend, and load it into four FPACs
WSTI	Convert FPAC data and load into memory
WSTIX	Convert the four FPACs and load into memory

Table 2.15 Fixed-point to floating-point conversion and store instructions

Instruction	Function
LLEF	Load effective address
LLEFB	Load effective byte address
LPEF	Push address
LPEFB	Push byte address
WMOVR	Wide move right (convert byte pointer to word pointer)
XLEF	Load effective address
XLEFB	Load effective byte address
XPEF	Push effective address
XPEFB	Push effective byte address

Table 2.16 Load effective word and byte address instructions

Instruction	Function
DADI	Add signed integer to destination indicator
DAPS	Add signed integer to opcode pointer if sign flag is 0
DAPT	Add signed integer to opcode pointer if trigger is 1
DAPU	Add signed integer to opcode pointer
DASI	Add signed integer to source indicator
DDTK	Decrement a word in the stack by one and jump if word is nonzero
DEND	End edit subprogram
DICI	Insert characters immediate
DIMC	Insert character j times
DINC	Insert character once
DINS	Insert character a or character b depending on sign flag
DINT	Insert character a or character b depending on trigger
DMVA	Move j alphabetical characters
DMVC	Move j characters
DMVF	Move j float
DMVN	Move j numerics
DMVO	Move digit with overpunch
DMVS	Move numeric with zero suppression
DNDF	End float
DSSO	Set sign flag to 1
DSSZ	Set sign flag to 0
DSTK	Store in stack
DSTO	Set trigger to 1
DSTZ	Set trigger to 0

Table 2.17 Edit subprogram instructions

Instruction	Function
DAD *	Add two unsigned BCD numbers in two accumulators
DSB *	Subtract two unsigned BCD numbers in two accumulators

Table 2.18 BCD arithmetic instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
DHXL *	Double hex shift left
DHXR *	Double hex shift right
HXL *	Hex shift left
HXR *	Hex shift right

Table 2.19 Hex shift instructions

*ECLIPSE C/350 compatible instruction

Chapter 3
Floating-Point Instruction Summary

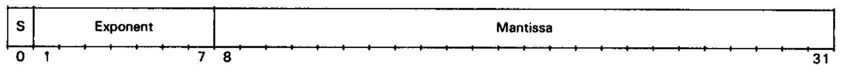
This chapter summarizes the floating-point data formats and floating-point instructions and describes the instructions used to manipulate the floating-point status register (FPSR). For further information, refer to *Principles of Operation, 32-Bit* ECLIPSE® *Systems.*

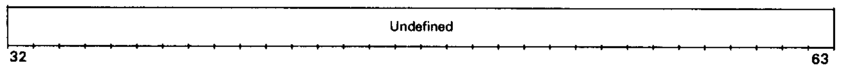
Floating-Point Data Formats

This section presents the floating-point accumulator (FPAC) formats for single- and double-precision floating-point numbers.

Single-Precision Numbers

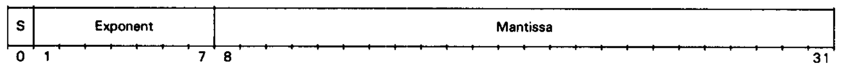
Double Word



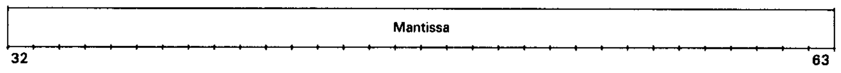


Double-Precision Numbers

Double Word 0



Double Word 1



Floating-Point Instructions

Tables 3.1 through 3.8 list the floating-point instructions.

Instruction	Function
FAD *	Add double (FPAC to FPAC)
FAS *	Add single (FPAC to FPAC)
LFAMD	Add double (memory to FPAC)
LFAMS	Add single (memory to FPAC)
XFAMD	Add double (memory to FPAC)
XFAMS	Add single (memory to FPAC)

Table 3.1 Floating-point addition instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
FSD *	Subtract double (FPAC from FPAC)
FSS *	Subtract single (FPAC from FPAC)
LFSMD	Subtract double (memory from FPAC)
LFSMS	Subtract single (memory from FPAC)
XFSMD	Subtract double (memory from FPAC)
XFSMS	Subtract single (memory from FPAC)

Table 3.2 Floating-point subtraction instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
FMD *	Multiply double (FPAC by FPAC)
FMS *	Multiply single (FPAC by FPAC)
LFMMD	Multiply double (FPAC by memory)
LFMMS	Multiply single (FPAC by memory)
XFMMD	Multiply double (FPAC by memory)
XFMMS	Multiply single (FPAC by memory)

Table 3.3 Floating-point multiplication instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
FDD *	Divide double (FPAC by FPAC)
FDS *	Divide single (FPAC by FPAC)
FHLV *	Halve (FPAC/2)
LFDMD	Divide double (FPAC by memory)
LFDMS	Divide single (FPAC by memory)
XFDMD	Divide double (FPAC by memory)
XFDMS	Divide single (FPAC by memory)

Table 3.4 Floating-point division instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
FCMP *	Compare two floating-point numbers (set N and Z)
FSEQ *	Skip on 0 (Z = 1)
FSGE *	Skip on greater than or equal to 0 (N = 0)
FSGT *	Skip on greater than 0 (N and Z = 0)
FSLE *	Skip on less than or equal to 0 (N and Z = 1)
FSLT *	Skip on less than 0 (N = 1)
FSND *	Skip on no 0 divide (DVZ = 0)
FSNE *	Skip on nonzero (Z = 0)
FSNER *	Skip on no error (ANY = 0)
FSNM *	Skip on no mantissa overflow (MOF = 0)
FSNO *	Skip on no overflow (OVF = 0)
FSNOD *	Skip on no overflow and no 0 divide (OVF and DVZ = 0)
FSNU *	Skip on no underflow (UNF = 0)
FSNUD *	Skip on no underflow and no 0 divide (UNF and DVZ = 0)
FSNUO *	Skip on no underflow and no overflow (UNF and OVF = 0)

Table 3.5 Floating-point skip on condition instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
FEXP *	Load exponent (AC0 17-23 to FPAC 1-7)
FAB *	Compute absolute value (set sign of FPAC to 0)
FFAS *	Fix to AC (FPAC to AC)
FINT *	Integerize (FPAC)
FLAS *	Float from AC (AC to FPAC)
FNEG *	Negate
FNOM *	Normalize (FPAC)
FRDS	Floating-point round double to single
FRH *	Read high word (FPAC 0-15 to AC0 16-31)
FSCAL *	Scale floating point
WFFAD	Wide fix from FPAC
WFLAD	Wide float from AC

Table 3.6 Floating-point binary conversion instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
WLDI	Convert a decimal and load into FPAC
WLDIX	Convert a decimal, extend, and load it into four FPACs
WSTI	Convert FPAC data and load into memory
WSTIX	Convert the four FPACs and load into memory

Table 3.7 Floating-point decimal conversion instructions

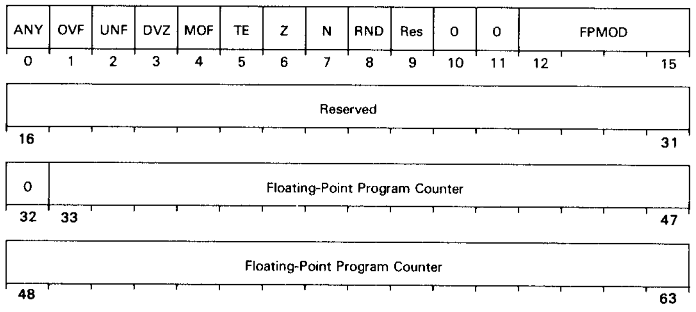
Instruction	Function
FMOV *	Move floating point (FPAC to FPAC)
LFLDD	Load floating-point double
LFLDS	Load floating-point single
LFSTD	Store floating-point double
LFSTS	Store floating-point single
WFPOP	Wide floating-point pop
WFPSH	Wide floating-point push
XFLDD	Load floating-point double
XFLDS	Load floating-point single
XFSTD	Store floating-point double
XFSTS	Store floating-point single

Table 3.8 Floating-point data movement instructions

*ECLIPSE C/350 compatible instruction

Floating-Point Status Register

Table 3.9 lists the instructions that manipulate the floating-point status register (FPSR). The format for the FPSR follows.



Instruction	Function
FCLE *	Clear errors (FPSR)
FTD *	Floating-point trap disable (resets TE)
FTE *	Floating-point trap enable (sets TE)
LFLST	Load FPSR
LFSST	Store FPSR
WFPSH	Push floating-point state
WFPOP	Pop floating-point state

Table 3.9 FPSR instructions

*ECLIPSE C/350 compatible instruction

Chapter 4
Stack Management Instruction Summary

This chapter summarizes the instructions that affect the wide stack. For further information, refer to *Principles of Operation, 32-Bit* ECLIPSE® *Systems.*

Table 4.1 lists the wide stack register instructions; Table 4.2 lists the instructions that access the wide stack; Table 4.3 lists the instructions that push or pop wide stack return blocks; and Table 4.4 lists the instructions that push or pop one or more double words onto the wide stack. Table 4.4 also lists the number of words that the instructions require beyond the wide stack limit for a stack fault return block.

Instruction	Function
LDAFP	Load accumulator with contents of the WFP register
LDASB	Load accumulator with contents of the WSB register
LDASL	Load accumulator with contents of the WSL register
LDASP	Load accumulator with contents of the WSP register
STAFP	Store accumulator in the WFP register
STASB	Store accumulator in the WSB register
STASL	Store accumulator in the WSL register
STASP	Store accumulator in the WSP register
WMSP	Wide modify WSP register

Table 4.1 Wide stack register instructions

Instruction	Function
DSZTS	Decrement the double word addressed by WSP (skip if 0)
ISZTS	Increment the double word addressed by WSP (skip if 0)
LDATS	Load accumulator with double word addressed by WSP
LPEF	Push address
LPEFB	Push byte address
LPSHJ	Push jump to subroutine (pop with WPOPJ)
STATS	Store accumulator into double word addressed by WSP
WFPOP	Wide floating-point pop
WFPSH	Wide floating-point push
WPOP	Wide pop accumulators (push with WPSH)
WPOPJ	Wide pop PC and jump (push with LPSHJ or XPSHJ)
WPSH	Wide push accumulators (pop with WPOP)
XPEF	Push address
XPEFB	Push byte address
XPSHJ	Push jump to subroutine (pop with WPOPJ)

Table 4.2 Wide stack double-word access instructions

Instruction	Function
BKPT	Breakpoint handler (return from breakpoint handler with PBX)
LCALL	Call subroutine (return from call with WRTN)
PBX	Pop block and execute (return from breakpoint handler)
WPOPB	Wide pop block
WRSTR	Wide restore from an interrupt
WRTN	Wide return via wide save (WSAVR, WSAVS, WSSVR, and WSSVS)
WSAVR	Wide save/reset overflow mask (used with LCALL and XCALL)
WSAVS	Wide save/set overflow mask (used with LCALL and XCALL)
WSSVR	Wide special save/reset overflow mask (used with LJSR and XJSR)
WSSVS	Wide special save/set overflow mask (used with LJSR and XJSR)
WXOP	Extended operation (return with WPOPB; used to expand instruction set)
XCALL	Call subroutine (return from call with WRTN)

Table 4.3 Wide stack return block instructions

Instruction	Description	Double Words Pushed (or Popped)	Double Words Required Beyond WSL for Stack Fault
ADD, etc.	Arithmetic with OVK enabled	0	11
FAD, etc.	Arithmetic with TE enabled	0	11
BKPT	Breakpoint handler	6	11
LCALL	Subroutine call	1	6
LPEF	Push address	1	6
LPEFB	Push byte address	1	6
LPSHJ	Push jump	1	6
PBX	Pop block and execute	(6)	5
WEDIT	Wide edit	16	27
WFPOP	Wide floating-point pop	(10)	5
WFPSH	Wide floating-point push	10	15
WPOP	Wide pop accumulators	(1-4)	5
WPOPB	Wide pop block	(6)	5
WPOPJ	Wide pop PC and jump	(1)	5
WPSH	Wide push accumulators	1-4	9
WRSTR	Wide restore	(10)	5
WRTN	Wide return	(6)	5
WSAVR	Wide save/reset OVK	5	10
WSAVS	Wide save/set OVK	5	10
WSSVR	Wide special save/reset OVK	6	11
WSSVS	Wide special save/set OVK	6	11
WXOP	Extended operation	6	11
XCALL	Subroutine call	1	6
XPEF	Push address	1	5
XPEFB	Push byte address	1	5
XPSHJ	Push jump to subroutine	1	5
XVCT	Vector on I/O interrupt	6	11

Table 4.4 Multiword wide stack instructions

Chapter 5
Program Flow Management

This chapter describes the program flow management capabilities of the ECLIPSE MV/10000 computer. The chapter lists the program flow instructions and discusses instruction addressing, interrupt handling, and fault handling. For further information on program flow management, refer to *Principles of Operation, 32-Bit* ECLIPSE® *Systems.*

Program Flow Instructions

Tables 5.1 through 5.6 list the instructions that affect program flow.

Instruction	Function
XCT *	Execute bits 16-31 of an accumulator as an instruction.

Table 5.1 Execute accumulator instruction

*ECLIPSE C/350 compatible instruction

Instruction	Function
LDSP	Dispatch
LJMP	Jump (with long displacement)
WBR	Branch (PC relative jump)
XJMP	Jump (with extended displacement)

Table 5.2 Jump instructions

Instruction	Function
FNS *	No skip
FSA *	Skip always
LNDO	Narrow do until greater than
LWDO	Wide do until greater than
XNDO	Narrow do until greater than
XWDO	Wide do until greater than
NBStc	Narrow search queue backward
NFStc	Narrow search queue forward
WBStc	Wide search queue backward
WFStc	Wide search queue forward

Table 5.3 Skip instructions

*ECLIPSE C/350 compatible instruction

Instruction	Function
BKPT	Breakpoint handler
LCALL	Call subroutine
LJSR	Jump to subroutine
LPSHJ	Push jump
PBX	Pop block and execute
WEDIT	Wide edit of alphanumeric
WPOPB	Wide pop block
WPOPJ	Wide pop PC and jump
WRTN	Wide return
WSAVR	Wide save/reset overflow mask
WSAVS	Wide save/set overflow mask
WSSVR	Wide special save/reset overflow mask
WSSVS	Wide special save/set overflow mask
WXOP	Wide extended operation
XCALL	Call subroutine
XJSR	Jump to subroutine
XPSHJ	Push jump

Table 5.4 Subroutine instructions

Instruction	Function
LCALL	Call subroutine
WPOPB	Wide pop block
WRTN	Wide return
XCALL	Call subroutine
WRSTR	Wide restore from an I/O interrupt

Table 5.5 Segment transfer instructions

Call Instruction or Sequence	Segment Crossing Permitted	Associated Save Instruction	Return Instruction
BKPT	no		PBX/WPOPB*
LCALL	yes	WSAVR	WRTN
	yes	WSAVS	WRTN
LJSR	no	WSSVR	WRTN
	no	WSSVS	WRTN
LPSHJ	no		WPOPJ
WEDIT	no		DEND
WXOP	no		WPOPB
XCALL	yes	WSAVR	WRTN
	yes	WSAVS	WRTN
XJSR	no	WSSVR	WRTN
	no	WSSVS	WRTN
XPSHJ	no		WPOPJ

Table 5.6 Sequence of subroutine instructions

*Use the BKPT/WPOPB instruction sequence when removing the BKPT instruction before returning from the breakpoint handler.

Instruction Addressing

The program counter (PC), which specifies the logical address of the instruction, controls the sequence of executing instructions. Address wraparound occurs within the current segment, because only bits 4 through 31 are used to increment the PC.

To address the next instruction (for normal program flow), the processor either increments the PC or forces an address into the PC. The processor increments the PC by:

- one when executing a one-word instruction (such as NADI)
- two when executing a two-word instruction (such as NADDI)
- three when executing a three-word instruction (such as LNADI)
- four when executing a four-word instruction (such as LCALL)

When the processor forces an address into the PC, the processor clears the instruction processor pipeline and initiates a different program sequence. Any one of the following events alters the normal program sequence:

- executing the XCT instruction
- executing a jump instruction
- executing a skip instruction
- executing a subroutine call or return instruction
- detecting a fault
- detecting an I/O interrupt request

Interrupt Handling

This section describes noninterruptible, restartable, and resumable instructions and discusses the servicing of interrupts.

When the processor honors an interrupt, program execution stops and the processor services the interrupt. How the processor halts program execution to service the interrupt depends upon the instruction currently executing within the program. The currently executing instruction will be one of the following three kinds:

- noninterruptible
- restartable
- resumable

Table 5.7 lists restartable or resumable instructions. Any instruction not listed as either restartable or resumable is noninterruptible.

Restartable (From Beginning)	Restartable (With Updated Values)	Resumable
LCALL	*BAM	*EDIT
LSBRA	*BLM	*LDI
LSBRS	*CMP	*LDIX
PATU	*CMT	*LSN
XCALL	*CMV	*STI
	*CTR	*STIX
	NBStc	WEDIT
	NFStc	WLDI
	ORFB	WLDIX
	RRFB	WSTI
	WBLM	WSTIX
	WBStc	
	WCMP	
	WCMT	
	WCMV	
	WCTR	
	WFStc	
	WLMP	

Table 5.7 Restartable or resumable instructions

*Denotes a C/350 instruction.

Noninterruptible Instructions

If an instruction is noninterruptible, the processor finishes executing that instruction before it services the interrupt. Examples of noninterruptible instructions are *Add, Load Accumulator,* and *Complement.*

The processor does not set any bits in the processor status register (PSR) if an interrupt occurs during a noninterruptible instruction.

Figure 5.1 shows the noninterruptible instruction interrupt sequence.

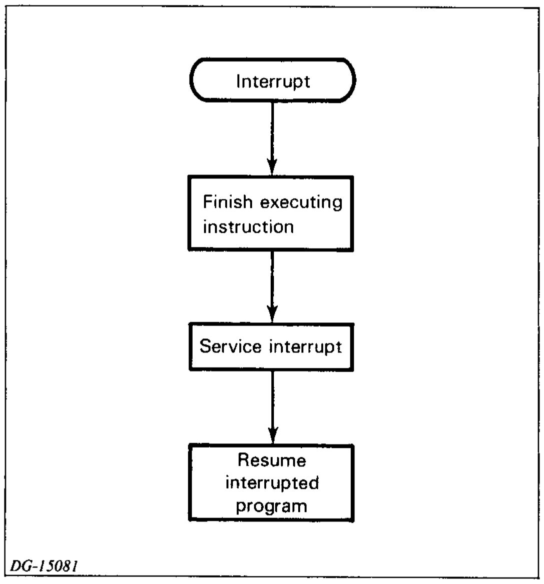


Figure 5.1 Noninterruptible instruction interrupt sequence

Restartable Instructions

If an instruction is restartable, the processor services the interrupt before the instruction finishes. When an interrupt occurs, the processor saves the address of the interrupted instruction in the PC and then services the interrupt. When servicing is complete, the processor can restart the interrupted instruction in one of the following two ways.

- If the parameters of the restartable instruction *have not changed*, the processor restarts the instruction from the beginning. For example, when an interrupt occurs during a *Floating-Point Divide* instruction, the processor restarts the instruction from the beginning, because the accumulators containing the operands have not changed.
- If the parameters of the interrupted instruction *have changed*, the processor restarts execution with the updated values. An instruction such as *Block Move,* for example, uses pointers to source and destination locations and updates them after each one-word move. After servicing the interrupt, the processor restarts execution with the current values of the source and destination pointers, not the original values.

Note that the processor sets bit 2 of the PSR to 1 when an interrupt occurs during a restartable instruction.

Figure 5.2 summarizes the interrupt sequence for a restartable instruction.

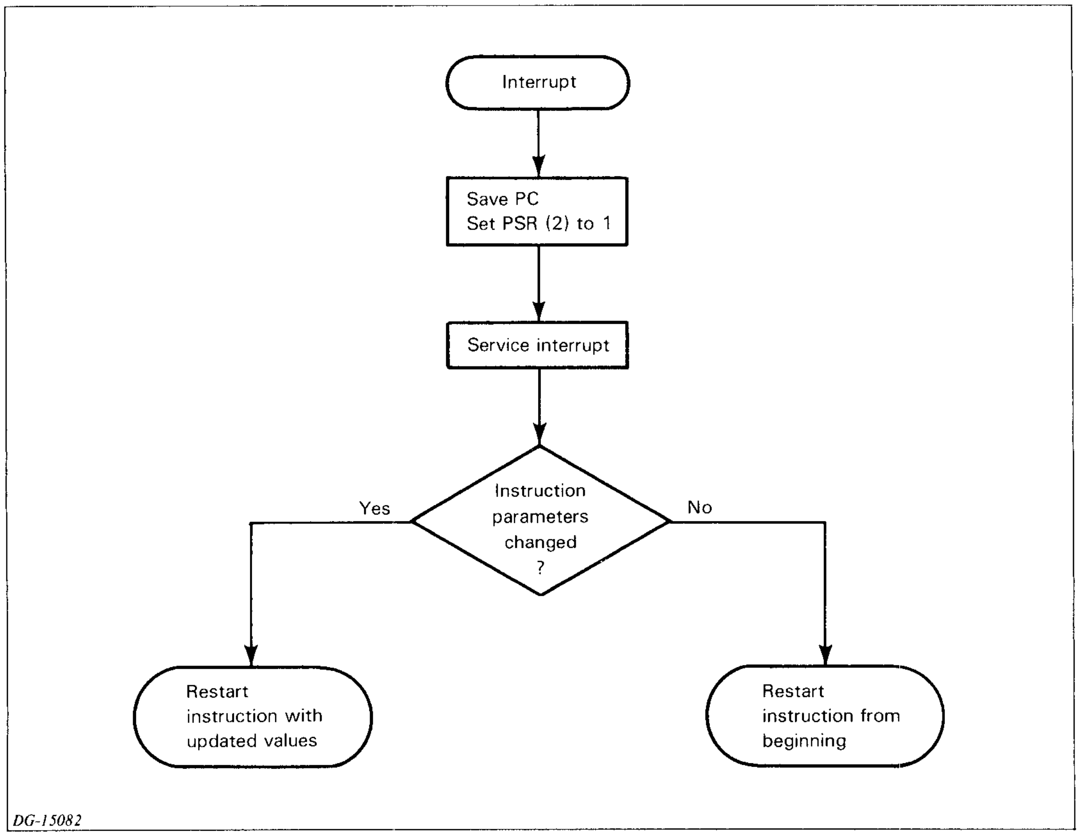

Interrupt
Save PC
Set PSR (2) to 1
Service interrupt
Instruction parameters changed ?
Yes
No
Restart instruction with updated values
Restart instruction from beginning
DG-15082

Figure 5.2 Restartable instruction interrupt sequence

Resumable Instructions

As with restartable instructions, the processor services an interrupt before finishing a resumable instruction. The processor must save a copy of the internal processor state if it is to restart a resumable instruction correctly.

