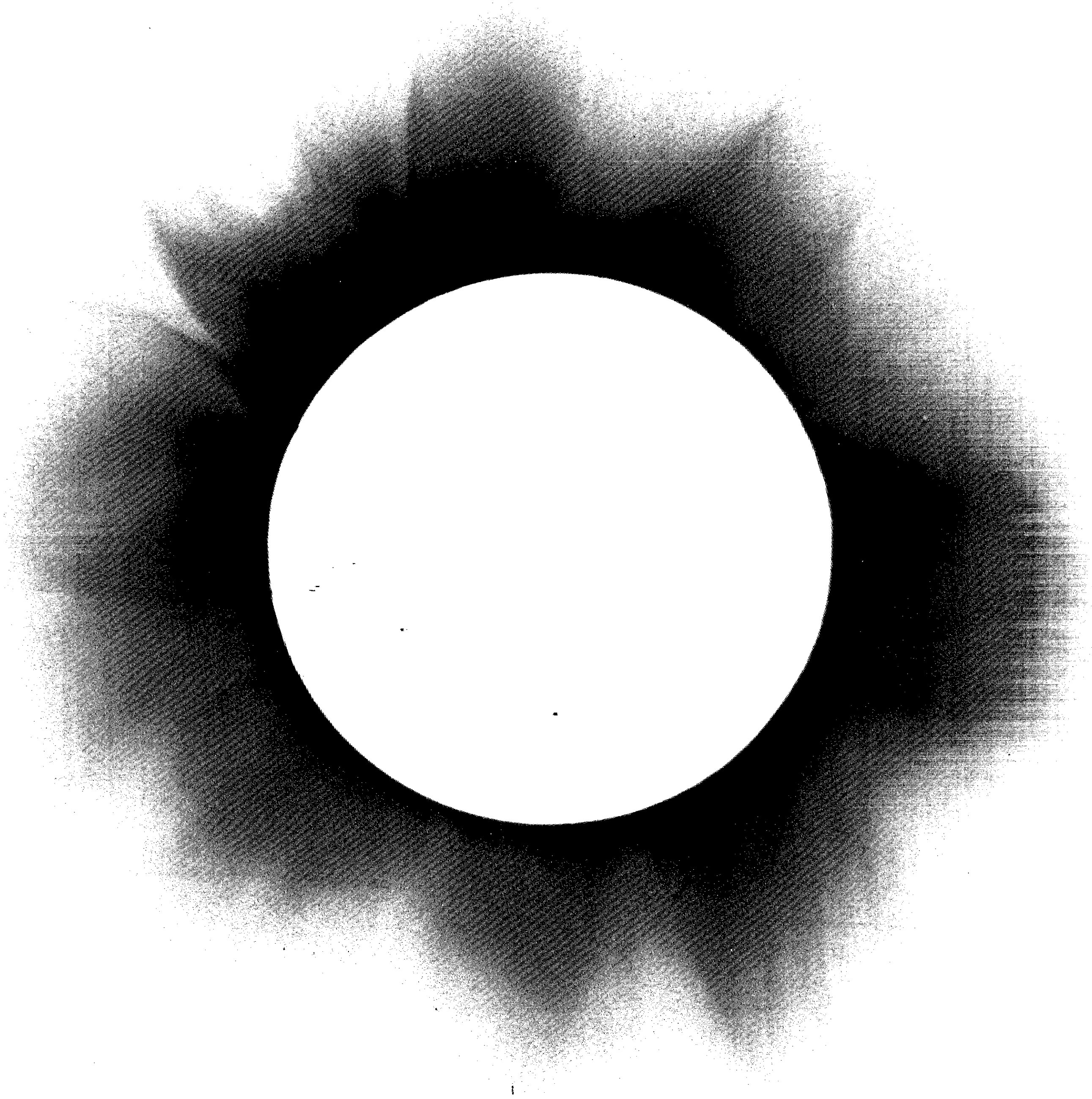
Data General

Microprogrammers' Reference, ECLIPSE MV/10000™ Computer



Microprogrammers' Reference, ECLIPSE MV/10000™ Computer

014-701003

Microprogrammers' Reference,
ECLIPSE MV/10000 Computer
014-701003

Original Release—May 1983

Contents

Preface p-1

Chapter 1 Introduction 1-1

Terminology 1-1
Microprogramming 1-2
MV/10000 Subsystems 1-2
MV/10000 Buses 1-4

Chapter 2 MV/10000 Architecture and Operation 2-1

Clocks and Timing 2-1
The Microsequencer 2-2
  Writable Control Store 2-2
  Microinstruction Register 2-2
  Microprogram Counters 2-3
  Microstack and Microstack Input Multiplexer 2-3
  Top of Stack Register 2-4
  RAM Address Multiplexer 2-5
  DSP Register, Crossbar Net, Dispatch Multiplexer 2-5
  AA Bus 2-6
  Flags 2-6
  SCP Control 2-7
  Tests 2-7
The Integer ALU 2-8
  Integer Register File 2-10
  Registers on the ID Bus 2-11
  Scratch Pad 2-12
  Transfer Register 2-13
  Hex Shifter 2-14
  ALU 2-15
  Carry-In Logic 2-16
  Commercial Test and Edit PROMs 2-17
  Bit Shifter 2-18
  Processor Status Register 2-18
  CPD Bus Register—PDR 2-19
The Floating-Point Unit 2-19
  FPU Buses 2-21
  Mantissa Logic 2-21
    General Logic 2-22
    Floating-Point Register File 2-22
    Floating-Point Status Register 2-22

Floating-Point STATE Register . . . 2-24
Hex Shifter . . . 2-24
MAG Register . . . 2-24
Mantissa ALU . . . 2-26
Working Register . . . 2-26
Multiply Hardware . . . 2-26
X and Y Registers . . . 2-27
YSEL Counter . . . 2-28
Multiply ALU . . . 2-28
Divide Hardware . . . 2-28
Divide Guard Digit Register . . . 2-29
Divide Partial Remainder Register . . . 2-29
Sign and Exponent Logic . . . 2-30
Exponent Working Register . . . 2-30
Exponent ALU . . . 2-31
SA and SB Registers . . . 2-31
Sign Logic . . . 2-31
The Address Generator . . . 2-31
Buses . . . 2-32
Register File . . . 2-33
Register File Addressing . . . 2-34
AGB Bus Sources . . . 2-35
RFIN Multiplexer . . . 2-36
Address Generator ALU . . . 2-36
The Address Translation Unit . . . 2-36
Address Translation Cache . . . 2-37
Referenced/Modified RAM . . . 2-38
Validity RAM . . . 2-38
Logical Address Translation . . . 2-38
Page Table Addressing Logic . . . 2-38
Ring Protection . . . 2-39
Indirection Protection . . . 2-39
Read/Write/Execute Protection . . . 2-40
ATU State . . . 2-40
ATU Diagnostic Register . . . 2-41
ATU Dispatch . . . 2-42
The CPD Bus and Transfer Register . . . 2-43
Instruction Processor . . . 2-43
Interrupts . . . 2-45
I/O Protocols . . . 2-46

Chapter 3 Micro-order Format and Instruction Set . . . 3-1

NAC—Next Address Control . . . 3-2
NAC:COP—Conditional OPcode . . . 3-4
NAC:TSEL—Test Selection . . . 3-6
Microsequencer Tests . . . 3-7
Address Translation Unit Tests . . . 3-9
Integer ALU Tests . . . 3-12
Floating-Point Tests . . . 3-18
NAC:UCOP—Unconditional OPcode . . . 3-20
NAC:DSR—Dispatch Address Source . . . 3-23
Address Generator Micro-orders . . . 3-24

AA and AB—The Register File Address Fields 3-24
AGB—The Address Generator Bus Field 3-28
AOP—Address Generator ALU Operation Field 3-29
AL—Address Generator Register Loading 3-29
Memory Control Micro-orders 3-30
MEMS—Memory Start 3-31
MEMC—Memory Complete 3-32
Bus Control Micro-orders 3-34
CPMS—CPM Bus Sources 3-34
CPDS—CPD Bus Sources 3-35
RAND—Random Micro-orders 3-39
RAND:GEN—General Random Micro-orders 3-40
RAND:GEN:REG0—General/ACSR/ACDR Micro-orders 3-40
RAND:GEN:REG1—Register Load Operations 3-48
RAND:GEN:SPAD—Scratch Pad Input Control 3-49
RAND:ATU—ATU Random Micro-orders 3-50
RAND:ATU:ATU0—ATU Operations 3-50
RAND:ATU:ATU1—Additional ATU Operations 3-58
RAND:ATU:SPAD—Scratch Pad Input Control 3-59
RAND:FIX—Fixed-point Random Micro-orders 3-60
RAND:FIX:COVS—Carry, Overflow and Status 3-60
RAND:FIX:LOAD—Load Registers 3-63
RAND:FIX:SPAD—Scratch Pad Input Control 3-63
RAND:FLT—Floating-Point Random Micro-orders 3-64
RAND:FLT:SGN—Floating-Point Sign 3-65
RAND:FLT:EXP—Floating-Point Exponent 3-66
RAND:FLT:SCNT—Shift Count Control 3-69
Integer ALU Micro-orders 3-71
IA and IB—Integer Register File Addressing 3-71
ID—ID Bus Source Control 3-75
RS—ALU Input Multiplexer Control 3-77
IOP—ALU Control and Shift Magnitude 3-77
IY—IY Bus Source 3-79
IL—Integer Register File Input 3-84
Floating-Point ALU Micro-orders 3-86
FR—FR Bus Source 3-86
FS—FS Bus Source 3-87
FOP—Mantissa Operations 3-88
FWR—Working Register Input 3-89
FCW—Floating-Point Register Write Address 3-90
FL—Register File Load Specifier 3-94
FRG—Floating-Point Register Load Control 3-95
FX—Excess-64 Control 3-99

Chapter 4 Microprogramming Examples 4-1

Memory Accesses 4-2
IPOP—Crossing Macroinstruction Boundaries 4-4
Indirection Resolution 4-5
Dispatching 4-6

Chapter 5 MV/10000 Microcode Macroassembler 5-1

The Macroassembler 5-1
CPM Bus 5-2
CPD Bus 5-3
Memory Starts and Address Generator Operations 5-4
Memory Completion 5-6
ALU Operation Constructs 5-6
IY Bus 5-6
ALU Test 5-9
Loading SPAR 5-10
Edit PROM 5-10
ID Bus 5-11
IR 5-11
FPU Operations 5-11
FD Bus 5-11
FA and FB Buses 5-12
WR == 5-13
Sign and Exponent Control 5-13
Shift Count 5-14
Multiply Control 5-14
FPU State 5-15
Divide Control 5-15
GEN Randoms 5-15
ACSR (SRC Register Pointer) Randoms 5-15
ACDR (DES Register Pointer) Randoms 5-16
Flag Manipulation 5-16
Skips 5-17
Miscellaneous Randoms (NPDR and XTND) 5-17
ATU Randoms 5-18
FIX Randoms 5-18
Next Address Sequence 5-19
Conditional Address Generation 5-19
Unconditional Address Generation 5-19
Pseudo-unconditional Address Generation 5-20
Test Definitions 5-20
Examples 5-24
Unassembled Example 5-24
Assembled Examples 5-26

Appendix A Page Faults A-1

Appendix B CPD Bus Legal Path Analysis B-1

Appendix C CPM Bus Legal Path Analysis C-1

Appendix D ALU Source and Destination Paths D-1

Appendix E Page Zero Locations E-1

Appendix F Fault Codes F-1

Appendix G Exceptions G-1

Appendix H Scratch Pad Addresses H-1

Figures

1-1 MV/10000 Subsystems 1-3

2-1 MV/10000 System Clocks 2-1
2-2 Microinstruction Register 2-2
2-3 Program Counters and Microstack 2-4
2-4 The Crossbar Network and RA Multiplexer 2-6
2-5 Integer ALU 2-9
2-6 Integer ALU Register File 2-11
2-7 Registers on the ID Bus 2-12
2-8 Scratch Pad 2-13
2-9 The Transfer Register 2-14
2-10 The Integer ALU and Associated Logic 2-15
2-11 CARRY Bit and Carry-In Logic 2-16
2-12 The Floating-Point Unit 2-20
2-13 The MAG Register Sources 2-25
2-14 Multiply Data Paths 2-27
2-15 Divide Data Paths 2-29
2-16 Sign and Exponent Logic 2-30
2-17 The Address Generator 2-32
2-18 The Address Translation Unit 2-37
2-19 Page Table Addressing Logic 2-39
2-20 The Instruction Processor 2-44
2-21 I/O Command Formats 2-46

3-1 The MV/10000 Microword 3-1
3-2 The NAC Field 3-2
3-3 RAND Mode Formats 3-40

A-1 Context Blocks A-3

Tables

2-1 RAND Mode and CIB 2-17
2-2 Processor Status Register 2-18
2-3 Address Translation Unit State 2-40
2-4 ATU State Fault Codes 2-41

3-1 MV/10000 Microaddresses 3-3
3-2 Conditional Microorders in the OP Field 3-4
3-3 COM1 Tests 3-14
3-4 COM2 Tests 3-15
3-5 IOT Tests 3-16
3-6 Unconditional OP Microorders 3-21
3-7 Dispatch Address Source 3-23
3-8 RM Field Micro-orders 3-39
3-9 CNST Microorders for RAND MFS0 and MFS1 3-44
3-10 CNST Microorders for RAND AF46 and AF57 3-46
3-11 ATU Restored State 3-51
3-12 CP Mode Code 3-52

4-1 Microword Header Abbreviations 4-1

Preface

The *Microprogrammers' Reference, ECLIPSE MV/10000 Computer* describes the microcode for the ECLIPSE MV/10000™ computer.

Who Should Read This Manual?

This manual is intended as a reference for microprogrammers. It assumes some prior knowledge of the MV/10000 hardware and instruction set.

Manual Organization

The two major sections of this manual (Chapters 2 and 3) describe the MV/10000 hardware and MV/10000 micro-orders. The hardware descriptions are oriented towards microcode control. The micro-order descriptions are arranged by field. Chapter 4 presents examples of MV/10000 microprogram segments; these illustrate typical microcode operations. Chapter 5 describes the MV/10000 microcode macrolanguage and provides examples of microassembler input and output. Appendixes provide supplemental information and microprogramming aids.

Prerequisite Manuals

- *ECLIPSE MV/10000 System Functional Characteristics* (014-000724)
- *Principles of Operation, 32-bit ECLIPSE Systems, Programmers Reference Series* (014-000704)

Other Related Manuals

- *μLink Microcode Linker Manual* (093-400029)
- *μASM Microassembler Manual* (093-400030)
- *SMI Microcode Simulator Manual* (093-400031)

Contacting Data General

- To order any Data General manual, notify your sales representative and supply the manual title and order number.
- If you have software problems, please notify your local Data General systems engineer.
- If you have hardware problems, please notify the Field Engineering Dispatch Center.

End of Preface

Chapter 1
Introduction

The ECLIPSE MV/10000™ computer implements the ECLIPSE MV instruction set. The MV/10000 CPU is microprogrammable. This manual describes the CPU microcode. In this chapter, we begin with a brief discussion of terminology and microprogramming. Then we describe each microcode-controlled subsystem in the MV/10000 CPU. We conclude by describing the CPU buses.

Terminology

Microprogramming terminology is similar to programming terminology. The following special terms are peculiar to microprogamming.

- Microcode—Code written with microinstructions.
- Control store—The local memory, either RAM or ROM, that holds the microcode for a computer.
- Microinstruction—The contents of a location in control store. An MV/10000 microinstruction is 104 bits wide.
- Microassembler—A program that lets you use symbolic names when writing microcode.
- Microfield—A predefined segment of a microinstruction, usually associated with a particular control function.
- Micro-order—A possible value for a microfield. The number of micro-orders available for a microfield depends on the width of the field. In this text, we will usually refer to micro-orders by their microassembly names. For example, "FOP:SUB" means the SUB (subtract) micro-order in the FOP (floating-point operation) microfield.
- Microroutine—The set of microinstructions needed to carry out a complete operation, such as adding two numbers.
- Macroinstruction—A machine-language instruction. A macroinstruction is implemented by one or more microinstructions.

Microprogramming

The MV/10000 machine-language instruction set is interpreted by microcode. In some computers, the machine language is "hard-wired." The signals that control various parts of the computer are generated by logic in the Central Processing Unit (CPU). This logic produces a different set of signals for each instruction the computer can execute. In the MV/10000 processor, microcode generates these same signals. Microcode has the advantage that, unlike hard-wired logic, it can be changed easily to accomodate changes to the instruction set.

Like the logic that preceded it, microcode controls the machine at a primitive level and uses the hardware to interpret machine-language instructions. Each macroinstruction is implemented by a microroutine residing in control store. This microroutine interprets an instruction much as a machine-language program might interpret a higher-level language.

MV/10000 microcode uses Writable Control Store (WCS), which means that microcode is stored in RAM and must be reloaded each time the machine is booted. (Some computers store their microcode in ROM, so that it is available even when the machine first starts up.) MV/10000 microcode is loaded by the System Control Processor (SCP).

MV/10000 Subsystems

The MV/10000 CPU has six separate subsystems: the Instruction Processor, the Microsequencer, the Address Generator, the Address Translation Unit, the Integer ALU, and the Floating-Point Unit. All of these are under microcode control. These subsystems are connected to the System Control Processor (which acts as a system console), the I/O Controller (which connects the MV/10000 processor to peripheral devices), and main memory. Figure 1-1 shows the MV/10000 subsystems.

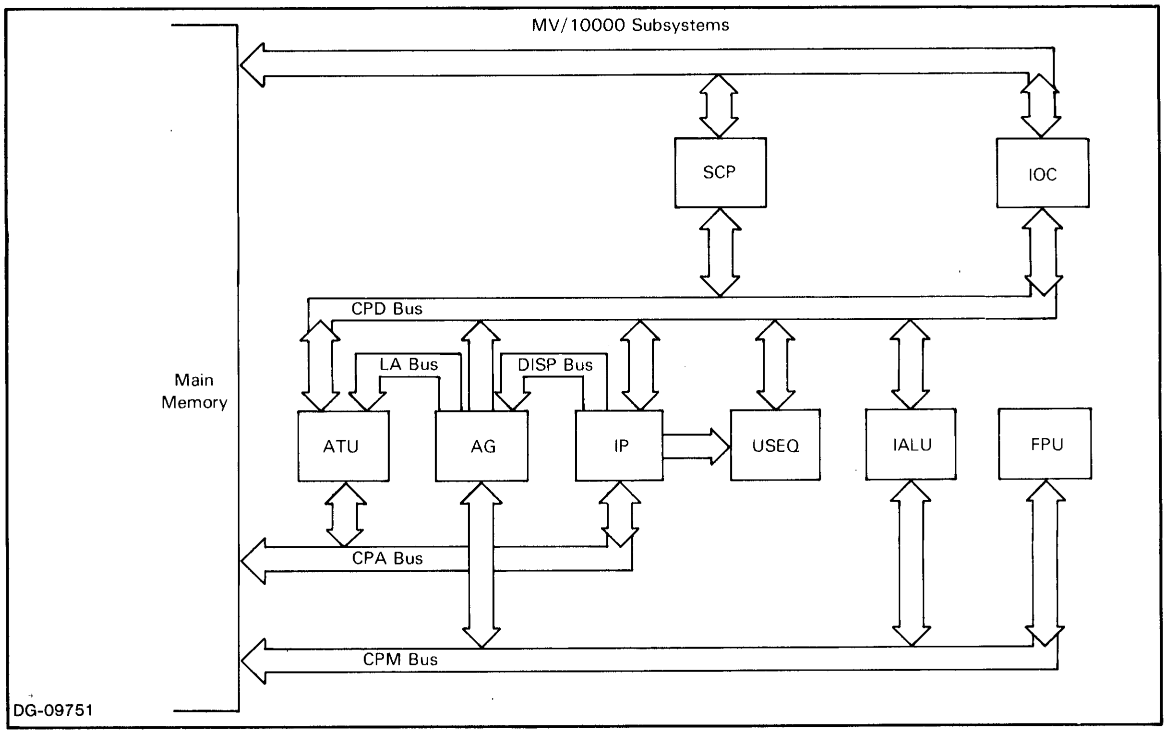


Figure 1-1. MV/10000 Subsystems

The rest of this chapter briefly describes each microcode-controlled subsystem. Chapter 2 describes them in greater detail.

The Instruction Processor

The Instruction Processor (IP) decodes macroinstructions. Decoding an instruction means dividing it into component fields and producing a starting WCS address. The starting address points to the beginning of the microroutine that will execute (interpret) the instruction. The IP is pipelined: while one instruction is executing, several other instructions may be in various stages of decoding.

The IP contains the program counter, the instruction register, and the Instruction Cache (Icache). The Icache speeds up the fetching of instructions.

The Microsequencer

The microsequencer generates addresses into WCS. It determines which microinstruction will execute next. When microcode is being loaded, the microsequencer takes addresses from the SCP, along with the microroutines to be loaded.

The microsequencer contains the microprogram counter, the microstack, and the microaddress-generating logic. It can construct addresses from several different sources, depending on the needs of the microroutine. At the beginning of a microroutine that interprets a macroinstruction, the IP provides the microsequencer with a starting WCS address.

The Address Generator

The Address Generator (AG) constructs logical addresses for the MV/10000 processor. These addresses are 32-bit references to the 4-gigabyte, 8-segment ECLIPSE MV logical address space. The AG provides addresses for the Address Translation Unit, which translates them into physical addresses for main memory.

The AG contains a register file, an ALU, and several individual registers. Decode logic in the AG, under microcode control, determines how an address is constructed.

The Address Translation Unit

The Address Translation Unit (ATU) changes logical addresses to physical addresses. Main memory uses a paging procedure that brings in logical pages from secondary memory only when they are needed. The ATU determines the physical location in main memory of the logical page addressed by the AG.

The ATU has an address translation cache, in which it stores recently used logical-to-physical translations. It also has special logic that lets it access page tables (which locate pages in main memory) very rapidly. In addition, the ATU has protection logic that checks memory references for validity.

The Integer ALU

The Integer ALU (IALU) adds and subtracts fixed-point numbers. It also shifts numbers, translates and validates commercial data, and performs logical operations.

The IALU contains a register file, a shifter, a scratchpad memory, and an ALU.

The Floating-Point Unit

The Floating-Point Unit (FPU) adds and subtracts floating-point numbers and multiplies and divides both floating- and fixed-point numbers. It can manipulate floating-point numbers up to 64 bits wide.

The FPU has separate exponent and mantissa sections. These share a 64-bit-wide register file; however, each section has its own ALU and logic. In addition, the FPU has sign logic to determine the sign for each result.

MV/10000 Buses

The following buses carry data within the MV/10000 CPU. They connect the various subsystems to other subsystems and to external devices:

- *The CPM Bus* is a 32-bit bus that carries data between the CPU and main memory. The AG, the IALU, and the FPU can all source and sink this bus.

- *The CPD Bus* is a 32-bit bus that carries data among the subsystems of the CPU and between the CPU and the I/O Controller. The CPD Bus connects to all the CPU subsystems except the FPU.

- *The LA Bus* carries 32-bit logical addresses between the AG and the ATU.

- *The CPA Bus* carries physical addresses from the ATU to main memory.

End of Chapter

Chapter 2
MV/10000 Architecture and Operation

This chapter describes elements of the MV/10000 processor, and explains how they are interconnected. Most of the descriptions are oriented towards a microprogrammer's point of view, with frequent references to specific micro-orders. Chapter 3 contains full descriptions of the micro-orders.

In this chapter, we examine the clocks and the microsequencer for the MV/10000 processor; then the Arithmetic Logic Units; and finally the addressing logic and instruction processor. We also look briefly at the protocols for I/O and memory references.

Clocks and Timing

The basic clock for the MV/10000 processor is SYS clock, which has a cycle of 70 nanoseconds. From SYS clock each board derives its own clock, typically called CP clock, which has a cycle of 140 nanoseconds. CP clock is the instruction-cycle clock for microinstructions. Thus, the basic system timing cycle is 140 nanoseconds.

The basic timing cycle can be extended by coding RAND:<GEN:REG0 or ATU:ATU0>:XTND. This code extends the CP clock cycle by two SYS clock periods, so that CP clock takes 270 nanoseconds. Figure 1 shows the basic MV/10000 clocks.

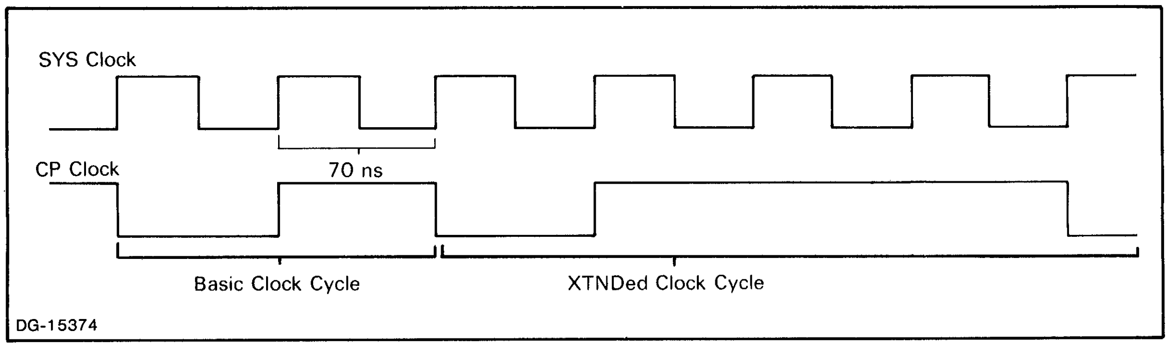


Figure 2-1. MV/10000 System Clocks

The Microsequencer

The microsequencer determines the next microinstruction to be executed. The microsequencer contains part of the Writable Control Store, the microinstruction register, and the next address logic.