Before interrupting resumable instructions, you should ensure that:

- You define a stack.
- The interrupt handler uses WPOPB, WRSTR, WRTN, or LPSR to return to the interrupted program. These instructions restore the PSR when interrupt service completes.

When an interrupt occurs, the processor saves the address of the interrupted instruction and pushes a copy of all necessary processor information (the microstate block) onto the current stack.

The information needed depends upon the interrupted instruction. If the processor is interrupted during execution of a WEDIT instruction, the processor sets bit 2 of the PSR (IRES) to 1. If the processor is interrupted during execution of a resumable or restartable instruction resulting from a PBX instruction, the processor sets bit 3 of the PSR (IXCT) to 1.

After pushing the block, the processor checks for a stack overflow. If it detects a stack overflow, the processor performs the following steps:

1. Services the interrupt.
2. Returns to the interrupted program.
3. Services the stack fault (if necessary).
4. Resumes the interrupted instruction.

Next, the processor restores the PSR using the appropriate return instruction. If a resumable instruction was interrupted, then the processor tests bits 2 and 3. If either bit contains a 1, the processor examines the microstate block on the current wide stack to determine the type of microinterrupt.

If the microstate block is valid, the processor resumes executing the interrupted instruction.

If the block is invalid, the next sequence of events depends on the type of instruction interrupted as follows:

- An MV/10000-system-specific instruction causes a protection fault to occur. Accumulator 1 (AC1) will contain the code 12 to indicate the invalid microstate block.
- A C/350 floating-point instruction causes a floating-point fault to occur.
- A C/350 *Decimal/ASCII* instruction causes a narrow decimal/ASCII fault to occur. AC1 will contain the code 5 to indicate the invalid microstate block.

If the interrupted instruction was inserted into the instruction stream as a result of a PBX instruction, then the processor sets the IXCT flag in the PSR and pushes the op-code of the executing instruction onto the wide stack.

Table 5.8 shows the processor settings of bits 2 and 3 of the PSR when an interrupt occurs during execution of a resumable instruction. Figure 5.3 summarizes the sequence of events upon the interruption of a resumable instruction.

NOTE: *When an interrupt occurs during a segment crossing, the saved PC points to the first instruction of the called procedure.*

Instruction	PSR Bit 2 (IRES)	PSR Bit 3 (IXCT)
C/350	1	1 if executed via PBX; otherwise, 0.
MV/10000-specific	1	1 if executed via PBX; otherwise, 0.

Table 5.8 State of PSR bits 2 and 3

Servicing an Interrupt

When servicing an interrupt, the processor disables further interrupts by setting the interrupt on (ION) flag to 0. Then, depending on whether or not the MV/10000 address translator is enabled, one of two sets of events occurs.

When the MV/10000 address translator is disabled — that is, when the processor is in *physical mode* — the processor fetches the contents of physical location 1 and prepares to resolve any indirection. The processor treats this address as the address of the MV/10000 interrupt handler.

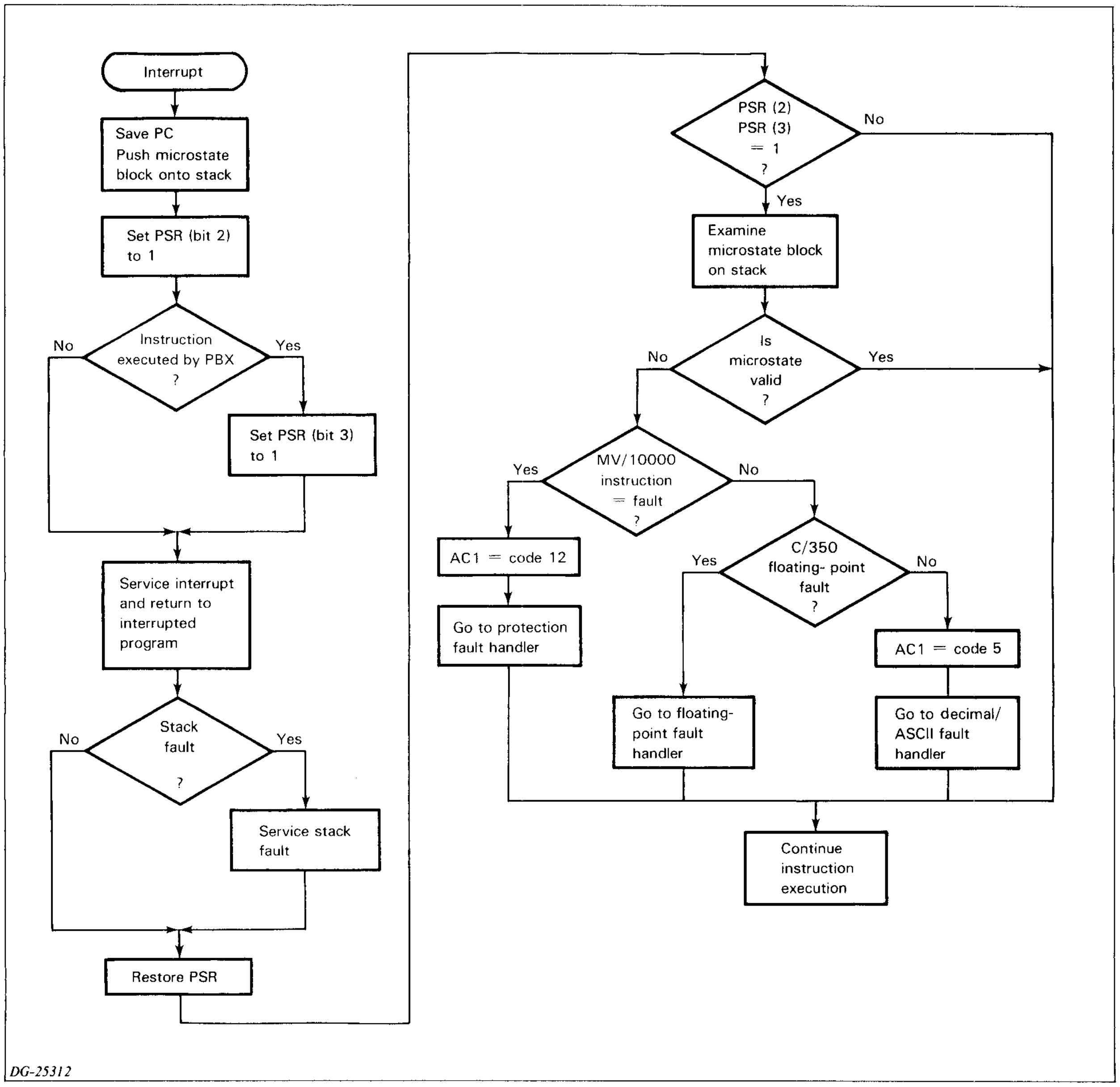

Interrupt
Save PC Push microstate block onto stack
Set PSR (bit 2) to 1
Instruction executed by PBX ?
No
Yes
Set PSR (bit 3) to 1
Service interrupt and return to interrupted program
Stack fault ?
No
Yes
Service stack fault
Restore PSR
PSR (2) PSR (3) = 1 ?
No
Yes
Examine microstate block on stack
Is microstate valid ?
No
Yes
MV/10000 instruction = fault ?
Yes
No
AC1 = code 12
Go to protection fault handler
C/350 floating- point fault ?
Yes
No
AC1 = code 5
Go to floating-point fault handler
Go to decimal/ ASCII fault handler
Continue instruction execution
DG-25312

Figure 5.3 Resumable instruction interrupt sequence

When the MV/10000 address translator is enabled, the processor fetches the contents of logical location 1 in page zero of segment 0. This location contains the address of the interrupt handler. The processor then determines the current segment of execution. If it is not segment 0, the processor performs a segment crossing to segment 0. Next, the processor must resolve the interrupt handler address.

If the fetched address of the interrupt handler is indirect, the processor resolves it to a final direct address. This address is the address of the first instruction of the handler.

The first instruction of the MV/10000 interrupt handler will be one of the following three types:

- An XVCT instruction.
- Any other MV/10000-system-specific instruction (Type 1).
- C/350 instructions, WBR, and some MV/10000-system-specific memory to accumulator instructions (Type 2).

Figure 5.4 summarizes the process of accessing the interrupt handler.

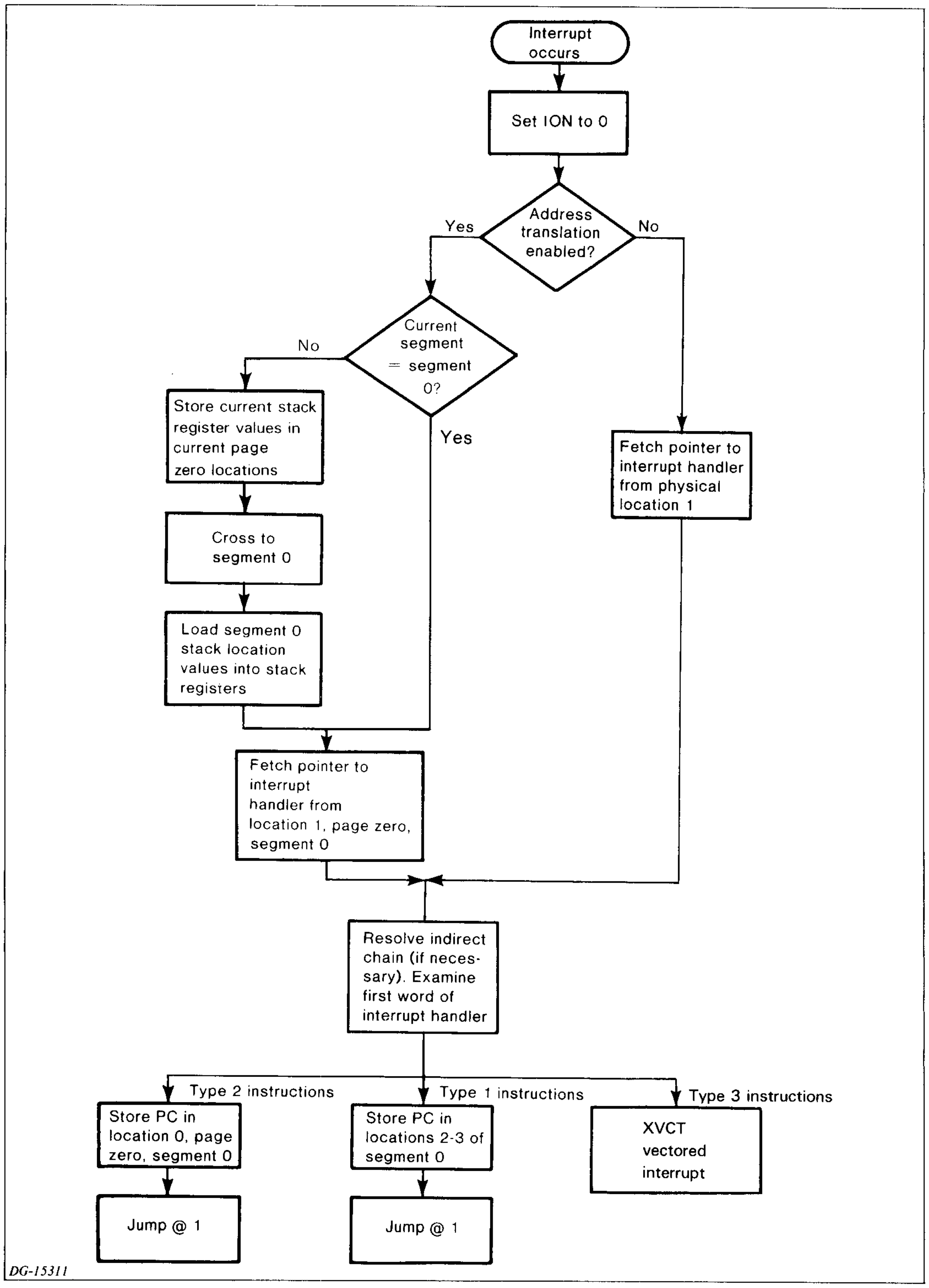

Interrupt occurs
Set ION to 0
Address translation enabled?
Yes
No
Current segment = segment 0?
No
Yes
Store current stack register values in current page zero locations
Cross to segment 0
Load segment 0 stack location values into stack registers
Fetch pointer to interrupt handler from location 1, page zero, segment 0
Fetch pointer to interrupt handler from physical location 1
Resolve indirect chain (if necessary). Examine first word of interrupt handler
Type 2 instructions
Type 1 instructions
Type 3 instructions
Store PC in location 0, page zero, segment 0
Store PC in locations 2-3 of segment 0
XVCT vectored interrupt
Jump @ 1
Jump @ 1
DG-15311

Figure 5.4 Accessing the interrupt handler

Fault Handling

While executing an instruction, the processor performs certain checks on the operation and the data. If the processor detects an error, a *privileged fault* or *nonprivileged fault* occurs before execution of the next instruction.

With the MV/10000 address translator enabled, the processor detects the following faults:

Fault Generated	**Fault Type**
Protection violation fault	Privileged
Page fault	Privileged
Stack fault	Nonprivileged
Fixed-point overflow	Nonprivileged
Floating-point fault	Nonprivileged
Decimal/ASCII fault	Nonprivileged

Appendix F lists the error codes returned to AC1 and the types of faults generated. For further information on fault handling, refer to *Principles of Operation, 32-Bit* ECLIPSE® *Systems.*

Privileged Faults

The two types of privileged faults — page faults and protection violation faults — are described in Chapter 8 and in *Principles of Operation, 32-Bit* ECLIPSE® *Systems*, respectively.

Nonprivileged Faults

Principles of Operation, 32-Bit ECLIPSE® *Systems* describes the handling of nonprivileged faults.

Execution of C/350 instructions does not generate fixed-point faults. Certain C/350 arithmetic instructions such as ADD and DIV set the state of the Carry bit. If a program is to detect a particular fault, a subroutine must be set up to check the state of the Carry bit upon completion of these instructions. A carry from accumulator bit 16 affects the MV/10000 system's Carry bit upon execution of these C/350 instructions. The instruction dictionary in *Principles of Operation, 32-Bit* ECLIPSE® *Systems* describes the C/350 instruction set and lists the instructions that affect the Carry bit.

Note that all faults that occur with the execution of C/350 instructions use the narrow stack.

Chapter 6
Queue Management Instruction Summary

This chapter summarizes the queue instructions. For further information, refer to *Principles of Operation, 32-Bit* ECLIPSE® *Systems.*

Table 6.1 lists the queue instructions.

Instruction	Action
ENQH	Queue toward the head; add a data element to queue
ENQT	Queue toward the tail; add a data element to queue
DEQUE	Dequeue a queue data element; delete a data element
NBStc	Narrow search queue backward; 16-bit test condition
NFStc	Narrow search queue forward; 16-bit test condition
WBStc	Wide search queue backward; 32-bit test condition
WFStc	Wide search queue forward; 32-bit test condition
WMESS	Wide mask; skip and store if equal

Table 6.1 Queue instructions

Chapter 7
Device Management

This chapter summarizes the general I/O instructions and describes the instructions for the manipulation of the following devices:

- central processing unit
- programmable interval timer
- real-time clock
- primary asynchronous line input/output
- system control processor
- data channel and burst multiplexor channel
- universal power supply controller

Appendix E lists device codes, device mnemonics and priority mask bit assignments.

General I/O Instructions

Table 7.1 lists the general I/O instructions; Tables 7.2 and 7.3 list the device flags mnemonics. For further information, refer to *Principles of Operation, 32-Bit* ECLIPSE® *Systems.*

Instruction	Function
DIA*[f]* *	Data in A (from A buffer of device).
DIB*[f]* *	Data in B (from B buffer of device).
DIC*[f]* *	Data in C (from C buffer of device).
DOA*[f]* *	Data out A (to A buffer of device).
DOB*[f]* *	Data out B (to B buffer of device).
DOC*[f]* *	Data out C (to C buffer of device).
IORST *	I/O reset.
NIO*[f]* *	No I/O transfer (initialize a BUSY/DONE flag).
PIO	Issue a programmed I/O command to a device.
SKP*t* *	I/O skip (test a BUSY/DONE flag and skip on condition).

Table 7.1 General I/O instructions

*The *[f]* or *t* defines the optional device flag handling.

*The * identifies ECLIPSE C/350 compatible instructions.

Assembler Code for *f*	Bits 8 9	I/O BUSY	I/O DONE	CPU ION
(option omitted)	0 0	No effect	No effect	No effect
S	0 1	Set to 1	Set to 0	Set to 1
C	1 0	Set to 0	Set to 0	Set to 0
P	1 1	Pulses a special I/O bus control line		No effect

Table 7.2 Device flags for general devices

Assembler Code for *t*	Bits 8 9	I/O	CPU
BN	0 0	Test for BUSY = 1	Test for ION = 1
BZ	0 1	Test for BUSY = 0	Test for ION = 0
DN	1 0	Test for DONE = 1	Test for power fail = 1
DZ	1 1	Test for DONE = 0	Test for power fail = 0

Table 7.3 Device flags for skip instruction

Central Processor

Device Code
77_8

Assembler Mnemonic
CPU

Priority Mask Bit
None

Device Flags

Device flag commands to the CPU determine whether or not the processor can interrupt the current program with a program interrupt request. When the interrupt enable flag (ION) equals 1, the processor can interrupt the program (once the instruction following the enable has begun). When the interrupt enable flag equals 0, the processor cannot interrupt the program. The CPU interrupt enable flag is controlled by the device flag commands as follows:

f=**S**	Sets the interrupt enable flag to 1.
f=**C**	Sets the interrupt enable flag to 0.
f=**P**	Causes an unimplemented instruction interrupt.

The assembler interprets the I/O instructions for the CPU using either the standard I/O instruction format or a special I/O instruction format. The instruction that initializes the devices and sets the priority mask bits to 0 uses the following standard form:

DIC*[f] ac*,**CPU**

The same instruction can take the following special form:

IORST

The special **IORST** assembler statement is equivalent to the following standard assembler statement:

DICC 0,CPU

Both statements set all the BUSY and DONE flags to 0. You cannot append a device flag (**S, C,** or **P**) to the special form of a CPU instruction.

NOTE: *The assembler detects a fatal format error if you append a device flag to a special CPU instruction.*

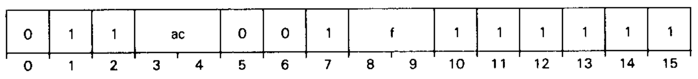
CPU Instructions

Table 7.4 lists the I/O instructions — both standard and special — that affect the CPU device.

Assembler Statement	Function
READS *ac* DIA*[f]* *ac*, CPU	Reads console switches (places the contents of the console data switches into an accumulator)
PRTSEL NIO CPU	Sets the default I/O channel to contents of AC0.
PRTRST PIO *acs, acd*	Initializes an I/O subsystem (channel 0 or 1).
INTA *ac* DIB*[f]* *ac*,CPU	Returns the device code of the interrupting device (interrupt acknowledge)
IORST DIC*[f]* *ac*,CPU	Initializes the I/O system (resets the BUSY and DONE flags and all the priority mask bits to 0; clears certain CPU registers and disables the DCH mapping and address translator)
MSKO *ac* DOB*[f]* *ac*,CPU	Initializes or changes the priority mask
HALT DOC*[f]* *ac*,CPU	Stops the processor
INTDS NIOC CPU	Sets ION flag to 0 (interrupt disable)
INTEN NIOS CPU	Sets ION flag to 1 (interrupt enable)
SKP*t* CPU	Tests the condition of the ION flag or power fail flag, and when true, it skips the next word in the program

Table 7.4 I/O instructions for CPU

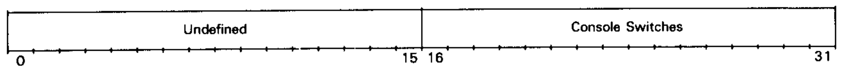
Read Switches
READS *ac*
DIA*[f]* *ac*,CPU



The *Read Switches* instruction places the contents of the console switches into bits 16 through 31 of the specified accumulator. After the transfer, the instruction sets the ION flag according to the function specified by *[f]*.

NOTE: *The assembler recognizes the special mnemonic READS* ac *to be equivalent to DIA* ac*, CPU.*

The format of the specified accumulator after instruction execution is:



Console Switches: 1=ON; 0=OFF

I/O Channel Select
PRTSEL
NIO 3,CPU

0	1	1	0	0	1	0	1	1	0	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The *I/O Channel Select* instruction specifies, in AC0, the I/O channel that the ECLIPSE C/350 compatible I/O instructions use. This channel is also called the default I/O channel. The format of AC0 is:

Undefined	Reserved	I/O Chan
0–15	16–28	29–31

NOTE: *An I/O reset does not change the default I/O channel. On a power up or after a system reset, the default I/O channel is 0.*

With a device code other than 77_8 (the CPU device):

- The ECLIPSE C/350 compatible I/O instructions use the default I/O channel.
- The PIO instruction (with an ECLIPSE C/350 compatible I/O instruction) uses I/O channel 0 or 1.

In either case, results are undefined with any other channel number.

Table 7.5 shows the effect of using the I/O channel numbers with a device code of 77_8.

I/O Instruction	Default I/O Channel	PIO to I/O Channel *n* *	PIO to I/O Channel 7
READS *ac* DIA*[f] ac*,CPU	Read CPU Switch Register	Undefined	Undefined
INTA *ac* DIB*[f] ac*,CPU	INTA to channel *n*	INTA channel *n*	INTA highest priority channel/device
IORST *ac* DIC*[f] ac*,CPU	Reset all I/O	PRTRST channel *n*	PRTRST all channels
MSKO *ac* DOB*[f] ac*,CPU	MSKO to channel *n*	MSKO channel *n*	MSKO all channels
HALT *ac* DOC*[f] ac*,CPU	HALT to CPU	Undefined	Undefined
INTDS NIOC *ac*,CPU	Disable Interrupts (ION = 1)	Undefined	Undefined
INTEN NIOS *ac*,CPU	Enable Interrupts (ION = 0)	Undefined	Undefined
SKPBN CPU	SKP on ION = 1	Undefined	Undefined
SKPBZ CPU	SKP on ION = 0	Undefined	Undefined
SKPDN CPU	SKP if power fail	Undefined	Undefined
SKPDZ CPU	SKP if no power fail	Undefined	Undefined

Table 7.5 CPU device instructions with I/O channels

**N* equals 0 or 1

I/O Channel Reset
PRTRST
PIO *acs,acd*

1	acs	acd	1	0	1	1	1	0	1	1	0	0	1
0	1 2	3 4	5	6	7	8	9	10	11	12	13	14	15

The *I/O Channel Reset* instruction sends a reset signal to all devices on the I/O channel specified in *acs*, instructing those devices to clear their states. In addition, the instruction sets the 16-bit priority mask to 0 and clears the I/O channel mask bit (MK0 or MK1). The *acd* accumulator is not used. The format of *acs* is:

Undefined	0	I/O Chan	0	1	0	1	1	0	1	1	1	1	1	1
0 15	16	17 19	20	21	22	23	24	25	26	27	28	29	30	31

A PRTRST instruction issued to channel 0 resets channel 0. Issued to channel 1, PRTRST resets channel 1; issued to channel 7, it resets channels 0 and 1. All other channel numbers produce undefined results.

Interrupt Acknowledge
INTA *ac*
DIB*[f]* *ac*,CPU

0	1	1	ac	0	1	1	f	1	1	1	1	1	1
0	1	2	3 4	5	6	7	8 9	10	11	12	13	14	15

The *Interrupt Acknowledge* instruction places the 9-bit device code of that device requesting an interrupt that is physically closest to the CPU on the I/O bus into bits 23 through 31 of the specified accumulator, setting bits 0 through 22 to 0. After the transfer, the instruction sets the ION flag according to the function specified by *[f]*.

NOTE: *The assembler recognizes the special mnemonic* **INTA** ac *to be equivalent to DIB* ac, *CPU. Do not use the DIBP* ac, *CPU instruction. On the ECLIPSE C/350, it is reserved for the VCT instruction.*

The format of the specified accumulator after instruction execution is:

0 — 0	Device Code
0 22	23 31

Bits	Name	Contents or Function
0-22	Reserved	Returned as 0.
23-31	Device Code	Device code of interrupting device. Bits 23 through 25 set to 0 if INTA was issued on the default I/O channel. Bits 23 through 25 contain channel number if INTA was issued with a PIO to channel *n*.

I/O Reset
IORST
DIC*[f]* *ac*,CPU

0	1	1	ac	1	0	1	f	1	1	1	1	1	1
0	1	2	3 4	5	6	7	8 9	10	11	12	13	14	15

The *I/O Reset* instruction sends a reset signal to all devices to clear their states. The instruction sets the 16-bit priority mask to 0; sets the I/O channel mask register flag for channel 0 to 0; disables logical address translation; sets the PSR to 0; sets bits 0 through 9 of FPSR to 0; sets bits 0, 3, 4, 7, 8, 9, and 14 of the I/O channel definition register (6000_8) to 0; and sets the ION flag according to the function specified by *[f]*.

If you use the standard form of the *I/O Reset* instruction (DIC *[f] ac*, CPU), you must code an accumulator to avoid assembly errors. During execution, the processor ignores the accumulator field, and the contents of the accumulator remain unchanged.

NOTE: *The assembler recognizes the special mnemonic IORST to be equivalent to DICC 0,CPU. This instruction sets the BUSY and DONE flags as described above and sets the ION flag to 0.*

Mask Out
MSKO *ac*
DOB*[f]* *ac*,CPU

0	1	1	ac	1	0	0	f	1	1	1	1	1	1
0	1	2	3 4	5	6	7	8 9	10	11	12	13	14	15

The *Mask Out* instruction places the contents of bits 16 through 31 of the specified accumulator in the priority mask. After the transfer, the instruction sets the ION flag according to the function specified by *[f]*. The contents of the specified accumulator remain unchanged. A 1 in a bit position disables interrupt requests at devices that use that bit as a mask.

NOTE: *Masking out a device when interrupts are enabled is not recommended.*

The assembler recognizes the special mnemonic MSKO *ac* to be equivalent to DOB *ac*, CPU. The contents of the specified accumulator are:

Unused	Priority Mask
0 15	16 31

Halt
HALT
DOC*[f]* *ac*,CPU

0	1	1	ac	1	1	0	f	1	1	1	1	1	1
0	1	2	3 4	5	6	7	8 9	10	11	12	13	14	15

The *Halt* instruction sets the ION flag according to the function specified by *[f]* and then stops the processor.

NOTE: *The assembler recognizes the special mnemonic HALT to be equivalent to DOC 0,CPU.*

Interrupt Disable
INTDS
NIOC CPU

0	1	1	0	0	0	0	0	1	0	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The *Interrupt Disable* instruction sets the ION flag to 0.

Interrupt Enable
INTEN
NIOS CPU

0	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The *Interrupt Enable* instruction sets the ION flag to 1.

If the instruction changes the state of the ION flag, the CPU allows one more instruction to execute before the first I/O interrupt can occur. However, if the instruction is interruptible, then interrupts can occur as soon as the instruction begins to execute.

CPU Skip
SKP*t* CPU

0	1	1	0	0	1	1	1	t		1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The *CPU Skip* instruction tests the specified flag. If the test condition is true, the processor skips the next sequential word. (Table 7.3 lists the possible test conditions.)

Programmable Interval Timer

Device Code
43_8

Assembler Mnemonic
PIT

Priority Mask Bit
11

The programmable interval timer (PIT) is a CPU-independent time base that you can set to initiate program interrupts at fixed intervals ranging from 100 microseconds to 6.5536 seconds in increments of 100 microseconds. It can also be sampled with I/O instructions at any point in its cycle to determine the time until the next interrupt. Use the PIT in multiprogram operating systems to allocate CPU time to different programs on a “time-slice” basis.

The PIT consists of a 16-bit initial count register and a 16-bit counter. During operation, the processor loads the PIT counter with the contents of the initial count register. The processor then increments the counter at 100-microsecond intervals until the count reaches 177777_8. The PIT then initiates a program interrupt request. At the end of the next 100-microsecond interval, the processor again loads the PIT counter with the contents of the initial count register, and the counting process is repeated. A BUSY flag and a DONE flag control the operation of the device.

To obtain a particular time interval between program interrupt requests, load the two’s complement of the number of 100-microsecond intervals between interrupt requests into the initial count register. When you first start the PIT, the processor immediately loads the count into the counter. At the first 100-microsecond pulse, the processor again loads the count into the counter. This is done to synchronize the program and the counter.

Device Flags

Device flag commands to the PIT start or stop the counting cycle for program interrupts.

f=S	Sets the BUSY flag to 1 and the DONE and interrupt request flags to 0; begins the counting cycle.
f=C	Sets the BUSY and DONE flags and the interrupt request flag to 0; stops the counting cycle.
f=P	No effect.

PIT Instructions

Table 7.6 lists the I/O instructions that affect the PIT device.

Assembler Statement	Function
DIA*[f]* *ac*,PIT	Reads the counter value into the accumulator
DOA*[f]* *ac*,PIT	Loads the counter with a value (PIT initializes the counter with the value each time the counter starts or overflows)
IORST	Stops the counting cycle and sets the BUSY and DONE flags, the interrupt mask bit 11, and the counter to 0

Table 7.6 I/O instructions for PIT

Read Count
DIA*[f]* *ac*,PIT

0	1	1	ac		0	0	1	f		1	0	0	0	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The *Read Count* instruction places the value of the PIT counter in bits 16 through 31 of the specified accumulator, destroying the accumulator's previous contents. After the data transfer, the instruction performs the function specified by *[f]*. The format of the specified accumulator is as follows:

Reserved		Count	
0	15	16	31

Bits	Name	Contents or Function
0-19	Reserved	Reserved for future use
20-31	Count	Current value of the PIT counter within one count cycle (two's complement)

Specify Initial Count
DOA*[f]* *ac*,PIT

0	1	1	ac	0	1	0	f	1	0	0	0	1	1
0	1	2	3 4	5	6	7	8 9	10	11	12	13	14	15

The *Specify Initial Count* instruction loads bits 16 through 31 of the specified accumulator into the initial count register of the PIT. After the data transfer, the instruction performs the function specified by *[f]*. The contents of the specified accumulator remain unchanged. The format of the accumulator is as follows:

Reserved	Initial Count
0 19	20 31

Bits	Name	Contents or Function
0-19	Reserved	Reserved for future use
20-31	Initial Count	The number of 100-microsecond intervals between interrupts (two's complement)

Real-Time Clock

Device Code
14_8
Assembler Mnemonic
RTC

Priority Mask Bit
13

The real-time clock (RTC) generates low-frequency I/O interrupts for performing time calculations independent of CPU timing. You can use these interrupts as a time base in programs that require it. The frequency of the clock is program-selectable to ac-line frequency and 10 Hz, 100 Hz, and 1000 Hz. The BUSY and DONE flags control the operation of the device.

Once you start the RTC, the first program interrupt request can come at any time up to the selected clock period. After the first interrupt has occurred, succeeding interrupts come at the clock frequency, provided that the program always sets the BUSY flag to 1 before the clock period expires. After power up or when an IORST instruction is issued, the processor sets the clock to the line frequency. After power up, the line frequency pulses are available immediately, but 5 seconds must elapse before a steady pulse train is available from the clock for other frequencies.

Device Flags

Device flag commands to the RTC determine the enabling or disabling of RTC interrupts.

f=S	Sets the BUSY flag to 1 and the DONE and interrupt request flags to 0; enables RTC interrupts.
f=C	Sets the BUSY, DONE, and interrupt request flags to 0; disables RTC interrupts.
f=P	No effect.

RTC Instructions

Table 7.7 lists the I/O instructions that affect the RTC device.

Assembler Statement	Function
DOA*[f]* *ac*,RTC	Selects a clock frequency with a value from an accumulator
IORST	Disables RTC interrupts and selects the ac-line frequency; also, sets the BUSY and DONE flags and the interrupt mask bit (bit 13) to 0

Table 7.7 I/O instructions for RTC

Select RTC Frequency
DOA*[f]* *ac*,RTC

0	1	1	ac		0	1	0	f		0	0	1	1	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The *Select RTC Frequency* instruction sets the clock frequency according to bits 30 and 31 of the specified accumulator. The contents of the specified accumulator remain unchanged; bits 0 through 29 are ignored. The format of the specified accumulator is as follows:

Reserved	RTC
0–29	30 31

Bits	Name	Contents or Function
0-29	Reserved	Reserved for future use (set to 0)
30-31	RTC	Selects the clock frequency as follows: **Bits** **Frequency Selected** 00 ac-line frequency 01 10 Hz 10 100 Hz 11 1000 Hz

Primary Asynchronous Line Input/Output

INPUT Device Code	**OUTPUT Device Code**
10_8	11_8
Assembler Mnemonic	**Assembler Mnemonic**
TTI	TTO
Priority Mask Bit	**Priority Mask Bit**
14	15

The asynchronous line controller (ALC) is the communication link between the processor and the master terminal. It supports asynchronous communication at selected rates from 110 to 4800 baud in 7-bit codes with program-generated parity or 8-bit codes with no parity. You can use one or two stop bits with either format.

Because the asynchronous communications input and output can generate program interrupts independently, each has its own device code and is controlled by its own set of BUSY and DONE flags. The ALC is program-compatible with Data General's Model 4010 controller.

The ALC is set up to transmit and receive 8-bit characters without parity checking. A process can send or receive 7-bit characters with even, odd, or mark parity by using the high-order bit in the 8-bit character (bit 8 in the accumulator) as a parity bit. On transmission, the program that drives the ALC calculates and inserts the correct parity bit. On reception, the program calculates and checks parity on the received character.

There are timing constraints on the *receive* portion of the controller. As the ALC receives each character, it places the character in an input character buffer and sets the DONE flag to 1 and the BUSY flag to 0. If the program controlling the receiver does not transfer the character before receiving the next character, the contents of the input character buffer are overwritten and the previous character is lost. Typically, the intercharacter time at 110 baud is 100 milliseconds; at 4800 baud, the intercharacter time is approximately 2.08 milliseconds.

Device Flags

Device flag commands to the TTI/TTO determine the flag settings and the transmission of an output character.

f=**S**	Sets the BUSY flag to 1 and the DONE flag to 0. When the S flag is used with the TTO device, the ALC transfers the character from the output buffer to the shifter and begins transmission of the character. The ALC sets the BUSY flag to 0 and the DONE flag to 1 when the character passes from the output buffer to the shifter.
f=**C**	Sets the BUSY, DONE, and interrupt request flags to 0.
f=**P**	No effect.

TTI/TTO Instructions

Table 7.8 lists the I/O instructions that affect the TTI/TTO device.

Assembler Statement	Function
DIA*[f]* *ac*,TTI	Reads a character from the device into an accumulator
DOA*[f]* *ac*,TTO	Sends a character from an accumulator to the device
IORST	Sets the BUSY and DONE flags and the interrupt mask bits (bits 14 and 15) to 0

Table 7.8 I/O instructions for TTI and TTO

Read Character Buffer
DIA*[f]* *ac*,TTI

0	1	1	ac		0	0	1	f		0	0	1	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The *Read Character Buffer* instruction places the contents of the controller's input buffer in bits 24 through 31 of the specified accumulator. After the data transfer, the instruction sets the controller's BUSY and DONE flags according to the function specified by *[f]*. The format of the specified accumulator is as follows:

Reserved	Character
0–23	24–31

Bits	Name	Contents or Function
0-23	Reserved	Reserved for future use
24-31	Character	The 8-bit character (or 7-bit character with parity in bit position 8) to be placed in the output buffer

Load Character Buffer
DOA*[f]* *ac*,TTO

0	1	1	ac	0	1	1	f	0	0	1	0	0	1
0	1	2	3 4	5	6	7	8 9	10	11	12	13	14	15

The *Load Character Buffer* instruction loads bits 24 through 31 of the specified accumulator into the controller's output buffer. After the data transfer, the instruction sets the controller's BUSY and DONE flags according to the function specified by *[f]*. The contents of the specified accumulator remain unchanged. The format of the specified accumulator is as follows:

Reserved	Character
0 23	24 31

Bits	Name	Contents or Function
0-23	Reserved	Reserved for future use
24-31	Character	The 8-bit character (or 7-bit character with parity in bit position 8) read from the input buffer

System Control Processor

Device Code
45_8

Assembler Mnemonic
SCP

Priority Mask Bit
15

The system control processor (SCP), as described in Chapter 1, is a system within the MV/10000 computer that has its own microcomputer.

The SCP runs programs designed to help isolate hardware problems. It maintains an error log consisting of the type of error, its location, and the time it occurred; and it provides all the system timing for the MV/10000 computer.

Device Flags

Device flag commands to the SCP determine the settings of the BUSY and DONE flags.

f=**S**	Sets the BUSY flag to 1 and the DONE flag to 0.
f=**C**	Sets the BUSY and DONE flags to 0.
f=**P**	No effect.

SCP Instructions

Table 7.9 lists the instructions that permit the CPU to communicate with the SCP.

Mnemonic	Name	Function
DOBS *ac*,SCP	Enable/Disable Error Reporting	Enable/disable CPU error reporting, and perform indicated command
DIBC *ac*,SCP	Return SCP Status	Return the current status of the SCP
SKP*t* SCP	Skip Test	Test the SCP BUSY/DONE flag and skip next instruction if true
IORST	I/O Reset	Disable CPU error reporting

Table 7.9 SCP instructions

The SKP and IORST instructions are described earlier in this chapter. Note that the DIA, DOA, DIC, DOC, and NIO instructions to the SCP are no-ops.

Before issuing a DOBS SCP instruction, the process should check the SCP BUSY flag. If the BUSY flag is 0, the SCP is ready to accept the next DOBS instruction.

The process should also test the CPU-to-SCP buffer for availability. The protocol requires that the process set word 0 of this buffer to non-zero before using it; the SCP sets word 0 to zero after the SCP has used the information in it.

The S pulse of the DOBS *ac*, SCP instruction notifies the SCP that the B register is full. The SCP sets the BUSY flag to 0 when the B register is again available to the process.

The contents of the accumulator specified in the DOBS *ac*, SCP instruction are interpreted by the SCP as a function request and an SCP log enable/disable request. (See the description of the Enable/Disable Error Reporting instruction.)

The SCP then reads the B register and performs the indicated function. The SCP sets the BUSY flag to 0. If the CPU-to-SCP buffer is used by the indicated function, the SCP sets word 0 of the buffer to zero after it has finished using the information stored in that buffer.

Enable/Disable Error Reporting
DOBS *ac*,SCP

0	1	1	ac	1	0	0	0	1	1	0	0	1	0	1
0	1	2	3 4	5	6	7	8	9	10	11	12	13	14	15

The *Enable/Disable Error Reporting* instruction sets the SCP BUSY flag, clears the DONE flag, and uses the contents of the specified accumulator to enable or disable CPU error reporting and to perform the command (function) contained in the command field. Following is the accumulator format:

Reserved	E	Command	Interface Block
0 15	16	17 23	24 31

Bits	Name	Contents or Function
0-15	Reserved	These bits are reserved for future use.
16	E	The E flag enables the SCP error reporting (1 = enable; 0 = disable).
17-23	Command	The SCP performs the function defined by these bits:
		Command (octal) **Name**
		000 No-op
		001 Set time
		002 Select SCP power down mode
		003 Disable SCP power down mode
		004 Set block
		005 Enable all ERCC
		006 Mask ERCC page
		007 Mask soft ERCC
		010 Mask all sniff
		011 Disable all ERCC
		012 Undefined
		thru
		176 Undefined
		177 Enter diagnostic sequence
24-31	Interface Block	Depending on the command, the CPU or SCP uses bits 24 through 31 as a physical address pointer to a multiword block in page zero. Note: This address must be in the range of 0 to 377_8.

The E flag (or enable command) enables CPU error reporting. When the CPU or SCP wishes to report an error, it uses the page zero address specified by the last set block command as a pointer to a double-word physical address. This address in turn points to a 16-word block that the CPU or SCP can use to report error data. The first word of the block receives the error code. The remaining 15 words are available for reporting extended error status information.

If the SCP should interrupt the CPU, the SCP disables error reporting until the process issues a new enable command.

For example, under a Data General operating system, the CPU uses the first word of the error block as the SYSLOG code number. Any error that requires extended error status will also cause the entire 16-word block (including the code number) to be logged as the data area of the SYSLOG entry.

The commands are as follows:

- No-op (command 000_8)

  The No-op command allows the enabling or disabling of SCP-to-CPU logging without specifying any other function. The command enables ERCC error reporting, but does not change the current ERCC reporting mode.

- Set time (command 001_8)

  The Set Time command requires a CPU-to-SCP buffer of three words as follows:

Word	Contents
0	Number of days since December 31, 1967
1, 2	Number of seconds since midnight

NOTE: *Word 0 of the buffer will be set to zero on completion of this function.*

- Select SCP Power-Down Mode (command 002_8)

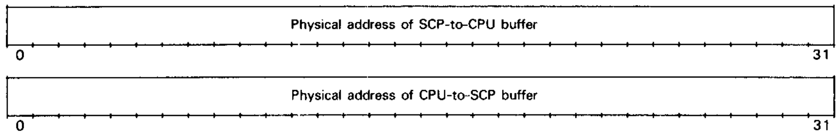
  The Select SCP Power-Down Mode command places the SCP in the power-down mode, with power fail interrupts on device code 0 reported as maskable SCP interrupts. The SCP does not use the page zero address entered with this command.

- Disable SCP Power-Down Mode (command 003_8)

  The Disable SCP Power-Down Mode command removes the SCP from the power-down mode. The SCP no longer intercepts power fail interrupts.

- Set Block (command 004_8)

  The Set Block command specifies to the SCP the address of the SCP/CPU interface block. The address points to a four-word block in page zero. The format of the four-word block is as follows:


Physical address of SCP-to-CPU buffer
0
31
Physical address of CPU-to-SCP buffer
0
31

  Note: Both physical addresses are nonindirectable.

  CPU-to-SCP buffer lengths are specified by the functions that use them. SCP-to-CPU logging requires an SCP-to-CPU buffer of 16 words.

- Enable All ERCC Error Reporting (command 005_8)

  The Enable All ERCC Error Reporting command enables the SCP to detect and report any memory error.

  – Single-bit -- 1-bit ERCC error detected during memory read

  – Multibit -- 2-bit (or more) ERCC error detected during memory read

  – Soft-sniff -- 1-bit ERCC error detected during memory refresh

  – Hard-sniff -- 2-bit (or more) ERCC error detected during memory refresh

  This function requires an SCP-to-CPU buffer of five words.

  NOTE: After a reset or power restore, reporting of any ERCC codes is turned off until this function code (005_8) is issued.

- Mask ERCC Page (command 006_8)

  The Mask ERCC Page command re-enables error reporting. The CPU-to-SCP page 0 address specifies the address of a word containing the page number to be masked out.

- Mask Soft ERCC Error Reporting (command 007_8)

  The Mask Soft ERCC Error Reporting command disables all of memory from single-bit, soft-sniff, and hard-sniff error reporting; detection and correction remain enabled. The processor reports multibit memory errors.

- Mask All Sniff Error Reporting (command 010_8)

  The Mask All Sniff Error Reporting command disables all of memory from soft-sniff and hard-sniff error reporting; detection and correction remain enabled. The processor reports single-bit and multibit memory errors.

- Disable All ERCC (command 011_8)

  The Disable All ERCC command disables all of memory from single-bit, multibit, soft-sniff, and hard-sniff error reporting; detection and correction remain enabled.

- Enter Diagnostic Sequence (command 177_8)

  Disable CPU error reporting and all previously enabled functions. The SCP does not use the page zero address entered with this DOBS SCP instruction. The SCP uses the previous page zero address as a pointer to the SCP/CPU interface block. The SCP clears its BUSY flag. The SCP remains in diagnostic mode until either a console reset occurs or the process issues another DOBS SCP instruction.

  When the process issues the second DOBS SCP instruction, the SCP first places the contents of bits 16 through 31 of the specified accumulator into word 0 of the *SCP-to-CPU* buffer. The SCP then reads words 1 through 7 from the *CPU-to-SCP* buffer, inverts them, and writes them back to their respective locations in the *SCP-to-CPU* buffer. Upon completion, the SCP transmits a status 0 to the host, sets the DONE flag, and interrupts the CPU.

  Diagnostic mode can also be cleared by issuing an IORST instruction. The SCP clears diagnostic mode on RESET, START, and BOOT commands.

  NOTE: The SCP-to-CPU interface block address will be lost when diagnostic mode is terminated.