Writable Control Store

The MV/10000 Writable Control Store consists of 8K microwords divided into 1K pages. (The WCS address is 14 bits; however, only 13 are used at this time.) Each microword consists of 104 bits. WCS is divided so that 48 bits of each microword are on the microsequencer card and 56 bits are on the Address Generator (AG) card. The System Control Processor (SCP) loads microroutines into WCS when the system is booted.

Microinstruction Register

The microinstruction register is 104 bits wide, divided into 26 fields, 2 parity bits, and 4 unused bits. Figure 2-2 shows the microinstruction register. The fields in the microinstruction register are decoded to provide the control signals that operate the MV/10000 computer.

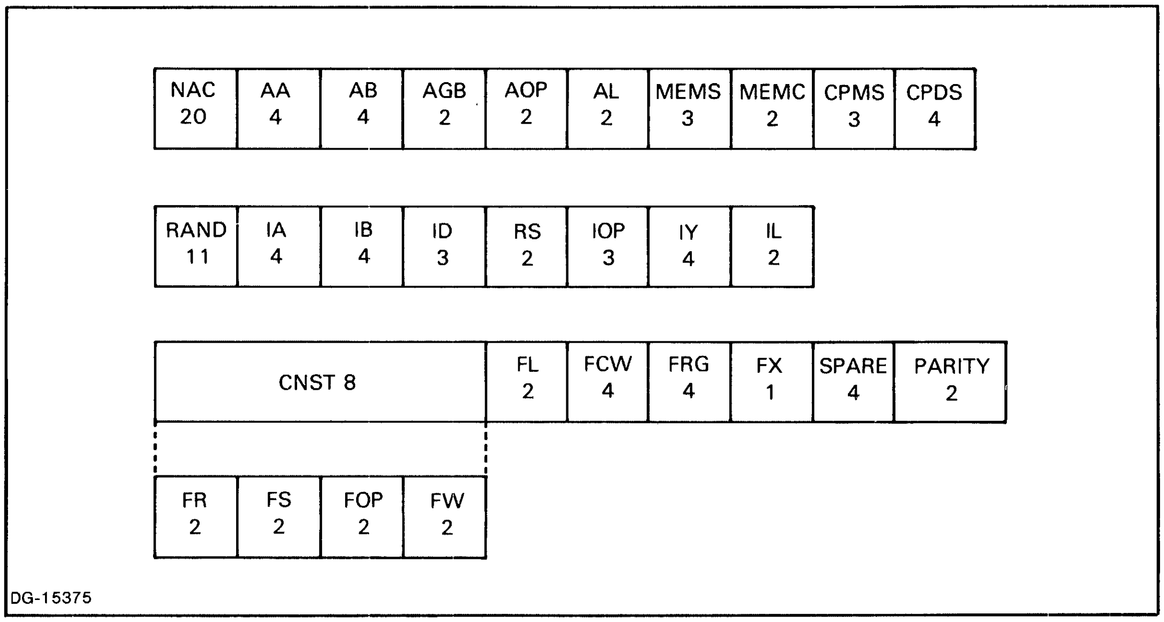


Figure 2-2. Microinstruction Register

Microprogram Counters

The MV/10000 processor has a microprogram counter (uPC) and an incremented microprogram counter (uPC+1). Both of these are available to microroutines. uPC+1 increments modulo 1024; that is, if uPC points to the upper boundary of a 1K page, uPC+1 addresses word zero in that page.

Both PCs are loaded whenever an address is sent to WCS. uPC+1 is automatically incremented when it is loaded. Figure 2-3 shows the two microprogram counters.

Microstack and Microstack Input Multiplexer

The microstack is a hardware stack that microroutines can use for calls and traps. It contains fifteen 16-bit words. The current value at the top of the stack is kept in the top of stack (TOS) register.

You can push either a 14-bit address or a 16-bit value from the CPD Bus onto the microstack. Inputs to the stack come through a multiplexer controlled by the NAC:COP or NAC:UCOP fields in the microword. The stack logic, controlled by these fields, determines whether the stack is pushed or popped. In addition, the stack control logic signals empty and IPOP. (IPOP occurs when a microinstruction pops an empty microstack; this operation dispatches to the microroutine for the next macroinstruction.) Figure 2-3 shows the microstack.

The NAC:COP and NAC:UCOP fields control the stack input multiplexer. This multiplexer can select one of the following to be pushed on the stack:

- the uPC+1
- the AA Bus (described below)
- the CPD Bus, bits 0-15 (inverted)

uPC+1 and AA are 14-bit addresses that the microcode can use to address WCS.

The most significant 16 bits of the CPD Bus can be pushed onto the stack also. This operation is used primarily to restore state.

In addition to uPC+1, AA, and CPD, the hardware can also push the current uPC and the current test result. It does this during a hardware TRAP; the uPC cannot be pushed under microcode control. To restore from a TRAP, the micro-order NAC:COP:CRST must be performed. This micro-order pops the test result from the stack and makes it the current test result. This result appears as TOS14 in the TOS register.

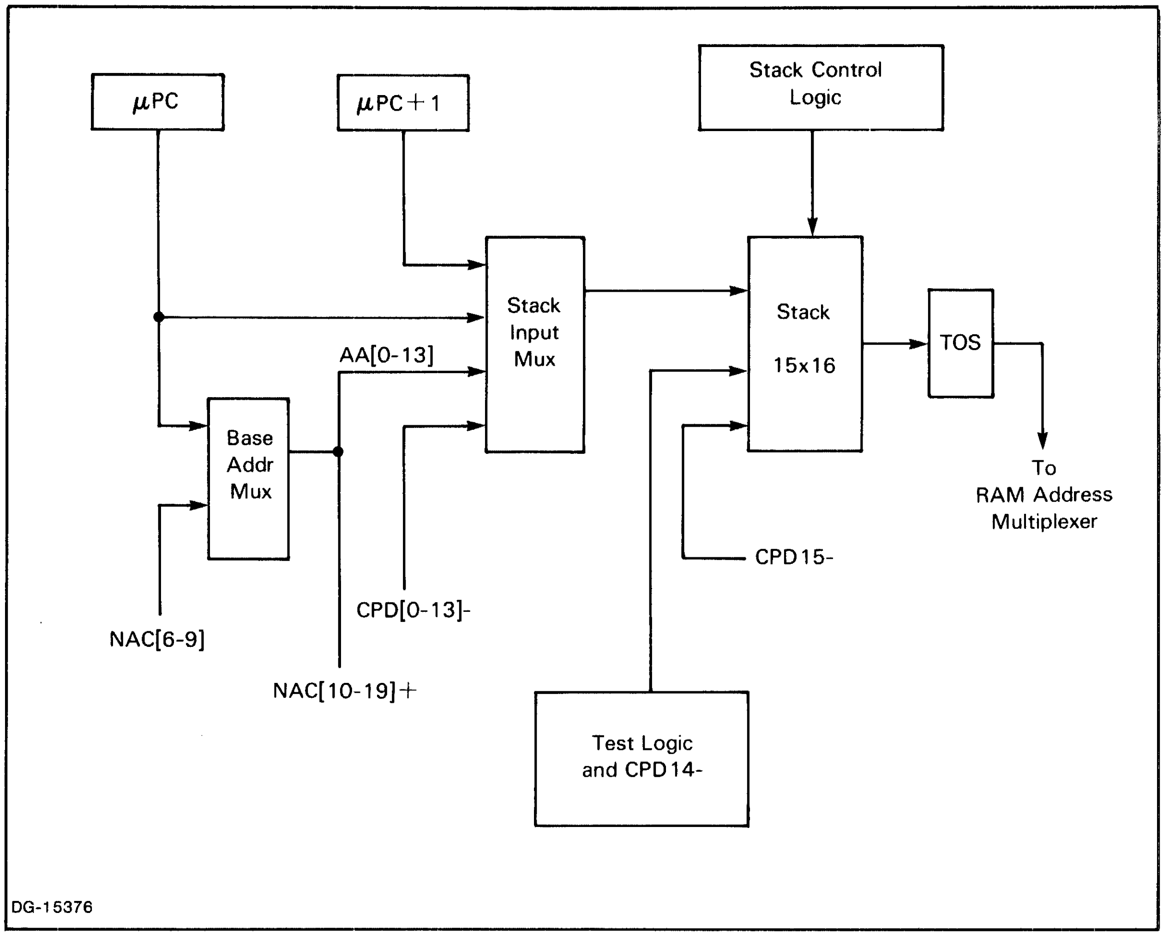


Figure 2-3. Program Counters and Microstack

Top of Stack Register

The top of stack (TOS) register holds the current value at the top of the microstack. (Figure 2-3 and Figure 2-4 show the TOS register.) This value can be sent as an address to WCS or it can be gated onto the CPD Bus. When the microstack is empty, the TOS register is disabled by the stack control logic. When the logic disables the TOS register, it enables a multiplexer (NTOS select) that supplies a new address. This multiplexer selects between the starting microaddress (STUAD), derived from the next macroinstruction, and CRA, an address the System Control Processor supplies. (Figure 2-4 shows the multiplexer.) The CRA is used for initial microcode loading and is not enabled in a running system.

RAM Address Multiplexer

The RAM Address (RA) multiplexer selects the control store address. (Figure 2-4 shows the RA multiplexer.) The multiplexer is controlled by the NAC:COP and NAC:UCOP fields, and selects among four possible addresses for the WCS:

- The TOS or STUAD address (described above)
- The address from the dispatch register and crossbar network
- The AA Bus
- The incremented microprogram counter (uPC+1)

During a trap, this multiplexer forces an address to the appropriate trap routine.

DSP Register, Crossbar Net, Dispatch Multiplexer

The dispatch (DSP) register is used to form an address for WCS. The 8-bit register is loaded from CPD[24-31]-. The output of the dispatch register goes to a cross-bar network that provides three address formats. These are constructed from a combination of the AA Bus, two bits (ATD[0-1]) from the Address Translation Unit (ATU), and the dispatch register. The three formats are:

- AA[0-9],DSP[4-7]
- AA[0-5],DSP[0-7]
- AA[0-9],0,ATD[0-1],0

The dispatch multiplexer chooses among these addresses. The NAC:DSR field selects which of these addresses will go to the RA multiplexer. The DSR field is part of all microinstructions that specify a dispatch address.

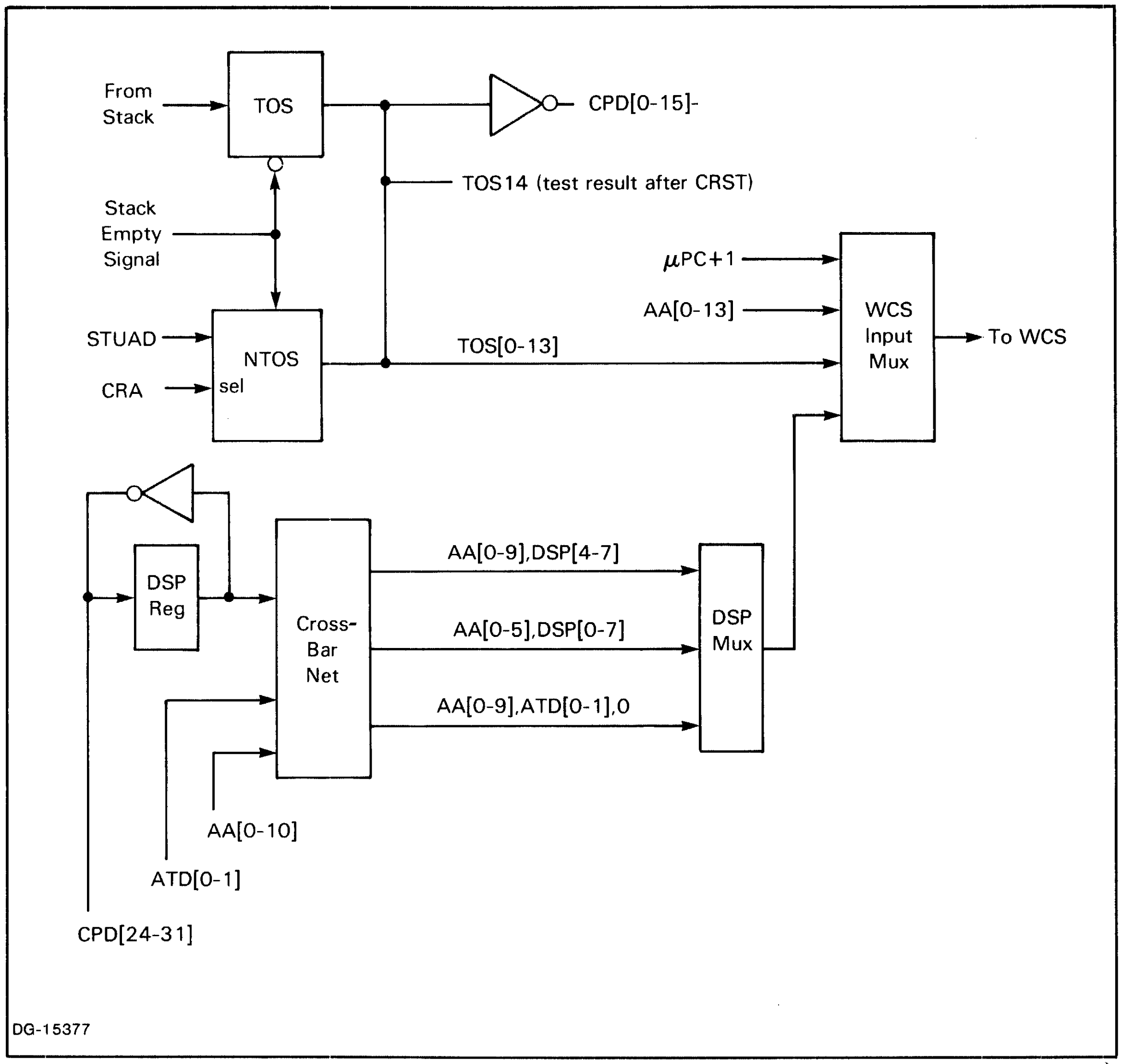


Figure 2-4. The Crossbar Network and RA Multiplexer

AA Bus

The AA Bus is the main address bus in the microsequencer. It is fourteen bits wide, is sourced by the NAC:ADDRESS field of the microword and by uPC, and sources both the RA multiplexer and the crossbar network. The least significant ten bits of the AA Bus come from the NAC:ADDRESS field of the microword. The four most significant bits come from either NAC:ADDRESS or from the page bits [0-3] of the uPC, depending on the type of operation coded: unconditional or conditional, respectively.

Flags

The microsequencer card has eight flags that microcode can test. At IPOP time, the IP decode RAM sets flags 0-3, which refer specifically to the current macroinstruction.

Flag 0	Flag 0 indicates the width of the data in the ALU and controls the word/sign extension on the IY Bus (see the ALU section). For FLAG0=0, the width is 16 bits; for FLAG0=1, the width is 32 bits.
Flag 1	Flag 1 indicates the width of data on the Logical Address (LA) Bus. For narrow width addressing, the 15 least-significant bits and LA0 will be driven by the Address Generator. The ATU will supply the current ring bits, setting LA[1-3] equal to CRE[1-3]; bits LA[4-16] will be zero. For wide addressing, the Address Generator will drive all the bits on the LA Bus, unless the RAND:ATU:ATU1:AC micro-order is coded in the same cycle. For FLAG1=0, addressing is narrow; for FLAG1=1, addressing is wide.
Flag 2	Flag 2 indicates operand precision (single or double) for operations of the floating-point unit. For FLAG2=0, operands are single precision; for FLAG2=1, operands are double precision.
Flag 3	Flag 3 indicates the ALU test width. Test widths from the ALU can be either 32 bits or 16 bits. For FLAG3=0, width is 16 bits; for FLAG3=1, width is 32 bits.

The remaining 4 flags are set to zero at IPOP time; these flags are defined only by their use in the microprogram. All eight flags can be manipulated by the microprogram. The CPDS:USS micro-order gates the flags onto CPD[16-23]- as part of the microsequencer state.

SCP Control

The microcode in WCS is loaded from the System Control Processor. When the system is booted, the SCP scans in microroutines. Only after microcode is loaded is control turned over to the microsequencer.

Tests

The microsequencer can perform actions conditional on the outcome of various tests. There are 64 tests that can be specified by the NAC:TSEL field. These tests fall into four categories:

- Microsequencer tests, including microstack empty and flag tests.
- ATU tests
- Integer ALU tests
- Floating-Point tests

In addition, the TSEL polarity bit can invert the test result, thus altering its interpretation. The test result determines which action is performed by a micro-order in the NAC:COP field.

The Integer ALU

The Integer ALU (IALU) performs arithmetic and logic operations on 16-bit and 32-bit integers. It consists of a register file, an ALU, a hex shifter, a bit shifter, a scratch-pad file, and miscellaneous additional logic. Figure 2-5 shows the integer ALU.

Narrow and Wide Operations

The Integer ALU can perform either 16-bit or 32-bit arithmetic. Width is determined by FLAG0 (for bus widths) and FLAG3 (for test widths). The Instruction Processor automatically sets these flags when it is decoding a macroinstruction.

When FLAG0=1, the IY Bus in the ALU is effectively a 32-bit bus. When FLAG0=0, the IY Bus is effectively a 16-bit bus. Values that are sourced onto the bus when FLAG0=0 will go into the least-significant 16 bits (IY[16-31]). For FLAG0=0, data from the ALU and bit shifter will be sign-extended if written to the IALU's or Address Generator's register files, or to the Scratch Pad (SPAD); other destinations will be one-filled in their most-significant bits. Data from the hex-shifter always contains zeros in the most-significant bits, regardless of destination.

When FLAG3=1, Integer ALU tests apply to 32-bit quantities; when FLAG3=0, the tests apply to 16-bit quantities. The exact effect of this on any given test is explained in Chapter 3 under the individual test explanations.

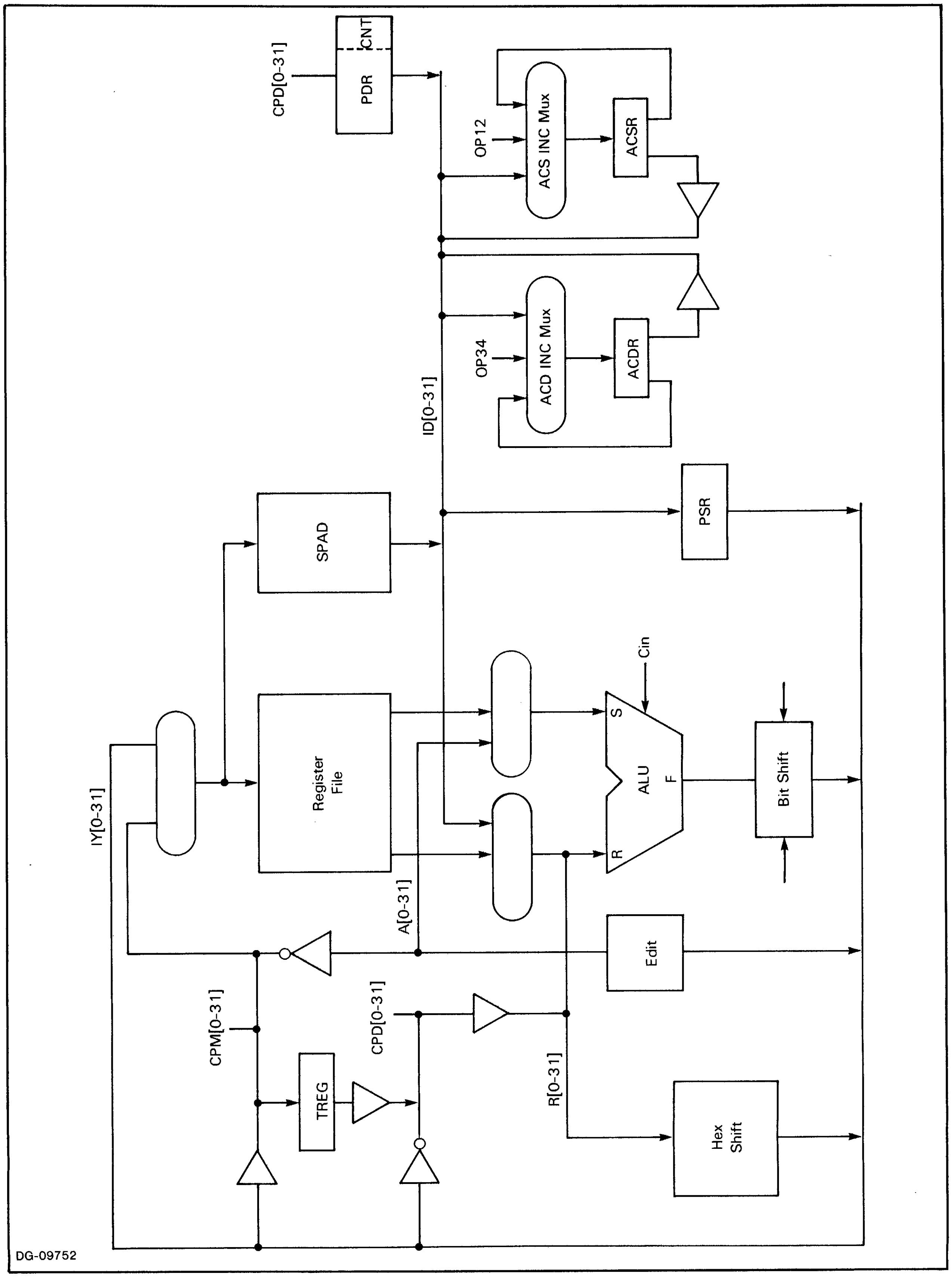


Figure 2-5. Integer ALU

Integer Register File

The Integer ALU general register file consists of sixteen 32-bit registers with two separately addressable output ports and a single input port. The register file input is through a multiplexer that selects either the CPM Bus or the IY Bus. The output from the register file is to the A Bus and the ID Bus. Figure 2-6 shows the register file.

By convention, registers in the file are assigned particular meanings as follows:

Register	Meaning
0	Programmer-visible Accumulator 0 (must contain same value as AG reg. 0 at IPOP)
1	Programmer-visible Accumulator 1 (must contain same value as AG reg. 1 at IPOP)
2	Programmer-visible Accumulator 2 (must contain the same value as AG reg. 2 at IPOP)
3	Programmer-visible Accumulator 3 (must contain the same value as AG reg. 3 at IPOP)
4	Wide frame pointer
5	Wide stack limit
6	Wide stack base
7	Constant (-1)
8	Microprogram general register
9	Microprogram general register
10	Microprogram general register
11	Microprogram general register
12	Microprogram general register
13	Microprogram general register
14	Register addressed by ACSR
15	Register addressed by ACDR

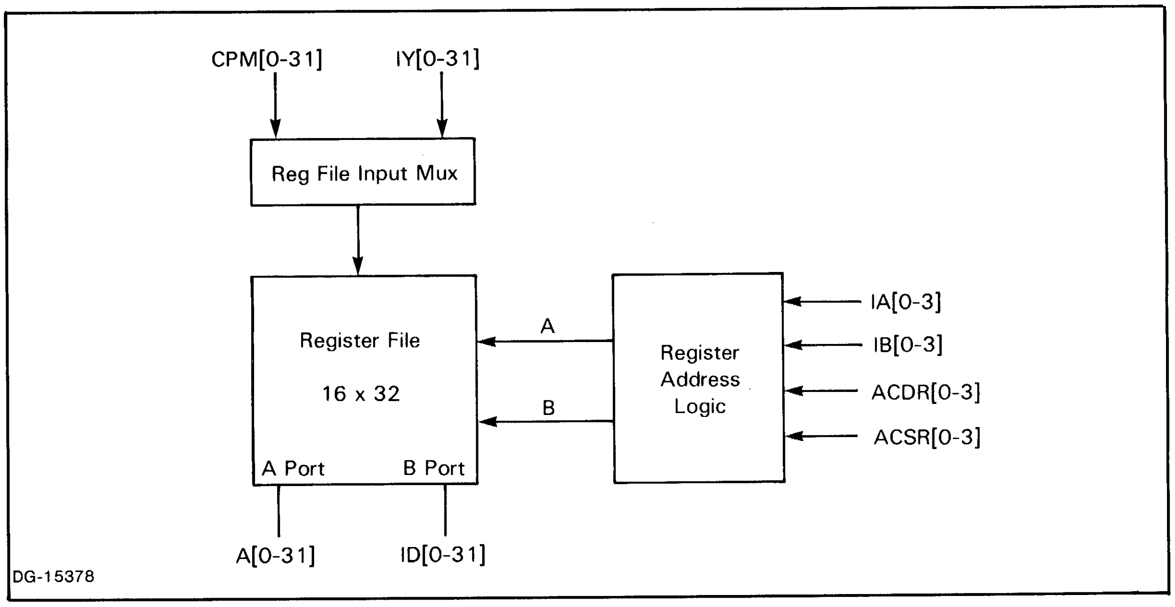


Figure 2-6. Integer ALU Register File

The IA and IB fields of the microinstruction supply addresses for the A and B output ports. The following table shows the relationship between values in the IA and IB fields and the registers addressed:

IA or IB	Register Addressed
=<D	The field directly addresses the register file.
=E	The Accumulator Source Register (ACSR) addresses the register file.
=F	The Accumulator Destination Register (ACDR) addresses the register file.

The IB address field supplies addresses for the input port of the integer register file. The CPM Bus and the IY Bus source the register file through a multiplexer. This multiplexer is controlled by the IL field of the microinstruction.

Registers on the ID Bus

Figure 2-7 shows the registers that source the ID Bus. Data from these registers can go to either input of the ALU or to the Processor Status Register. The ACDR and ACSR registers can both source and sink the bus. The B output port of the Integer ALU Register File (ALU BREG) is another source, described above. The Scratch Pad (SPAD) can also source the ID Bus; it is described in the section "Scratch Pad," below. PDR is a 32-bit register that transfers data between the CPD and ID Bus. It is described in "CPD Bus Register—PDR," below.

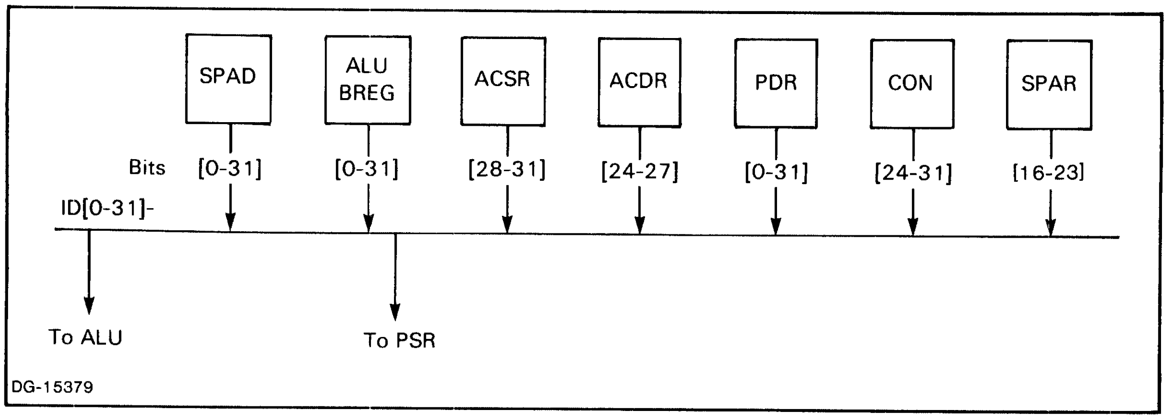


Figure 2-7. Registers on the ID Bus

ACSR and ACDR Registers

The Accumulator Source and Destination Registers (ACSR and ACDR) are 4-bit registers that address the integer register file. They generally contain the values from the ACS and ACD fields of a macroinstruction. The registers can be incremented, decremented, or loaded. The RAND:GEN:REG0 field controls these registers.

CON and SPAR registers

The constant (CON) and Scratch Pad Address Register (SPAR) are 8-bit registers that address locations in the scratch pad (see below). CON contains the value in the CNST field of the microinstruction. SPAR can be loaded from the IY Bus or from CON. In addition, SPAR can be loaded with a hardware-generated address that indexes into bit masks kept in the first thirty-two scratch-pad locations. Besides addressing the scratch pad, both CON and SPAR can source the ID Bus.

Scratch Pad

The scratch pad (SPAD) consists of 256 thirty-two-bit registers. It stores temporary values and constants used in various microcode routines. Appendix H lists the scratch pad constants. Figure 2-8 shows the scratch pad and its addressing logic.

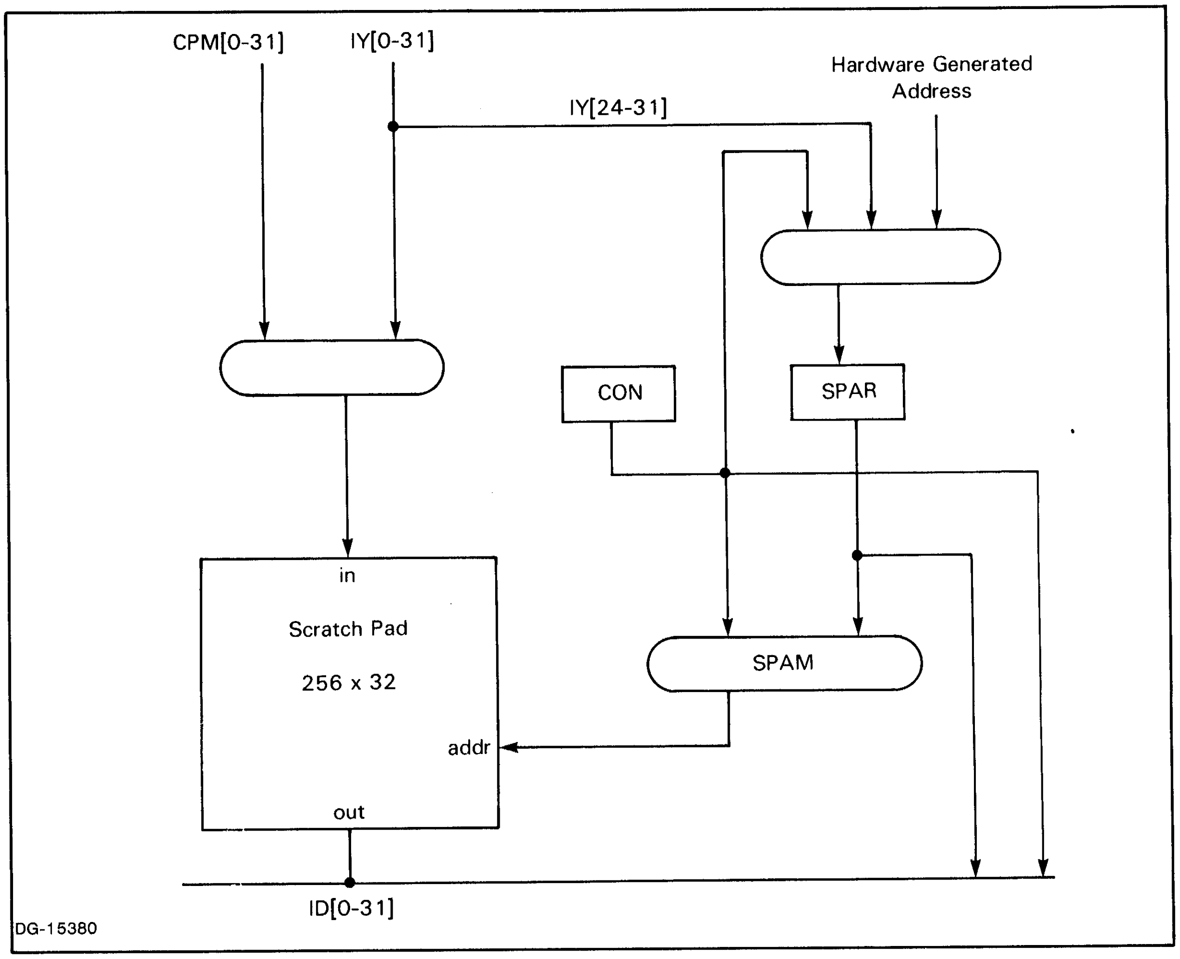


Figure 2-8. Scratch Pad

Data is written into the scratch pad from the IY Bus or the CPM Bus; the scratch pad sources data to the ID Bus. The scratch pad can be addressed by the CNST field of the microinstruction or by the scratch pad address register (SPAR). Micro-orders in the RAND:GEN:REG0, SPAD, and ID fields control the scratch pad.

Special logic generates the scratch pad address for the WSKBO and WSKBZ instructions. The address is constructed from the macro instruction register and loaded into SPAR as follows:

```
0,0,0,IR[1-3],IR[10-11]
```

The table of bit masks for WSKBO and WSKBZ resides in the first thirty-two scratch pad locations. The constructed address indexes into the scratch pad for the proper mask.

Transfer Register

The transfer register (TREG) moves data from the CPM Bus to the CPD Bus. Figure 2-9 shows the position of the TREG.

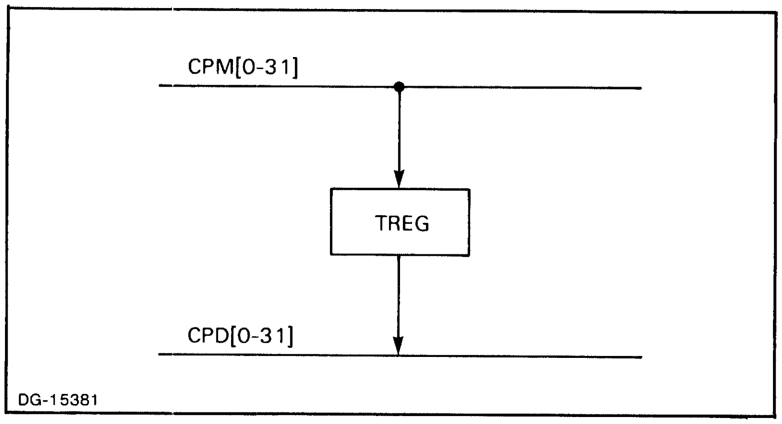


Figure 2-9. The Transfer Register

TREG is controlled by the LT micro-order, which can be used in the RAND FIX:LOAD, ATU:ATU1, or GEN:REG1 fields, and the CPDS:TRG micro-order.

Hex Shifter

The hex shifter can shift left, shift right, or rotate 32 bits in 4-bit increments. A shifted number is zero filled. In addition, the hex shifter can sign-extend words and bytes, and can zero- extend words. Figure 2-10 shows the hex shifter. The size of the shift is controlled by the IOP field of the microword, and so, generally, you cannot use the ALU and the shifter simultaneously.

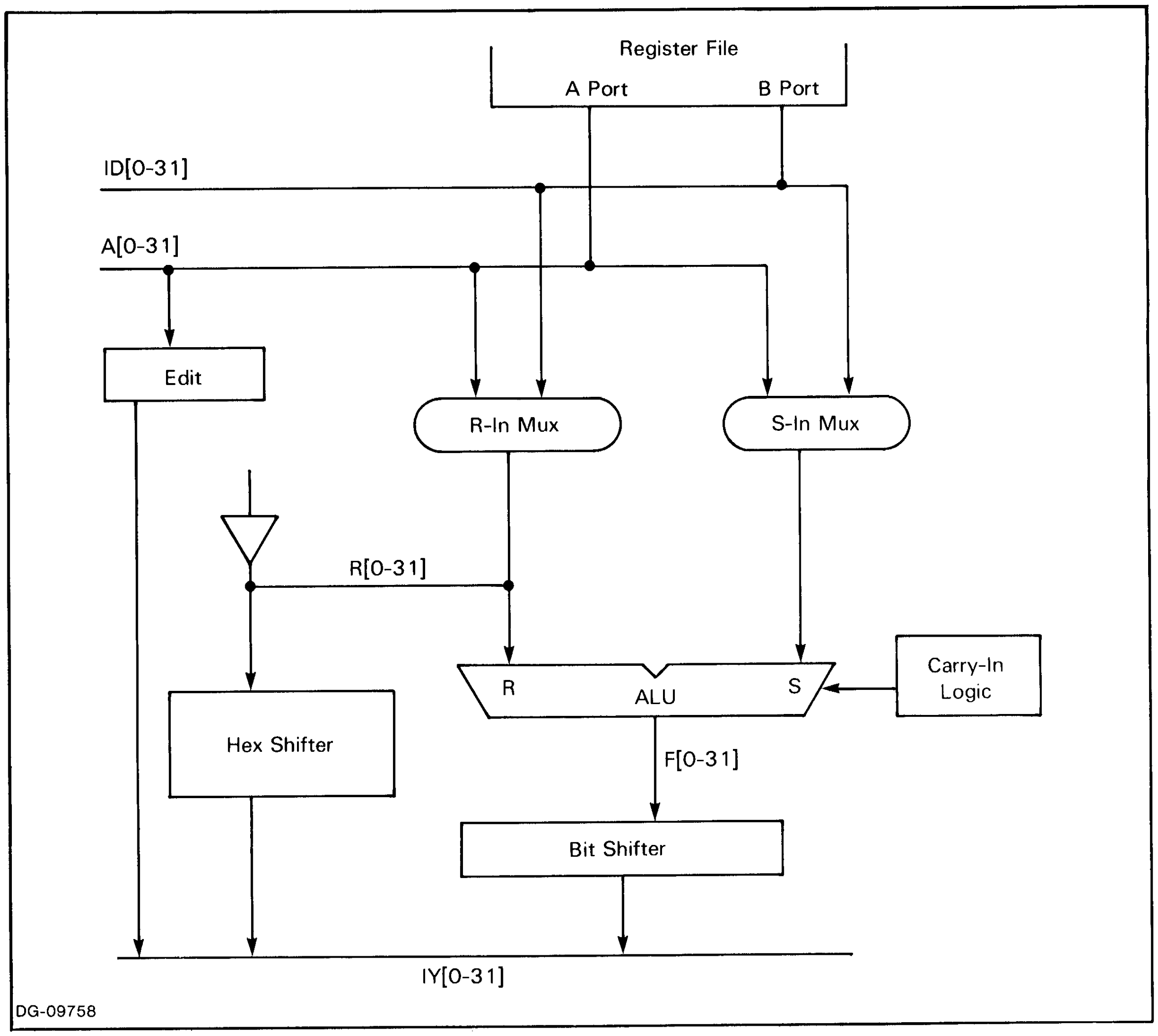


Figure 2-10. The Integer ALU and Associated Logic

ALU

The ALU performs the arithmetic and logical functions for the integer ALU section of the CPU. Figure 2-10 shows the ALU and its relationship to other logic. The inputs to the ALU are through the R-in and S-in multiplexers, which can take data from the A and ID buses. They are controlled by the RS field of the microinstruction. The R input can also take data from the CPD Bus. In addition the ALU has a carry-in bit for arithmetic functions. The output of the ALU goes through the bit shifter to the IY Bus.

The IOP field controls the R and S inputs and the polarity of the carry-in input to the ALU. The ALU can perform the following functions:

- R *AND* S
- R *OR* S
- R *AND* S'

- R *XOR* S
- R' + S + CIB'
- R + S + CIB'
- R' + S + CIB
- R + S + CIB

Note that the IOP field also specifies the hex-shifter count when HL0, HR0, or HRT is coded in the IY field.

Carry-In Logic

The carry-in logic for the ALU determines the value of the carry-in base (CIB). The CIB is normally a zero. As can be seen in the preceding section, the arithmetic operations allow selection of either polarity for the CIB. The fixed-point mode randoms also allow the CARRY bit to be selected as the CIB. This facilitates multiword arithmetic operations. Figure 2-11 shows the carry-in logic.

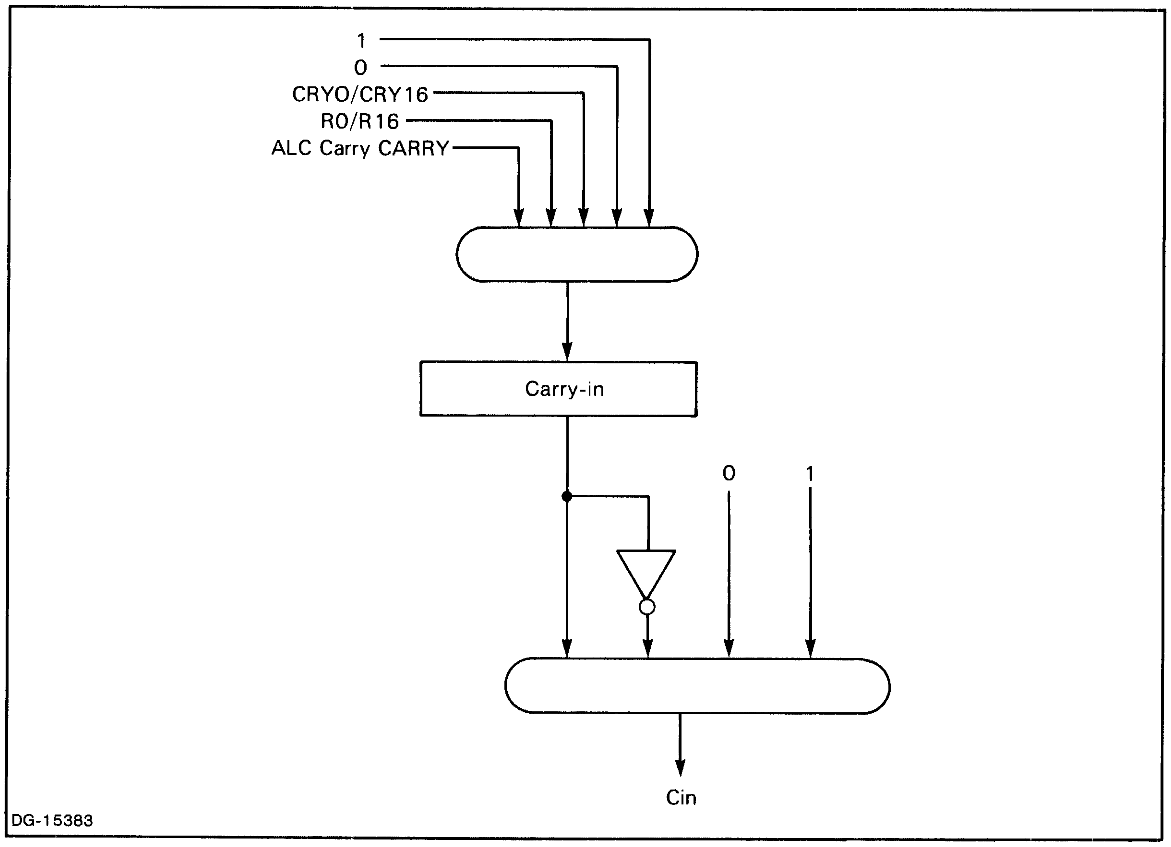


Figure 2-11. CARRY Bit and Carry-In Logic

The CIB can be either the CARRY register or zero, depending on the RAND mode. Table 2-1 shows the CIB that goes with each RAND mode.

Table 2-1. RAND Mode and CIB

RAND Mode	CIB
GEN	0
ATU	0
FIX (XC)	CARRY
FIX (XZ)	0
FLT	0

The polarity of the CIB is controlled by the IOP field. See "ALU," above, for the possible CIB polarities.

The CARRY bit may be set as follows:

1) One—CARRY is set to one.

2) Zero—CARRY is set to zero.

3) CRY0/CRY16—CARRY is set to the carry-out from the ALU; depending on whether the operation is wide (FLAG3=1) or narrow (FLAG3=0), the carry-out will be for a 32-bit result (CRY0) or a 16-bit result (CRY16).

4) R Bus—CARRY is set to bit 0 or bit 16 of the R Bus (the output of the R-input multiplexer—see Figure 2-10) depending on whether the operation is wide or narrow (FLAG3=1 or 0).

5) ALC carry—CARRY is set according to an ALC macroinstruction. The carry is dependent on IR[10-11], which specify the carry operation; on IR[5-7], which specify the macroinstruction function; and IR[8-9], which specify the shift function and its effect on the carry.

The CARRY register is controlled by the RAND:<XC XZ>:COVS field of the microinstruction.

Commercial Test and Edit PROMs

The test and edit PROMs test the validity of the least-significant byte on the A Bus as commercial data. The PROMs are enabled by the TSEL:COM1 and TSEL:COM2 micro-orders. The test is specified by a code in the CNST field; the test result may be used by the microsequencer like any other test result. In addition, the PROMs translate commercial data to BCD, which is sourced to the IY Bus and may be accessed by the IY:EDT micro-order.

The same PROMs also generate the IOT test functions. The TSEL:IOT micro-order enables these functions. Micro-orders in the CNST field specify the tests.

Bit Shifter

The bit shifter can pass 32-bit or 16-bit data, swap the two least-significant bytes of data, or shift the output of the ALU to the IY Bus one bit right or left for 32-bit or 16-bit data. The bit shifter is shown in Figure 2-10. The shift can be either zero or one filled, depending on the micro-order in the IY field. In addition, if RAND:<XC XZ>:COVS:ALC is coded, the input bit to the shift will be forced to the ALC carry, which is determined from the macroinstruction (see "Carry-in Logic," above). For narrow operations (FLAG0=0), the most-significant sixteen bits on the IY Bus are not used. They are filled with ones, except when the result goes to the Address Generator, SPAD, or the register file. In these cases, the 16-bit result (IY[16-31]) is sign extended into [0-15] by the input multiplexers of those destinations.

Processor Status Register

The Processor Status Register (PSR) contains information about the state of the MV/10000 processor. Table 2-2 shows the bits in the PSR.

Table 2-2. Processor Status Register

Bit	Name	Description
0	OVK	Overflow mask
1	OVR	Fixed-point overflow indicator
2	IRES	Interrupt resume
3	IXCT	The interrupted instruction was Executed via XCT or PBX.
4-15	—	Reserved

The overflow mask indicates whether an overflow trap may occur; when OVK is zero, the trap is disabled. The overflow indicator tells whether a fixed-point overflow has occurred. Note that unless both OVR and OVK are set, overflow faults cannot happen.

The RAND:COVS field controls the OVK and OVR status bits. If a micro-order is coded to update OVR (from the current ALU computation) during an IPOP cycle, and the result produces an overflow error, a microtrap is always generated. The trap microcode examines the OVK bit to determine whether the fault should also be serviced at the macro level.

The IRES bit is used by the interrupt routines for resumable instructions. During an interrupt, resumable instructions save state on the user stack, and must restore it when the interrupt is over. When an instruction is interrupted, IRES is set. The microroutine for a resumable instruction must test IRES before it begins to execute. From IRES, it determines whether it has just started or is being resumed.

If the instruction can resume at more than one place, assign a number to each entry and use that number to determine the proper entry point. To protect the system, do not push a microaddress onto the user stack.

The IXCT bit is used by interrupt routines. It indicates that the executing instruction was inserted into the I-stream by the CPU. The instruction was originally in an accumulator and was executed as a result of an XCT or PBX instruction.

The PSR is controlled by the RAND:<XZ or XC>:COVS field.

CPD Bus Register—PDR

PDR is a 32-bit register that transfers data between the CPD Bus and ID Bus. In addition, it can act as a counter. It can be read using the ID:PD micro-order.

The least-significant eight bits of PDR may be used as a counter. The micro-order TSEL:CNT4 and TSEL:CNT8 will increment the eight-bit counter. CNT4 tests for the four least-significant bits equal to zero; CNT8 tests for all eight bits equal to zero. The count should not be tested until two cycles after the counter has been loaded with its initial value.

Normally, PDR is loaded every time the CPD Bus is active (i.e., whenever the CPDS field is not coded with N). However, loading is suppressed during an LAT routine, a Cache Block Crossing routine, or when the micro-orders RAND:ATU:ATU0:NPDR or RAND:GEN:REG0:NPDR are coded. Loading of PDR is also suppressed when a page fault occurs, up until the time that the ATU state is read.

The Floating-Point Unit

The Floating-Point Unit (FPU) performs all floating-point arithmetic as well as doing integer multiply and divide. The FPU is synchronous with other MV/10000 units. Figure 2-12 shows the floating-point unit.

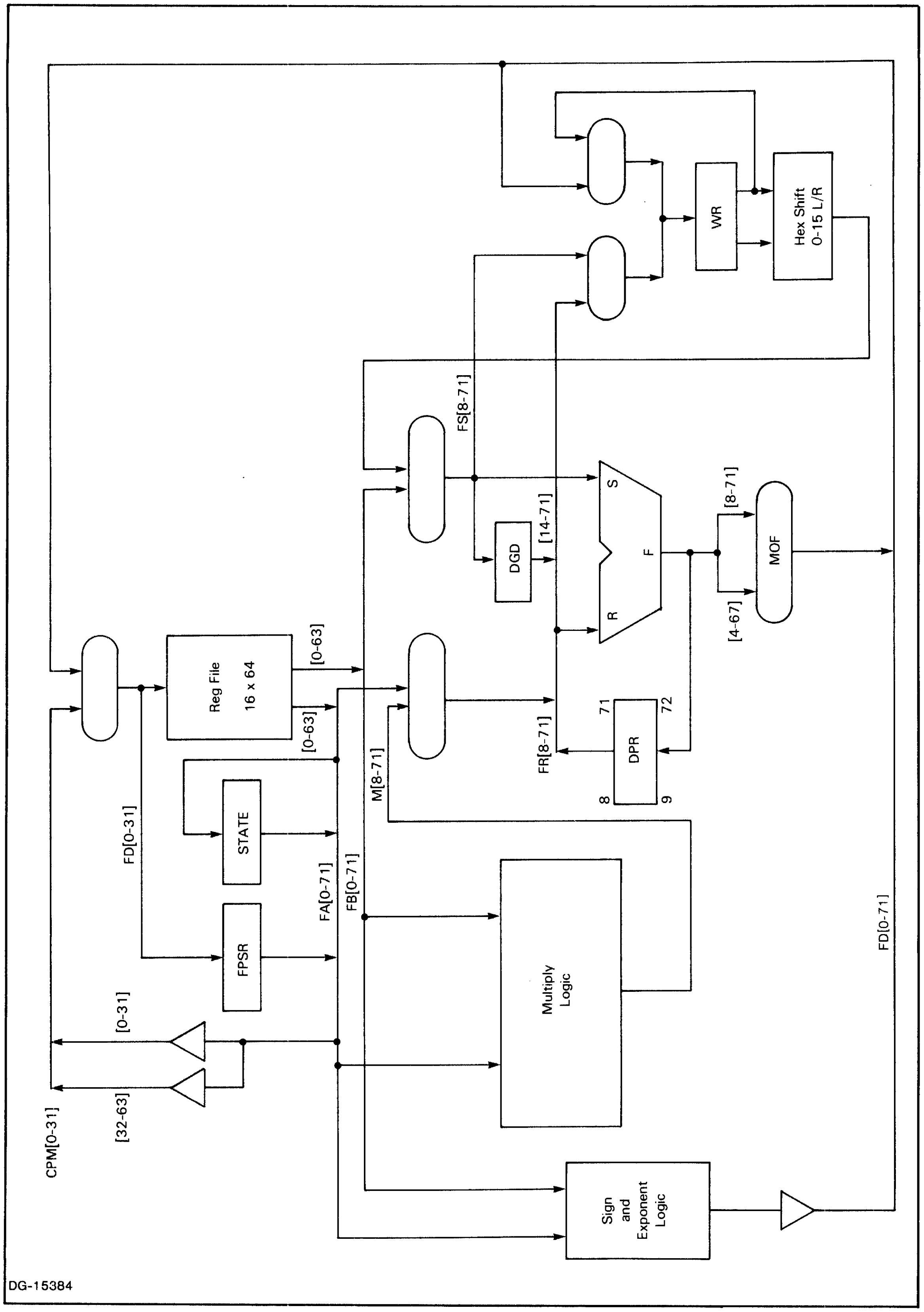


Figure 2-12. The Floating-Point Unit

FPU Buses

FPU internal buses are 72 bits wide. Bits 0-7 form the sign and exponent of the floating-point number and generally go to the sign and exponent sections of the FPU. Bits 8-71 are manipulated by the mantissa section of the FPU. Bits 64-71 are guard bits; they ensure sufficient bits for rounding during all operations. For single-precision arithmetic (FLAG2=0), only the most-significant bits on a bus are used. The effects of this are noted for the individual buses.