Return SCP Status
DIB ***ac*****,SCP**
DIBC ***ac*****,SCP**

0	1	1	ac		0	1	1	1	0	1	0	0	1	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

NOTE: *The DIB* ac, *SCP and DIBC* ac, *SCP instructions are equivalent.*

The *Return SCP Status* instruction clears the SCP BUSY and DONE flags and returns a code to the specified accumulator denoting the current status of the SCP. The CPU expects all information except status information to be passed using the SCP-to-CPU buffer via the SCP/CPU interface block. If there is information in this buffer, the CPU does not expect this data to be valid until after it has issued the DIB instruction. Following is the accumulator format:

Reserved		Status	
0	15	16	31

Bits	Name	Contents or Function
0-15	Reserved	These bits are reserved for future use
16-31	Status	The codes returned to these bits denote the current status of the SCP as follows:

Status (octal)	Meaning
000000	Log information is in current SCP-to-CPU buffer. SCP logging is disabled. The first word of the log buffer (word 0) is interpreted as a log code; the use of the remaining 15 words are dependent on the log code (extended status). The status codes and their definitions are:

Code (octal)	Definition
007	Power fail detected
050	Power restore detected
053	Single-bit ERCC error detected (See ERCC extended status)
054	Multibit ERCC error detected (See ERCC extended status)
055	Sniff or I/O detected multibit ERCC error (See ERCC extended status)

ERCC extended status

The four-word extended status for ERCC codes 053, 054, and 055 is:

Word	Contents
0	Status code (053, 054, or 055)
1	Status
2	Physical page number
3	Double-word on module
4	Syndrome bits

Word 1 Status:

Bits	Definition
0-11	reserved
12	CPU access
13	I/O access
14	reserved
15	sniff access

Code (octal)	Definition
140	SCP error logging enabled
141	SCP error logging disabled
142	Processor halted
143	SCP BOOT command has been executed
144	Power down
145	Power up
146	Reserved
147	Reserved
150	Battery backup complete
153	Microsequencer parity error
154	System cache parity error
155	System cache to Bank Controller parity error
156	IOC parity error
157	Sbus timeout
160	Sbus parity error
161	Operating system error
162	SCP error log disk error

Bits	Name	Contents or Function
		163 Infinite protection fault
		164 Infinite page fault
		165 Instruction cache enabled
		166 Instruction cache disabled
		167 Reserved
		170 Reserved
		171 SCP RESET command has been executed
		172 Address Translation Unit enabled
		173 Address Translation Unit disabled
		174 Illegal PIO command
		175 Reserved
		176 Reserved
		177 SCP HALT command has been executed
		200 SCP CONTINUE command has been executed
		201 SCP START command has been executed
		202 SCP INIT command has been executed
		203 SCP has disabled interrupts initiated by the Bank Controller for soft ERCC errors. (This occurs when there are multiple "stuck" soft ERCC errors and the interrupt frequency gets so high that it locks out the system console.)
		204 Reserved
		205 Hard interrupt from an unknown source
		000001 SCP reset. The SCP is reset and must be reinitialized with the *DOBS ac,SCP* instruction and a command 4. (All previously enabled functions have been disabled, and the SCP-to-CPU interface block address has been lost.)
		000002 SCP function request acknowledge. This acknowledgment indicates to the process that a requested function has been completed by the SCP.
		000003 SCP-requested function is in error. The SCP reports an unknown error with this code. For instance, if a required SCP/HOST interface block has not been defined, or if an undefined function request is made, or if invalid data is passed to the SCP (through the HOST-to-SCP buffer), the SCP issues this code.
		177777 SCP is in diagnostic sequence.

Data Channel/Burst Multiplexor Channel

The data channel (DCH) provides I/O communication for medium-speed devices and synchronous communications. The burst multiplexor channel (BMC) is a high-speed communications pathway that transfers data directly between main memory and high-speed peripherals. I/O-to-memory transfers for both DCH and BMC always bypass the address translator.

DCH/BMC Maps

A map controls a DCH or BMC. This map is a series of contiguous map slots, each of which contains a pair of map registers — an even-numbered register and its corresponding odd-numbered register.

The MV/10000 computer supports 16 DCH maps, each of which contains 32 map slots. The DCH sends to the processor a logical address with each data transfer. The processor translates the logical address into a physical address using the appropriate map slot for that address.

The device controller performing the data transfer controls the BMC. No program control or CPU interaction is required, except when setting up the BMC's map table. The BMC has two address modes and contains its own map.

BMC Address Modes

The BMC operates in either the unmapped mode — that is, the *physical* mode — or the mapped mode — that is, the *logical* mode.

In the unmapped mode, the BMC receives 20-bit addresses from the device controllers and passes them directly to memory. As the BMC transfers each data word to or from memory, it increments the destination address, causing successive words to move to or from consecutive locations in memory.

If the controller specifies the mapped mode for a data transfer, the high-order 10 bits of the logical address form a logical page number, which the BMC map translates into a 10-bit physical page number. This page number, combined with the 10 low-order bits from the logical address, forms a 20-bit physical address, which the BMC uses to access memory.

BMC Map

The BMC uses its own map to translate logical page numbers into physical ones. The map table contains 1024 map registers, the odd-numbered registers each containing a 10-bit physical page number. The BMC uses the logical page number as an index into the map table, and the contents of the selected map register become the high-order 10 bits of the physical address.

Note that when the BMC performs a mapped transfer, it increments the destination address after it moves each data word. If the increment causes an overflow out of the 10 low-order bits, this selects a new map register for subsequent address translation. Depending on the contents of the map table, this could mean that the BMC cannot transfer successive words to or from consecutive pages in memory.

DCH/BMC Registers

The MV/10000 system contains 512 DCH registers and 1024 BMC registers. The map registers are numbered from 0 through 7777_8, as explained in Table 7.10 and depicted in Figure 7.1.

Registers (Octal)	Description
0000-3776	Even-numbered registers are the most significant half of BMC map positions 0-1777
0001-3777	Odd-numbered registers are the least significant half of BMC map positions 0-1777
4000-5776	Even-numbered registers are the most significant half of DCH map positions 0-777
4001-5777	Odd-numbered registers are the least significant half of DCH map positions 0-777
6000	I/O channel definition register
6001-7677	Reserved
7700	I/O channel status register
7701	I/O channel mask register
7702-7777	Reserved

Table 7.10 Device map registers 0000-7777

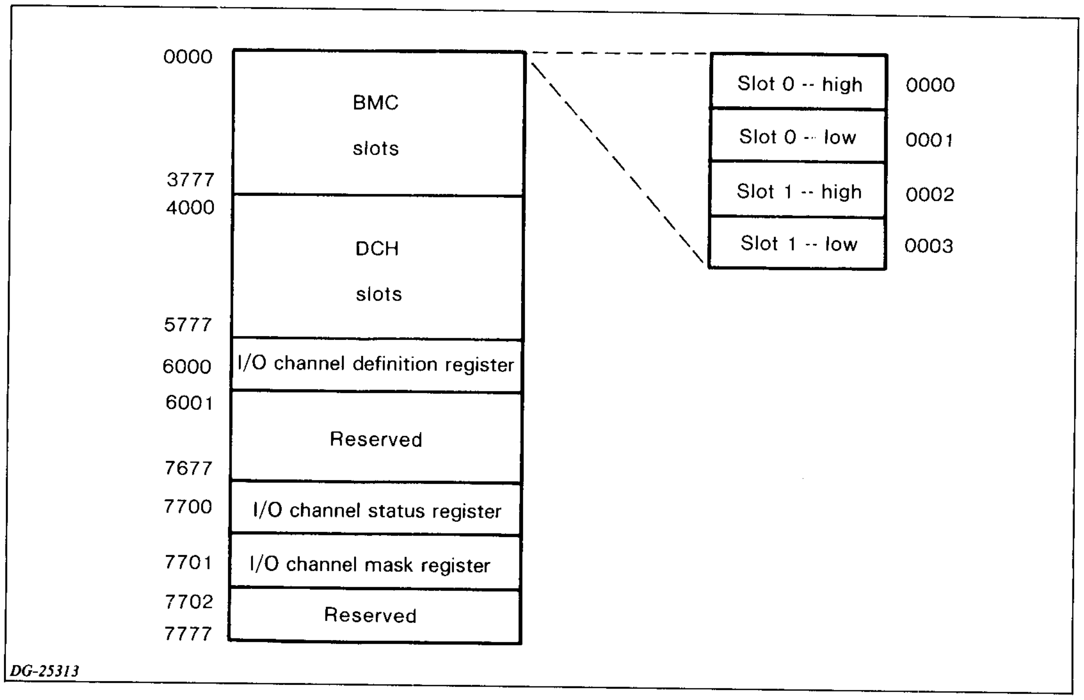


Figure 7.1 DCH/BMC registers

The device map registers and their formats follow.

BMC/DCH Even-Numbered Register Formats

The processor translates the contents of the BMC and DCH even address registers (0000-3776$_8$ and 4000-5776$_8$, respectively) as:

V	D	Hardware Reserved
0	1	2–15

Bits	Name	Contents or Function
0	V	Validity bit; if 1, the processor denies access
1	D	Data bit If 0, the channel transfers data If 1, the channel transfers zeros
2-15	Hardware Reserved	Write to with zeros; reading these bits returns an undefined state

BMC/DCH Odd-Numbered Register Formats

The processor translates the contents of the BMC and DCH odd address registers (0000-3777$_8$ and 4001-5777$_8$, respectively) as:

Res	Physical Page Number
0–1	2–15

Bits	Name	Contents or Function
0-1	Res	Hardware reserved; write to with zeros. Reading these bits returns an undefined state
2-15	Physical Page Number	Physical page number associated with the logical page that referred to that particular slot

I/O Channel Definition Register Format

The I/O channel definition register (6000$_8$) provides status information.

E	Reserved	BV	DV	Res	BX	A	P	DIS	I/O Channel	M	0
0	1–2	3	4	5	6	7	8	9	10–13	14	15

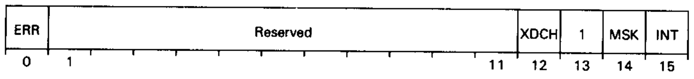
NOTE: *Writing to bits 3, 4, 7, or 8 with a 1 complements these bits.*

The C/350 IORST instruction clears bits 0, 3, 4, 7, 8, 9, and 14.

Bits	Name	Contents or Function
0	E	Error flag; if 1, an error has occurred on the I/O channel (0 only when all other error bits are 0).
1,2	Reserved	Bits 1 and 2 are reserved for future use and are returned as zero.
3	BV	BMC validity error flag; if 1, BMC validity protect error has occurred.
4	DV	DCH validity error flag; if 1, DCH validity protect error has occurred.
5	Res	Bit 5 is reserved for future use and is returned as zero.
6	BX	BMC transfer flag; if 1, BMC transfer is in progress (read only bit).
7	A	BMC address error; if 1, the channel has detected an address parity error.
8	P	BMC data error; if 1, the channel has detected a data parity error.
9	DIS	Disable block transfer; if 1, disables BMC block transfers to and from I/O memory port (read/write bit).
10-13	I/O channel	I/O channel number.
14	M	DCH mode; if 1, DCH mapping is enabled.
15	0	Always set to 0.

I/O Channel Status Register Format

The read-only I/O channel status register (7700_8) provides I/O channel status information.



Bits	Name	Contents or Function
0	ERR	If 1, the I/O channel has detected an error indicated by the IOC status register or a memory parity error.
1-11	Reserved	Bits 1 through 11 are reserved for future use.
12	XDCH	If 1, extended DCH map slots and operations are supported.
13	—	Always set to 1.
14	MSK	If 1, MSK prevents all devices connected to the channel from interrupting the CPU. However, the INTA instruction returns the device code of any device with its DONE flag set.
15	INT	Interrupt pending; if 1, the channel is attempting to interrupt the CPU.

I/O Channel Mask Register Format

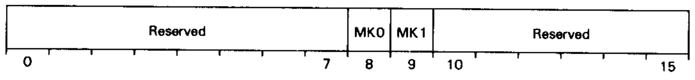
The write-only I/O channel mask register (7701_8) specifies a mask flag for each channel. When an I/O channel mask flag is set to 1, the processor ignores all interrupt requests from devices on that channel.

The INTA instruction with channel 0 or 1 returns on that channel the device code of the highest priority interrupting device, which has its DONE flag set. With channel 7, the INTA instruction returns the device code of the highest priority interrupting device on the highest priority channel, regardless of the state of the I/O channel mask register flags.

An I/O channel Bus Reset PRTRST instruction zeroes the mask bit for one channel (0 or 1) or for both channels (7).

NOTE: *A CIO read to the I/O channel mask register produces undefined results.*

The format of the I/O channel mask register is as follows:



Bits	Name	Contents or Function
0-7	Reserved	Bits 0 through 7 are reserved for future use.
8	MK0	If 1, MK0 prevents all devices connected to channel 0 from interrupting the CPU. A system reset sets MK1 to a 0.
9	MK1	If 1, MK1 prevents all devices connected to channel 1 from interrupting the CPU. A system reset sets MK1 to a 1.
10-15	Reserved	Bits 10 through 15 are reserved for future use.

DCH/BMC Map Instructions

The CIO, CIOI, and WLMP instructions initiate DCH/BMC map loads and reads when in mapped mode, with the LPHY instruction used for loads in unmapped mode. The channel sets its BUSY flag to 1 when a map load or read is in progress. There is no DONE flag, and the channel never causes program interrupts. Table 7.11 lists the instructions that affect the DCH and BMC maps.

Assembler Statement	Function
WLMP	Loads BMC/DCH map slots from memory
CIO, CIOI	Returns BMC/DCH status or loads map registers (1/2 slot) from accumulators
LPHY	Translates logical addresses to physical addresses
IORST	Clears bits 0, 3, 4, 7, 8, 9, and 14 of the I/O channel definition register, which disables data channel maps

Table 7.11 DCH/BMC map instructions

The CIO, CIOI, and LPHY instructions are described in *Principles of Operation, 32-Bit* ECLIPSE® *Systems*. The IORST and NIOC instructions are presented earlier in this chapter.

Wide Load Map
WLMP

1	0	1	0	0	1	1	1	1	1	1	1	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The WLMP instruction in conjunction with three accumulators loads successive double words from memory into successive DCH or BMC map slots.

The double word contained in AC0 refers to the first map slot in the specified I/O channel that the WLMP instruction will load. AC1 contains a 16-bit unsigned count of the number of map slots in the I/O channel to be loaded. AC2 contains the effective address of the first double word to be loaded into the referenced I/O channel slots.

For each map slot loaded, the accumulators are incremented or decremented as follows:

AC0 is incremented by one.

AC1 is decremented by one.

AC2 is incremented by two.

Upon completion of the WLMP instruction:

AC0 references the map slot following the last slot loaded;

AC1 contains a 0 in the 16 least significant bits;

AC2 contains the address of the word following the last double word loaded.

NOTE: *If AC1 is initially 0, a no-op is performed.*

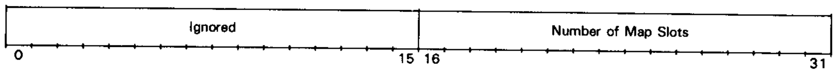
The accumulator formats for the WLMP instruction follow.

AC0 contains a double word:

Reserved	Channel	Map Slot Number
0 – 17	18 – 20	21 – 31

Bits	Name	Contents or Function
0-17	Reserved	Bits 0 through 17 are reserved for future use.
18-20	Channel	The channel number can be 0, 1, or 7; channel 7 affects both channels.
21-31	Map Slot Number	Indicates which map slot the instruction will load. **Number** — **Meaning** $0\text{-}1777_8$ — Loads a BMC slot $2000\text{-}2177_8$ — Loads a DCH slot

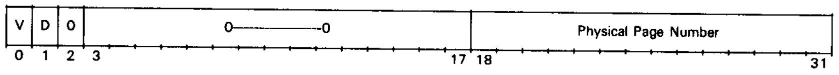
AC1 contains a 16-bit unsigned count:



Bits	Name	Contents or Function
0-15	Ignored	Bits 0 through 15 are ignored by the WLMP instruction.
16-31	Number of Map Slots	Unsigned count of the number of map slots that the WLMP instruction will load.

AC2 contains an effective address that refers to the first double word the WLMP instruction will load.

The contents of these double words are in the following format:



Bits	Name	Contents or Function
0	V	Valid. Set to 0 implies valid; set to 1 implies access denied.
1	D	Data. Set to 0 implies transfer data; set to 1 implies transfer zeros.
2-17	0	Must be set to 0.
18-31	Physical Page Number	Physical effective address containing the first double word that the instruction will load.

The effect of the setting of the V and D bits and the direction of the transfer are:

V	D	Transfer Direction	Action
0	0	From I/O port	Transfer data.
0	1	From I/O port	Transfer zeros from either DCH or BMC device.
1	—	From I/O port	Transfer aborted — flag error to device.
0	0	To I/O port	Transfer data.
0	1	To I/O port	Transfer zeros to either DCH or BMC device.
1	—	To I/O port	Transfer aborted — flag error to device.

NOTE: *"From I/O port" implies memory to device; "to I/O port" implies device to memory.*

Upon detection of an invalid map entry due to an active device:

For the BMC — The active BMC requesting device is flagged.

For the DCH — Bit 4 of the IOC Status Register is set to 1.

WLMP is a privileged and interruptible instruction.

Universal Power Supply Controller

Device Code
4_8

Assembler Mnemonic
UPSC

Priority Mask Bit
13

The universal power supply controller (UPSC) is a daughter board inside the power supply. With the help of an on-board microprocessor, the UPSC performs a power-up diagnostic self test, monitors the system power, and reports failures, problems, and status information to the MV/10000 computer.

The UPSC controls power-up and power-down sequencing; the transfer to battery operation; I/O operations with the MV/10000; and output voltage margining.

In addition, the UPSC monitors problems on the power supplies (such as overtemperature and overcurrent conditions); AC overvoltages and undervoltages; reed switches for sensing overload on +5V (or determining that the power switch was turned off); battery backup faults; and fan failures.

Device Flags

Device flag commands to the UPSC determine the enabling or disabling of UPSC interrupts.

f=**S**	Sets the BUSY flag to 1 and the DONE flag to 0.
f=**C**	Sets the BUSY and DONE flags to 0.
f=**P**	Sets the BUSY flag to 1 and the DONE flag to 0.

UPSC Instructions

Table 7.12 lists the I/O instructions that affect the UPSC.

Assembler Statement	Function
DOAS *ac*,UPSC	Write data to UPSC
DOAP *ac*,UPSC	Request data from UPSC
DIA *[f] ac*,UPSC	Read data from UPSC
IORST	Clears BUSY and DONE flags and interrupt mask bit (bit 13)

Table 7.12 I/O instructions for UPSC

Write Data to UPSC
DOAS*[f]* *ac*,UPSC

0	1	1	ac		0	1	0	0	1	0	0	0	1	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

When the MV/10000 computer issues the DOAS instruction, the UPSC sets the BUSY flag. The UPSC resets the BUSY flag and sets the DONE flag when the UPSC completes the operation.

The *Write Data to UPSC* instruction sends the contents of the accumulator to the UPSC. The four registers that can be written on the UPSC are defined as follows:

Register	Name	Contents or Function
0	Control register	Selects reporting mode, power margining, and enable/disable battery backup.
1	Power margining register	When the backpanel is jumpered for margining or margining is selected using the control register, the +5V logic, +5V memory, -5V memory, and +12V memory voltages can be increased or decreased.
2	Reserved	Reserved for future use.
3	Diagnostic test register	Verify the data path between the MV/10000 computer and UPSC or enable battery test.

The bit descriptions in the following tables explain the bit function when a bit is set.

Register 0
Control Register

Undefined		0	0	Res	BT	ALT	COM	BBU	PFM
0	7	8	9	10	11	12	13	14	15

Bits	Name	Contents or Function
0-7	Undefined	Bits 0 through 7 are undefined.
8,9	Register 0	The control register bits 8 and 9 equal zero.
10	Res	Bit 10 is reserved for future use.
11	BT	Remove AC power to allow battery testing.
12	ALT	Mask out power fail interrupts. When ALT is 1, power fail skips (SKPDN and SKPDZ) will always behave as if there is no power fail.
13	COMM	UPSC can interrupt MV/10000 when a fault occurs.
14	BBU	Disables the battery backup unit.
15	PFM	Enable power margining through program control.

Register 1
Power Margining Register

Undefined		0	1	+5LI	+5LD	ALLI	ALLD	+5MI	+5MD
0	7	8	9	10	11	12	13	14	15

A voltage is in the nominal state when the corresponding bit is 0. The voltage is margined when the corresponding bit is 1 and the MV/10000 computer is jumpered or programmed for margining.

All percentages are additive. For instance, when bits 12 and 15 are used together, the voltage for +5V memory increases approximately 5 percent, while the −5V memory and +12V memory increase approximately 8 percent.

WARNING: *Do not margin any voltage greater than 8%.*

Bits	Name	Contents or Function
0-7	Undefined	Bits 0 through 7 are undefined.
8,9	Register 1	The power margining register bits 8 and 9 equal 01_2.
10	+5LI	Increase +5V logic approximately 2.5%.
11	+5LD	Decrease +5V logic approximately 5%.
12	ALLI	Increase +5V memory, -5V memory, and +12V memory voltages approximately 8%.
13	ALLD	Decrease +5V memory, -5V memory, and +12V memory voltages approximately 8%.
14	+5MI	Increase +5V memory approximately 3%.
15	+5MD	Decrease +5V memory approximately 3%.

Register 3
Diagnostic Test Register

0	0	0	0	0	0	0	0	1	1	0	0	0	0	BTE	COMP
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The UPSC performs the battery test or bit test as specified by bits 14 and 15 of the accumulator. To complete the command, the UPSC requires a second DOAS *ac*,UPSC instruction.

When the UPSC fails to detect the second DOAS instruction, the UPSC automatically exits the diagnostic test. The UPSC indicates a timeout by setting the DONE flag and the appropriate fault code in the fault code register.

Bits	Name	Contents or Function
0-7	0 — 0	Bits 0 through 7 are reserved and must be zero.
8,9	Register 3	The diagnostic test register bits equal 11_2.
10-13	0 — 0	Bits 10 through 13 are reserved and must be zero.
14	BTE	Battery Test Enable. If the accumulator contains 2_8, the battery test is enabled. You initiate the test with the second DOAS to bit 11 of register 0 (BT). **NOTE:** The BTE bit must be set before the BT bit.
15	COMP	Complement. If the accumulator contains 0_8 or 1_8, the UPSC reads the data from the second DOAS (A buffer), complements it if COMP is 1, and then returns the data to the A buffer. The A buffer can then be read with the DIA instruction.

Request Data From UPSC
DOAP*[f]* *ac*,UPSC

0	1	1	ac	0	1	0	1	1	0	0	0	1	0	0
0	1	2	3 4	5	6	7	8	9	10	11	12	13	14	15

The *Request Data From UPSC* instruction uses bits 13 through 15 of the accumulator to request specific information from the UPSC.

0	0	0	0	0	0	0	0	0	0	0	0	0	octal value
0	1	2	3	4	5	6	7	8	9	10	11	12	13 15

Bits	Name	Contents or Function
0-12	Reserved	Bits 0 through 12 are reserved and must be 0.
13-15	Octal value	
	0	Read control bits.
	1	Read battery backup and margining bits.
	2	Read power supply system status.
	3	Read fault code register.
	4	Read UPSC code revision number.

Read Data From UPSC
DIA*[f]* *ac*,UPSC

0	1	1	ac	0	0	1	0	0	0	0	0	1	0	0
0	1	2	3 4	5	6	7	8	9	10	11	12	13	14	15

The *Read Data From UPSC* instruction loads the data from the UPSC A buffer into the accumulator. The previous *Request Data From UPSC* instruction defines the data read from the A buffer.