The *FA Bus* is sourced by the A output of the floating-point register file and by the Floating-Point STATE Register and the floating-point STATUS register. The bus sources the R input of the mantissa ALU, the X register of the multiply logic, the STATE register, and the R input of the exponent and sign logic. Bits 32-71 are zero for single-precision operations (FLAG2=0) and bits 64-71 are zero for double-precision operations (FLAG2=1).

The *FB Bus* is sourced by the B output of the register file. The bus sources the S input of the Mantissa ALU, the Y register of the multiply logic, and the S input of the exponent and sign logic. When the FB Bus sources the ALU, bits 32-71 of the S input are zero for single-precision operations (FLAG2=0) and bits 64-71 are zero for double-precision operations (FLAG2=1).

The *FD Bus* is sourced by the mantissa ALU (bits 8-71) and the exponent ALU (bits 1-7). It can source the working register and the floating-point register file.

The *FR Bus* is sourced by the Divide Partial Remainder register, the multiply ALU, the FA Bus, the Divide Guard Digit register, and the round logic. It sources the working register and the R input of the mantissa ALU.

The *FS Bus* is sourced by the hex shifter and the FB Bus. It sources the DGD register, the working register, and the S input of the mantissa ALU. Bits 40-71 are zero for single-precision operations (FLAG2=0); bits 64-71 are zero for double precision operations (FLAG2=1) when the FB Bus is the source.

When the working register is the source, bits 40-71 are zero for single-precision operations (FLAG2=0). When the FB Bus is the source, bits 32-71 are zero for single-precision operations and bits 64-71 are zero for double-precision operations.

It is also possible to provide zeros on bits 8-71 of the FS bus for use in passing data on the FR Bus through the ALU.

The *M Bus* sources the FR Bus and is sourced by the multiplier ALU.

Mantissa Logic

The mantissa logic of the FPU performs arithmetic on the mantissas of floating-point numbers and does division and multiplication on integers.

General Logic

Much of the mantissa logic is used by more than one arithmetic algorithm. This includes such things as the register file, various FPU state registers, and the hex shifter.

Floating-Point Register File

The floating-point register file is a 16-word by 64-bit file that holds the operands for floating-point operations and for integer multiply and divide. The register file has two output ports, A and B, which are addressed by the IA and IB fields of the microword. The single input port is addressed by the FCW field.

The A output port sources the FA Bus while the B port sources the FB Bus.

Floating-Point Status Register

The Floating-Point Status Register (FPSR) contains bits that specify floating-point overflow, underflow, divide by zero, or mantissa overflow. Any of these conditions is a floating-point error, but they are handled differently in microcode. The UNF and OVF bits cause a microtrap to a handler when they are updated by FRG:UFS. The DVZ and MOF bits must be set by microcode, which must explicitly test for the presence of the related error conditions. The need for these tests occurs infrequently.

Additional bits perform other functions. The following table describes the bits in the FPSR:

Bits	Name	Description
0	ANY	The value of the logical OR of FPSR[1-4].
1	OVF	An exponent overflow occurred during processing of a floating-point number; the result is correct except that the exponent is 128 too small. This will cause a microtrap when the FPSR is updated by FRG:UFS.
2	UNF	An exponent underflow occurred during processing of a floating-point number; the result is correct except that the exponent is 128 too large. This will cause a microtrap when the FPSR is updated by FRG:UFS.
3	DVZ	Microcode detected a zero divisor during a divide. DVZ is a floating-point error that is detected by microcode, which is responsible for jumping to an error handler.
4	MOF	Microcode detected a mantissa overflow during the FSCAL, FFAS, FFMD or WFFAD instruction. MOF is a floating-point error that is detected by microcode. Microcode is responsible for jumping to an error handler.
5	MOF	This bit indicates mantissa overflow (MOF). While processing a FSCAL instruction, the FPU shifted the mantissa left. During a FFAS, FFMD, or WFFAD instruction, the result contained more than 15 bits for single-word results or 31 bits for double-word results. MOF is a floating-point error that is detected by microcode, which is responsible for jumping to an error handler.

Bits	Name	Description
	TE	If this bit is set (=1), a 1 in any of the FPSR bits 1-4 will result in a floating-point macro trap. Microcode tests for this trap and jumps to the appropriate handler.
6	Z	The result of the last floating-point operation was true zero.
7	N	The result of the last floating-point operation was negative.
8	RND	If this bit is set (=1), then unbiased rounding is used for floating-point operations. If this bit is not set (=0), then truncation is used for floating-point operations.
9	RES	Microcode sets this bit to indicate that an interrupt has occurred during execution of resumable code.
10-11	—	Reserved for future use. Must be zero.
12-15	FPMOD	Floating-point ID code. Hardwired to 0111.

Floating-Point STATE Register

The floating-point STATE register contains various bits that are necessary to restore the Floating-Point Unit after its state has been altered. The following fields are included in the state register:

Bits	Mnemonic	Description
0	SA	Sign register for A operand
1	SB	Sign register for B operand
2	A.EQ.B	Result from a compare operation. The A operand equals the B operand.
3	A.LT.B	Result from a compare operation. The A operand is less than the B operand in magnitude.
4	SWAP	This bit causes the A and B addresses for the register file to be exchanged. It is used after a compare operation to force the larger operand onto the FA Bus. This bit has no effect on write addressing.
5	X.GT.15	Result from a compare operation. The absolute exponent difference exceeds 15.
6	^XEWR	The most significant bit of the exponent working register (EWR). This bit is an inverted extension bit.
7	^ER0	The second most significant bit of the exponent R Bus. This will be the EWR0 if RAND:FLT:EXP:N is coded.
8-11	YSEL[0-3]	The contents of the byte-selection counter (see below).
12-15	^MAG[0-3]	The contents of the hex-shifter magnitude register (see below).

Hex Shifter

The hex shifter can shift the output of the working register (WR) either left or right by up to 8 bytes in four-bit increments. The shifter sources the FS Bus through a multiplexer controlled by the FS field of the microword. Micro-orders in this field also designate whether the shift is left or right. When the shifter is the FS Bus source, the magnitude of the shift is determined by the MAG portion of the floating-point STATE register.

MAG Register

The MAG register controls the hex shifter and provides a value that can be arithmetically manipulated by the exponent ALU. MAG is part of the floating-point STATE register (bits 12-15). The MAG register is controlled by the RAND:FLT:SCNT field. Figure 2-13 shows the MAG register.

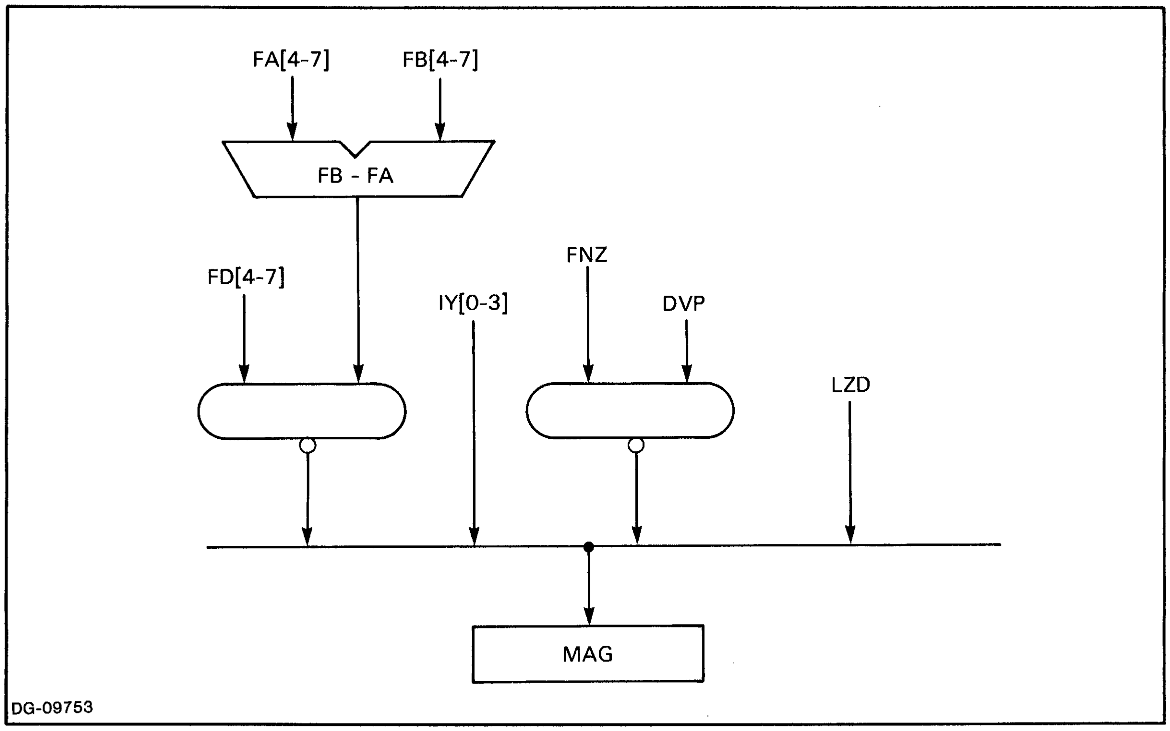


Figure 2-13. The MAG Register Sources

The MAG register is sourced by the following:

A) The Leading Zero Detector (LZD)—The LZD determines the number of leading hexadecimal zeros in the mantissa and places the result in MAG. MAG can then be used to left-shift the mantissa and to adjust the exponent in order to normalize the number. Note that on the first cycle the LZD gives a result for only the first two hexadecimal digits in the mantissa. If the first three digits are zero, the full result must be used on the next cycle to provide the correct value for MAG. If the first two hexadecimal digits are not both zero, then all leading zeros have been detected and the initial result is correct. Hardware resolves the correct MAG value, invisibly to microcode, except that RAND:FLT:SCNT:LZD must be coded twice (see Chapter 3).

B) The value of bits 4-7 of the EF Bus (the output of the exponent ALU).

C) The absolute value of the difference between two exponents.

D) The value of bits 12-15 of the FA Bus.

E) The IY field—The value in the IY field of the microword can source the MAG register.

F) First Nibble Zero logic—If the first nibble of the mantissa is 0, MAG is set to -1; otherwise, MAG is set to 0.

G) Divide Prescale logic—MAG is set to 1 if there was a carry-out from the mantissa; otherwise, MAG is set to 0.

Mantissa ALU

The mantissa ALU is used for all floating-point operations, as well as integer multiplication and division. It adds and subtracts mantissas after they have been aligned. During multiplication, it sums the partial products produced by the multiply ALU. During division, it calculates the quotient by adding or subtracting the divisor from the partial remainder.

Working Register

The working register holds the quotient in divide operations and the partial product in multiply operations. For addition and subtraction prescaling, it holds the smaller of the two operands. It also holds the unnormalized result of all operations that use rounding and normalization. The FR Bus, FS Bus, and FDI Bus can all source the working register. During division, the working register performs 1-bit left shifts and gets its least-significant bit from the Q-bit. The Q-bit is an extension to the mantissa ALU and is derived from the mantissa carry, the ALU operation, and the most significant bit of the ALU output. The hex shifter can left-shift or right-shift the output of the working register.

Multiply Hardware

FPU multiplication uses the multiply ALU, the mantissa ALU, the X and Y registers, and the working register. The operands are placed in the X and Y registers. The X register is multiplied by a single byte of the Y register; that byte is selected by the YSEL counter. Each multiplication by the multiply ALU produces a partial product that is added to the accumulated partial product in the working register. Figure 2-14 shows the mantissa hardware used in multiplication.

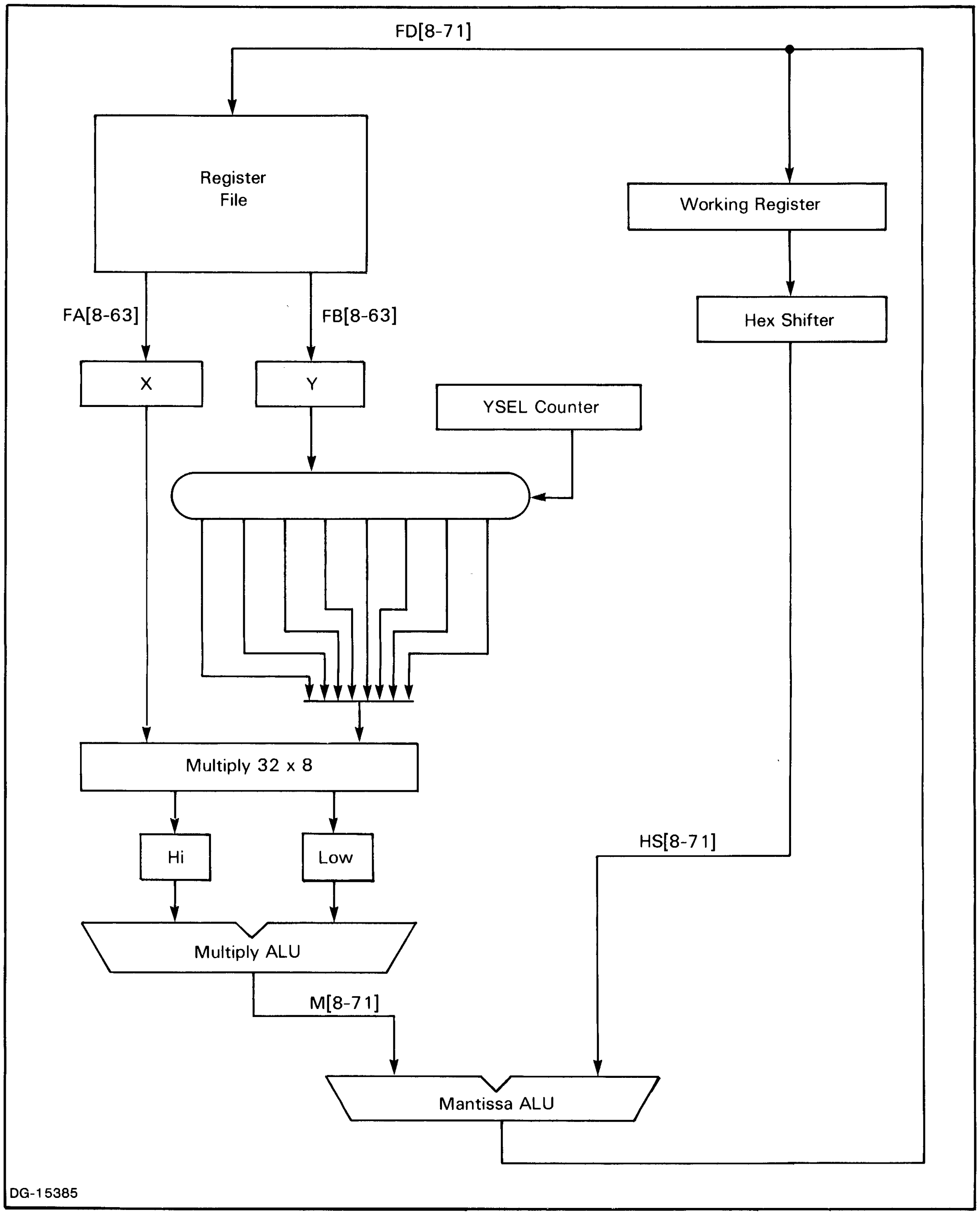


Figure 2-14. Multiply Data Paths

X and Y Registers

The X and Y registers hold the multiplicand and the multiplier, respectively, for integer and mantissa multiplication. The Y register outputs a single byte at a time, selected by the YSEL counter.

YSEL Counter

The YSEL counter is a 4-bit counter that designates which byte of the Y (multiplier) register forms the current partial product. This counter is controlled by the FRG field of the microword, and loaded from the IY field. Note that YSEL is part of the floating-point STATE register (bits 8-11).

Multiply ALU

The multiply ALU multiplies the 56-bit X register by 8 bits of the Y register. This operation forms a 64-bit partial product. Partial products are accumulated in the working register to form the final product.

Divide Hardware

FPU division uses the mantissa ALU, the Divide Partial Remainder (DPR) register, the working register, and the Divide Guard Digit (DGD) register. The DGD register preserves the guard digits of the prescaled dividend when that value must be temporarily written back to the register file (which has no guard-digit storage).

The floating-point ALU implements a nonrestoring division algorithm. The hardware performs the following functions:

- The working register shifts in the quotient one bit at a time.
- The DPR register stores the remainder, left shifted by one bit.
- The mantissa ALU adds or subtracts the divisor from the remainder.
- The register file stores the divisor. During division, the FCW and IB fields must address the divisor in order for the register outputs to be stable. Register file port B sources the divisor to the S side of the mantissa ALU.

Figure 2-15 shows the divide data paths in the FPU. Note that two division cycles are performed for each microinstruction cycle.

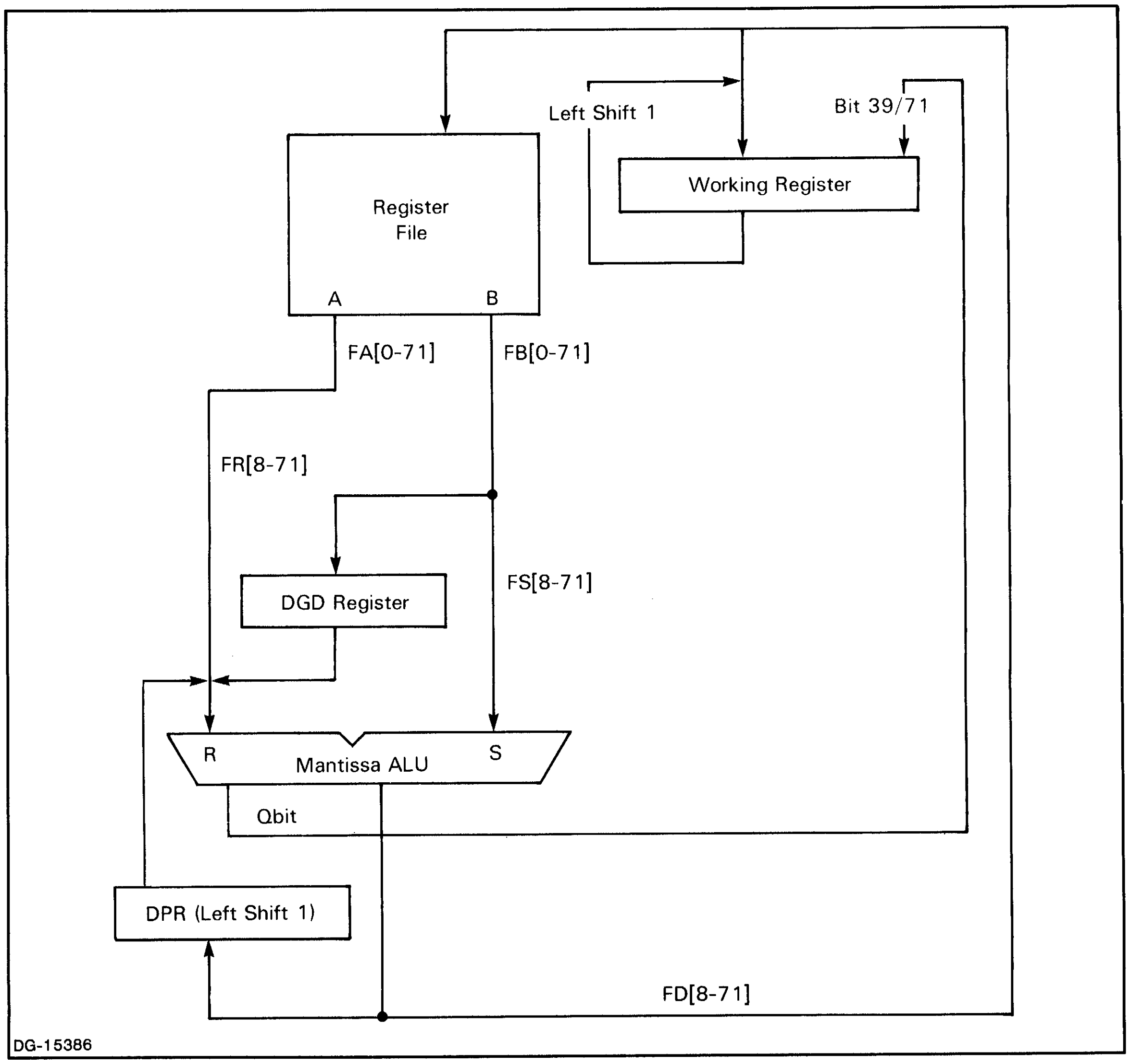


Figure 2-15. Divide Data Paths

Divide Guard Digit Register

The Divide Guard Digit (DGD) register holds the least-significant 8 bits of the dividend at the beginning of a divide operation. The most-significant bits are in a register in the register file. The DGD register and the A port of the register file together source the entire dividend onto the FR Bus. The divisor is subtracted from the dividend during the initial division cycle, and the result is loaded into the Divide Partial Remainder (DPR) register.

Divide Partial Remainder Register

The Divide Partial Remainder (DPR[8-71]) register holds the intermediate dividend during a divide operation. The DPR is sourced by the mantissa ALU and sources the R side of that ALU. When a value is passed through the DPR, it is left-shifted by one bit and the least-significant digit is zero filled.

Sign and Exponent Logic

The sign and exponent logic determines the signs of the results of arithmetic operations and computes the exponents of floating-point numbers. Figure 2-16 shows the sign and exponent logic.

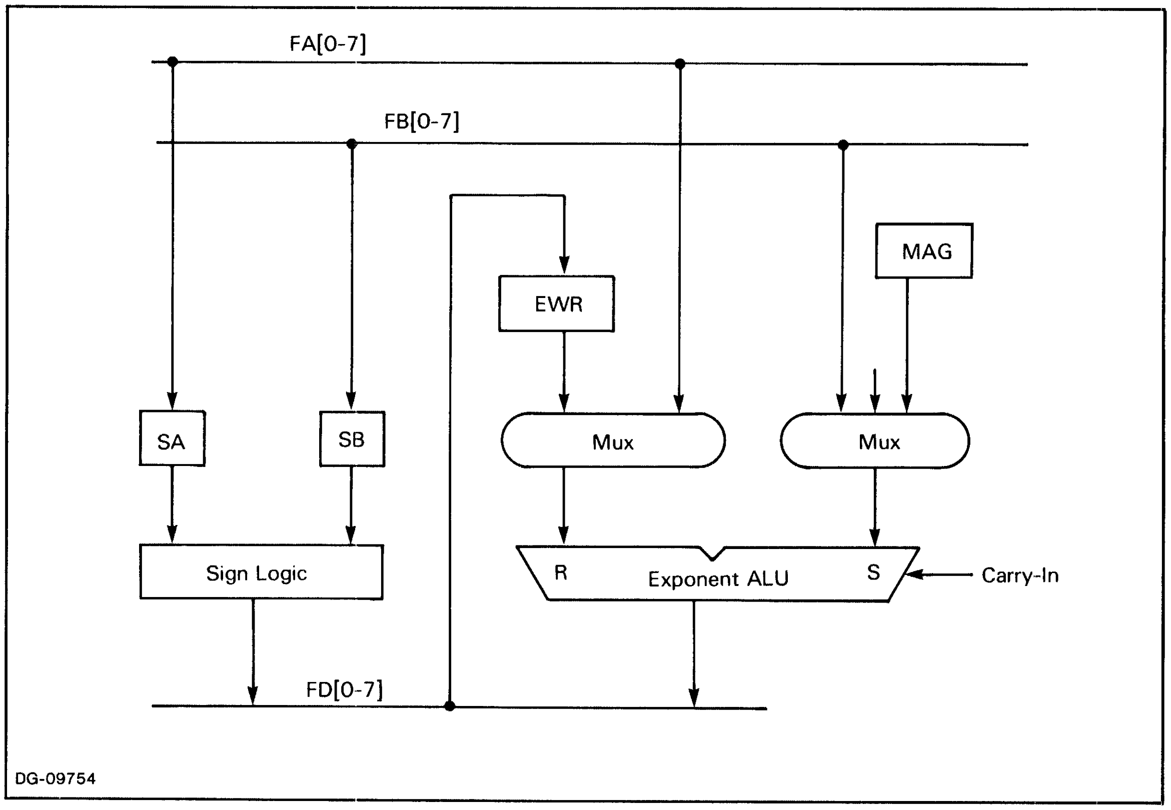


Figure 2-16. Sign and Exponent Logic

The operands for any operation are available on the FA and FB Buses, which also source the sign registers (SA and SB). The sign logic uses the values in these registers to determine the sign of the result. The RAND:FLT:SGN field of the microword controls the sign logic.

Exponents in MV/10000 floating-point numbers are in excess-64 form. If exponents are added or subtracted, as in multiply or divide, the excess-64 form must be restored by subtracting or adding 64 to the result. These operations are performed by the RAND:FLT:EXP micro-orders A64 and S64 in conjunction with the FX:X64 micro-order.

Exponent Working Register

The Exponent Working Register (EWR) provides temporary storage for exponent values. The EWR is one of the possible inputs to the exponent ALU; the MAG register can be added to or subtracted from the EWR.

Exponent ALU

The exponent ALU manipulates the exponents of floating-point numbers. The R side of the ALU accepts input from either the EWR or the FA Bus; the inputs to the S side can be the FB Bus, the MAG register, or zero. The exponent ALU can add its inputs or subtract its S input from its R input. Output from the ALU is to the FD Bus and the EWR. An exponent value can be stored in the EWR for use in a future cycle. In addition to the R and S inputs, the ALU can add 1 to the exponent to correct for a carry-out (MOF) from the mantissa that is adjusted by a right shift.

SA and SB Registers

The SA and SB registers are single-bit registers that hold the signs of the current operands. They are sourced by the most-significant bits of the FA and FB Buses. In turn, they source the sign logic.

Sign Logic

The sign logic determines the sign of the result of an FPU operation. The RAND:FLT:SGN field determines how the sign is computed.

The Address Generator

The Address Generator (AG) provides logical addresses, which can then be transformed into physical addresses by the Address Translation Unit (ATU). The AG input comes from either the IP via the DISP bus, or from main memory via the CPM bus. The AG can source data to the CPM bus, the CPD bus, the LA bus, and the DISP bus. Figure 2-17 shows the Address Generator.

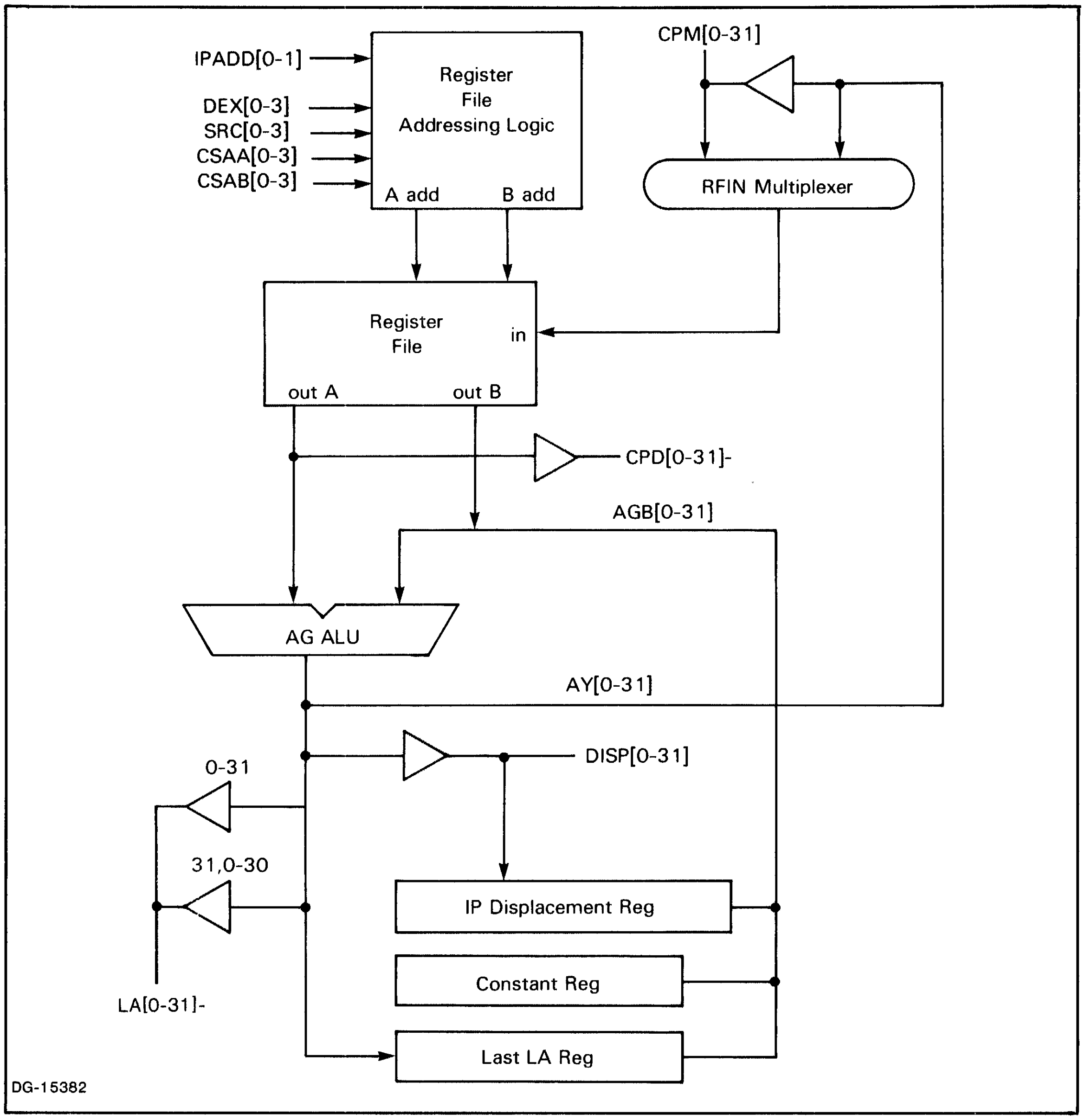


Figure 2-17. The Address Generator

Buses

The *DISP bus* is a 32-bit, bidirectional bus that carries data between the Instruction Processor (IP) and the AG. The IP ALU sources the DISP bus, while both the IPPC register and the ICP register take data off the bus. The IP normally drives the DISP bus. It uses the bus to provide instruction displacements (either addressing offsets or immediate data) to the AG. If an address calculation is specified as PC relative, the IP will add the PC to the address offset before sending it to the AG. The AG ALU can source the DISP bus with the AY bus during branches so the IP can update the IPPC and ICP registers.

The *Logical Address (LA) bus* carries logical addresses from the AG to the Address Translation Unit (ATU). FLAG1 determines the width of the LA bus. If FLAG1=0, the LA bus is narrow and the AG drives the 15 least-significant bits and LA0; the Address Translation Unit will supply the current ring bits (LA[1-3]) from CRE[1-3]. If FLAG1=1, the

LA bus is wide and the AG drives all the bits on the LA bus, except when RAND:ATU:ATU1:AC is coded.

The LA bus is sourced by the AG ALU via the AY bus. Two separate drivers can drive AY[0-31] or AGB[31],AY[0-30] onto the LA bus. Unless overridden by the RAND:ATU:ATU0 micro-orders BYTE and WORD, the type of memory start determines which bits go onto the bus: word and double word addresses use AY[0-31]; byte addresses use AGB[31],AY[0-30].

The *AY bus* is a 32-bit, internal AG bus. It is sourced by the AG ALU and transfers data to:

A) the last Logical Address (LA) register,

B) the Logical Address (LA) bus,

C) the IP DISP bus,

D) the AG register file (through the RFIN multiplexer), and

E) the CPM bus.

The *Address Generator Bus* (AGB) is a 32-bit, internal AG bus that drives the B input to the AG ALU. The AGB is sourced by:

A) the Displacement (DISP) register,

B) the B output port of the AG register file,

C) the last LA register, and

D) the 8-bit constant register (with sign fill).

Register File

The AG register file has sixteen 32-bit registers, a single input port, and two output ports. The output ports, A and B, are separately addressable. The A output goes to the A input of the AG ALU and also drives the CPD bus; the B output goes to the AGB bus. Input to the register file is through a multiplexer that selects either the AY bus or the CPM bus.

By convention, registers in the file are assigned particular meanings as follows:

Register	Meaning
0	Identical to macroprogram accumulator 0 at IPOP (must contain same value as ALU reg. 0 at IPOP)
1	Identical to macroprogram accumulator 1 at IPOP (must contain same value as ALU reg. 1 at IPOP)
2	Identical to macroprogram accumulator 2 at IPOP (must contain the same value as ALU reg. 2 at IPOP)
3	Identical to macroprogram accumulator 3 at IPOP (must contain the same value as ALU reg. 3 at IPOP)
4	Wide stack pointer
5	Constant (=1)
6	Constant (=2)
7	Reserved register for Long Address Translation (LAT)
8	Microprogram general register
9	Microprogram general register
10	Microprogram general register
11	Microprogram general register
12	Microprogram general register
13	Microprogram general register
14	Register addressed by ACSR
15	Register addressed by ACDR

Register File Addressing

The register file addressing logic controls the selection of particular registers within the file. The logic produces separate addresses for the A and B output ports; the B address is also the address for the input port. The addresses for the register file can be supplied from fields in the microword, fields in the macroword, or registers in the integer ALU.

When the AG is not performing effective address (EFA) calculations, the AA (CSAA[0-3]) and AB (CSAB[0-3]) fields of the microword determine which registers will be addressed. For AA or AB from 0 to D, the value in the field directly specifies the register in the file. For AA or AB equal to E, the register is specified by the ACSR register (SRC[0-3]) of the integer ALU; for AA or AB equal to F, the file address is specified by the ACDR register (DES[0-3]) of the integer ALU.

When the AG is performing EFA calculations, the A output port addressing and the ALU operation are controlled by the index bits from the macroinstruction:

A-Port Address	Address Mode and Operation
00	Absolute macroaddressing—The A port is not used for EFA calculations.
01	PC relative addressing—The A port is not used for EFA calculations.
10	AC2 relative addressing—The A port reads AC2, which is added to the displacement from the AGB bus.
11	AC3 relative addressing—The A port reads AC3, which is added to the displacement from the AGB bus.

AGB Bus Sources

The *AGB bus* sources the B input of the AG ALU. The bus itself has four sources. The AGB field of the microword selects a particular source. The sources are:

- the register file,
- the IP displacement register,
- the constant register with sign extension, and
- the last Logical Address (LA) register.

The *register file* is discussed above.

The *IP displacement* register is sourced by DISP[0-31]. DISP[0-31] is the output of the IP ALU. Usually, the displacement register contains the displacement portion of the next macroinstruction from the Instruction Processor.

The *constant register* is loaded from the CONSTANT field of the microinstruction. The register is loaded on every microinstruction, but when the field is being used for a floating-point instruction, the bits in the register may be meaningless.

The *last logical address register* holds the address used for the most recent memory start. Its input comes from the AY bus. It is loaded automatically on memory starts, except for Long Address Translations (LATs) or Cache Block Crossings. LATs occur when the ATU does not have an encached translation for the logical address from the AG. Thus, on return from an LAT routine, the last logical address register holds the address of the original memory request. Cache block crossings (CBXs) occur when a double word access is performed for a memory address that ends in (octal) 7 (i.e., the eighth and last word in a block).

RFIN Multiplexer

The Register File In (RFIN) multiplexer selects the source of the data that is read into the AG register file. This data can come from the AY bus or the CPM bus. The multiplexer is controlled by the AL field of the microword.

Address Generator ALU

The Address Generator (AG) ALU is 32 bits wide, with A and B inputs and a carry-in. Its A input comes from the register file and its B input comes from the AGB bus. Its output goes to the AY bus. The ALU is controlled by the AOP field of the microword. The following functions are available:

- A + B
- B - A
- Pass the B input through unchanged
- Try to perform an Effective Address (EFA) calculation. If the index bits in the macroinstruction are 00 or 01, the ALU passes the B input (sourced by the displacement register). If the index bits are 10 or 11, the ALU performs A + B, where A is sourced by AG2 or AG3, respectively, and B is sourced by the displacement register.

The Address Translation Unit

The Address Translation Unit (ATU) changes logical addresses from the Address Generator into physical addresses that reference main memory. Because the logical address space is much larger than the physical address space, it is necessary for the ATU to map any logical address into a smaller physical address. Figure 2-18 shows the various parts of the ATU.

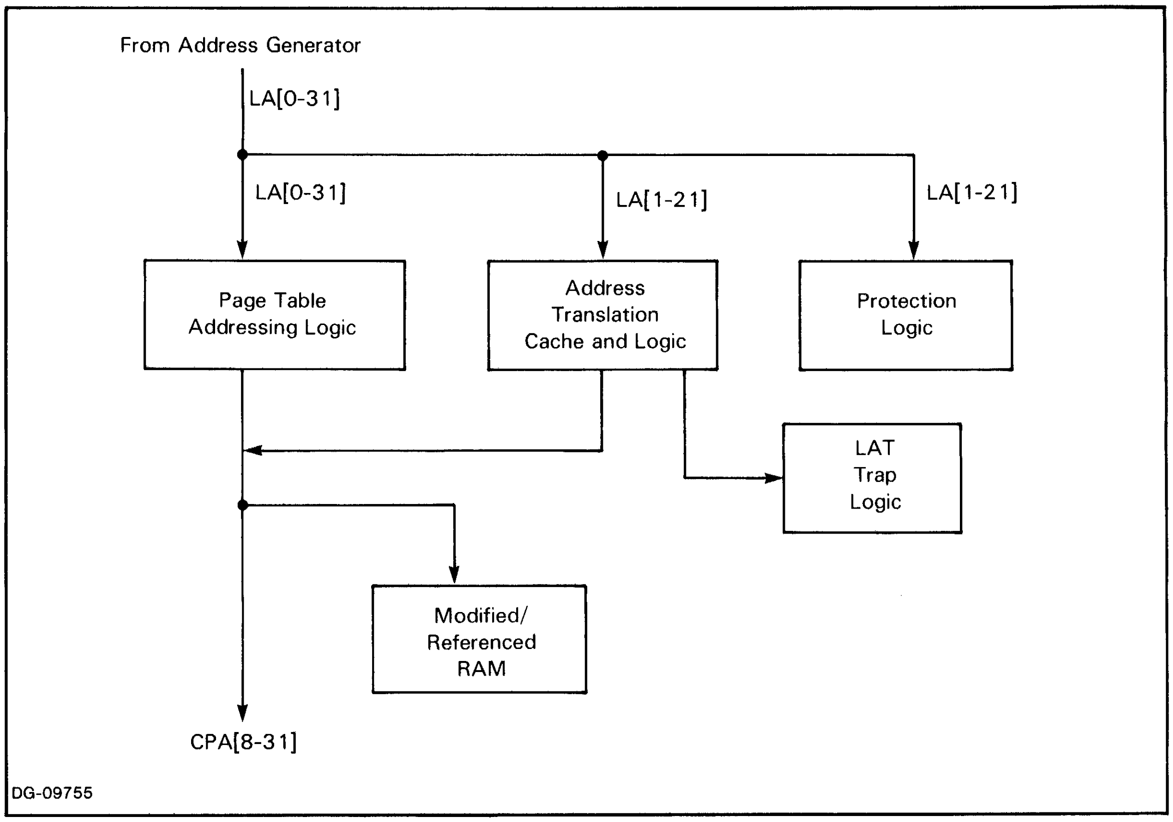


Figure 2-18. The Address Translation Unit

Address mapping is done on a page (2K-byte) basis. Software maintains page tables in physical memory that specify the current mapping of all logical pages. Parts of those tables are also kept in a cache in the ATU. If a mapping is encached, then the ATU can immediately translate a logical address to a physical address. Otherwise, microcode must go to main memory and examine the page tables to determine the correct physical address and to load the cache with a new value.

The ATU also maintains the Segment Base Registers (SBRs). The SBRs registers point to the page tables in main memory. They also contain certain status bits for a segment.

Besides addressing, the ATU also takes care of protection by checking each address for ring maximization, indirection depth, and read/write/execute access. In addition, it maintains the modified/referenced RAM, which keeps track of the current status of physical pages in memory.

Address Translation Cache

The Address Translation Cache (see Figure 2-18) contains 1024 translations of logical to physical addresses. It is addressed by the Logical Address (LA) bus. The cache outputs a 14-bit physical page address. The page address is concatenated with the 10-bit page offset to form a full physical address on the CPU Physical Address (CPA) bus. The physical address goes to the memory system. Microcode can load, flush, and read the cache.

Referenced/Modified RAM

The referenced/modified RAM contains bits for each physical page that indicate whether that particular page has been referenced (read or written) or modified (written). This information is used by the operating system to determine which pages need to be swapped out to secondary memory (i.e., disk) and whether a page can be overwritten.

The referenced and modified bits are part of the ATU state. This state can be read by using the CPDS:ATS micro-order. The referenced and modified bits can be written by the RAND:ATU:ATU0:WRRM micro-order. In addition, referenced bits can be read and reset by the RAND:ATU:ATU0:RSRF micro-order. The specific use of the RSRF micro-order is described in Chapter 3.

Validity RAM

The ATU validity RAM indicates the current state of the address translation cache. There are 1K bits in the RAM: one bit for each translation. Whenever an LAT loads a valid address translation into the cache, the appropriate validity bit is set. The RAND:ATU:ATU0:PRGA resets all the bits in the RAM. If the validity bit for a particular translation is not set, then an LAT routine is necessary to produce the proper logical-to-physical address translation.

Logical Address Translation

If the logical address presented to the ATU is encached, then the ATU can simply place a physical address on the CPA bus and the memory reference can continue. However, if the logical address is not encached, then the ATU causes a Long Address Translation (LAT) trap to occur. The micromachine goes into LAT mode and executes an LAT routine.

The LAT routine aborts the main memory start and inhibits any pending start, constructs the new logical to physical address translation by going to page tables in memory, and then restarts memory the same way that it was started before the LAT routine began. The new translation is stored in the ATU cache.

Page Table Addressing Logic

The ATU has logic that expedites page table addressing. The micro-order RAND:ATU:ATU0:RSBR automatically constructs an address from the current address in the Logical Address Register (LAR). The logic addresses the appropriate SBR (determined by bits 1-3 of the LAR) and concatenates it to bits 4-12 or 13-21 of the LAR. Which set of bits is chosen depends on the level bit (bit 1) of the SBR: for one-level page table addressing, the logic chooses bits 13-21; for two-level addressing, bits 4-12. This address is gated to the CPA bus for an immediate memory reference. Figure 2-19 shows the page table addressing logic.

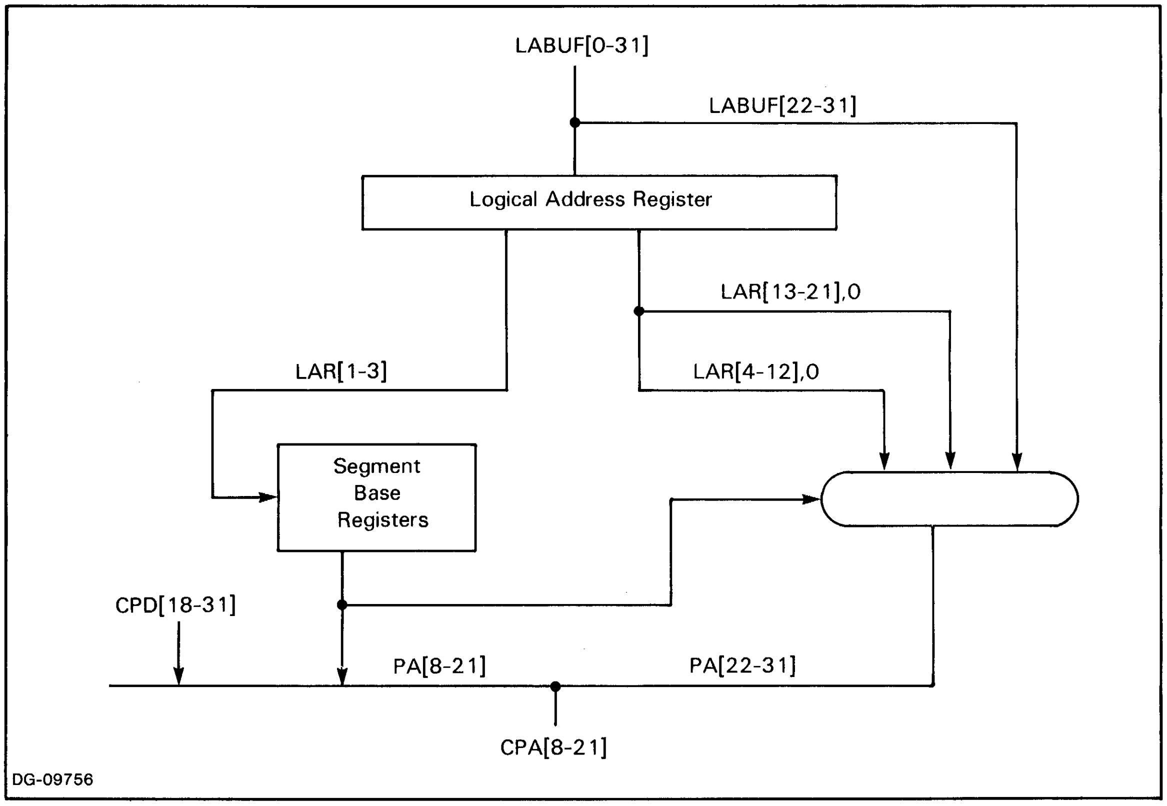


Figure 2-19. Page Table Addressing Logic

Ring Protection

The ATU maintains two registers for keeping track of protection rings. These are the Current Ring of Execution (CRE) register and the Effective Source Register (ESR). The CRE register is set to the ring in which the presently running program resides. The ESR register is used in indirection chains; it represents the ring for the last memory reference.

At macroinstruction boundaries, ESR is set to CRE. During an indirection chain, the protection logic checks the newly started address against the ESR. All memory references must be outward from the ring of execution. On valid memory references, the ESR is reset to the new address; this process continues until the end of the indirection chain or until it reaches an indirection depth of fifteen.

Indirection Protection

The defer counter keeps track of indirection depth. The counter is set to zero at the beginning of a macroinstruction; the only exception occurs when there was an EFA started at IPOP during the previous macroinstruction. The counter will be incremented each time an indirection is resolved. (Incrementing occurs when the RAND:ATU:ATU1:DF micro-order is coded and the test for ending the indirection chain is false.)

The defer counter will allow fifteen levels of indirection for all the indirection resolution required by a macroinstruction. For instance, if an instruction takes a stack fault and must resolve a pointer, that resolution is added to the defer count. At fifteen levels of indirection, a protection trap occurs if the ATU is on; if the ATU is off, there is no protection.

If more than one indirection resolution occurs in a macroinstruction, microcode must reset ESR to CRE at the beginning of each such resolution. This is done with the following steps:

1) Code RAND:ATU:ATU1:AC, which sets the logical address ring bits to CRE, and

2) Code RAND:ATU:ATU0:LCRE, which loads both CRE and ESR from the logical address ring bits.

Read/Write/Execute Protection

The ATU contains a read/write/execute RAM. This RAM is loaded from bits 2-4 of a PTE and reflects the current protection status of pages with translations in the address translation cache. The bits are written whenever a new address translation is produced. These bits can cause a memory protection error if microcode attempts to access a page illegally.

ATU State

The Address Translation Unit (ATU) has 32 bits of state. Table 2-3 explains these bits.

Table 2-3. Address Translation Unit State

Bits	Name	Description
0	^RESTART	Indicates whether the instruction can be restarted after a page fault.
1-3	ESR[1-3]	The Effective Source Ring
4-6	^FLT CODE[0-2]	The fault code for a hardware protection trap.
7	^ASMPND	A cache-block crossing, read double-word assembly is pending.
16	^MEMSTARTED	A memory start is still pending.
17	^SXCT STRT	An XCT start is pending.
18	^IPST STRT	Indicates whether the last non-LAT start was an Instruction Processor start (IPST).
19	^ICAT STRT	Indicates whether the last non-LAT start was an Instruction Cache Address Translation (ICAT).
20	^CPWRITE@	Indicates whether the last non-LAT start was a write.
21-23	^CPMODE@[0-2]	The mode bits for the last non-LAT start.
24	^CBXRTIN	Indicates whether the last non-LAT start was during a Cache Block Crossing.
25	^AT:SET LAT	Indicates whether the processor is in the LAT state.
26	MOD	The modify bit of the page addressed the preceding cycle.

Table 2-3. Address Translation Unit State
(Continued)

Bits	Name	Description
27	REF	The reference bit of the page addressed the preceding cycle.
28-31	REF[0-3]	The four reference bits for the quad page addressed the preceding cycle.

These state bits can be sourced to the CPD bus with the CPDS:ATS micro-order. The ESR and ^CPMODE bits can be restored from the LA and CPD buses with the RAND:ATU:ATU0:LATS micro-order. The ^CPMODE bits are explained under the LATS micro-order in Chapter 3. Table 2-4 shows the meanings of values in the FLTCODE field.

Table 2-4. ATU State Fault Codes

Code	Meaning
0	Read protection fault
1	Write protection fault
2	Execute protection fault
4	Inward reference
5	Defer protection (more than 15 indirect references)

ATU Diagnostic Register

The ATU Diagnostic Register is a 24-bit register that can hold either the current value on the CPA bus or various internal state bits of the CPU. These bits can be sourced (inverted) on the CPD Bus with the CPDS:ATD micro-order. The register captures data whenever a CPU memory start is coded (including IP starts). If RAND:ATU:ATU0:XTND is coded, then CPA[8-31] sources the register; if XTND is not coded, the register will hold the following bits:

Bits	Signal	Description
8-9	—	Reserved
10	^BP:CPRT	The read transfer signal on the backplane.
11	^BP:CPWT	The write transfer signal on the backplane.
12	^PERMITRD	The read-enable bit in the read/write/execute RAM for the page addressed by the current logical address.