Read Control Bits

0	0	0	0	0	0	0	0	0	0	0	JFM	ALT	COM	BBU	PFM
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Bits	Name	Contents or Function
0-10	Reserved	Returned as zero.
11	JFM	Power margining is enabled through hardware jumpering.
12	ALT	Power fail is masked out.
13	COMM	UPSC can interrupt the MV/10000 computer when a fault occurs.
14	BBU	The battery backup unit is disabled.
15	PFM	Power margining is enabled through program control.

Read Battery Backup and Margining Bits

0	0	0	0	0	0	0	0	BAT	0	+5LI	+5LD	ALLI	ALLD	+5MI	+5MD
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Bits	Name	Contents or Function
0-7	Reserved	Returned as zero.
8	BAT	The battery backup is connected and in use. (The bit is cleared if system is not running on batteries, a battery fault occurs, or the BBU flag is set.)
9	Reserved	Returned as zero.
10	+5LI	+5V logic is increased approximately 2.5%.
11	+5LD	+5V logic is decreased approximately 5%.
12	ALLI	+5V memory, -5V memory, and +12V memory voltages are increased approximately 8%.
13	ALLD	+5V memory, -5V memory, and +12V memory voltages are decreased approximately 8%.
14	+5MI	+5V memory is increased approximately 3%.
15	+5MD	+5V memory is decreased approximately 3%.

Read Power Supply System Status

0	0	0	0	0	0	0	0	0	0	0	0	PART	FUL	RUN	CHR
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Bits	Name	Contents or Function
0-11	Reserved	Returned as zero.
12	PART	The system is equipped with partial battery backup.
13	FULL	The system is equipped with full battery backup.
14	RUN	The system is running on the batteries.
15	CHAR	The batteries are recharging.

Read Fault Code Register

0	0	0	0	0	0	0	0	0	Fault Code	Fault Category
0	1	2	3	4	5	6	7	8	9 12	13 15

Bits	Name	Contents or Function
0-8	Reserved	Returned as zero.
9-12	Fault Code	Specifies the fault code for a specific fault category. (See Table 7.12.)
13-15	Fault Category	Specifies the fault categories (in a range of 0 through 7)

When the UPSC power system detects a fault, it loads the fault code and category into the fault code register and then flashes the code on the MV/10000 front panel LEDs. The fault code register retains the code of the last fault, even if the fault clears. For example, if a fan takes too long to come up to speed, it can cause a fan fault. However, when the fan is running, the fault code register retains the fault, even though the fault clears. Appendix F lists the fault codes by fault category.

Chapter 8
Memory and System Management

This chapter describes the address translator, the system and memory management instructions, the sequence of events initiated by a privileged fault, and the reserved memory.

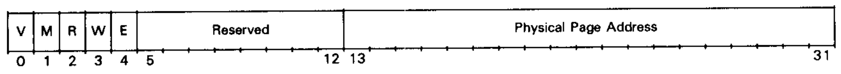
Address Translator

The CPU *address translator* converts the logical address of a piece of data into a physical address in memory.

To perform the translation, the address translator uses a series of *page tables*, which contain information on the pages of logical memory. These tables contain one entry for each page. The tables indicate whether or not the page is currently in physical memory and whether or not the page is valid (and the process can access it) and provide the information needed for logical-to-physical address translation.

To avoid referring to a page table for every memory reference, the address translator maintains a table of address translations and access privileges for 1024 recently referenced pages (128 per segment). The hardware checks the address translator's table for entries before referring to a page table in memory.

Page Table Entry Format



Bits	Name	Contents or Function
0	V	Valid access flag. 0 Indicates invalid page. 1 Indicates valid page.
1	M	Memory-resident page. 0 Indicates disk-resident page. 1 Indicates memory-resident page.
2	R	Read access flag. 0 Indicates read access denied. 1 Indicates read access.
3	W	Write access flag. 0 Indicates write access denied. 1 Indicates write access.
4	E	Execute access flag. 0 Indicates execute access denied. 1 Indicates execute access.
5-12	Reserved	Bits 5 through 12 are reserved for future use.
13-31	Physical Page Address	The physical address of a page in memory.

Because the memory references for a procedure tend to be clustered in several pages, a page translation is likely to be in the address translator's table of address translations. The address translator updates the entries in this table as execution continues.

Referenced and Modified Bits

The address translator also controls two memory management bits for each page: the *modified bit* and the *referenced bit*. The operating system uses these bits during *page faults*.

A *page fault* occurs when a process refers to a page that is not currently in physical memory. Each time a page fault occurs, the *page fault handler* must transfer a new page from disk to physical memory. The page fault handler can remove a page from physical memory to make room for the new page. The modified bit indicates whether or not the old page is the same as it was when it came into physical memory.

- If the modified bit for the old page is 1, it indicates that it is a modified page, and the page fault handler must save the modified page on the disk before it can bring in the new page.
- If the modified bit is 0, the copy of the old page on disk is still valid, and the page fault handler can move the new page immediately into memory.

The referenced bit helps determine which page in memory the page fault handler can replace with a new page from disk. In general, the page to which the processor refers most infrequently is the page replaced. The referenced bit allows the operating system to determine the frequency of references to individual pages.

Protection Validation

The address translator performs all protection system hardware checks. These checks include access validation, page validation, segment crossing validation, and others. If any of the checks fails, the address translator initiates a protection fault to the operating system. For more information on types of protection checks, refer to *Principles of Operation, 32-Bit* ECLIPSE® *Systems.*

Memory/System Management Instructions

Table 8.1 lists the memory/system management instructions. For further information, refer to *Principles of Operation, 32-Bit* ECLIPSE® *Systems.* Appendix C lists the accumulator formats for the *Load CPU Identification* and *Narrow Load CPU Identification* instructions.

Instruction	Function
ECLID	Load CPU identification
LCPID	Load CPU identification
LMRF	Load modified and referenced bits
LSBRA	Load all segment base registers
LSBRS	Load segment base registers 1-7
NCLID	Narrow load CPU identification
ORFB	OR referenced bits
PATU	Purge address translator
RRFB	Reset referenced bits
SMRF	Store modified and referenced bits
WDPOP	Pop context block (return from page fault)

Table 8.1 Memory and system management instructions

Privileged Faults

Upon detection of a privileged fault, the address translator generates either a page or protection fault. The interpretation of the validity and appropriate access bits in a page table entry, coupled with the occurrence of one of the following conditions, initiate a page fault.

- An attempt to refer to a location that is part of the logical address space, but is not part of the physical address space.
- The result of a logical address reference that requires a two-level page table, but is only allocated a one-level page table.

Page Faults

When a page fault occurs, the following actions result:

- If the current segment is not 0, the processor stores the frame pointer and stack pointer in their respective locations in page zero of the current segment and performs a segment crossing to segment 0.
- The processor uses the contents of locations 32_8 and 33_8 of segment 0 as a base address to store a context block (the internal state of the machine) in memory. (See Appendix D for context block structure.)
- The processor initializes the segment 0 stack from page zero of segment 0.

- The processor stores the fault code in AC1.

Fault Code	Explanation
0	Multiple ERCC fault
1	Page table depth
2	Page table page fault
3	Reserved
4	Normal object reference

- The processor disables interrupts for one instruction, jumps indirect through locations 30_8 and 31_8 of segment 0, and executes the first instruction of the page fault handler.

  NOTE: *If an additional page fault occurs during any of these actions, the processor halts.*

Once the page fault handler corrects the fault (for example, once it brings the page into physical memory or creates a two-level page table), the execution of a WDPOP instruction restarts the program. The WDPOP instruction restores the processor state from information contained in the context block. Figure 8.1 summarizes the actions taken upon detection of a page fault.

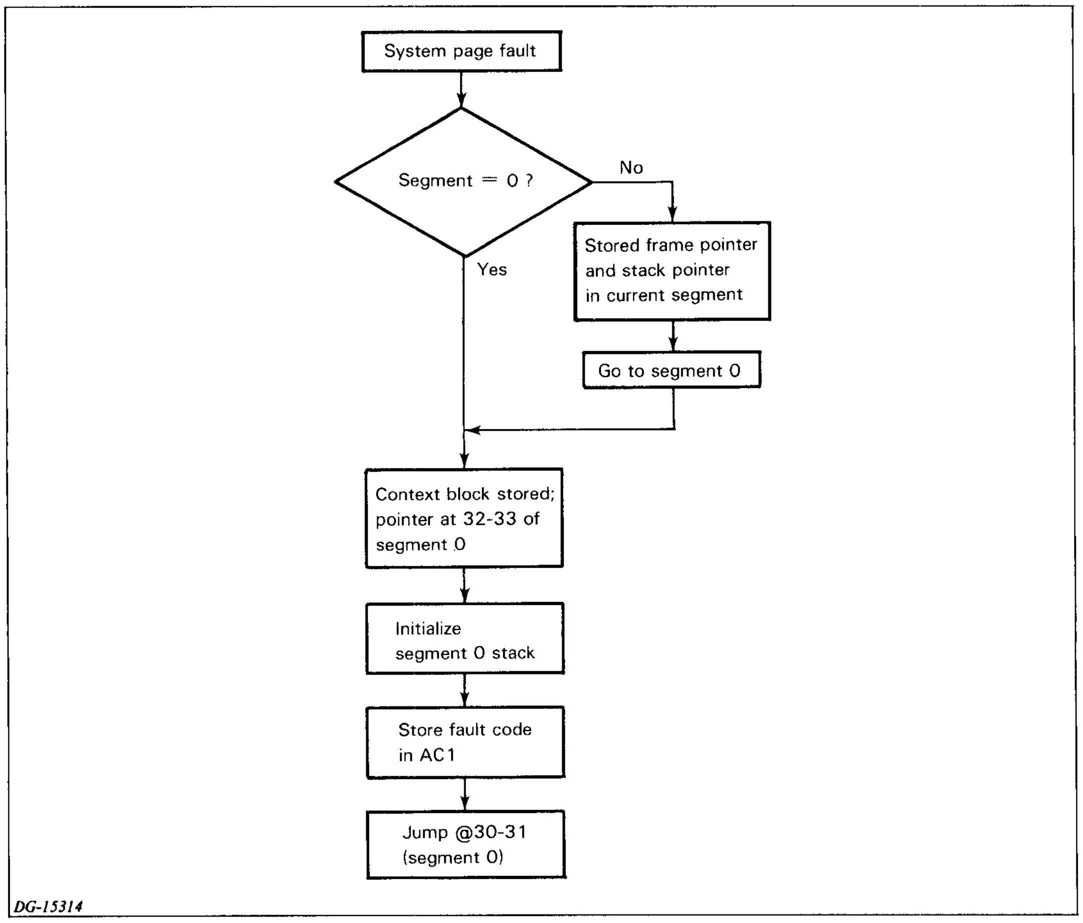

System page fault
Segment = 0 ?
No
Yes
Stored frame pointer and stack pointer in current segment
Go to segment 0
Context block stored; pointer at 32-33 of segment 0
Initialize segment 0 stack
Store fault code in AC1
Jump @30-31 (segment 0)
DG-15314

Figure 8.1 Page fault sequence

Protection Faults

With the address translator enabled, the following (in descending order of priority) will produce a protection violation fault:

- privileged or I/O instruction violation
- defer (indirect) address violation
- inward reference violation
- segment validity violation
- page table validity violation
- read, write, or execute access violation
- segment crossing violation

When a fault occurs, AC1 receives a code indicating the type of fault. (Refer to Appendix F.) *Principles of Operation, 32-Bit* ECLIPSE® *Systems* describes the remainder of the protection violation fault procedure.

Reserved Memory

When a privileged/nonprivileged fault occurs, the processor transfers control to an appropriate fault handler. A reserved storage location in page zero of each segment contains the starting address of the fault handler.

The processor interprets segment 0, page zero locations differently from segments 1 through 7, page zero locations. For example, segment 0 contains pointers to privileged fault handlers, and segments 1 through 7 reserve these locations. Appendix D describes these locations for all segments.

NOTE: *The first instruction of the protection fault handler executes before the processor honors interrupts.*

The privileged *Store State Pointer* (SSPT) instruction is a no-op.

Chapter 9
C/350 Programming

The ECLIPSE MV/10000 computer is capable of executing ECLIPSE C/350 16-bit programs with only slight program instruction modification.

This chapter describes the operation of the MV/10000 system when it implements C/350 instructions. In this chapter we explain:

- register implementation
- C/350 instruction compatibility
- program flow management
- fault handling
- reserved memory
- CPU identification

Registers

The following C/350 registers are implemented on the MV/10000 computer:

- four 64-bit floating-point accumulators
- four 16-bit fixed-point accumulators
- one 32-bit floating-point status register
- one 15-bit program counter
- one 1-bit Carry flag

The four 64-bit MV/10000 floating-point accumulators are identical to the C/350 floating-point accumulators.

The ECLIPSE C/350 16-bit fixed-point accumulators correspond to bits 16 through 31 of the MV/10000 accumulators.

The 32-bit floating-point status register (FPSR) corresponds to bits 0 through 15 and 49 through 63 of the 64-bit MV/10000 FPSR.

The C/350 15-bit program counter (PC) corresponds to bits 17 through 31 of the MV/10000 31-bit PC.

Execution of C/350 instructions does not generate fixed-point faults thereby leaving the processor status register unaffected. Certain C/350 arithmetic instructions such as

ADD and DIV set the state of the Carry bit. If a program is to detect a particular fault, you must set up a subroutine that checks the state of the Carry bit upon completion of these instructions. A carry from accumulator bit 16 affects the MV/10000 Carry bit upon execution of these C/350 instructions. The instruction dictionary contained in *Principles of Operation, 32-Bit* ECLIPSE® *Systems* describes the C/350 instruction set and the instructions that affect the carry bit.

C/350 instructions function with the narrow stack, and thus use reserved memory locations for stack management without affecting the MV/10000 stack management registers.

Appendix C illustrates the register fields.

Instruction Compatibility

C/350 program flow instructions are limited to an address range of 64 Kbytes.

C/350 instructions that load AC3 with the address of the next instruction *(jump to subroutine)* or push the address of the next instruction onto the narrow stack *(push and jump)* calculate effective addresses within the lower 64 Kbytes of the present segment.

The MV/10000 system does not support the following C/350 instructions:

- XOP, XOP1 (replaced by XOP0)
- Floating-point function instructions FCOSD, FCOSS, FEXPD, FEXPS, FLOGD, FLOGS, FSIND, FSINS, FPLYD, FPLYS, FSQRD and FSQRS
- VCT, SYC, and LMP

Tables 9.1 through 9.4 list the C/350 instructions and the equivalent MV/10000 system instructions.

C/350 Instruction	C/350 Instruction Function	Equivalent Instruction
BAM	Block add and move	—
BLM	Block move	WBLM
BTO	Set bit to 1	WBTO
BTZ	Set bit to 0	WBTZ
CLM	Compare to limits and skip	WCLM
CMP	Character compare	WCMP
CMT	Character move until true	WCMT
CMV	Character move	WCMV
COB	Count bits	WCOB
CTR	Character translate and compare	WCTR
DSZ	Decrement and skip if 0	XNDSZ *
EDIT	Edit decimal and 16-bit alphanumeric data	WEDIT
EDSZ	Extended decrement and skip if 0	XNDSZ
EISZ	Extended increment and skip if 0	XNISZ
ELDA	Extended load accumulator	XNLDA
ELDB	Extended load byte (from memory to AC)	XLDB
ESTA	Extended store accumulator	XNSTA
ESTB	Extended store byte (right byte of AC to byte in memory)	XSTB
ISZ	Increment and skip if 0	XNISZ *
LDA	Load accumulator	XNLDA *
LDB	Load byte (from memory to AC)	WLDB
LSN	Load sign	WLSN
POP	Pop multiple accumulators	WPOP
PSH	Push multiple accumulators	WPSH
SNB	Skip on nonzero bit	WSNB
SZB	Skip on 0 bit	WSZB
SZBO	Skip on 0 bit and set to 1	WSZBO
STA	Store accumulator	XNSTA *
STB	Store byte (right byte of AC to byte in memory)	WSTB

Table 9.1 C/350 fixed-point computing instructions

*With 32-bit processors, the equivalent instruction requires two words.

C/350 Instruction	C/350 Instruction Function	Equivalent Instruction
FAMD	Add double (memory to FPAC)	XFAMD
FAMS	Add single (memory to FPAC)	XFAMS
FDMD	Divide double (FPAC by memory)	XFDMD
FDMS	Divide single (FPAC by memory)	XFDMS
FFMD	Fix to memory (FPAC to memory)	WFFAD *
FLDD	Load floating-point double	XFLDD
FLDS	Load floating-point single	XFLDS
FLMD	Float from memory	WFLAD *
FLST	Load floating-point status register	LFLST **
FMMD	Multiply double (FPAC by memory)	XFMMD
FMMS	Multiply single (FPAC by memory)	XFMMS
FPOP	Pop floating-point state	WFPOP
FPSH	Push floating-point state	WFPSH
FSMD	Subtract double (memory from FPAC)	XFSMD
FSMS	Subtract single (memory from FPAC)	XFSMS
FSST	Store floating-point status register	LFSST **
FSTD	Store floating-point double	XFSTD
FSTS	Store floating-point single	XFSTS
LDI	Load integer (memory to FPAC)	WLDI
LDIX	Load integer extended (memory to FPAC)	WLDIX
STI	Store integer (FPAC to memory)	WSTI
STIX	Store integer extended (FPAC to memory)	WSTIX

Table 9.2 C/350 floating-point computing instructions

*The WFFAD and WFLAD instructions use a 32-bit accumulator, while the equivalent C/350 instruction uses two memory words.

**The LFLST or LFSST instruction is a triple-word instruction; the C/350 instruction is a double-word instruction.

C/350 Instruction	C/350 Instruction Function	Equivalent Instruction
DSPA	Dispatch	LDSP
EJMP	Extended jump	XJMP
EJSR	Extended jump to subroutine	XJSR
ELEF	Extended load effective address	XLEF
JMP	Jump	—
JMP, 1	Jump, relative to the program counter	WBR
JSR	Jump to subroutine	—
LEF	Load effective address	—
POPB	Pop block and execute (return from XOP0)	WPOPB
POPJ	Pop PC and jump (return with PSHJ)	WPOPJ
PSHJ	Push jump (return with POPJ)	XPSHJ
PSHR	Push return address (pop with POPJ)	—
RSTR	Restore	WRSTR **
RTN	Return	WRTN *
SAVE	Save (used with JSR)	WSSVR, WSSVS *
SAVZ	Save without arguments (used with JSR)	WSSVR, WSSVS *
XOP0 ***	Extended operation (return with POPB)	WXOP ***

Table 9.3 C/350 program flow management instructions

*The WRTN, WSSVS, and WSSVR instructions modify the OVK fixed-point overflow mask and use a return block of six double words.

**The WRSTR instruction uses the wide stack and is equivalent to RSTR.

***The XOP0 and WXOP instructions are double-word instructions.

C/350 Instruction	C/350 Instruction Function	Equivalent Instruction
MSP	Modify stack pointer	WMSP
POP	Pop multiple accumulators	WPOP
POPB	Pop block and execute (return from XOPO)	WPOPB
POPJ	Pop PC and jump	WPOPJ
PSH	Push multiple accumulators	WPSH
PSHJ	Push jump	XPSHJ
PSHR	Push return address	—
RSTR	Restore	WRSTR **
RTN	Return	WRTN *
SAVE	Save (used with JSR)	WSSVR, WSSVS *
SAVZ	Save without arguments (used with JSR)	WSSVR, WSSVS *
XOPO ***	Extended operation (return with POPB)	WXOP ***

Table 9.4 C/350 stack management instructions

*The WRTN, WSSVS, and WSSVR instructions modify the OVK fixed-point overflow mask and use a return block of six double words.

**The WRSTR instruction uses the wide stack and is equivalent to RSTR.

***The XOP0 and WXOP instructions are double-word instructions.

Program Flow

The program counter governs program flow management as described in Chapter 5.

For any C/350 program executing on the MV/10000 computer, when the PC contains 77777_8 and increments to refer to the next instruction, the PC does not wrap around to 0. The PC increments to 100000_8, and the processor fetches the next instruction from this location. This will affect certain data movement instructions (for example, BAM, BLM, CMT, CMV, CTR and EDIT). If data movement is backward (descending addresses) and the process attempts a segment crossing, the address translator indicates a protection violation.

The C/350 program flow instructions load bits 17 through 31 of the PC with the address generated by the program flow instruction. Bits 1 through 3 remain unchanged; bits 4 through 16 are set to 0.

Appendix C illustrates the PC contents.

Fault Handling

The handling of faults is identical to the handling of MV/10000 system nonprivileged faults as described in *Principles of Operation, 32-Bit* ECLIPSE® *Systems.*

Note that all faults that occur with the execution of C/350 instructions use the narrow stack.

Appendix F lists the error codes returned to AC1 upon the occurrence of a decimal/ASCII fault and denotes the type of fault generated.

Reserved Memory

The MV/10000 computer does not implement C/350 auto-increment and auto-decrement locations 20_8 through 37_8. The processor reserves these locations for storage of certain system parameters.

CPU Identification

The ECLID and NCLID instructions return central processor information.

The NCLID instruction loads the CPU identification into bits 16 through 31 of three accumulators (AC0, AC1, and AC2). The NCLID instruction can execute only with the LEF mode disabled. With the LEF bit enabled, this instruction becomes a LEF instruction.

Appendix C lists the accumulator formats.

Appendix A
Instruction Summary

The table in this appendix lists the machine-specific instructions alphabetically by assembler-recognizable mnemonic and gives the format of the instruction, the data type used, the action performed, and location contents before and after instruction execution.

The C/350 compatible instructions are identified with an asterisk.

Principles of Operation, 32-Bit ECLIPSE® *Systems* presents a summary of the instructions used by all ECLIPSE MV/Family computers.