Bits	Signal	Description
13	^PERMITWR	The write-enable bit in the read/write/execute RAM for the page addressed by the current logical address.
14	^PERMITXEQ	The execute-enable bit in the read/write/execute RAM for the page addressed by the current logical address.
15	NEWXLAT	LOAD STATUS bit for translation cache, protection, and validity RAM.
16	FRC LAT	The Force Logical Address Translation bit in the of the CBUS register R1 (bit 5). This bit lets the SCP force LAT after every memory reference from microcode. This bit is valid only when the ATU is on and it is not in a LAT routine.
17	^GDXLATA	The output of the address-translation cache comparator that compares LA[4-11] with TAGLA[4-11].
18	^GDXLATB	The output of the address-translation cache comparator that compares LA[12-14] to TAGLA[12-14]. If FRC LAT is set, this output will always be zero.
19	VLDSETSEL	The set-selector bit for the validity RAM. When this bit equals 1, it designates set A; when 0, set B.
20	^VALID TAG	The output of the validity RAM designated by VLDSETSEL. The RAM is addressed by the current logical address or by the purge counter.
21-31	TAGLA[4-14]	The output of the tag store for the address translation cache.

ATU Dispatch

The ATU can produce a two-bit code that goes to the microsequencer and can be used as part of a dispatched microaddress. This type of dispatching is used in Long Address Translation (LAT). The code provides a quick means to branch on the differences between one-level and two-level page tables and between memory start types. The NAC:DSR:A micro-order implements dispatching.

Micro-orders in the RAND:ATU:ATU0 field distinguish the various types of memory start. The possible meanings of the dispatch code, as generated by the ATU, are as follows:

Code	Meaning
00	Start memory for the second Page Table Entry (PTE). This means that the first memory start for the first PTE has already been performed using an RBSR micro-order. This second memory start must use a LPTA micro-order during the memory start in order to correctly address the second PTE.
01	Start memory using an IPST micro-order, i.e., an IP start caused the original LAT routine, and memory must be restarted in the same manner.
10	Start memory using an ICAT miro-order, i.e., an instruction cache translation caused the original LAT routine, and memory must be restarted in the same manner.
11	Start memory using an OPTA micro-order. OPTA means that the final address has been formed and is sourced to the CPA bus for this memory start.

Note: Within an LAT routine, IPST and ICAT have the meaning of OPTA in addition to their regular meanings. When a routine is simply examining PTEs, any of the last three dispatch codes means that the final translation has been encountered.

The CPD Bus and Transfer Register

The ATU cannot take data directly off the CP Memory (CPM) bus. To get information from memory (e.g., a PTE during an LAT), data is passed through the transfer registers (TREG) of the integer ALU. See the integer ALU section for the details of this register.

Instruction Processor

The Instruction Processor (IP) decodes macroinstructions from the instruction stream. It provides starting microaddresses to the WCS for the microroutines that implement the instructions. The IP decode process is pipelined, so that at any given time as many as four different instructions may be in various stages of decoding. The IP has an instruction cache that contains the currently executing instructions. It also maintains the macroinstruction program counter (IPPC). Figure 2-20 shows the IP.

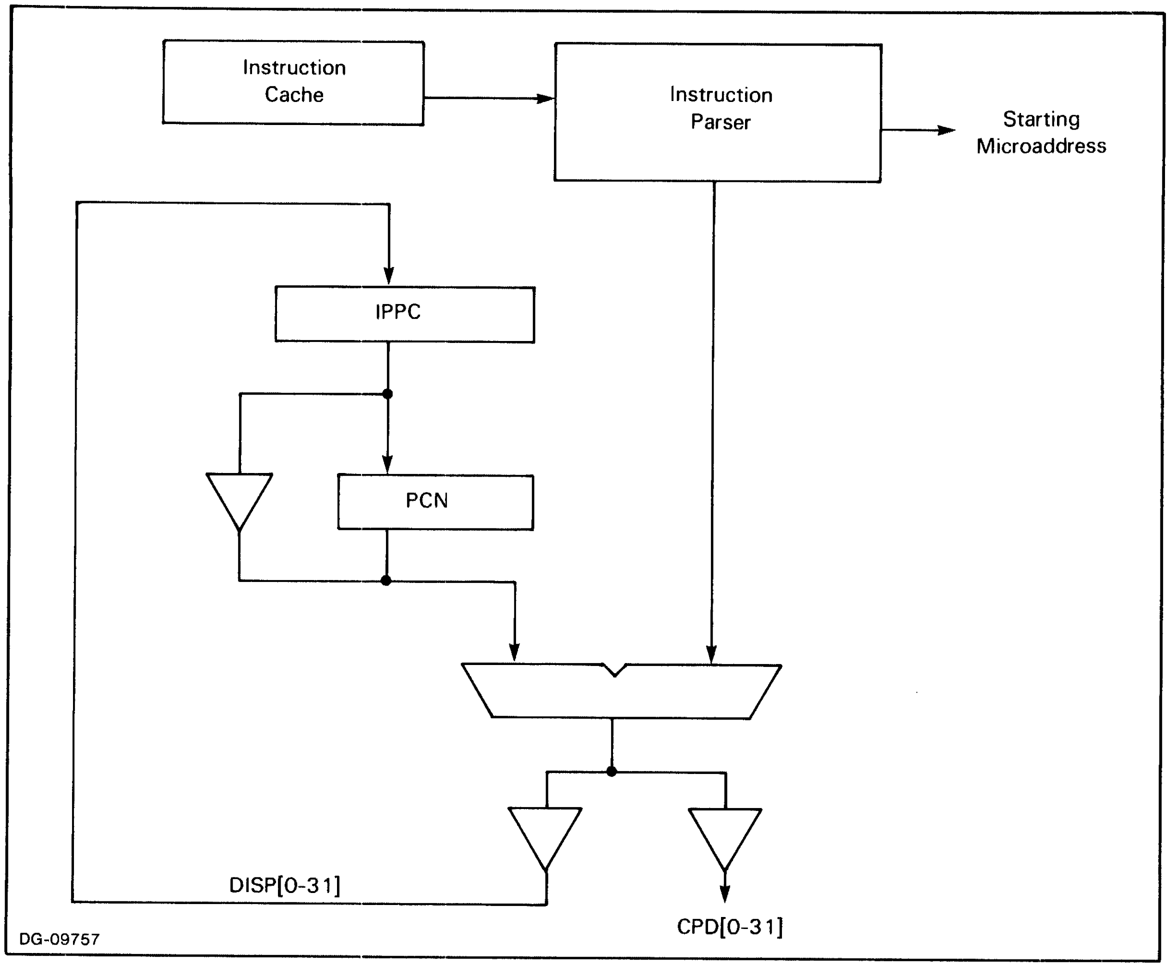


Figure 2-20. The Instruction Processor

When the IP executes instructions sequentially, PCX (which is derived from PCN and the instruction length) points to the executing instruction, PCN points to instruction to be executed next, and IPPC points to the instruction after that. When the next instruction is out of sequence, e.g., because of a JMP, microcode loads a new location into the IPPC register. PCX always points to the currently executing macroinstruction.

Microcode can examine PCX and PCN using micro-orders in the CPDS field. In addition, it can examine the next instruction location whether it comes from PCN or IPPC.

Displacement fields from macroinstructions can go to the Address Generator over the DISP bus.

Instruction Processor State

IP state consists of the following registers:

- The Instruction Processor Program Counter (IPPC)
- The Next Program Counter (PCN)

- The LPCX[0-1] register, which contains the length of the currently executing instruction
- The ION flag, which is the interrupt mask bit
- The XCTFLG, which indicates whether the current macroinstruction resulted from an XCT

The CPDS:IPS micro-order sources IP state onto the CPD bus as follows:

CPD Bits	IP State Bits	Description
30-31	LPCX[0-1]	Length of currently executing instruction
29	ION	Master interrupt mask bit
28	XCTFLG	Bit indicating that the current macroinstruction was the result of an XCT instruction

Interrupts

Interrupts are handled by the IP. Interrupts are normally taken between macroinstructions as a result of the IP forcing the starting microaddress (STUAD) to the beginning of the interrupt routine.

If an instruction takes longer to execute than the specified interrupt latency (12 microseconds), microcode must acknowledge interrupts within that instruction. By convention, microcode tests for interrupts within 64 instruction cycles if an instruction requires more than 80 cycles to execute. We classify interruptable instructions as "restartable" or "resumable."

A *restartable* instruction can safely back out of the current operation or can leave the accumulators in such a state the instruction can procede with its operation. These instructions manage interrupts by either backing out or updating the accumulators, pointing the IPPC at themselves, and performing an IPOP. At IPOP the IP forces the STUAD to the interrupt routine. On returning from the interrupt, execution starts again at the interrupted instruction. A wide character move (WCMV) is an example of a restartable instruction. Because the data in the accumulators is updated with each byte moved, the instruction can be restarted at any point.

A *resumable* instruction cannot back out of the current operation and requires more state than the accumulators. In this case, state is pushed on the user's stack in the same ring as the interrupted instruction (to maintain system integrity, a microaddress may not be pushed on the user's stack). PSR2 is set, which indicates that an instruction must be resumed, IPPC is pointed at the instruction itself, and an IPOP is performed. When the instruction reexecutes, it must check PSR2 to discover whether it must begin fresh or continue from where it left off.

I/O Protocols

The MV/10000 CPU communicates with the I/O Controller (IOC) (and therefore all peripherals) via the CPD bus. The IOC uses the least-significant 16 bits of the bus to transmit and receive data. TREG is the only valid CPD source for communication with the IOC. Bits 14-15 are command bits that tell the IOC what the CPU expects it to do. These bits are coded as follows:

CPD[14-15]	Explanation of Command
00	Clear (No op)
01	*Instruction*: the data on CPD[16-31] is interpreted as a command. See below for commands.
10	*Input*: the data on CPD[16-31] is irrelevant.
11	*Output*: the data on CPD[16-31] goes to the IOC.

There are two separate instruction formats: one for programmed I/O and one for nonprogrammed I/O. These formats are shown in Figure 2-21.

Programmed I/O Format:

16	17-19	20	21-23	24-25	26-31
0	Port	R	OPcode	f	Device Code

Nonprogrammed I/O Format:

16	17-19	20-31
1	Port	Register

Figure 2-21. I/O Command Formats

The first bit in the command specifies programmed or non-programmed I/O. The remaining fields have the following meanings:

- Port—The IOC port address. Port 7 is the broadcast port. Port 0 and 1 are supported.
- R—Reserved.

- OPcode—The programmed I/O command: NIO, DIA, DOA, etc.
- f—The control bits for Busy/Done flags. These are S, C, P for non-SKP instructions, and BZ, BN, DZ, DN for SKP instructions.
- Device Code—The code for a device on the I/O bus.
- Register—The address of a register in the port. This is the register that will be read from or written to.

The following sequence transfers data to and from the IOC:

1) Issue an instruction (01) command.

2) Issue either an input (10) command or an output (11) command.

3) Wait two cycles and then test I/O Busy (TSEL:IOB).

4) Wait for I/O Busy to clear, and, if Step 2 was input, read the input data from the CPD bus.

5) Issue a clear (00) command to prepare for next instruction.

End of Chapter

Chapter 3
Micro-order Format and Instruction Set

This chapter describes the micro-orders in the MV/10000 microword. The microword contains 104 bits, including spare and parity bits (see Figure 3-1).

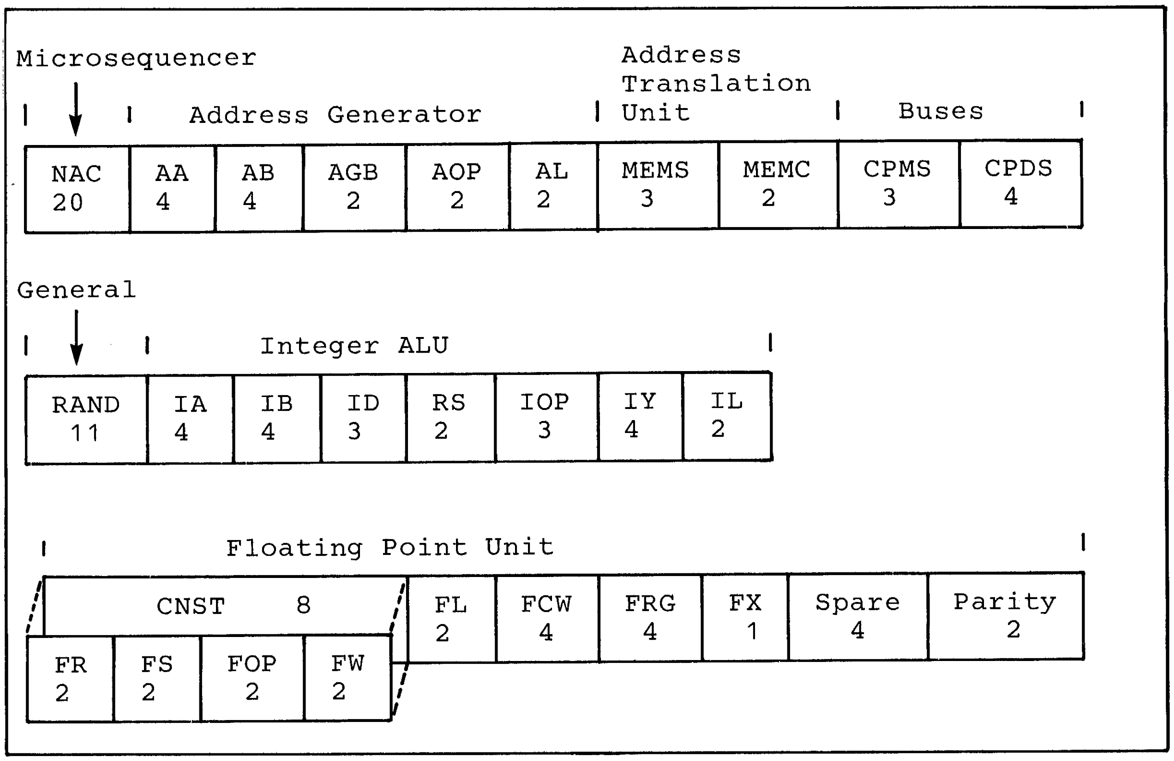


Figure 3-1. The MV/10000 Microword

The microword is divided according to its control functions. Each subsystem in the MV/10000 processor has its own microword fields; in addition, the RAND field contains micro-orders that control several different subsystems.

Each micro-order is described separately in this chapter. Each description begins with a descriptive title, followed by the microassembler mnemonic and the value (in hexadecimal) for that micro-order. For example:

Sample micro-order

SAMP

00

Description of micro-order.

NAC—Next Address Control

The Next Address Control (NAC) field controls logic on the microsequencer board. The NAC field has two separate formats, depending on whether the microinstruction is conditional or unconditional. Figure 3-2 shows the NAC subfields.

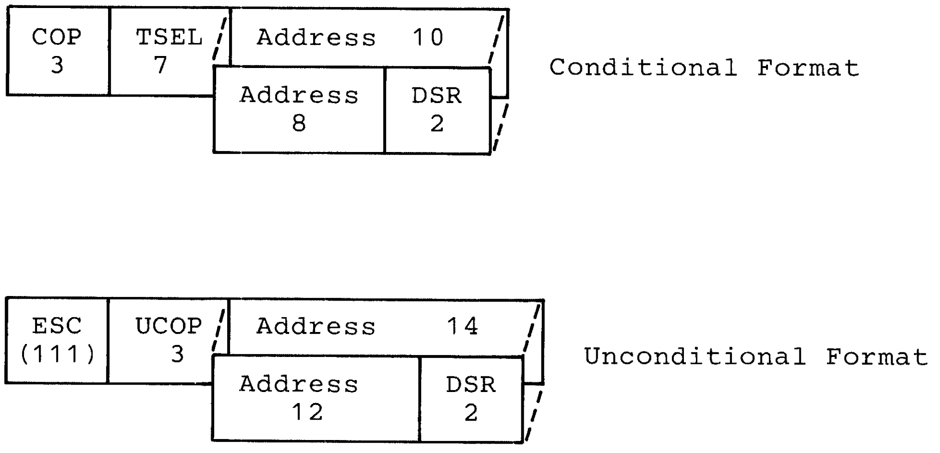


Figure 3-2. The NAC Field

The NAC field controls:

- stack operation and address selection,
- test selection, and
- dispatching.

Note that a micro-order contains either a COP and TSEL field or a UCOP field. All micro-orders beginning '111' are unconditional. (This escape is automatically coded by the microassembler whenever a UCOP micro-order is specified.)

Addresses

Microsequencer addresses can come from:

1) the incremented microprogram counter (uPC+1),

2) the top-of-stack (TOS) register,

3) the AA Bus, or

4) the dispatch register.

The incremented microprogram counter value is the address immediately following the current one with wrap-around on 1K pages. The top-of-stack register value is an address that was previously pushed onto the stack by an NAC:COP or NAC:UCOP micro-order.

The AA Bus is sourced differently depending on whether a micro-order is conditional or unconditional. For conditional micro-orders, the least significant ten bits come from the NAC:ADDRESS field and the most significant bits are equal to the current page; thus conditional micro-orders cannot address beyond the current page boundaries. For unconditional micro-orders, all fourteen bits on the AA Bus come from the NAC:ADDRESS field; thus unconditional micro-orders can address all of WCS.

Addresses from the dispatch register maintain the same distinction between conditional and unconditional. Like the AA Bus addresses, conditional dispatched addresses take the current page number for the four most significant bits. However, the AA Bus supplies only the most-significant address bits for micro-orders that use dispatching. The least-significant bits are provided from the dispatch register or the Address Translation Unit (ATU). Table 3-1 shows the possible address forms for MV/10000 microcode.

Table 3-1. MV/10000 Microaddresses

Symbol	Description
uPC+1	The 14-bit address from the incremented microprogram counter.
TOS	The address that is currently at the top of the microstack.
AA	The 14-bit address from the microinstruction. [unconditional, undispatched]
PA	The current page number concatenated to the 10-bit page offset in the microinstruction (NAC[10-19]). [conditional, undispatched]
ADA	The address from the dispatch multiplexer that is based on the NAC:ADDRESS field. [unconditional, dispatched]
PDA	The address from the dispatch multiplexer that is based on the NAC:ADDRESS field concatenated to the current page number. [conditional, dispatched]

Stack Control and Address Selection

The stack control and address selection logic is controlled by the COP and UCOP subfields of the NAC field. This logic includes:

- the AA selector logic, which determines the source of the AA Bus;
- the microaddress selector logic, which determines the source of the WCS address;
- the microstack source selector logic, which determines what will be pushed onto the microstack; and
- the microstack control logic, which determines whether the stack will be pushed or popped.

NAC:COP—Conditional OPcode

The COP field controls conditional actions by the microsequencer, i.e., those actions that depend on the outcome of a test. The micro-order specifies the next address and the stack operation for both the true and the false test results. The particular test is specified in the NAC:TSEL field. Table 3-2 summarizes COP field micro-orders. (In the table .+1 means uPC+1.)

Table 3-2. Conditional Microorders in the OP Field

Mnemonic	Value	True		False		Description
		uPC	uStack	uPC	uStack	
CJMP	0	PA	-	.+1	-	Conditional jump
CJSR	1	PA	PSH .+1	.+1	-	Jump and save return
CDSP	2	PDA	-	.+1	-	Conditional dispatch
CABT	3	PA	POP	.+1	POP	Cond jump, abort TOS
CRTN	4	TOS	POP	PA	-	Conditional return
TWB	5	TOS	POP	PA	POP	Two-way branch
CRST	6	TOS	POP	PA	—	Conditional restore
—	7	—	—	—	—	Escape to unconditionals

Conditional Jump

CJMP

0

If the test is true, CJMP transfers control to the address specified in NAC:ADDRESS. This address must be within the current page; CJMP's 10-bit address field cannot address more than a page at a time. If the test is false, CJMP goes to uPC+1. CJMP has no effect on the microstack.

Jump and Save Return

CJSR

1

CJSR is identical to CJMP, except that if the test is true, uPC+1 is pushed onto the microstack. At some later point, control can be returned by popping the stack.

Conditional Dispatch

CDSP

2

If the test is true, CDSP forms an address using the dispatch register or bits from the ATU. If the test is false, control transfers to the following instruction. Note that CDSP requires the use of a DSR micro-order to further specify the dispatching.

Conditional Jump, Abort TOS

CABT

3

If the test is true, CABT transfers control to the address specified in the NAC:ADDRESS field. If the test is false, control transfers to the following instruction. In this regard, CABT is identical to CJMP. However, CABT has the additional effect that regardless of the test outcome, the microstack is popped.

Conditional Return

CRTN

4

If the test is true, CRTN transfers control to the address at the top of the stack and pops the microstack. If it is false, control transfers to the address in the NAC:ADDRESS field and the microstack is unchanged.

Two-way Branch

TWB

5

If the test is true, TWB transfers control to the address at the top of the microstack. If it is false, control transfers to the address in the NAC:ADDRESS field. Regardless of the test result, TWB pops the microstack.

Conditional Restore

CRST

6

If the test is true, CRST:

A) transfers control to the address at the top of the microstack,

B) pops the microstack, and

C) forces the test condition for the next microcycle to come from the restored microstack (TOS14).

If the test is false, control transfers to the address in the NAC:ADDRESS field and the microstack is unchanged.

Note that CRST is the only micro-order that restores the test condition; it is used primarily by trap routines.

NAC:TSEL—Test Selection

The test select (TSEL) field selects tests for COP micro-orders. There are a total of 64 tests. Unless otherwise noted, the test result applies to the test condition in the preceding microcycle.

The polarity bit controls the value of the test return: if the polarity is changed from the value in the table, the meaning of the test is reversed. For instance, if the polarity bit for

IOB is changed from 1 to 0, the meaning becomes "I/O not busy."

Note: The microassembler automatically changes the polarity bit if a mnemonic is preceded by "N," e.g., "NIOB." Also, the assembler recognizes FALSE as having the meaning NTRUE.

Microsequencer Tests

The following tests apply particularly to the microsequencer.

True

TRUE

Polarity=0 Value=0

The outcome of the TRUE test is always true, forcing the action specified in the COP field. Coding FALSE will reverse the polarity bit and force the false choice.

CPD31 Equals 1

CPD31

Polarity=1 Value=1

CPD31 tests the value of the least significant bit on the CPD Bus. If the bit is 1, the test is true; if 0, false.

Microstack Empty

USMT

Polarity=1 Value=2

USMT is true if there is no more data in the microstack; if values remain to be popped, the test is false.

Interrupt Pending

INTR

Polarity=1 Value=3

INTR is true if there is an unserviced interrupt waiting and the interrupt on flag (ION) is set to 1.

IO Busy

IOB

Polarity=1 Value=4

IOB is true if the I/O Controller (IOC) is busy, i.e., if it cannot receive or source data at this time. See the section "I/O Protocols" in Chapter 2.

IP Started

IPST

Polarity=0 Value=5

This test is true if, during the current macroinstruction, a RAND:ATU:ATU0:IPST micro-order has been issued. This means that a new value has been loaded into IPPC.

SCP Command Valid

CIRV

Polarity=1 Value=5

CIRV is true if the CIR register on the SCP has valid data; i.e., an SCP request is ready.

Macroinstruction Executed

XCTF

Polarity=0 Value=6

XCTF is true if an *Execute* microinstruction sequence inserted the current macroinstruction in the instruction stream (rather than the instruction coming from memory). Note that both XCT and PBX instructions perform *Execute* sequences.

Perform Rounding

RND

Polarity=0 Value=7

RND is true if bit 8 of the Floating Point Status Register (FPSR) is 1. FPSR8 specifies whether rounding or truncation takes place in floating-point arithmetic. FPSR8=1 indicates unbiased rounding; FPSR8=0 indicates truncation.

Flag Tests

FLG[0-7]

Polarity=0 Value=[8-F]

The FLG# tests are true if the appropriate flag is 1, and false if the flag is 0. The "Flags" section of Chapter 2 explains the meanings of the individual flags.

Address Translation Unit Tests

The following tests are used by the Address Translation Unit (ATU).

Test Most Significant Bit

INDR

Polarity=1 Value=10

This micro-order tests the most-significant bit in the data coming from memory. If a double-word memory start was initiated in MEMS, then it tests CPM0; if a single-word memory start was initiated, it tests CPM16. The test is true if the tested bit is 1.

This test is used to examine the indirection bit in data brought from main memory.

Current Ring Equal to Zero

RNG0

Polarity=1 Value=11

This micro-order tests to see if the Current Ring of Execution (CRE) is zero, i.e., if the program is currently in the operating system segment. The test is true if CRE = 0.

Check for Inward Reference

RMAX

Polarity=1 Value=12

Checks the current reference to see whether it is less than the Current Ring of Execution (CRE) or the Effective Source Ring (ESR). If the micro-order RAND:ATU:ATU1:DF (the defer micro-order) was coded with the last memory start, then the processor is in an indirection chain, and the test uses the ESR. Otherwise, the test uses the CRE.

Note that memory need not be started for this test: simply sourcing the address to the LA Bus is sufficient. If memory is started, then the memory protection mechanism is also armed.

Ring Less Than Effective Source Ring

LESR

Polarity=0 Value=13

Checks to see whether the ring bits for the current logical address are less than the Effective Source Ring (ESR). The test is true for LA[1-3] < ESR[1-3] .

Ring Greater Than Current Ring of Execution

GCRE

Polarity=0 Value=14

Checks to see whether the ring bits for the current logical address are greater than the Current Ring of Execution (CRE). The test is true for LA[1-3] > CRE[1-3].

Ring Equal to Current Ring of Execution

ECRE

Polarity=0 Value=15

Checks to see whether the ring bits of the current logical address are equal to the Current Ring of Execution (CRE). The test is true for LA[1-3] = CRE[1-3].

Ring Less Than Current Ring of Execution

LCRE

Polarity=0 Value=16

Checks to see whether the ring bits of the current logical address are less than the Current Ring of Execution (CRE). The test is true for LA[1-3] < CRE[1-3].

Detect Cache Block Boundary

CBLK

Polarity=1 Value=17

Checks to see whether the current logical address is on the upper boundary of a cache block in main memory. Cache blocks contain eight 16-bit words; the least-significant three bits in a logical address determine a word's location in the block. This micro-order detects whether the least-significant three bits are all ones. The test is true if LA[29-31] = 7.

Address Translation Unit On

ATON

Polarity=0 Value=18

This test is true if the Address Translation Unit is on this cycle.

Address Translation Unit Purging

PRGB

Polarity=0 Value=19

This test is true if the Address Translation Unit is currently purging its translation cache.

Check for Valid Page Table Entry

VPTE

Polarity=1 Value=1A

This micro-order checks the two most-significant bits on the CPM Bus. If the last memory start was with RAND:ATU:ATU0:<RSBR or LPTA> these bits will be the "valid" and "resident" bits of a Page Table Entry. This test is true if CPM[0-1]=3.

Check for Valid Segment Base Register

VSBR

Polarity=1 Value=1B

This micro-order checks the "valid" and "length" bits of Segment Base Register (SBR). If the length bit is zero (1-level page table), then bits 4-12 of the current logical address should be zero. This test is true if:

```
SBR0=1 AND (SBR1=1 OR (SBR1=0 AND LAR[4-12]=0))
```

Check Valid Bit

VLD

Polarity=1 Value=1C

Depending the coding of RAND:ATU:ATU0, this micro-order examines the valid bit for a Segment Base Register or a Page Table Entry (PTE).

If RAND:ATU:ATU0:RSBR was coded last cycle, then this test is true if SBR0=1.

If RAND:ATU:ATU0:<OPTA or LPTA> is coded, then this test is true if CPD0=1. (Presumably, CPD0 is the valid bit for a PTE that was addressed by RSBR or LPTA in a previous microinstruction.)

Macroinstruction Decoded

IVLD

Polarity=0 Value=1D

IVLD is true if the Instruction Processor has a macroinstruction decoded. If an instruction is decoded, then IPOP can proceed.

I/O Allowed

IOEN

Polarity=0 Value=1E

This test is true if I/O is allowed in the current ring.

PC Relative Addressing

IXPC

Polarity=0 Value=1E

This test is true if the index bits of the macroinstruction indicate PC relative addressing (=01). This test is valid only during the first cycle of a macroinstruction.

Integer ALU Tests

The following tests apply to functions in the integer ALU.

Test Bit 28 on the Y Bus

Y28

Polarity=0 Value=28

This test is true if bit Y28 is one, and false if it is zero.

Test Bit 29 on the Y Bus

Y29

Polarity=0 Value=29

This test is true if bit Y29 is one, and false if it is zero.

Test Bit 30 on the Y Bus

Y30

Polarity=0 Value=2A

This test is true if bit Y30 is one, and false if it is zero.

Test Bit 31 on the Y Bus

Y31

Polarity=0 Value=2B

This test is true if bit Y31 is one, and false if it is zero.

Test Bit 31 on the D Bus

D31

Polarity=1 Value=2C

This test is true if bit D31 is one, and false if it is zero.

Test the Sign Bit on the D Bus

DSGN

Polarity=0 Value=2D

This test is true if the D Bus sign bit is 1. FLG3 determines which bit is the sign bit: if FLG3=0, D16 is the sign bit; if FLG3=1, D0 is the sign bit.

Compare the Source and Destination Addresses

COMP

Polarity=0 Value=2E

This test is true if the ACSR and ACDR point to the same register in the integer register file.

Test the Resumable Instruction Bit

IRES

Polarity=0 Value=2F

The Processor Status Register (PSR) bit 2 indicates whether a resumable instruction has been interrupted. All resumable instructions must test this bit when they begin execution. If it is set, they must restore state from the user stack.

Commercial Data Validity Test 1

COM1

Polarity=0 Value=30

The COM1 test validates commercial data. A PROM tests the least-significant byte on the A Bus of the integer ALU section, i.e., the data must be available at the A output port of the integer register file. The test to be performed is specified in the CNST field. Table 3-3 describes the tests. The inputs are those octal values that will cause COM1 = true.

Table 3-3. COM1 Tests

Mnemonic	Value	Description
VCB	0	Validate Character Byte Inputs: 040 101-132 141-172
VSB	1	Validate Sign Byte Inputs: 053 055
VSL	2	Validate Sign Low (Low nibble in byte) Inputs: 014 015 017
VCS	3	Validate Commercial Sign Byte Inputs: 015 055 175 112-122

Commercial Data Validity Test and Translation

COM2

Polarity=0 Value=31

The COM2 test validates commercial data. A PROM tests the least-significant byte on the A Bus of the integer ALU section, i.e., data must be available at the A output port of the integer register file. The test to be performed is specified in the CNST field. Table 3-4 describes the tests. The inputs are those octal values that will cause COM2 = true. The outputs are the hexadecimal values that the inputs are translated into. The micro-order IY:EDT makes the outputs available on the IY Bus.

Table 3-4. COM2 Tests

Mnemonic	Value	Description
VSO	4	Validate Sign Overpunch and Translate Data Inputs: 040 053 055 060-071 101-111 112-122 173 175 Outputs: 0 0 0 0-9 1-9 1-9 0 0
VDB	5	Validate Digit Byte Inputs: 040 060-071 Outputs: 0 0-9
VDL	6	Validate Low Digit (Low nibble in byte) Inputs: 000-011 Outputs: 0-9
VDH	7	Validate High Digit (High nibble in byte) Inputs: 000-011 Outputs: 0-9

I/O Tests

IOT

Polarity=0 Value=32

The IOT micro-order is used for I/O skips and for decoding NOVA I/O instructions. The test is specified in the CNST field. The data to be tested must be on the A Bus of the integer ALU, i.e., it must be available at the A output port of the integer register file. Table 3-5 shows the CNST field micro-orders that specify the tests.

Table 3-5. IOT Tests

Mnemonic	Value	Description
CPUD	0	CPU Device Code: A[24-31] = xx11 1111
SKPT	1	Skip test: A[28-31] = 000x or 011x or 10x0 or 11x1
IONF	2	ION Flag change: A[24-25] = 10 or 01

Test the Least-Significant ALU Bit

F31

Polarity=0 Value=33

This test is true if F31, the ALU output's least-significant bit, equals 1.

Carry from Least-Significant 4 Bits

CRY28

Polarity=0 Value=34

This test is true if the carry-out from bits 28-31 of the ALU equals 1.

Test the Sign on the R Bus

RSGN

Polarity=0 Value=35

The test is true if the R Bus sign bit equals 1. For FLAG3=1, the sign bit is R0; for FLAG3=0, the sign bit is R16.

Test the Most-Significant Bit on the Y Bus

Y0

Polarity=0 Value=37

This test is true if IY[0] equals 1. For narrow operations from the bit shifter, this test is *always* true.

Test 8-bit PDR Counter

CNT8

Polarity=1 Value=38

Test and increment the CPD Bus register (PDR) in the integer ALU. This test is true if PDR[24-31] is all ones. Note that, for this micro-order, PDR is considered an 8-bit counter. After testing, the counter is incremented by one. (This micro-order is functionally equivalent to incrementing the counter and testing for 0.)

Test 4-bit PDR Counter

CNT4

Polarity=1 Value=39

Test and increment the CPD Bus register (PDR) in the integer ALU. This test is true if PDR[28-31] is all ones. Note that, for this micro-order, PDR is considered a 4-bit counter. After testing, the counter is incremented by one. Note that CNT4 increments the entire eight-bit counter. (This micro-order is equivalent to incrementing the counter and testing PDR[28-31] for 0.)

Carry

CRY

Polarity=0 Value=3A

Test the carry-out from the ALU. For wide tests (FLAG3=1), this test is true if CRY0=1; for narrow tests (FLAG3=0), if CRY16=1.

Test the ALU Sign Bit

FSGN

Polarity=0 Value=3B

This test is true if the ALU output's sign bit equals one (i.e., if the value is negative). For wide tests (FLAG3=1), the sign bit is F0; for narrow tests (FLAG3=0), the sign bit is F16.

Overflow

OVF

Polarity=0 Value=3C

This test is true if there is overflow from an ALU operation. For wide tests (FLAG3=1), the overflow is from a 32-bit result; for narrow tests (FLAG3=0), from a 16-bit result.

Test for ALU Result Equal to Zero

FZR

Polarity=1 Value=3D

This test is true if the F Bus (the ALU output) equals zero. For wide tests (FLAG3=1), the test is for 32 bits (F[0-31]); for narrow tests (FLAG3=0), 16 bits (F[16-31]).

Signed Greater Than or Equal

SGE

Polarity=1 Value=3E

This test is true if the signed value on the S input is greater than or equal to the signed value on the R input. For this test to work correctly, on the previous cycle you must subtract (IOP:CSR) the quantities you wish to compare.

For wide tests (FLAG3=1), SGE tests all 32 bits; for narrow tests (FLAG3=0), SGE tests only the least-significant 16 bits.

CARRY Equal to One

CRRY

Polarity=1 Value=3F

Test for CARRY equal to one this cycle.

Floating-Point Tests

Floating-point tests in the TSEL field compare the magnitudes of two numbers or check for mantissa or exponent carry-out. A compare test is based on the compare status bits in the STATE register. These bits are set by the RAND:FLT:SCNT:CMP micro-order. The signed magnitude tests are based on the values in SA and SB. In order for the comparison tests to work, the CMP micro-order must be executed and, if necessary, RAND:FLT:SGN:LAB must be executed at least two cycles before the microinstruction containing the test.

A Equals B, Unsigned

UAEB

Polarity=0 Value=27

This test is true if the absolute values of the numbers on the FA and FB buses were equal when CMP was executed.

A Less Than B, Unsigned

UALB

Polarity=0 Value=26

This test is true if the absolute value of the number on the FA Bus was less than the absolute value of the number on the FB Bus when CMP was executed.

A Greater Than B, Unsigned

UAGB

Polarity=0 Value=25

This test is true if the absolute value of the number on the FA Bus was greater than the absolute value of the number on the FB Bus when CMP was executed.

A Equals B, Signed

SAEB

Polarity=1 Value=24

This test is true if the signed value of the number on the FA Bus was equal to the signed value of the number on the FB Bus.

A Less Than B, Signed

SALB

Polarity=0 Value=23

This test is true if the signed value of the number on the FA Bus was less than the signed value of the number on the FB Bus.

A Greater Than B, Signed

SAGB

Polarity=0 Value=22

This test is true if the signed value of the number on the FA Bus was greater than the signed value of the number on the FB Bus.

Mantissa Carry-out

FCRY

Polarity=0 Value=21

This test is true if there was a carry-out from the mantissa ALU.

Exponent Carry-out

ECRY

Polarity=0 Value=20

This test is true if there was a carry-out from the exponent ALU.

NAC:UCOP—Unconditional OPcode

The UCOP field controls unconditional actions by the microsequencer, i.e., actions that occur without regard to any test condition. Table 3-6 summarizes the micro-orders in the UCOP field.

Table 3-6. Unconditional OP Microorders

Mnemonic	Value	uPC	uStack	Description
LEAP	0	AA	—	14-bit jump
LSR	1	AA	PSH .+1	Leap and save return
DSPA	2	ADA	—	14-bit dispatch
DSPR	3	ADA	PSH .+1	Dispatch and save return
LPOP	4	AA	POP	Leap and pop TOS
PUSH	5	.+1	PSH AA	Push a 14-bit address
PCPD	6	AA	PSH CPD	Push stack state from CPD bus
TPSH	7	TOS	PSH AA	Go to TOS and push address

The NAC:ADDRESS field for UCOP micro-orders is 14 bits long, so that these micro-orders can address all of WCS.

Fourteen-bit Jump

LEAP

0

LEAP transfers control to NAC:ADDRESS.

Fourteen-bit Jump and Save

LSR

1

LSR pushes the uPC+1 onto the microstack and transfers control to NAC:ADDRESS.

Fourteen-bit Dispatch

DSPA

2

DSPA constructs an address using bits from the dispatch register or the ATU. The DSR field must be coded with DSPA to specify how the address is constructed.

Fourteen-bit Dispatch and Save Return

DSPR

3

DSPR constructs an address using bits from the dispatch register or the ATU. In addition, it pushes uPC+1 onto the stack for a future return. The DSR field must be coded with DSPA to specify how the address is constructed.

Jump and Pop the Microstack

LPOP

4

LPOP transfers control to NAC:ADDRESS and pops the microstack. Note that the current top of stack (TOS) value is lost.

Push a Fourteen-bit Address

PUSH

5

PUSH transfers control to the following instruction (uPC+1) and pushes NAC:ADDRESS onto the microstack.

Push State from CPD

PCPD

6

PCPD transfers control to NAC:ADDRESS and pushes the most significant bits from the CPD Bus (CPD[0-15]-) onto the microstack. This micro-order can be used to store state that can be recovered at some later time by popping the stack.

Go to TOS and Push a Fourteen-bit Address

TPSH

7

TPSH transfers control to the address at the top of the microstack and pushes NAC:ADDRESS onto the microstack.

NAC:DSR—Dispatch Address Source

The DSR portion of the NAC field controls the cross-bar network and the dispatch multiplexer. It is used by microinstructions that specify dispatch addressing, specifically, the NAC field micro-orders COP:CDSP, UCOP:DSPA, and UCOP:DSPR.

Table 3-7. Dispatch Address Source

Mnemonic	Value	Description
A	0	ATU dispatch: AA[0-9],0,ATD[0-1],0
F	1	Four-bit dispatch: AA[0-9],DSP[4-7]
E	3	Eight-bit dispatch: AA[0-5],DSP[0-7]

The dispatch register is loaded from the CPD Bus. Using the dispatch register, the microprogram can branch on the basis of some external value (e.g., a value from a macroinstruction or from an I/O controller). The cross-bar network can also construct an address with two bits supplied by the Address Translation Unit.

ATU Dispatch

A

0

The A micro-order constructs an address using the most significant ten bits from the AA Bus and two bits supplied by the ATU over the ATD Bus. The resulting address is:

```
AA[0-9],0,ATD[0-1],0
```

ATU addresses are used to direct page-table searches for the Long Address Translation (LAT) routine.

Four-bit Dispatch

F

1

The F micro-order constructs an address with the ten most significant bits from the AA Bus and the four least significant bits from the dispatch register. The resulting address is:

```
AA[0-9],DSP[4-7]
```

You can use the four-bit dispatch for such things as quick access to a table (with the AA address as the table base and DSP as an index).

Eight-bit Dispatch

E

3

The E micro-order is similar to F, except that it uses eight bits from the dispatch register and only six bits from the AA Bus. The resulting address is

```
AA[0-5],DSP[0-7]
```

The additional bits allow dispatching to a greater range of addresses (e.g., for larger tables).

Address Generator Micro-orders

The Address Generator portion of the microword contains the following fields:

- *AA*—specifies the A output of the register file.
- *AB*—specifies the B output of the register file and also the input register for the register file.
- *AGB*—specifies the sources for the AGB Bus.
- *AOP*—specifies the operation for the AG ALU.
- *AL*—specifies the source for loading the register file.

AA and AB—The Register File Address Fields

The AA and AB fields designate the sources for the register file's A and B output ports. In addition, the AB field designates the input register. The following micro-orders can appear in both the AA and AB fields.

Macroinstruction Accumulator 0

AG0

0

This micro-order specifies the first register in the register file. At IPOP, this register must contain the same value as Accumulator 0 in the Integer ALU register file.

Macroinstruction Accumulator 1

AG1

1

This micro-order specifies the second register in the register file. At IPOP, this register must contain the same value as Accumulator 1 in the Integer ALU register file.

Macroinstruction Accumulator 2

AG2

2

This micro-order specifies the third register in the register file. At IPOP, this register must contain the same value as Accumulator 2 in the Integer ALU register file.

Macroinstruction Accumulator 3

AG3

3

This micro-order specifies the fourth register in the register file. At IPOP, this register must contain the same value as Accumulator 3 in the Integer ALU register file.

Wide Stack Pointer

SP

4

The wide stack pointer for the current ring (page-zero-location 12_{16}) is copied into this register. By convention, these copies are not always identical. The register contains the valid copy.

Constant 1

ONE

5

This register always contains a 1. The microprogrammer can use it to increment or decrement values.

Constant 2

TWO

6

This register always contains a 2. The microprogrammer can use it to increment or decrement a value by 2. For instance, the 2 can be used to increment the wide stack pointer for WPSH.

Reserved Register for Long Address Translation

LAT

7

This register is used by the LAT routine, and must not be used for general microprogramming.

General Register 0

AR0

8

The microprogrammer may use this register for general purposes. It has no assigned meaning.

General Register 1

AR1

9

The microprogrammer may use this register for general purposes. It has no assigned meaning.

General Register 2

AR2

A

The microprogrammer may use this register for general purposes. It has no assigned meaning.

General Register 3

AR3

B

The microprogrammer may use this register for general purposes. It has no assigned meaning.

General Register 4

AR4

C

The microprogrammer may use this register for general purposes. It has no assigned meaning.

General Register 5

AR5

D

The microprogrammer may use this register for general purposes. It has no assigned meaning.

Register Addressed by ACSR

SRC

E

This micro-order takes the address for the AG register file from the Accumulator Source (ACSR) address register of the integer ALU. Thus, for this micro-order, the AG register will correspond to the ACSR register. For example, if the ACSR register holds 1, it addresses macroaccumulator 1 and the macroaccumulator's copy in register 1 of the AG register file.

Register Addressed by ACDR

DES

F

This micro-order takes the address for the AG register file from the Accumulator Destination (ACDR) address register of the integer ALU. Thus, for this micro-order, the AG register will correspond to the ACDR register. For example, if the ACDR register holds 1, it addresses macroaccumulator 1 and the macroaccumulator's copy in register 1 of the AG register file.

AGB—The Address Generator Bus Field

The AGB field controls the sources to the B input of the AG ALU. The following micro-orders are used in this field:

Displacement Register

D

0

The Instruction Processor (IP) Displacement Register is chosen as the source for the AGB Bus. The IP controls the loading of this register from the DISP Bus. Microcode should use this register only to perform EFA calculations; its contents are indeterminant to a microprogram. The D micro-order must always be used in the AGB field at IPOP.

Register File B Port

B

1

The B output port of the AG register file sources the AGB Bus. The B port address comes from the AB field of the microinstruction.

Constant Register

C

2

The constant register sources the AGB Bus. This register is loaded from the CONSTANT field of the microinstruction. Note that if the CONSTANT field is being used for floating-point operations, the value in the constant register may be meaningless.

Last Logical Address Register

L

3

The last Logical Address (LA) register sources the AGB Bus. The last LA register is loaded only on memory starts, except during LAT or Cache Block Crossings (CBXs). CBXs are performed by incrementing the last LA register to access the second half of the double word.

AOP—Address Generator ALU Operation Field

The AOP field controls the actions of the AG ALU. The ALU is used for address calculations. The following micro-orders are available for this field:

B minus A

SUB

0

Subtract the A input to the AG ALU from the B input. This operation has a carry-in of 1 (2's complement subtraction).

A plus B

ADD

1

Add the A input of the AG ALU to the B input. This operation has no carry-in.

Pass AGB Bus

PSB

2

Pass the B input (the AGB Bus) through the AG ALU unchanged. The A input is ignored.

Effective Address Calculation

EFA

3

Try to perform the next Effective Address (EFA) calculation. Note that this micro-order uses the index bits of the next macroinstruction to calculate the A-port address for the AG register file and to determine the AG ALU function. Therefore, the microprogrammer cannot use the AA field in an IPOP cycle. This micro-order must be coded during IPOP.

AL—Address Generator Register Loading

The AL field of the microinstruction determines which source is used when the AG register file is loaded. Note that the address for this file is always the B address, specified in the AB field. The following micro-orders can be coded in the AL field:

No Load

N

0

Do not load the AG register file on this microinstruction.

Load from CPM

M

1

Load the AG register file from the CPM Bus.

Load from AY

Y

2

Load the AG register file from the AY Bus.

Load on True

C

3

Load the AG register file from the CPM Bus if the test coded on this cycle is true.

Memory Control Micro-orders

The memory control portion of the microword contains the following fields:

- *MEMS*—starts a main memory reference.
- *MEMC*—completes a main memory reference.

MEMS—Memory Start

The following micro-orders start memory prior to a transfer of data to or from the CPU. The Address Generator forms the reference addresses during the same microinstruction as the memory start. The type of start determines how the logical address is formed:

Start Type	Address Formation
Word or double word	LA[0-31] = AGB[0],AY[1-31]
Byte	LA[0-31] = AY[31],AY[0-30]

If FLAG1=0 (narrow addressing), then LA[1-3] = CRE[1-3] and LA[4-16] are zeroed.

No Operation

N

0

No operation takes place.

Read a Word

RW

1

Start a memory cycle to read a word.

Read a Double Word

RD

2

Start a memory cycle to read a double word.

Read a Byte

RB

3

Start a memory cycle to read a byte.

Machine State Determined Start

S@

4

Start memory according to the information from the current macroinstruction (read/write, byte/word/double word, and indirection). In a LAT routine, start memory according to the last non-LAT memory start. S@ must be coded during IPOP.

Write a Word

WW

5

Start a memory cycle to write or read/modify a word.

Write a Double Word

WD

6

Start a memory cycle to write or read/modify a double word.

Write a Byte

WB

7

Start a memory cycle to write or read/modify a byte. For byte writes, only the integer ALU buses IA and IY correctly align themselves for the cache. Other sources must be aligned by microcode: for even addresses, CPM[16-23]; for odd addresses CPM[24-31]. The cache looks at only the specified bits on the CPM Bus.

MEMC—Memory Complete

MEMC micro-orders complete memory transfers that were started by micro-orders in the MEMS field.

No Operation

N

0

No operation takes place.

Read or Read/Modify Complete

R

1

Complete a read operation started on a previous cycle by a micro-order in the MEMS field. An R complete checks read protection. You perform a read/modify operation by coding a write start followed by a read complete. Memory will remain started until you code a write complete.

Write Complete

W

2

Complete a write operation started on a previous cycle by a micro-order in the MEMS field. If possible, avoid coding W in the same microinstruction with a read memory start, because memory will cause a delay in the microinstruction cycle. Never code W with a read start during IPOP.

Execute Complete

X

2

Complete a read-word memory start (MEMS:RW) to the Instruction Processor (IP). Execute protection will be checked if the start was accompanied by the RAND:ATU:ATU0:<IPST or ICAT> micro-orders.

Abort

A

3

Abort a memory start. An abort always inhibits protection, except for indirection depth. Coding A enables the ATU diagnostic register. This is described in Chapter 2 under "ATU Diagnostic Register."

Bus Control Micro-orders

The Bus Control portion of the microword contains the following fields:

- *CPMS*—specifies the source for the CPM Bus.
- *CPDS*—specifies the source for the CPD Bus.

CPMS—CPM Bus Sources

The CPMS field controls the sources of the CPM Bus.

No Op

N

0

The CPM Bus is not driven.

Main Memory

MM

1

Main memory (the system cache) drives the CPM Bus. See the MEMS and MEMC micro-order fields.

Address Generator

AG

2

The AY Bus of the Address Generator drives the CPM Bus.

ALU IY Bus

IY

3

The IY Bus of the integer ALU drives the CPM Bus. The IY Bus carries data from the ALU, the hex shifter, and the edit RAMs.

ALU A Bus

IA

4

The A Bus of the integer ALU drives the CPM Bus. The A Bus is the A output of the integer ALU register file.

Most-Significant Floating-Point Word

HF

5

The most-significant bits from the floating-point register file (FA[0-31]) source the CPM Bus.

Least-Significant Floating-Point Word

LF

6

The least-significant bits from the floating-point register file (FA[32-63]) source the CPM Bus.

CPDS—CPD Bus Sources

The CPD Bus is the major bus connecting the various boards of the MV/10000 CPU. Most data that is internal to the CPU travels over this bus, and the I/O controller is connected to this bus.

The PDR register is loaded whenever the CPD Bus is sourced (i.e., any micro-order other than N). The only exceptions to this are during a LAT routine and when RAND:ATU:ATU0:NPDR is coded.

Note: Sources designated as ''slow'' cannot be used for arithmetic or during IPOP. These sources start and stop driving the CPD Bus later than normal sources. A normal source should not be coded in a cycle immediately following a slow source. If this were done, the normal source signals would be garbled by the signals from the slow source.

No Op

N

0

No source drives the CPD Bus. The bus contains zeros and the PDR register is not loaded.

ALU IY Bus

IY

1

The IY Bus of the integer ALU drives the CPD Bus.

Transfer Register

TRG

2

The transfer register (TREG) in the integer ALU sources the CPD Bus. TREG is loaded from the CPM Bus. TREG drives the CPD Bus fast enough so that it can source the integer ALU. The result from the ALU can be driven to the Address Generator on the CPM Bus during the same microinstruction cycle.