The following abbreviations are used in the table of instructions:

Abbreviation	**Meaning**
#	Integer
→	Returned to
+	Addition
=	Equality
OR	Logical OR
?	Unpredictable result
&	Ties together two (or more) items to be operated upon as one
ac	Fixed-point accumulator
acs	Source ac
acd	Destination ac
PSR	Processor status register
sp	Narrow stack pointer
fp	Narrow frame pointer
sl	Narrow stack limit
sa	Narrow stack fault address
E	Calculated effective address
(#)page zero	Address in page zero
x	Unknown and soon to be lost
displ.	Displacement
PC	Program counter
ION	Interrupt on flag

NOTE: *For all operations, unless otherwise indicated:*

Before instruction execution:	**Upon instruction completion:**
OVR=x	unchanged
CRY=x	unchanged
overflow=x	unchanged
FPSR bits=x	updated
BUSY, DONE flags=x	unchanged

Instruction Format	**Action**	**Before (Location =)**	**After (Location =)**
ECLID	CPU id→AC0	AC0=x	CPU id
*HALT NOTE: *HALT=DOC 0,CPU*	Stops the processor	ION flag=x	unchanged
*INTA *ac* NOTE: *INTA ac=DIB ac,CPU*	device code→ac	ac=x ION flag=x	device code unchanged
*INTDS NOTE: *INTDS=NIOC CPU*	0→ION flag	ION flag=x	0
*INTEN NOTE: *INTEN=NIOS CPU*	1→ION flag	ION flag=x	1
*IORST NOTE: *IORST=DICC 0,CPU*	Clear all I/O devices 0→priority mask	ION flag=x BUSY,DONE flags=x	0 0

*MSKO *ac*
NOTE: *MSKO ac=DOB ac,CPU*
	ac→priority mask	ac=#
		ION flag=x
		mask=x
			unchanged
			unchanged
			ac

NCLID	CPU id→AC0&AC1&AC2	AC0=x
		AC1=x
		AC2=x
			model number
			microcode rev
			memory size

*READS *ac* NOTE: *READS ac=DIA ac,CPU*	console switches→ac	ac=x ION flag=x	result unchanged
*SKP *t device*	If t=true =skip	BUSY,DONE flags=x	unchanged

WLMP	(E)→map slots	AC0=#(1st slot #)
		AC1=#(# slots)
		AC2=E
			last
			0
			last E+2

Appendix B
Instruction Execution Times

The table in this appendix lists the average execution times of the instructions supported by the ECLIPSE MV/10000 computer. The table shows execution times in microseconds.

This discussion of execution times assumes the following:

- Physical memory modules consist of 1- or 2 Mbytes of memory.
- All logical-to-physical address translations are resident in the address translator, the system cache, and the instruction cache.
- The EDIT and WEDIT subopcodes that process commercial numeric data are processing a data type of 4, and the source pointer into the data (j) is never moved out of the bounds of the data.

If these conditions do not apply, the execution times must be adjusted as follows:

To Every Memory Reference	Add (microseconds)
If logical-to-physical address translation is not in the address translation unit (ATU) cache	
For one-level page table	0.56
For two-level page table	0.84
If indirection is specified by the instruction	0.28 per level of indirection
If a double-word reference address ends in 7_8	0.28
If data is not in system cache	0.28

Any Instruction	Add (microseconds)
If any of the following faults occur	
Stack overflow/underflow	2.8
Fixed-point fault	4.2 + 2.8 if stack fault
Protection fault	Min. 4.2, Avg. 4.76, Max. 6.3 } +2.8 @ indirect, +2.8 if stack fault
If instruction is not in instruction cache	0.28

The C/350 compatible instructions listed in the table are identified with an asterisk. Any instruction capable of specifying indirection is identified with a tilde (~).

Mnemonic	Timing (microseconds)	Mnemonic	Timing (microseconds)
ADC *	0.14 + 0.28 if skip	DHXR *	0.7
ADD *	0.14 + 0.28 if skip	DIV *	2.45
ADDI *	0.14	DIVS *	3.22
ADI *	0.14	DIVX *	3.08
ANC *	0.14 + 0.28 if skip	DLSH *	0.98 if count=0 1.4 (min) 1.82 (max)
AND *	0.14 + 0.28 if skip	DSB *	0.77
ANDI *	0.14	DSPA *	1.54 + 0.28(each level of indirect addressing) + 0.14 if entry = -1
BAM *	1.40 + 0.28(number of words moved) + 0.28(each level of indirect addressing)	DSZ *	0.42 + 0.28 if skip
BKPT	1.12 + 0.28(each level of indirect addressing)	DSZTS	0.56 + 0.28 if skip
BLM *	1.40 + 0.28(number of words moved) + 0.28(each level of indirect addressing)	ECLID	0.84
BTO *	0.56 + 0.42 (each level of indirect addressing)	EDIT *	1.3 + sum of sub-op execution times that are processed
BTZ *	0.56 + 0.42 (each level of indirect addressing)		DADI 1.1
CLM *	0.63 (1.05 if acs=acd)		DAPS 0.8 (w/o add) 1.1 (with add)
CMP *	3.9 + 0.5/byte (min) 5.2 + 0.9/byte (max)		DAPT 0.8 (w/o add) 1.1 (with add)
CMT *	1.8 + 0.8/byte + 0.6 if no delimiter		DAPU 1.1
CMV *	2.9 + 0.1/byte (min) 2.8 + 0.4/byte (max)		DASI 1.4 (type 4) 2.0 (type 5)
COB *	0.56 (min) 1.02 (max)		DDTK 2.1 + 0.3 if k is in narrow stack
COM *	0.14 + 0.28 if skip		DEND 1.4
CRYTC	0.14		DICI 0.4 + 0.4 per char. insert
CRYTO	0.14		DIMC 1.7 + 0.3 per char. insert + 0.3 if parameter j is located in the narrow stack
CRYTZ	0.14		DINC 1.0
CTR *	Translate and move 0.8 + 0.7/byte Translate and compare 1.1 + 0.7/byte		DINS 1.1
CVWN	0.42		DINT 1.1
DAD *	1.77		DMVA 1.0 + 0.7 per char. moved + 0.3 if parameter j is located in the narrow stack
DEQUE	Queue not empty: 1.68 + 0.14 if AC1=-1 + 0.42 if final element Queue empty: 0.7		DMVC 1.4 + 0.42 per char. moved + 0.3 if parameter j is located in either stack
DERR	0.98		
DHXL *	0.84		

Mnemonic	Timing (microseconds)	Mnemonic	Timing (microseconds)
EDIT* (continued)	DMVF 1.5 + 1.7 per digit moved + 0.3 if parameter j is located in either stack	FLDD *	0.28 ~
	DMVN 1.8 + 1.3 per digit moved + 0.3 if parameter j is located in either stack	FLDS *	0.14 ~
	DMVO 2.5		
	DMVS 1.5 + 1.4 per digit moved + 0.3 if parameter j is located in either stack	FLMD *	0.7 ~
	DNDF 1.1	FLST *	0.56
	DSSO 1.0	FMD *	1.4
	DSSZ 1.0	FMMD *	1.68 ~
	DSTK 1.4 + 0.2 if k is in the narrow stack	FMMS *	0.98 ~
	DSTO 1.0	FMOV *	0.14
	DSTZ 1.0	FMS *	0.84
EDSZ *	0.42 + 0.28 if skip ~	FNEG *	0.28
EISZ *	0.42 + 0.28 if skip ~	FNOM *	0.28
EJMP *	0.42 ~	FNS *	0.14
EJSR *	0.42 ~	FPOP *	2.66
ELDA *	0.14 ~	FPSH *	2.1
ELDB *	0.28 + 0.42 if PC-relative	FRDS *	0.7
ELEF *	0.28 ~	FRH *	0.28
ENQH	1.54 + 0.42 if queue not empty	FSA *	0.42
ENQT	1.54 + 0.42 if queue not empty	FSCAL*	1.26
ESTA *	0.14	FSD *	0.42
ESTB *	0.28 + 0.42 if PC-relative	FSEQ *	0.28 + 0.28 if skip
FAB *	0.14	FSGE *	0.28 + 0.28 if skip
FAD *	0.42	FSGT *	0.28 + 0.28 if skip
FAMD *	0.7	FSLE *	0.28 + 0.28 if skip
FAMS *	0.56	FSLT *	0.28 + 0.28 if skip
FAS *	0.42	FSMD *	0.7 ~
FCLE *	0.42	FSMS *	0.56 ~
FCMP *	0.42	FSND *	0.28 + 0.28 if skip
FDD *	4.48 (FPSR bit 8=0) 5.04 (FPSR bit 8=1)	FSNE *	0.28 + 0.28 if skip
FDMD *	4.76 (FPSR bit 8=0) 5.32 (FPSR bit 8=1)	FSNER*	0.28 + 0.28 if skip
FDMS *	2.38 (FPSR bit 8=0) 2.94 (FPSR bit 8=1)	FSNM *	0.28 + 0.28 if skip
FDS *	2.24 (FPSR bit 8=0) 2.80 (FPSR bit 8=1)	FSNO *	0.28 + 0.28 if skip
FEXP *	0.28	FSNOD*	0.28 + 0.28 if skip
FFAS *	0.84	FSNU *	0.28 + 0.28 if skip
FFMD *	1.26 ~	FSNUD*	0.28 + 0.28 if skip
FHLV *	0.56	FSNUO*	0.28 + 0.28 if skip
FINT *	0.7	FSS *	0.42
FLAS *	0.56		

Mnemonic	Timing (microseconds)	Mnemonic	Timing (microseconds)
FSST *	0.56 ~	LFSMD	0.7 ~
FSTD *	0.28 ~	LFSMS	0.56 ~
FSTS *	0.14 ~	LFSST	0.42 ~
FTD *	0.28	LFSTD	0.28 ~
FTE *	0.42	LFSTS	0.14 ~
FXTD	0.42	LJMP	0.42 ~
FXTE	0.42	LJSR	0.42 ~
HLV *	0.28 + 0.21 if ac is negative	LLDB	0.28 + 0.42 if PC-relative
HXL *	0.28	LLEF	0.28 ~
HXR *	0.28	LLEFB	0.14 + 0.42 if PC-relative
INC *	0.14 + 0.28 if skip	LMRF	1.33
IOR *	0.14	LNADD	0.14 ~
IORI *	0.14	LNADI	0.14
ISZ *	0.42 + 0.28 if skip ~	LNDIV	2.52 ~
ISZTS	0.56 + 0.28 if skip ~	LNDO	0.84 (no termination) 1.40 (for termination)
JMP *	0.42 ~	LNDSZ	0.42 + 0.28 if skip
JSR *	0.42 ~	LNISZ	0.42 + 0.28 if skip
LCALL	Intra-ring (same ring) 0.84 + 0.28 per indirect ~ Inter-ring (cross ring) 3.64 + 0.28 per indirect + 0.42 per argument	LNLDA	0.14 ~
LCPID	0.84	LNMUL	1.82 ~
LDA *	0.14 ~	LNSBI	0.14
LDAFP	0.14	LNSTA	0.14 ~
LDASB	0.14	LNSUB	0.14 ~
LDASL	0.14	LOB *	0.42 + [0.14 * (number leading zero nibbles)]
LDASP	0.28	LPEF	0.42 ~
LDATS	0.28	LPEFB	0.42 + 0.42 if PC-relative
LDB *	0.28	LPHY	0.98 (1 level valid) 1.26 (2 level valid) 0.98 (invalid)
LDI *	11.1 (Type 4, length 7)	LPSHJ	0.42 ~
LDIX *	40.9 (Type 4, length 31)	LPSR	0.42
LDSP	1.96 ~	LRB *	0.84 + (0.14 if acs<>acd) + [0.14 * (number leading zero nibbles)]
LEF *	0.28 ~	LSBRA	3.36
LFAMD	0.7 ~	LSBRS	3.08
LFAMS	0.56 ~	LSH *	0.42 if count = 0 0.84 (min) 1.26 (max)
LFDMD	4.76 (FPSR bit 8=0) 5.32 (FPSR bit 8=1)	LSN *	1.7 + 1.3/leading zero digit
LFDMS	2.38 (FPSR bit 8=0) 2.94 (FPSR bit 8=1)	LSTB	0.28 + 0.42 if PC-relative
LFLDD	0.28 ~	LWADD	0.14 ~
LFLDS	0.14 ~	LWADI	0.14 ~
LFLST	0.56 ~	LWDIV	4.06 ~
LFMMD	1.68 ~	LWDO	0.84 (no termination) 1.40 (for termination)
LFMMS	0.98 ~	LWDSZ	0.42 + 0.28 if skip ~

Mnemonic	Timing (microseconds)	Mnemonic	Timing (microseconds)
LWISZ	0.42 + 0.28 if skip ~	SAVZ	1.68
LWLDA	0.14 ~	SBI *	0.14
LWMUL	2.38 ~	SEX	0.14
LWSBI	0.14 ~	SGE *	0.14 + 0.28 if skip
LWSTA	0.14 ~	SGT *	0.14 + 0.28 if skip
LWSUB	0.14 ~	SMRF	1.12
MOV *	0.14 + 0.28 if skip	SNB *	0.56 + (0.42*indirects) + 0.28 if skip
MSP *	0.70	SNOVR	0.42 + 0.28 if skip
MUL *	1.40	SPSR	0.14
MULS *	1.61	STA *	0.14 ~
NADD	0.14	STAFP	0.14
NADDI	0.14	STASB	0.28
NADI	0.14	STASL	0.28
NBStc	1.86 + 0.42(n-1) + 0.56 if end is encountered	STASP	0.14
NCLID	0.98	STATS	0.28
NDIV	2.38	STB *	0.28
NEG *	0.14 + 0.28 if skip	STI *	13.9 (Type 4, length 7)
NFStc	1.86 + 0.42(n-1) +0.56 if end is encountered	STIX *	56.1 (Type 4, length 31)
NLDAI	0.14	SUB *	0.14 + 0.28 if skip
NMUL	1.68	SZB *	0.42 + 0.28 if skip
NNEG	0.14	SZBO *	0.7 + 0.42 (each level of indirection) + 0.28 if skip
NSALA	0.14 + 0.28 if skip	VWP	0.42 + 0.42 (each level of indirection)
NSALM	0.42 + 0.28 if skip	VBP	0.28
NSANA	0.14 + 0.28 if skip	WADC	0.14
NSANM	0.42 + 0.28 if skip	WADD	0.14
NSBI	0.14	WADDI	0.14
NSUB	0.14	WADI	0.14
ORFB	1.26 + 0.98(count), count = AC0 + 1	WANC	0.14
PATU	0.84	WAND	0.14
PBX	2.94 + executed instruction	WANDI	0.14
POP *	0.7 + 0.14 per ac	WASH	0.42 if count = 0, [0.84 (min), 1.26 (max)] + 0.28 if left shift
POPB *	1.54	WASHI	0.28 if count = 0, [0.84 (min), 1.26 (max)] + 0.28 if left shift
POPJ *	0.84	WBLM	1.4 + 0.28(number of words moved) + 0.28(each level of indirect addressing)
PSH *	0.7 + 0.14 per ac	WBR	0.42
PSHJ *	0.7	WBStc	1.72 + 0.42(n-1) + 0.56 if end is encountered
PSHR *	0.7	WBTO	0.56 + 0.42 (each level of indirection)
RRFB	1.68 + 0.28(count) count = AC0 +1	WBTZ	0.56 + 0.42 (each level of indirection)
RSTR *	1.82	WCLM	if acs<>acd 0.56 if no skip 0.70 if skip if acs=acd 0.84 if no skip 0.98 if skip
RTN *	1.96	WCMP	3.9 + 0.5/byte + 0.7 for each descending string (min) 5.2 + 0.9/byte + 0.7 for each descending string (max)
SAVE *	1.68	WCMT	1.8 + 0.8/byte + 0.6 if no delimiter + 1.4 for each descending string (min)

Mnemonic	Timing (microseconds)	Mnemonic	Timing (microseconds)
WCMV	2.9 + 0.1/byte + 0.7 for each descending string (min) 2.8 + 0.4/byte + 0.7 for each descending string (max)		DMVS 1.5 + 1.4 per digit moved + 0.3 if parameter j is located in either stack
WCOB	0.42 (min) 1.54 (max)		DNDF 1.1
WCOM	0.14		DSSO 1.0
WCST	1.3 + 0.4/byte + 0.7 for each descending string		DSSZ 1.0
WCTR	Translate & move 0.8 + 0.7/byte Translate & compare 1.1 + 0.7/byte		DSTK 1.4 + 0.2 if k is in narrow stack
WDIV	3.92		DSTO 1.0
WDIVS	4.76		DSTZ 1.0
WDPOP	2.38 restartable 2.80 restartable XCT 9.80 resumable	WFFAD	0.84
WEDIT	1.3 + sum of sub-op execution times that are processed	WFLAD	0.56
	DADI 1.1	WFPOP	2.1
	DAPS 0.8 (w/o add) 1.1 (with add)	WFPSH	1.82
	DAPT 0.8 (w/o add) 1.1 (with add)	WFStc	1.72 + 0.42(n-1) + 0.56 if end is encountered
	DAPU 1.1	WHLV	0.6
	DASI 1.4 (type 4) 2.0 (type 5)	WINC	0.14
	DDTK 2.1 + 0.3 if k is in narrow stack	WIOR	0.14
	DEND 1.4	WIORI	0.14
	DICI 0.4 + 0.3 per char. insert	WLDAI	0.14
	DIMC 1.7 + 0.3 per char. insert + 0.3 if parameter j is located in the wide stack	WLDB	0.28
	DINC 1.0	WLDI	11.1 (Type 4, length 7)
	DINS 1.1	WLDIX	40.9 (Type 4, length 31)
	DINT 1.1	WLDO	0.84 (no termination) 1.40 (for termination)
	DMVA 1.0 + 0.7 per char. moved + 0.3 if parameter j is located in either stack	WLMP	0.42 + 3.64(Number of BMC/DCH slots) (min)
	DMVC 1.4 + 0.42 per char. moved + 0.3 if parameter j is located in either stack	WLOB	0.28 + [0.14 * (number leading zero nibbles)]
	DMVF 1.5 + 1.7 per digit moved + 0.3 if parameter j is located in either stack	WLRB	0.56 + (0.14 if acs<>acd) + [0.14 * (number leading zero nibbles)]
	DMVN 1.8 + 1.3 per digit moved + 0.3 if parameter j is located in either stack	WLSH	0.28 if count = 0, 0.70 (min) 1.12 (max)
	DMVO 2.5	WLSHI	0.28 if count = 0, 0.70 (min) 1.12 (max)

Mnemonic	Timing (microseconds)	Mnemonic	Timing (microseconds)
WLSI	0.42 (0.28 if shift = 0)	WSNB	0.56 + (0.42 * indirections) + 0.28 if skip
WLSN	1.7 + 1.3/leading zero digit	WSNE	0.14 + 0.28 if skip + 0.14 if compare to 0
WMESS	0.84 + 0.42 if successful	WSNEI	0.14 + 0.28 if skip
WMOV	0.14	WSSVR	1.12
WMOVR	0.14	WSSVS	1.12
WMSP	0.56	WSTB	0.28
WMUL	2.24	WSTI	13.9 (Type 4, length 7)
WMULS	2.31	WSTIX	56.1 (Type 4, length 31)
WNADI	0.14	WSUB	0.14
WNDO	0.84 (no termination) 1.40 (for termination)	WSZB	0.56 + (0.42*indirects) + 0.28 if skip
WNEG	0.14	WSZBO	0.7 + (0.42*indirects) + 0.28 if skip
WPOP	0.28 + 0.14 per ac	WUGTI	0.28 + 0.28 if skip
WPOPB	1.54 intra ring 2.80 cross ring	WULEI	0.28 + 0.28 if skip
WPOPJ	0.84	WUSGE	0.14 + 0.28 if skip + 0.14 if compare to 0
WPSH	0.28 + 0.14 per ac	WUSGT	0.14 + 0.28 if skip + 0.14 if compare to 0
WRSTR	2.8 intra ring 4.06 cross ring	WXCH	0.28
WRTN	1.68 intra ring 2.94 cross ring	WXOP	2.1 + 0.28(indirect)
WSALA	0.14 + 0.28 if skip	WXOR	0.14
WSALM	0.42 + 0.28 if skip	WXORI	0.14
WSANA	0.14 + 0.28 if skip	XCALL	Intra-ring (same ring) ~ 0.84 + 0.28 per indirect Inter-ring (cross ring) 3.64 + 0.28 per indirect + 0.42 per argument
WSANM	0.42 + 0.28 if skip	XCH *	0.28
WSAVR	0.98	XCT *	1.26 + executed instruction
WSAVS	0.98	XFAMD	0.7 ~
WSBI	0.14	XFAMS	0.56 ~
WSEQ	0.14 + 0.28 if skip + 0.14 if compare to 0	XFDMD	4.76 (FPSR bit 8=0) ~ 5.32 (FPSR bit 8=1) ~
WSEQI	0.14 + 0.28 if skip	XFDMS	2.38 (FPSR bit 8=0) ~ 2.94 (FPSR bit 8=1) ~
WSGE	0.14 + 0.28 if skip + 0.14 if compare to 0	XFLDD	0.28 ~
WSGT	0.14 + 0.28 if skip + 0.14 if compare to 0	XFLDS	0.14 ~
WSGTI	0.28 + 0.28 if skip	XFMMD	1.68 ~
WSKBO	0.28 + 0.28 if skip	XFMMS	0.98 ~
WSKBZ	0.28 + 0.28 if skip	XFSMD	0.7 ~
WSLE	0.14 + 0.28 if skip + 0.14 if compare to 0	XFSMS	0.56 ~
WSLEI	0.28 + 0.28 if skip	XFSTD	0.28 ~
WSLT	0.14 + 0.28 if skip + 0.14 if compare to 0	XFSTS	0.14 ~

Mnemonic	Timing (microseconds)	Mnemonic	Timing (microseconds)
XJMP	0.42 ~	XPEF	0.42 ~
XJSR	0.42 ~	XPEFB	0.56 + 0.14 if PC-relative
XLDB	0.28 ~ + 0.42 if PC-relative	XPSHJ	0.42 ~
XLEF	0.28 ~	XSTB	0.28 + 0.42 if PC-relative
XLEFB	0.42 + 0.28 if PC-relative	XVCT	11.2 base level, inline 7.98 intermediate level, inline
XNADD	0.14 ~		10.5 base level, interrupt
XNADI	0.14 ~		8.82 intermediate level, interrupt (minimums)
XNDIV	2.52 ~	XWADD	0.14 ~
XNDO	0.84 (no termination) 1.40 (for termination)	XWADI	0.14 ~
XNDSZ	0.42 + 0.28 if skip ~	XWDIV	4.06 ~
XNISZ	0.42 + 0.28 if skip ~	XWDO	0.84 (no termination) 1.40 (for termination)
XNLDA	0.14 ~	XWDSZ	0.42 + 0.28 if skip ~
XNMUL	1.82 ~	XWISZ	0.42 + 0.28 if skip ~
XNSBI	0.14 ~	XWLDA	0.14 ~
XNSTA	0.14 ~	XWMUL	2.38 ~
XNSUB	0.14 ~	XWSBI	0.14 ~
XOP0 *	2.1 + 0.28(indirect)	XWSTA	0.14 ~
XOR *	0.14	XWSUB	0.14 ~
XORI *	0.14	ZEX	0.14

Appendix C
Register Fields

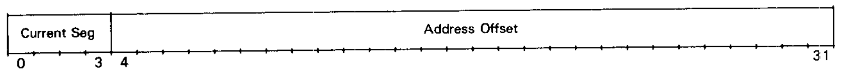
This appendix contains the formats for the programmer-accessible registers available on the MV/10000 computer for both MV/10000-system-specific and C/350 compatible formats.

Register	Purpose
Program Counter	Contains the logical address of the currently executing instruction
Processor Status Register	Contains information pertaining to fixed-point computations
Floating-Point Status Register	Contains information pertaining to floating-point computations
Segment Base Registers	Contain information pertaining to MV/10000 logical address translation
DCH/BMC Status Registers	Contain information pertaining to data channel and burst multiplexor channel maps
CPU Identification	Accumulators contain information pertaining to the CPU

Program Counter

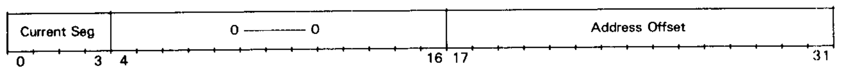
The 31-bit program counter (PC) contains the logical address of the currently executing instruction. The PC formats follow.

PC Format for Execution of MV/10000-System-Specific Programs



Bits	Name	Contents or Function
1-3	Current Segment	The current segment of program execution.
4-31	Address Offset	The 28-bit address of the currently executing instruction.

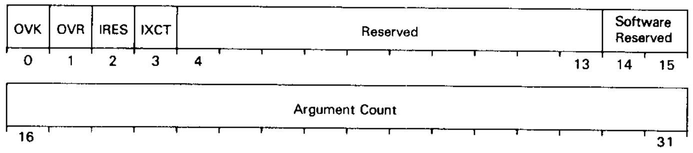
PC Format Altered by C/350 Program Flow Instructions



Bits	Name	Contents or Function
1-3	Current Segment	The current segment of program execution.
4-16	0-0	Set to 0 by instruction.
17-31	Address Offset	The 15-bit address formed by the program flow instruction.

Processor Status Register

Only MV/10000-system-specific instructions affect the 32-bit PSR. The format of the PSR follows:



Bits	Name	Contents or Function
0	OVK	Overflow Mask 0 indicates no fixed-point overflow trap. 1 indicates trap on OVR set to 1.
1	OVR	Fixed-point overflow indicator; set to 1 when calculating a two's complement number that does not fit in the specified location or register, or when attempting to divide by 0. If OVK equals 1, then the setting of OVR to 1 results in a fixed-point overflow fault
2	IRES	Micro-interrupt resume flag; set to 1 when the processor receives an I/O interrupt request while executing a resumable instruction such as WEDIT.
3	IXCT	Interrupt execute flag; set to 1 when the processor receives an I/O interrupt request while executing an instruction that was inserted into the instruction stream — for example, a PBX instruction.
4-13	Reserved	Bits 4 through 13 are reserved for future use.
14-15	Software Reserved	Bits 14 through 15 are software reserved in return block.
16-31	Argument Count	Bits 16 through 31 contain the number of arguments to be passed with LCALL or XCALL.

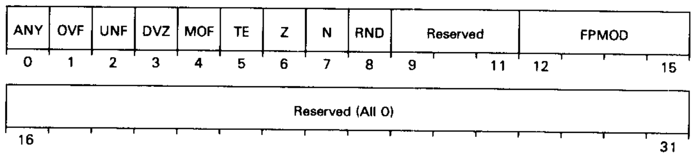
NOTE: *Any instruction that loads the OVK and OVR bits as part of its execution will not cause an overflow fault even if both are set to 1.*

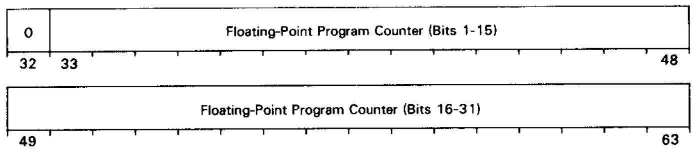
For all C/350 instructions, overflow equals 0, leaving OVR unchanged.

Floating-Point Status Register

MV/10000-system-specific and C/350 instructions affect the 64-bit floating-point status register (FPSR). The FPSR format follows.

NOTE: *When the C/350 FLST and FSST instructions write to or read from the FPSR, the instructions ignore bits 16 through 48.*

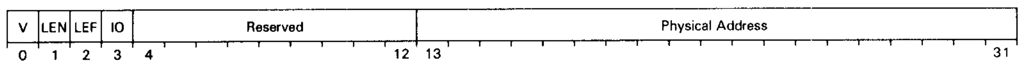



0
Floating-Point Program Counter (Bits 1-15)
32
33
48
Floating-Point Program Counter (Bits 16-31)
49
63

Bits	Name	Contents or Function
0	ANY	Indicates the setting to 1 of any of bits 1 through 4.
1	OVF	Exponent overflow indicator.
2	UNF	Exponent underflow indicator.
3	DVZ	Divide by 0.
4	MOF	Mantissa overflow.
5	TE	Trap enable; if set to 1, setting of any of bits 1 through 4 will result in a floating-point fault.
6	Z	Zero bit.
7	N	Negative bit.
8	RND	Floating-point rounding mode.
9-11	Reserved	Bits 9 through 11 are reserved for future use and must be set to 0.
12-15	FPMOD	Floating-point model; should be set to 0111.
16-31	Reserved	Bits 16 through 31 are reserved for future use; these should be set to 0.
32	0	Should be set to 0.
33-63	Floating-Point Program Counter	Floating-point program counter. In the event of a floating-point fault, this is the address of the first floating-point instruction that caused the fault.

Segment Base Registers

The 32-bit segment base registers (SBR) contain information for the MV/10000-system-specific logical address translation mechanism and for I/O protection. The format follows:



Bits	Name	Contents or Function
0	V	Segment validity bit — indicates the process' ability to refer to a segment. 0 indicates an invalid SBR. 1 indicates a valid SBR.
1	LEN	Length bit — indicates the maximum range of the logical memory address. 0 indicates a one-level page table. 1 indicates a two-level page table.
2	LEF	LEF enable — indicates whether the processor will operate in LEF or I/O mode. 0 indicates I/O mode. 1 indicates LEF mode.
3	IO	I/O enable — indicates if an I/O protection violation will occur upon an execution of an I/O instruction. 0 indicates protection violation will occur. 1 indicates the I/O instruction will execute.
4-12	Reserved	Bits 4 through 12 are reserved for future use.
13-31	Physical	Identifies the physical page address in memory of the indicated page table.

DCH/BMC Status Registers

The three registers described in this section include the I/O channel definition register, the I/O channel status register, and the I/O channel mask register.

I/O Channel Definition Register Format

The I/O channel definition register (6000_8) provides status information. The format for the register follows:

E	Reserved	BV	DV	Res	BX	A	P	DIS	I/O Channel	M	0
0	1 2	3	4	5	6	7	8	9	10 13	14	15

Bits	Name	Contents or Function
0	E	Error flag; if 1, an error has occurred on the I/O channel (0 only when all other error bits are 0).
1,2	Reserved	Bits 1 and 2 are reserved for future use and are returned as zero.
3	BV	BMC validity error flag; if 1, BMC validity protect error has occurred.
4	DV	DCH validity error flag; if 1, DCH validity protect error has occurred.
5	Res	Bit 5 is reserved for future use and is returned as zero.
6	BX	BMC transfer flag; if 1, BMC transfer is in progress (read only bit).
7	A	BMC address error; if 1, the channel has detected an address parity error.
8	P	BMC data error; if 1, the channel has detected a data parity error.
9	DIS	Disable block transfer; if 1, disables BMC block transfers to/from I/O memory port (read/write bit).
10-13	I/O channel	I/O channel number.
14	M	DCH mode; if 1, DCH mapping is enabled.
15	0	Always set to 0.

NOTES: *Writing to bits 3, 4, 7, or 8 with a 1 complements these bits.*

The C/350 IORST instruction clears bits 0, 3, 4, 7, 8, 9, and 14.

I/O Channel Status Register

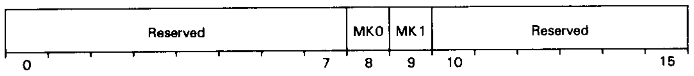
The read-only I/O channel status register (7700_8) provides I/O channel status information. The format for the register follows:

ERR	Reserved	XDCH	1	MSK	INT
0	1 11	12	13	14	15

Bits	Name	Contents or Function
0	ERR	If 1, the I/O channel has detected an error indicated by the IOC status register or a memory parity error.
1-11	Reserved	Bits 1 through 11 are reserved for future use.
12	XDCH	If 1, extended DCH map slots and operations are supported.
13		Always set to 1.
14	MSK	If 1, MSK prevents all devices connected to the channel from interrupting the CPU. However, the INTA instruction will return the device code of any device with its DONE flag set.
15	INT	Interrupt pending; if 1, the channel is attempting to interrupt the CPU.

I/O Channel Mask Register Format

The write-only I/O channel mask register (7701_8) specifies a mask flag for each channel. The format for the register follows:



Bits	Name	Contents or Function
0-7	Reserved	Bits 0 through 7 are reserved for future use.
8	MK0	If 1, MK0 prevents all devices connected to channel 0 from interrupting the CPU. A system reset sets MK1 to a 0.
9	MK1	If 1, MK1 prevents all devices connected to channel 1 from interrupting the CPU. A system reset sets MK1 to a 1.
10-15	Reserved	Bits 10 through 15 are reserved for future use.

NOTES: *A PRTRST instruction zeroes the mask bit for one or for both channels.*

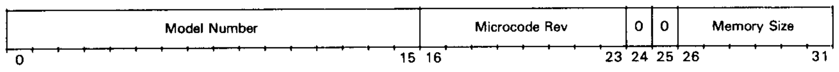
A CIO read to the I/O channel mask register produces undefined results.

CPU Identification

The three Load CPU Identification instructions return the information shown below to the specified accumulators.

LCPID and ECLID Instructions

The LCPID and ECLID instructions load a double word into AC0.

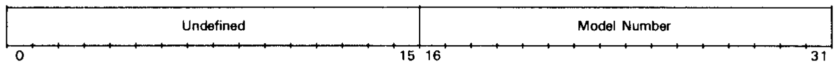


Bits	Name	Contents or Function
0-15	Model Number	The binary value of the model number allocated to the processor (10001001001100_2).
16-23	Microcode Rev	Current microcode revision.
24,25	0	Set to 0.
26-31	Memory Size	Amount of physical memory available: (in increments of 256 Kbytes) A 0 indicates 256 Kbytes. A 1 indicates 512 Kbytes to a maximum of 63, indicating 16 Mbytes.

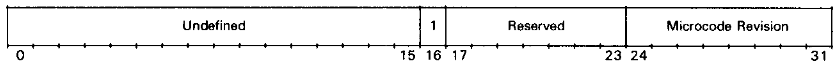
NCLID Instruction

The NCLID instruction loads the result into the low-order 16 bits of the three accumulators.

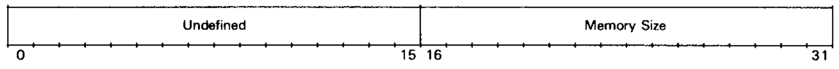
Returned in AC0:



Returned in AC1:



Returned in AC2:



AC#	Name	Contents or Function
0	Model Number	Binary representation of the machine's model number (10001001001100_2).
1	Microcode Revision	Current microcode revision **Bits** **Meaning** 16 Always set to 1 17-23 Reserved for future use 24-31 Current microcode revision If AC1 contains 177777_8, you should load the microcode.
2	Memory Size	Amount of physical memory available: (in increments of 32 Kbytes) A 0 indicates 32 Kbytes. A 1 indicates 64 Kbytes; etc.

Appendix D
Reserved Memory Locations and Context Block Formats

This appendix describes the reserved memory locations (see Tables D.1 and D.2) and the context block formats (see Table D.3).

Reserved Memory Locations

The processor reserves memory locations 0 through 47_8 of page zero (locations 0 through 377_8) of each segment for storage of certain parameters and fault handler addresses. The processor translates these locations as shown in Tables D.1 and D.2.

Some of the pointers are 16 bits long. As a result, they can only refer to locations in the first 64 Kbytes of the segment containing the pointer. If the pointer is indirect, all pointers in the indirect chain can also only refer to the first 64 Kbytes of the segment.

Page Zero Locations for Segment 0

When an MV/10000-system-specific interrupt occurs, segment 0 locations 0 through 47_8 have the meanings listed in Table D.1.

With the MV/10000 address translator enabled, the processor interprets all locations as logical.

Word	Name	Contents or Function
0	Interrupt Level	Level of interrupt processing; 0 indicates base-level processing; non-zero indicates intermediate-level processing.
1	I/O Handler	Address of the I/O interrupt handler; indirectable.
2-3	I/O Return Address	Address of the I/O interrupt return. (Word 2 contains the high order; word 3 contains the low order.)
4	Vector Stack Pointer	Low-order 16 bits of vector stack pointer, base, and frame pointer; high-order bits are zeroes.
5	Current C/350 Mask	Current C/350 interrupt priority mask.
6	Vector Stack Limit	Low-order 16 bits of vector stack limit.
7	Vector Stack Fault Address	Address of the vector stack fault handler; indirectable.
10-11	Breakpoint Address	Address of the breakpoint handler; indirectable.
12-13	WXOP Origin Address	Address of the beginning of the extended operations table; indirectable.
14	MV/10000 Stack Fault Address	Address of the MV/10000 stack fault handler; indirectable.
15-17	Reserved	Reserved.
20-21	WFP	MV/10000 frame pointer; nonindirectable.
22-23	WSP	MV/10000 stack pointer; nonindirectable.
24-25	WSL	MV/10000 stack limit; nonindirectable.
26-27	WSB	MV/10000 stack base; nonindirectable.
30-31	MV/10000 Page Fault Handler	Address of the MV/10000 page fault handler; indirectable.
32-33	Context Block Pointer	Address of the base of context block save area; indirectable.
34-35	WGP	Gate pointer; address of the gate array; nonindirectable.
36	Protection Fault Handler Address	Address of the protection fault handler; indirectable.
37	Fixed-Point Fault Handler Address	Address of the fixed-point fault handler; indirectable.
40	Stack Pointer	Address of the top of the C/350 stack; nonindirectable.
41	Frame Pointer	Address of the start of the current C/350 frame minus 1; nonindirectable.
42	Stack Limit	Address of the last normally usable location in the C/350 stack.
43	C/350 Stack Fault Address	Address of the C/350 stack fault handler; indirectable.
44	XOPO Origin Address	Address of the beginning of the C/350 extended operations table.
45	Floating-Point Fault Address	Address of the floating-point fault handler; indirectable.
46	Decimal/ASCII Fault Handler	Address of the Decimal/ASCII fault handler; indirectable.
47	DERR Error Handler	Address of the DERR error/trap handler; nonindirectable.

Table D.1 Page zero locations for segment 0

Page Zero Locations for Segments 1 through 7

Table D.2 shows the page zero locations for segments 1 through 7 with the address translator enabled.

Word	Name	Contents or Function
0-7	Reserved	Reserved.
10-11	MV/10000 Breakpoint Address	Address of the MV/10000 breakpoint handler; indirectable.
12-13	WXOP Origin Address	Address of the beginning of the extended operations table; indirectable.
14	MV/10000 Stack Fault Address	Address of the MV/10000 stack fault handler; indirectable.
15-17	Reserved	Reserved.
20-21	WFP	MV/10000 frame pointer; nonindirectable.
22-23	WSP	MV/10000 stack pointer; nonindirectable.
24-25	WSL	MV/10000 stack limit; nonindirectable.
26-27	WSB	MV/10000 stack base; nonindirectable.
30-33	Reserved	Reserved.
34-35	WGP	Gate pointer; address of the gate array; nonindirectable.
36	Reserved	Reserved.
37	Fixed-Point Fault Handler Address	Address of the fixed-point fault handler; indirectable.
40	Stack Pointer	Address of the top of the C/350 stack; nonindirectable.
41	Frame Pointer	Address of the start of the current C/350 frame minus 1; nonindirectable.
42	Stack Limit	Address of the last normally usable location in the C/350 stack.
43	C/350 Stack Fault Address	Address of the C/350 stack fault handler; indirectable.
44	XOPO Origin Address	Address of the beginning of the C/350 extended operations table.
45	Floating-Point Fault Address	Address of the floating-point fault handler; indirectable.
46	Decimal/ASCII Fault Handler	Address of the Decimal/ASCII fault handler; indirectable.
47	DERR Error Handler	Address of the DERR error/trap handler; nonindirectable.

Table D.2 Page zero locations for segments 1 through 7

Context Block Formats

The context block can be Type 1, which uses words 0 through 27, or Type 2, which uses words 0 through 105. Table D.3 shows the format of the context blocks.

Words in Block	Contents
0-1	PSR; argument count is zero.
2-3	AC0
4-5	AC1
6-7	AC2
8-9	AC3
10-11	CARRY, PC of offending (that is, executing) instruction.
12-13	Next PC. Double word containing the segment number in bits 1-3 of the next instruction to execute. The processor uses it to resolve on a microcycle basis the segment in which the instruction is actually executing. Because most instructions cannot cross segment boundaries, this double word reflects the same segment as the program counter of the executing instruction.
14-15	LAR; address that caused the page fault.
16-17	PBXED/IXCT_OPCODE.
18-19	CB state (see below).
20-21	IP state (see below).
22-23	ATU state (see below).
24-25	ALU state (see below).
26-27	MSEQ state (see below).
28-29	PDR
30-31	TREG
32-33	GR0
34-35	GR1
36-37	GR2
38-39	GR3
40-41	GR4
42-43	GR5
44-45	GR6
46-47	GR7
48-49	AG0
50-51	AG1
52-53	AG2
54-55	AG3
56-57	AR0
58-59	AR1
60-61	AR2
62-63	AR3
64-65	AR4
66-67	AR5
68-69	AR6
70-71	AR7
72-73	FG0H (FP GR <0-31>)
74-75	FG0L (FP GR <32-63>)
76-77	MICROSTACK0
:	:
:	: Variable size (0 to 28 words)
:	: (see MSEQ state below)
104-105	MICROSTACK13

Table D.3 Context block format

CB State Format

Bits	Description
12-15	Number of microstack entries pushed
30-31	Type of context block: 0 or 2 = Type 1, 1 = Type 2

IP State Format

Bits	Description
28	The macro instruction was the result of an execute micro sequence.
30-31	Length of currently executing instruction.

ATU State Format

Bits	Description
0	Instruction can be restarted after a page fault.
1-3	Effective source ring.
18	The last non-LAT start was an IPST.
19	The last non-LAT start was an ICAT.
20	The last non-LAT start was a write.
21-23	The mode bits of the last non-LAT start.
24	The last non-LAT start was during a Cache Block Crossing routine.

ALU State Format

Bits	Description
16-23	Scratch pad address register.
24-27	DES register pointer.
28-31	SRC register pointer.

MSEQ State Format

Bits	Description
0-13	Micro address at which the fault occurred.
14	The test condition the sequencer was using to calculate the next address.
16-23	Microcode Flag bits.
30-31	Contents of the dispatch data register.

For instructions and operations that can cross inward segment boundaries (for example, LCALL, XCALL, and processor-initiated calls for interrupts and protection faults), the processor changes the segment field to reflect the inner segment before modifying that inner segment's wide stack or its page zero parameters.

For instructions and operations that can cross outward segment boundaries (for example, WRTN, WRSTR, WPOPB, and processor-initiated returns from interrupts and protection faults), the segment field reflects the inner segment until the processor makes all modifications to that inner segment's wide stack and its page zero parameters. The processor then changes the segment field to reflect the outer segment before the processor modifies the outer segment's wide stack or its page zero parameters.

All other words in the context block contain information used by the microcode and other internal systems. The context block does not save the floating-point state. To save this information, use a *Push Floating-Point State* instruction.

Note that the processor assumes that the context block save area, the pointer to the save area, and all indirect chains accessed align on double-word boundaries. User programs should also make this assumption.

Appendix E
Standard I/O Device Codes

Octal Device Codes	Mnem	Priority Mask Bit	Device Name	Octal Device Codes	Mnem	Priority Mask Bit	Device Name
00	—	—	Reserved	40	DCU	4	Data control unit
01				41	DCU1	4	Second data control unit
02				42			
03				43	PIT	6	Programmable interval timer
04	UPSC	13	Universal power supply controller	44			
05				45	SCP	15	System control processor
06	MCAT	12	Multiprocessor adapter transmitter	46	MCAT1	12	Second multiprocessor transmitter
07	MCAR	12	Multiprocessor adapter receiver	47	MCAR1	12	Second multiprocessor receiver
10	TTI	14	TTY input	50	IAC1	11	Intelligent asynchronous controller 1
11	TTO	15	TTY output	51	IAC2	11	IAC2
12				52	IAC3	11	IAC3
13				53	IAC4	11	IAC4
14	RTC	13	Real-time clock	54	IAC5	11	IAC5
15				55	IAC6	11	IAC6
16				56	IAC7	11	IAC7
17	LPT	12	Line printer	57	LPT1	12	Second line printer
20				60			
21				61			
22	MTB	10	Magnetic tape	62	MTB1	10	Second magnetic tape
23				63			
24				64			
25				65	IAC	11/5	Host to IAC interface
26	DKB	9	Fixed-head DG/Disk	66	DKB1	9	Second fixed-head DG/Disk
27	DPF	7	DG/Disk storage subsystem	67	DPF1	7	Second DG/Disk storage subsystem
30	IAC13	11	IAC13	70	IAC8	11	IAC8
31	IAC14	11	IAC14	71	IAC9	11	IAC9
32	IAC15	11	IAC15	72	IAC10	11	IAC10
33	DKP	7	Moving head disk	73	IAC11	11	IAC11
34	ISC	4	Intelligent synchronous controller	74	IAC12	11	IAC12
35				75			
36				76		—	Reserved
37				77	CPU	—	CPU and console functions

Table E.1 Standard I/O device codes

Appendix F Fault Codes

Tables F.1 through F.4 contain an explanation of the fault codes returned in AC1 for protection, page, stack, and decimal/ASCII faults. Table F.5 contains an explanation of the universal power supply controller (UPSC) fault codes.

Protection Faults

Table F.1 lists the meanings of the codes returned in AC1 when an MV/10000 address translator protection fault occurs.

AC1 Code (octal)	Meaning
0	Read violation
1	Write violation
2	Execute violation
3	Validity bit protection (SBR or PTE)
4	Inward address reference
5	Defer (indirect) violation
6	Illegal gate — out of bounds or gate bracket access violation
7	Outward call
10	Inward return
11	Privileged instruction violation
12	I/O protection violation
14	Invalid microinterrupt return block

Table F.1 Protection fault codes

Page Faults

Table F.2 lists the page fault codes that the processor stores in AC1.

AC1 Code	Meaning
0	Multiple ERCC fault
1	Page table depth
2	Page table page fault
3	Reserved
4	Normal object reference

Table F.2 Page fault codes

Stack Faults

Table F.3 lists the MV/10000 stack fault codes. The processor does not return an error code for a narrow stack fault.

AC1 Code	Meaning
000000	Overflow on every stack operation except SAVE and WMSP, or ring crossing.
000001	Underflow or overflow would occur if the instruction were executed — WMSP, WSSVR, WSSVS, WSAVR, and WSAVS. (PC in return block refers to the instruction that caused the stack fault.)
000002	Too many arguments on a cross ring call.
000003	Stack underflow.
000004	Overflow due to a return block pushed as a result of a microinterrupt or fault.

Table F.3 Stack fault codes

Decimal/ASCII Faults

Table F.4 lists the decimal and ASCII fault codes. The first and second columns list the codes that appear in AC1. The third column lists the instructions that caused the faults. The last column describes the conditions that can cause the fault.

Code Returned in AC1		Faulting Instruction	Meaning
Narrow	**Wide**		
000000	100000	EDIT, WEDIT	An invalid digit or alphabetic character encountered during execution of one of the following subopcodes: DMVA, DMVF, DMVN, DMVO, or DMVS.
000001	100001	LDIX, STIX, EDIT, WEDIT, WLDIX, WSTIX	Invalid data type (7). Invalid data type (6 or 7).
000002	100002	EDIT, WEDIT	DMVA or DMVC subopcode with source data type 5; AC2 contains the data size and precision.
000003	100003	EDIT, WEDIT	An invalid opcode; AC2 contains the data size and precision.
000004	100004	STI, LDI, WSTI, WLDI STIX, LDIX, WSTIX, WLDIX	Number too large to convert to specified data type. Number > (10^{16}) - 1 Number too large to convert to specified data type. Number > (10^{32}) - 1
000006	100006	WLSN, WLDI, LSN, LDI, LDIX, WLDIX, EDIT, WEDIT	Sign code is invalid for this data type.
000007	100007	WLSN, WLDI, WLDIX, LSN, LDI, LDIX	Invalid digit.

Table F.4 Decimal and ASCII fault codes

UPSC Faults

Table F.5 lists the power system fault codes by fault category. A fatal power system fault causes a system shutdown.

NOTE: *Codes not shown are unused.*

Fault Code and Category Bits 9-15 (octal)	Operation	Fatal or Nonfatal Status
Category 0	System off or no fault or UPSC fault.	
000	System off or no fault.	—
170	Diagnostic mode timeout (MV/10000 failed to complete I/O).	Nonfatal
Category 1	Environment fault (VNR = Voltage Nonregulated Unit).	
011	VNR undervoltage.	Fatal > 300 msec
021	VNR overvoltage.	Fatal
031	Power supply over temperature.	Fatal > 15 sec
041	Chassis over temperature.	Fatal > 15 sec
Category 2	Fan failure.	
002	Blower or multiple fan failure.	Fatal > 15 sec
012	Failure of fan no. 1.	Fatal > 15 sec
022	Failure of fan no. 2.	Fatal > 15 sec
032	Failure of fan no. 3.	Fatal > 15 sec
042	Failure of fan no. 4.	Fatal > 15 sec
052	Failure of fan no. 5.	Fatal > 15 sec
062	Failure of fan no. 6.	Fatal > 15 sec
072	Cannot set for signals.	Nonfatal
Category 3	VNR fault.	
013	Battery backup fault indicated.	Nonfatal unless on batteries
Category 4	Power supply fault (includes undervoltages).	
004	+5V logic undervoltage.	Fatal > 1 msec
014	+5V logic current not sharing.	Nonfatal
044	+5V memory undervoltage, PS1.	Fatal > 1 msec
054	+5V memory undervoltage, PS2.	Fatal > 1 msec
064	+5V memory undervoltage, PS3.	Fatal > 1 msec
074	+12V memory or +12V undervoltage, PS1.	Fatal > 1 msec
104	+12V memory or +12V undervoltage, PS2.	Fatal > 1 msec
114	+12V memory or +12V undervoltage, PS3.	Fatal > 1 msec
124	−5V memory or −5V undervoltage, PS1.	Fatal > 1 msec
134	−5V memory or −5V undervoltage, PS2.	Fatal > 1 msec
144	−5V memory or −5V undervoltage, PS3.	Fatal > 1 msec
154	Undervoltage PS1, voltage unknown.	Fatal > 1 msec
164	Undervoltage PS2, voltage unknown.	Fatal > 1 msec
174	Undervoltage PS3, voltage unknown.	Fatal > 1 msec

Table F.5 Universal power supply controller fault codes (octal)

Fault Code and Category Bits 9-15 (octal)	Operation	Fatal or Nonfatal Status
Category 5	Overvoltage fault.	
005	Overvoltage on +5V.	Fatal
045	Overvoltage on +5V memory, PS1.	Fatal
055	Overvoltage on +5V memory, PS2.	Fatal
065	Overvoltage on +5V memory, PS3.	Fatal
075	Overvoltage on +12V or +12V memory, PS1.	Fatal
105	Overvoltage on +12V or +12V memory, PS2.	Fatal
115	Overvoltage on +12V or +12V memory, PS3.	Fatal
125	Overvoltage on −5V or −5V memory, PS1.	Fatal
135	Overvoltage on −5V or −5V memory, PS2.	Fatal
145	Overvoltage on −5V or −5V memory, PS3.	Fatal
155	Overvoltage PS1, voltage unknown.	Fatal
165	Overvoltage PS2, voltage unknown.	Fatal
175	Overvoltage PS3, voltage unknown.	Fatal
Category 6	Overcurrent fault.	
006	Reed switch sense low, +5V output, overcurrent to logic slots.	Fatal > 1 msec
016	Overcurrent on +5V, PS1.	Fatal > 1 msec
026	Overcurrent on +5V, PS2.	Fatal > 1 msec
036	Overcurrent on +5V, PS3.	Fatal > 1 msec
046	Overcurrent on +5V memory, PS1.	Fatal > 1 msec
056	Overcurrent on +5V memory, PS2.	Fatal > 1 msec
066	Overcurrent on +5V memory, PS3.	Fatal > 1 msec
067	Overcurrent on +12V or +12V memory, PS1.	Fatal > 1 msec
106	Overcurrent on +12V or +12V memory, PS2.	Fatal > 1 msec
116	Overcurrent on +12V or +12V memory, PS3.	Fatal > 1 msec
126	Overcurrent on −5V or −5V memory, PS1.	Fatal > 1 msec
136	Overcurrent on −5V or −5V memory, PS2.	Fatal > 1 msec
146	Overcurrent on −5V or −5V memory, PS3.	Fatal > 1 msec
156	Overcurrent PS1, voltage unknown.	Fatal > 1 msec
166	Overcurrent PS2, voltage unknown.	Fatal > 1 msec
176	Overcurrent PS3, voltage unknown.	Fatal > 1 msec
Category 7	UPSC fault.	
007	Checksum error on PROM (blinks).	Fatal at power-up
177	LED lamp test at power-up (short duration). (Present when ac power line is too low to complete power-up sequence. UPSC continues sequence when ac power is acceptable.)	Nonfatal

Table F.5 Universal power supply controller fault codes (octal), continued

Appendix G
Load Control Store Instruction

This appendix describes the *Load Control Store* instruction and its associated microcode file.

> **WARNING:** *The* **Load Control Store** *instruction changes various parts of the machine's internal state. This instruction is intended for diagnostic and special system applications.*

Load Control Store
LCS
NIO 2,CPU

0	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The *Load Control Store* instruction loads and verifies the soft internal states of the machine (for example, micro-store, decode rams, and scratch pad). In conjunction with bits 16 through 31 of three accumulators (AC0, AC1, AC2), the LCS instruction performs a load and verify, or verify only, using the contents of a microcode file.

AC0 contains the load and verify, or verify only, argument, and the destination code; AC1 contains the bit length of the code data; and AC2 contains a pointer to the first block of data.

> **NOTE:** *The LCS instruction loads a maximum of 16K words with each instruction. Therefore, it may be necessary to use multiple LCS instructions.*
>
> *This instruction is noninterruptible.*

The call sequence for the LCS instruction is:

LCS
error return
normal return

The formats for the three accumulators follow.

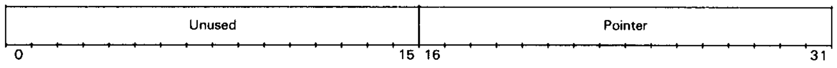
AC0

Unused	L/V	Destination Code
0–15	16	17–31

AC1

Unused	Bit Length
0–15	16–31

AC2



AC #	Contents	Meaning
0	L/V	Load/verify option. 0 Implies load and verify. 1 Implies verify only.
	Destination Code	Code for where the data is to be loaded.
1	Bit Length	Bit length of code data.
2	Pointer	Pointer to first block of data (nonindirectable).

The steps for LOAD and VERIFY are:

1. Parse microcode file blocks as follows: load Code blocks, fill Fill blocks, ignore Revision blocks, and print Comment blocks.

   Repeat this sequence until an End block is encountered.

2. Verify Code blocks that were loaded in step 1; ignore Fill, Comment, and Revision blocks.

If an End block is encountered, the LCS instruction is completed.

The sequence of events for the VERIFY ONLY is step 2 of the Load and Verify.

Microcode File Format

The microcode file format contains data for use in various parts of the machine's state. The microcode format is a block-oriented format (arranged into packets or blocks) that contains a description of the size of the block and the type of data it contains.

Figure G.1 shows the general format for each microcode file.

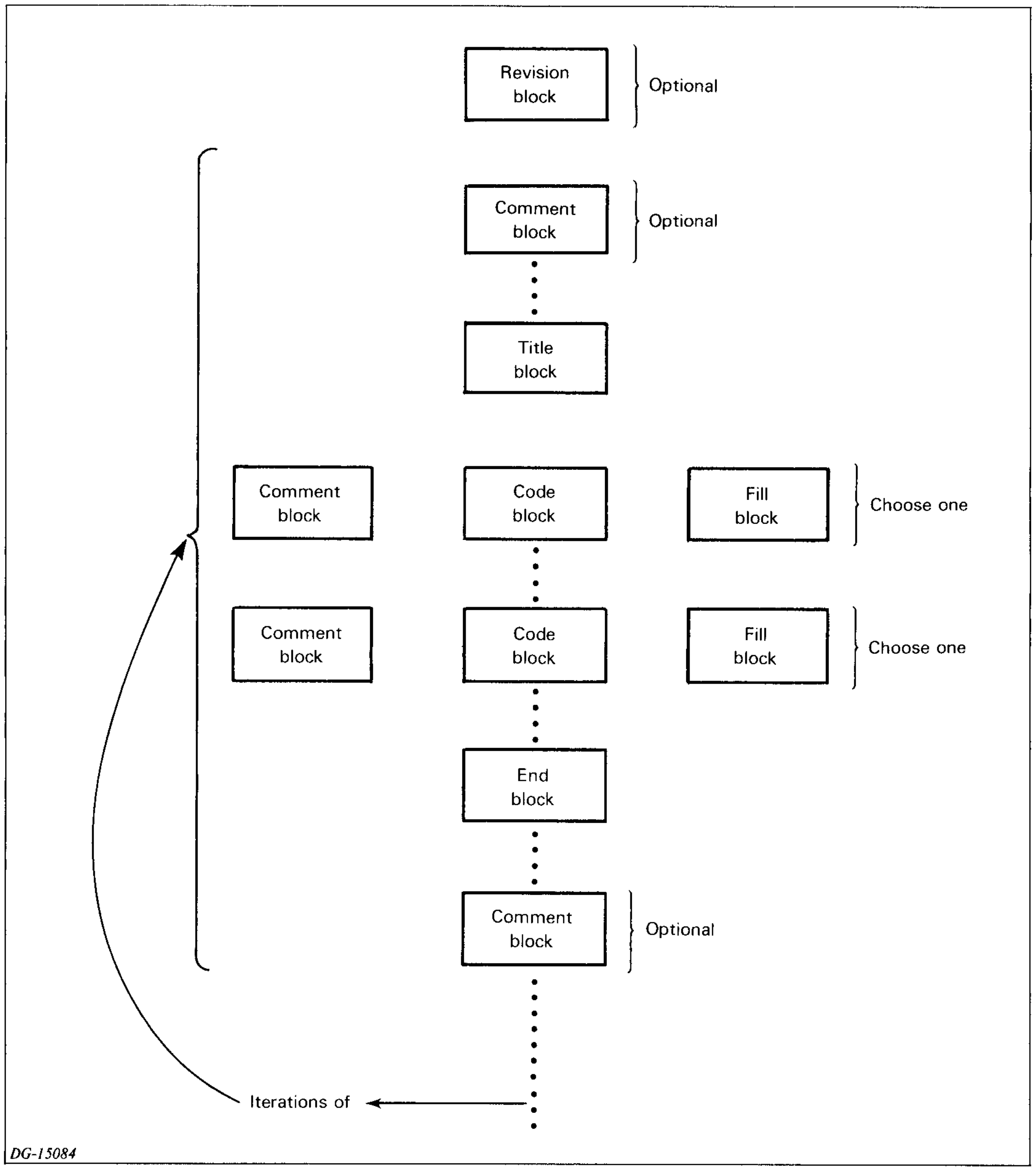


Figure G.1 Microcode file format

Microcode Block Format

Each microcode file must begin with a Title block and finish with an End block (Title/End block pair). Fill and Code blocks must be placed between the Title/End block pair. The Revision block precedes the first Title block. Comment blocks can appear anywhere within the microcode file.

Title blocks contain data pertaining to the code word's bit length, and the destination code. The data from the Title block should be used by the program issuing the LCS instruction as the data for AC0 and AC1.

End blocks contain the necessary data to continue execution or terminate the LCS instruction.

Code blocks contain code words and the starting location for storing each code word. Code blocks must appear between a Title/End block pair.

Fill blocks contain code words for use as background filler and the locations to receive this data. Fill blocks must appear between a Title/End block pair.

Comment blocks contain data that can be output to the system console or ignored. Comment blocks can appear anywhere within the microcode file structure. If the Comment block appears within the Title/End block pair (internal), the data is output to the system console; if the Comment block appears outside the Title/End block pair (external), the program issuing the LCS instruction decides whether to output or ignore the data.

Revision blocks contain the target CPU model number and the major and minor revision numbers for the microcode. Revision blocks should appear as the first block of the microcode file. The program issuing the LCS instruction determines whether the Revision blocks are ignored or output to the system console.

LCS Implementation

The LCS instruction performs the following functions:

- Recognizes Code blocks and loads the data contained into the proper destination addresses.
- Recognizes internal Comment blocks and prints the text string on the system console.
- Recognizes Fill blocks and performs a fill operation of the proper destination.
- Recognizes End blocks and performs a Verify operation upon the previously loaded data.
- Recognizes any of five error conditions (see the section entitled "Error Return") and returns the proper error code to AC0.

NOTE: *The LCS instruction operates on Code, Comment, Fill, and End blocks as described above. The program issuing the LCS instruction must parse out and set up the information from the Title and Revision blocks and any external Comment blocks.*

Microcode Blocks

Figure G.2 shows the general form of each microcode block.

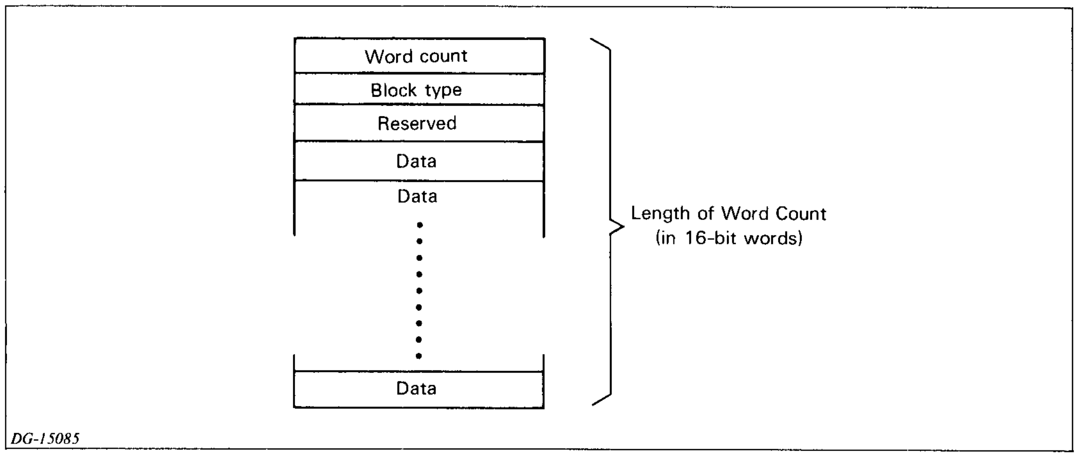


Figure G.2 Microcode block form

The first word of each block is the Word Count — the number of 16-bit words in the microcode block.

The second word of each block is the Block Type, which indicates the type of data contained in the block.

The third word is reserved for future use.

The remaining words contain the data pertaining to the block type.

The formats for the specific blocks follow.

TITLE

Format:

Word Count	7
Block Type	0
Reserved	
Data word 1	Code word's bit length.
Data word 2	Reserved for future use.
Data word 3	Reserved for future use.
Data word 4	Destination (code for where the data is to be loaded). Only positive non-zero 16-bit integers in the range 1 through 77777_8 are accepted by the processor.

The data from the first Title block is used by the program issuing the LCS instruction. For example:

AC0 $\leftarrow$ Data word 4 (destination)

AC1 $\leftarrow$ Data word 1 (code word's bit length)

END

Format:

Word Count	5
Block Type	1
Reserved	
Data word 1	Control word

Bits	Meaning
0-12	Reserved
13	Destination completion indicator 0 indicates more code of this destination may follow. 1 indicates no more code.
14	Switch from PROM to RAM Control Store 0 indicates to stay in current mode. 1 indicates switch to RAM.

	15	Start designator 0 indicates start Host (and continue SCP). 1 indicates start Master (SCP); data word 2 must be an address.
Data word 2	Address that is to be started.	

NOTE: *If this is -1 (177777_8), continue execution with the LCS normal/error return.*

The following chart summarizes the combined actions of Data word 1 (bit 15) and Data word 2:

Data Word 2 Contains	**Data Word 1 (bit 15)**	
	0	**1**
−1	Continue Host at LCS normal/error return.	Illegal.
Address	Start Host at this address; continue Master.	Start Master at this address; Host remains halted.

CODE

Format:

Word Count	Variable
Block Type	2
Reserved	
Data word 1	Location for storing the first code word in this block.
Data word 2 to N+1	First code word of the block.
Data word N+2 to 2N+1	Code word for the next sequential address.
Data word 2N+2 to 3N+1	Code word for the next sequential address.

.
.
.

Until end of block

NOTE: *Code data is in a word-aligned format: N is the number of 16-bit words that contain one code word [N = (word-bit-length + 15)/16]*

FILL

Format:

Word Count	N+5 [N=(word-bit-length + 15)/16]
Block Type	3

Reserved	
Data word 1	Starting location for storing code word.
Data word 2	Ending location for storing code word.
Data word 3 to N+2	Code word to be used as background filler.

The Fill block allows a method to "background fill" certain destinations of the machine; for example, zero-fill the control store to induce parity errors if an uninitialized location is erroneously entered during execution.

NOTE: *The Fill functionality can also be accomplished via Code blocks of the appropriate data.*

COMMENT

Format:

Word Count	Variable
Block Type	4
Reserved	
Data word 1	String length The length of the ASCII string (terminating NULL[s] are not counted). An odd string length indicates one terminating NULL; an even string length indicates two terminating NULLs.
Data word 2 to X+2	ASCII string (packed right to left) terminated by a NULL. [X = (String length + 1)/2]

REVISION

Format:

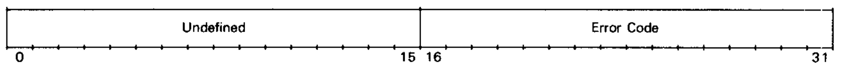
Word Count	6
Block Type	5
Reserved	
Data word 1	Target CPU model number.
Data word 2	Microcode major revision number.
Data word 3	Microcode minor revision number.

Error Return

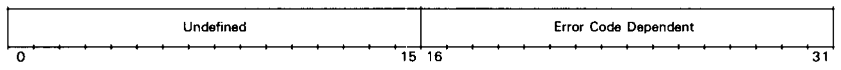
Upon encountering an error, the three accumulators (AC0, AC1, AC2) will contain an indication of the problem.

The formats for the accumulators follow.

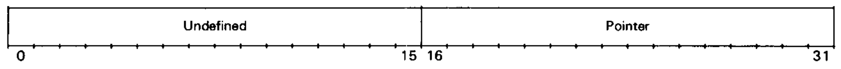
AC0



AC1



AC2



AC #	Contents	Meaning
0	Error Code	Code returned denoting type of error (defined below). **Code Error** 1 Verify error 2 Illegal code word length 3 Unexpected block type 4 Illegal block length 5 Unknown destination
1	Error Code Dependent	If unspecified AC1 is left unchanged.
2	Pointer	Pointer to erring block. **NOTE:** If an error occurs because of initial erroneous information in either AC0 or AC1, then AC2 is left unchanged.

Error codes returned to AC0:

Code	Meaning	Definition (AC1 Contents) (Possible Cause)
1	Verify error	Indicates that the data was not received properly by the destination. (AC1 will contain the code word location that is in error) (Possible hardware problem)
2	Illegal code word length	Code word bit length does not agree with length of code data as specified by the destination word in the same Title block. (AC1 is unchanged) (Possible attempt to load the wrong model microcode)
3	Unexpected block type	Block type other than allowable types (Code, Fill, End, Revision, or Comment) (AC1 is unchanged) (Possible missing block, or out of sequence) **Note** If any Title blocks are encountered between the Title/End block pair, the unexpected block type error will be returned.
4	Illegal block length	Block length is in error (AC1 is unchanged) (Block length of less than four was specified, or the code block did not contain an integral number of code words) For example: If the code word bit length is 80, then the length of all code blocks must be 4+N*(80+15)/16. N = number of code words per code block 16 = number of bits per word 4 = number of words at the beginning of each code block For this example, all code blocks must be of length 4+5*N
5	Unknown destination	Unknown location for loading of code word (AC1 is unchanged) (Possible attempt to load an incorrect model machine microcode file)

Kernel Functionality

The kernel is the minimum set of microcode necessary for the machine to function properly. With the kernel instruction set (including the LCS instruction) the processor can read in target microcode from an I/O device (using the kernel I/O instructions) and then load this microcode into the control store using the LCS instruction.

Because there is a 16K-word limit to the amount of data that can be loaded with a single LCS instruction, it may take several iterations of accessing the I/O device and executing the LCS instruction to completely change the machine from the kernel to the target.

NOTE: *Because the LCS instruction must return to the host after completion, the kernel instruction set must exist (in working order) after each execution of the LCS instruction.*

Index

Within the index, the letter *f* following a page entry indicates "and the following page"; the letters *ff* following a page entry indicate "and the following pages". The letter *t* following a page entry indicates that a table resides on the page.