Microsequencer State

USS

3

USS causes the microsequencer to drive its state onto the CPD Bus. This state consists of the current top of the stack (16 bits), the flags (8 bits), and the dispatch register (8 bits), as follows:

- TOS[0-15] goes to CPD[0-15]-
- FLG[0-7] goes to CPD[16-23]-
- DSP[0-7] goes to CPD[24-31]-

The top of stack data and the flags are read from CPD with their true values and written back to the bus in inverted form. The dispatch register bits are read inverted and written true. For state save and restore, data should be inverted before writing it to memory.

Next Sequential PC

PCN

4

The value on the CPD Bus is the currently executing PC plus the length of the currently executing macroinstruction. This micro-order only makes setup to the PDR register.

Executing PC

PCX

5

The value on the CPD Bus is the currently executing PC. This micro-order only makes setup to the PDR register.

Next PC

PC

6

The value on the CPD Bus is the next PC. This value will be identical to PCN if there is no IPST; otherwise, the value will be the IPST value (see RAND:ATU:ATU0:IPST). This micro-order only makes setup to the PDR register.

Instruction Processor State

IPS

7

The Instruction Processor (IP) state sources the CPD Bus, as follows.

CPD Bits	IP State Bits	Description
28	XCTFLG	Bit indicating that the current macroinstruction was the result of an XCT instruction
29	ION	Master interrupt mask bit
30-31	^LPCX[0-1]	Length of currently executing instruction

This is a slow source.

I/O Controller Data Register

IOC

8

The data register from the I/O controller sources the CPD Bus. This micro-order transfers data from peripherals and the I/O controller directly to the CPU. This is a slow source. See the I/O Protocols section of Chapter 2.

ATU Diagnostic Register

ATD

9

The ATU diagnostic register sources the CPD Bus. See the ATU Diagnostic Register section of Chapter 2. This is a slow source.

Logical Address Register

LAR

A

The Logical Address Register (LAR) sources the CPD Bus. LAR drives the bus fast enough so that it can source the integer ALU. The result from the ALU can drive the CPM Bus to the Address Generator during the same microinstruction cycle.

ATU State

ATS

B

The value on the CPD Bus is the Address Translation Unit (ATU) State. See the ATU state section of Chapter 2 and RAND:ATU:ATU0:LATS. This is a slow source.

SCP Instruction Register

CIR

C

The System Control Processor's instruction register sources the CPD Bus. This is a slow source.

SCP Data Register

CDR

D

The System Control Processor's data register sources the CPD Bus. This is a slow source.

Address Generator

AGA

E

The Address Generator's register-file A-port sources the CPD Bus. This micro-order cannot be used during IPOP.

Zero

ZER

F

Zeros are driven onto the CPD Bus. Unlike N, this micro-order loads PDR.

RAND—Random Micro-orders

The RAND field contains general micro-orders, as well as additional micro-orders for the IALU, the FPU, and the ATU. Micro-orders in the RAND field occur in four possible modes. The mode is specified by the first subfield (RM) in the RAND field. Table 3-8 shows the micro-orders in the RM field:

Table 3-8. RM Field Micro-orders

Mnemonic	Value	RAND Mode
GN	0	General (GEN)
AT	1	Address Translation Unit (ATU)
XC	2 CIB=0	Fixed-point (FIX), Carry-In Base is CARRY
XZ	2 CIB=1	Fixed-point (FIX), Carry-In Base is zero
FL	3	Floating point (FLT)

Note that FIX mode is invoked by either the XC or XZ micro-orders. These micro-orders also set the CIB field.

Each RAND mode has a corresponding set of subfields. The formats for each mode are shown in Figure 3-3.

RM Field	Corresponding Format			
GEN 2 (00)	REG0 5	REG1 2	SPAD 2	
ATU 2 (01)	ATU0 5	ATU1 2	SPAD 2	
FIX 2 (10)	CIB 1	COVS 4	LOAD 2	SPAD 2
FLT 2 (11)	SGN 3	EXP 3	SCNT 3	

Figure 3-3. RAND Mode Formats

RAND:GEN—General Random Micro-orders

The RAND:GEN portion of the microword contains the following fields:

- *REG0*—specifies general and ACS and ACD operations.
- *REG1*—specifies register control operations.
- *SPAD*—specifies scratch pad operations.

RAND:GEN:REG0—General/ACSR/ACDR Micro-orders

The RAND:GEN:REG0 field has micro-orders for general operations and for control of the ACSR and ACDR registers. The ACSR and ACDR micro-orders must be coded at least one cycle before the registers are used for addressing.

No Operation

N

0

No operation is performed.

Write Console Data

CDW

1

This micro-order gates the least-significant sixteen bits of the CPD Bus to the System Control Processor: CPD[16-31] goes to CDR[0-15].

Increment ACSR

INCS

4

INCS increments ACSR[2-3] by one. Note that ACSR[0-1] are unchanged.

Decrement ACSR

DECS

5

DECS decrements ACSR[2-3] by 1. Note that ACSR[0-1] are unchanged.

Load ACSR

LDAS

6

The Accumulator Source Register is loaded from the ID Bus (ID[28-31]).

Force ACSR

FRCS

7

The Accumulator Source Register is set to 'E' (hexadecimal).

Increment ACDR

INCD

8

INCD increments ACDR[2-3] by one. Note that ACDR[0-1] are unchanged.

Decrement ACDR

DECD

9

DECD decrements ACDR[2-3] by one. Note that ACDR[0-1] are unchanged.

Load ACDR

LDAD

A

The Accumulator Destination Register is loaded from ID[24-27].

Force ACDR

FRCD

B

The Accumulator Destination Register is set to 'F' (hexadecimal).

Force CARRY Bit

FCY

C

This micro-order forces the CARRY bit onto the CPM Bus when the A output of the integer register file sources that bus. CARRY goes to CPM0 if FLAG1=1; to CPM16 if FLAG1=0.

Enable Wide Skips

WSKP

D

This micro-order enables wide (32-bit) skips. The skip itself is dependent on test conditions coded in the CNST field. The tests are coded in the same way as those in the TSEL field. Note that they can be wide or narrow, depending on FLAG3. WSKP and the test conditions can be coded only at IPOP and cannot be coded with a memory complete. The table below lists the test conditions.

Mnemonic	**Value**	**Description**
CRY	0	Test the carry-out from the integer ALU (CRY0 or CR16 for FLAG3 = 1 or 0). The test is true if CRY# = 1.
NCRY	1	Test the carry-out from the integer ALU (CRY0 or CR16 for FLAG3 = 1 or 0). The test is true if CRY# = 0.
SGE	2	Compare the signed S input to the signed R input (bits 0-31 or 16-31 for FLAG3 = 1 or 0). The test is true for S >= R. For this test to work you must perform a subtract on the previous cycle.
NSGE	3	Compare the signed S input to the signed R input (bits 0-31 or 16-31 for FLAG3 = 1 or 0). The test is true for S < R. For this test to work you must perform a subtract on the previous cycle.
FZR	4	The test is true if the integer ALU output (F) equals zero (F[0-31] or F[16-31] for FLAG3 = 1 or 0).
NFZR	5	The test is true if the integer ALU output (F) does not equal zero (F[0-31] or F[16-31] for FLAG3 = 1 or 0).

Load ACSR and ACDR

LDSD

E

ACDR[0-3] are loaded from ID[24-27] and ACDS[0-3] are loaded from ID[28-31].

Force ACSR and ACDR

FRSD

F

The ACSR is set to ‘E’ (hexadecimal) and the ACDR is set to ‘F’ (hexadecimal).

Modify Flag Set 0

MFS0

10

MFS0 can modify flags 0, 1, 2, and 3, according to the coding in the CNST field.

Modify Flag Set 1

MFS1

11

MFS1 can modify flags 4, 5, 6, and 7, according to the coding in the CNST field.

CNST Field with MFS0 and MFS1

For the REG0 micro-orders MFS0 and MFS1, two bits in the CNST field specify what is to be done to each flag. Table 3-9 shows how these two bits are coded.

Table 3-9. CNST Microorders for RAND MFS0 and MFS1

Mnemonic	Value	Description
N	0	No operation to flag
S	1	Set flag to 1
C	2	Clear flag to 0
T	3	Toggle flag

For these micro-orders, the microassembler codes four micro-orders in the CNST field, rather than one. For example, to set flags 0-3 to 1,0,1,0, you would code:

```
    MODIFY_FLAGS_0123 (SET,CLEAR,SET,CLEAR)
```

The assembler would code the micro-order MFS0 in RAND:GEN:REG0, and the micro-orders S,C,S,C in the CNST field:

```
                  Bit  0   1   2   3   4   5   6   7
                      +---+---+---+---+---+---+---+---+
        CNST:         | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
                      +---+---+---+---+---+---+---+---+
                      |MFLG0: |MFLG1: |MFLG2: |MFLG3: |
                         S       C       S       C
```

MFLG0, MFLG1, MFLG2 and MFLG3 are two-bit subfields of CNST that correspond to the flags of the specified set.

Accumulate Test Results into Flags 4 and 6

AF46

12

AF46 puts test results into Flag 4 and Flag 6. Before the results are stored, they can be manipulated by codes in the CNST field.

Accumulate Test Results into Flags 5 and 7

AF57

13

AF57 puts test results into Flag 5 and Flag 7. Before the results are stored, they can be manipulated by codes in the CNST field.

CNST Field with Micro-orders AF46 and AF57

For the REG0 micro-orders AF46 and AF57, two 3-bit sections of the CNST field are used, one for each flag to be manipulated. Table 1 shows how these bits are coded.

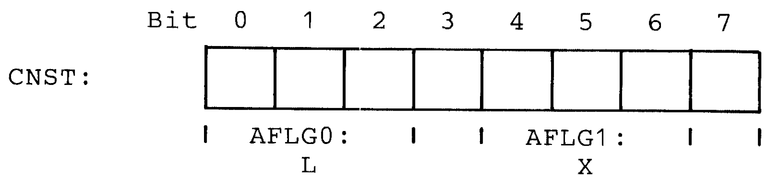
Table 3-10. CNST Microorders for RAND AF46 and AF57

Mnemonic	Value	Description
N	0	No operation to flag
X	1	XOR test to flag
S	2	Set flag to 1
A	3	AND test to flag
C	4	Clear flag to 0
O	5	OR test to flag
T	6	Toggle flag
L	7	Load test into flag

For these micro-orders the microassembler codes two micro-orders in the CNST field, rather than one. For example, to load a test result into flag 4 and invert flag 6 if the test result is 1, you would code:

```
MODIFY_FLAGS_46_WITH_TEST (LOAD XOR)
```

The assembler would code the micro-order L in the AFLG0 subfield of CNST and X in the AFLG1 subfield.



Load Flags From CPD

LFLG

14

LFLG loads the flags from CPD[16-23] (inverted). This micro-order does not load PDR.

Skip on False Test

SKFT

15

Skip over the next word in the instruction stream if the test selected this cycle is false. The microroutine must IPOP the next cycle in order for the skip to operate properly.

Even Parity

EPAR

16

Select even parity for WCS next cycle.

Load SPAR from Constant Register

SPCN

17

Designate CON[0-7], which contains the value in the CNST field of the micro-order, for loading SPAR.

Load Least Significant SPAR bits from IY Bus

SPY4

18

Designate the IY Bus for loading SPAR. SPAR[4-7] come from IY[28-31]; SPAR[0-3] remain unchanged.

SPCN and SPY4—SPAD Micro-orders

The scratch pad address register (SPAR) is one possible address source for the scratch pad. By default, SPAR is loaded from IY[24-31]; however, the SPCN and SPY4 micro-orders override this source. Note that these orders only select the input to SPAR; the micro-order RAND:<GEN,ATU,FIX>:SPAD:LS must be coded at the same time in order to load SPAR. These orders must not be coded at IPOP. For WSKBO and WSKBZ, the hardware generates an address that indexes the proper bit mask in the scratch pad (see Appendix H).

No Load PDR

NPDR

1E

Normally, the PDR is loaded whenever the CPD Bus is active. This micro-order prevents that loading.

Extended Clock

XTND

1F

This micro-order extends the microinstruction cycle for an additional cycle.

RAND:GEN:REG1—Register Load Operations

No Op

N

0

No operation takes place.

Append CRE

AC

1

Append the Current Ring of Execution (CRE) bits to the logical address. CRE[1-3] goes to LA[1-3].

Load the Dispatch Register

LD

2

LD loads the dispatch register from CPD[24-31]. The register must be loaded at least one microcycle before it is used.

Load Transfer Register

LT

3

Load the transfer register (TREG) from the CPM Bus: CPM[0-31] go to TREG[0-31].

RAND:GEN:SPAD—Scratch Pad Input Control

The SPAD field RAND micro-orders control the scratch pad, and the micro-orders in the field have the same meaning for the GEN, ATU, and FIX modes. The scratch pad is read when it is enabled onto the ID Bus by the micro-orders ID:SS and ID:SC. The scratch pad is loaded from the IY Bus or the CPM Bus, as selected by the IL field. It cannot be read and written on the same cycle.

No Operation

N

0

The scratch pad is not written to in this cycle; however, it may be read.

SPAR Addresses SPAD

WS

1

Load data into the scratch pad from the IY or CPM Bus, as selected by the IL field. The scratch pad address register (SPAR) addresses the scratch pad.

CON Addresses SPAD

WC

2

Load data into the scratch pad from the IY or CPM Bus, as selected by the IL field. The CON register (which contains the value in the CNST field) addresses the scratch pad.

Load SPAR

LS

3

Load data into the scratch pad address register (SPAR). The default input for SPAR is IY[24-31]. Other inputs are possible using micro-orders in the RAND:GEN:REG0 field.

RAND:ATU—ATU Random Micro-orders

Micro-orders in the RAND:ATU field are the principal micro-orders for the Address Translation Unit. The RAND:ATU portion of the microword contains the following fields:

- *ATU0*—specifies ATU operations.
- *ATU1*—specifies ATU operations.
- *SPAD*—specifies scratch pad operations.

RAND:ATU:ATU0—ATU Operations

No Op

N

0

No operation takes place.

Load the CRE and ESR Registers

LCRE

4

The Current Ring of Execution (CRE) register and the Effective Source Register (ESR) are loaded from the logical address bus: LA[1-3] goes to CRE[1-3] and ESR[1-3].

Start Memory in Mode 0

CM0

5

This micro-order is used for certain types of main memory references: cache-block-crossing reads, cache flushes, and XCT instructions. All of these references deal directly with the system cache and its functioning.

Cache Block Crossing

The MV/10000 main memory system is organized into blocks of four 32-bit double words. When a double-word (32-bit) read crosses a block boundary in the cache, there must be two separate memory reads—one for each 16-bit word in the reference. To code this memory start, use RAND:ATU:ATU0:CM0 along with MEMS:RW. The cache will assemble the two words into a double word and source it to the CPM Bus when the memory complete is coded. Note that no special memory start is necessary for cache-block-crossing writes.

Cache Flush

Normally, the system cache sends data back to main memory only when it is necessary to overwrite a block in the cache. However, it is possible to force the cache to write a block back to main memory arbitrarily. This ability is used for main memory diagnosis: you can move data out of a single main memory block, store it temporarily, and move it back to main memory without accessing any other blocks.

To move data this way, you must code a MEMS:RD micro-order along with CM0, and provide a block address, i.e., one that ends in three zeros. The addressed block will be read out from the cache and subsequently written back to main memory. In the fourth cycle following the memory start, you must code a MEMC:R micro-order; no other MEMC micro-orders are allowed.

XCT

In order to implement the XCT instruction, the opcode to be executed is sent to the Instruction Processor (IP) via the system cache. The following sequence of operations is used for XCT:

1) Code MEMS:WW and ATU:ATU0:CM0 in the same microinstruction.

2) Source the opcode onto the CPM Bus (from an accumulator in the integer ALU) and at the same time code a memory abort (MEMC:A).

3) Wait four cycles and IPOP.

Restore ATU State

LATS

6

This micro-order restores the state for the Address Translation Unit. The Effective Source Ring (ESR) register and the last memory start register (^CPWRITE@, ^CPMODE@[0-2]) are restored from the Logical Address Bus (LA[1-3] and LA[20-23]). Table 3-11 shows the ATU state that is restored.

Table 3-11. ATU Restored State

Bits	Name	Description
1-3	ESR[1-3]	The Effective Source Ring
20	^CPWRITE@	Indicates that the last non-LAT start was a write.
21-23	^CPMODE@[0-2]	The mode bits for the last non-LAT start.

The following table shows the possible values for the CPMODE state field:

Table 3-12. CP Mode Code

Code	Meaning
0	Word reference (16 bits)
1	Low byte reference
2	Double word reference (32 bits)
3	High byte reference
4	Assemble
5	Send execute data
6	Flush cache block
7	No operation

Write SBR

WSBR

7

This micro-order writes the SBR addressed by bits 1-3 of the LA Bus from the CPD Bus (inverted).

Address Page Table Entry with SBR

RSBR

8

This micro-order gates the address of a page table entry (PTE) onto the CPU Physical Address (CPA) Bus. The PTE address is formed from the address portion of the Segment Base Register (SBR) addressed by LAR[1-3] and either LAR[4-12],0 or LAR[13-21],0; the least significant bit of the PTE address is always zero because PTEs are aligned on 32-bit boundaries.

The number of page-table levels determines the specific bits from the LA Bus. Hardware makes this determination from bit 1 of the SBR and automatically inserts the correct bits into the PTE address.

For a one-level page table, the memory reference with this micro-order returns the physical page address corresponding to the current logical page address. The OPTA micro-order can combine the physical page address with the word-in-page offset (LA[22-31]) to produce a correct memory reference.

For a two-level page table reference, the memory returns the address of the second page table, and you must use the LPTA micro-order to get the physical page address.

Low-Order Page Table Addresses Memory

LPTA

9

This micro-order is used to address the second page table after the address from the first page table has been returned. The physical page address of the first page table is returned on CPD, and is combined with bits 13-21 of the Logical Address Register (LAR). The combination of CPD[18-31],LAR[13-21],0, is sourced to the CPA Bus for a memory reference. The memory returns the physical page address that corresponds to the logical page address. The OPTA micro-order combines this page address with the word-in-page offset (LA[22-31]) to form a physical address.

Load the Logical Address Register and Physical Page Address Register

LLAR

A

This micro-order loads the Logical Address Register (LAR) from the logical address Bus and PPAR[8-21] from CPA[8-21]

Load the Modified/Referenced RAM

WRRM

B

This micro-order sends data from CPD[26-27] to the modified and referenced bits addressed by the CPA Bus. CPD26 goes to the mod bit and CPD27 to the reference bit. The bits are addressed by the page address that the ATU gates onto the CPA Bus (CPA[8-21]). This address can be generated by either the address translation cache or the page table entry logic.

Read And Reset Reference Bits

RSRF

D

This RSRF micro-order will read and reset to zero the reference bits for eight pages simultaneously. The following prodedure must be followed to use this micro-order correctly:

1) Place the address for the first page in the eight-page block into the Physical Page Address Register (PPAR). This address must end in three zeros. The address is usually loaded by turning off the ATU, sourcing the physical address onto the LA Bus, and coding the ATU random LLAR.

2) Code RSRF. This sources the reference bits to the CPD Bus. Note that the bits will be ORed with any other data on the CPD Bus. The only possible destination for the reference bits is the PDR register in the integer ALU; there is not enough set-up time for any other registers on the CPD Bus.

3) Get the reference bits from the PDR[24-31]. Bit 24 is the reference bit for the page whose address ends in three zeros. The ATU is now on. ATU0:AOFF must be coded if the ATU should not be on.

Purge the Address Translation Cache

PRGA

E

This micro-order resets all the bits in the validity RAM for the address translation cache. In effect, it returns the cache to an empty state.

Page Table Addresses Memory

OPTA

F

This micro-order takes a page address from a Page Table Entry (PTE) and uses it to address a memory location. The page address itself comes from memory as a result of an RSBR or LPTA micro-order. The full address for memory is assembled from the PTE page address and the logical address's page offset. The page address is sourced to the ATU on the CPD Bus; the logical address is available on the LA Bus. The bits are sourced to the CPA Bus as follows:

```
CPA[8-31] = CPD[18-31],LABUF[22-31]
```

If the test condition in this microinstruction is true, then the ATU cache is also loaded. The cache will contain the physical address on CPA in the location pointed to by the logical address on the LA Bus. Thus, the next time this particular logical address is presented to the ATU, it will hit in the cache and avoid LAT.

Load Instruction Processor State

LIPS

10

This micro-order loads the Instruction Processor (IP) state and has the same effect as IPST on the ATU. The IP state consists of the following registers:

- The Program Counter (PC);
- The Next Program Counter (PCN);

- The LPCX[0-1] register, which contains the length of the currently executing instruction; and
- The XCTFLG, which indicates whether the current macroinstruction resulted from an XCT instruction.

In order to restore state to the IP:

1) Source the value for PCN to the Address Generator's AY Bus and code RAND:ATU:AT0:IPST.

   Note: No memory start should be coded while restoring IP state.

2) Source the value for the PC to the Address Generator's AY Bus and source XCTFLG on CPD28 and ^LPCX on CPD[30-31]. Code RAND:ATU:AT0:LIPS. (ION is not restored by this operation—see ION and IOFF.) If no memory start is coded, the random IPFL should be used to ensure that the IP is flushed and a good translation is provided.

Disable Interrupts

DISI

12

This micro-order disables interrupts *for one macroinstruction.*

Note: Do not code IPOP with this instruction.

Turn ION On

ION

13

This micro-order turns on the ION bit (part of the Instruction Processor state). This enables interrupts. This micro-order disables interrupts for one instruction cycle if ION changes state. NOTE: Do not code IPOP with this instruction.

Turn ION Off

IOFF

14

This micro-order turns off the ION bit (part of the Instruction Processor state). This action disables interrupts.

Note: Do not code IPOP with this instruction.

Instruction Cache Translation

ICAT

16

This micro-order loads the physical page register of the Instruction Processor (IP) with a physical address generated by the ATU. The IP uses the ATU to translate addresses, which the IP then uses to fetch instructions. Specifically, the following takes place:

```
PHY[8-21] = CPA[8-21]
```

Instruction Processor Start

IPST

17

This micro-order loads the Instruction Processor Program Counter (IPPC) and flushes the IP pipeline. The IPPC is loaded with the Current Ring of Execution (CRE) and the value on the AY Bus of the Address Generator. CRE also sources the Logical Address (LA) Bus:

```
IPPC = CRE[1-3],AY[4-31]

LA[1-3] = CRE[1-3]
```

At the same time, the physical page register is loaded with a physical address created by the ATU, as follows:

```
PHY[8-21] = CPA[8-21]
```

Send I/O Command or Data

SIO

18

This micro-order enables a command or data to the I/O controller. The command or data must be on the CPD Bus, and must follow the I/O protocols discussed in Chapter 2.

Force Byte Addressing

BYTE

1A

This micro-order forces byte addressing of main memory, regardless of what kind of memory start was initiated. Note: the LA register is not valid after this random.

Force Word Addressing

WORD

1B

This micro-order forces word addressing of main memory, regardless of what kind of memory start was initiated.

Turn On Address Translation Unit

AON

1C

This micro-order turns on the Address Translation Unit, so that logical addresses are converted into physical addresses before they go to main memory.

Turn Off Address Translation Unit

AOFF

1D

This micro-order turns off the Address Translation Unit. Effectively, all addresses are treated as physical addresses; there is no logical to physical translation.

Don't Load the PDR Register

NPDR

1E

This micro-order inhibits loading of the PDR register. Normally., the PDR is loaded every time the CPD Bus is used. NPDR allows the contents of PDR to remain unchanged regardless of CPD use.

Extend the CP Clock

XTND

1F

This micro-oorder extends the CP clock, which is normally 140 nanoseconds, by two SYS clock cycles to 270 nanoseconds. This micro-order also determines which of two sets of information is captured by the ATU diagnostic register. (See the MEMC:A micro-order, earlier in this chapter.)

RAND:ATU:ATU1—Additional ATU Operations

No Op

N

0

No operation takes place.

Append CRE

AC

1

This micro-order appends the current CRE to the most significant bits of the current logical address as follows:

```
LA[1-3] = CRE[1-3]
```

Increment DEFER Counter

DF

2

This micro-order increments the DEFER counter and, if the test this cycle is false, replaces the current ESR with bits 1-3 of the new logical address. In effect, this micro-order descends one more level in an indirection chain. If the defer counter goes beyond 15, it causes a trap.

When DF is coded, an inward reference is determined by comparing the started address's ring field to the ESR (not the CRE).

Load TREG

LT

3

This micro-order loads the transfer register (TREG) between the CPM and CPD buses from the CPM Bus. Later, TREG can source the CPD Bus.

RAND:ATU:SPAD—Scratch Pad Input Control

The SPAD field RAND micro-orders control the scratch pad, and the micro-orders in the field have the same meaning for the GEN, ATU, and FIX modes. The scratch pad is read when it is enabled onto the ID Bus by the micro-orders ID:SS and ID:SC. The scratch pad is loaded from the IY Bus or the CPM Bus, as selected by the IL field. It cannot be read and written on the same cycle.

No Operation

N

0

The scratch pad is not written to on this cycle; however, it may be read.

SPAR Addresses SPAD

WS

1

Load data into the scratch pad from the IY or CPM Bus, as selected by the IL field. The scratch pad address register (SPAR) addresses the scratch pad.

CON Addresses SPAD

WC

2

Load data into the scratch pad from the IY or CPM Bus, as selected by the IL field. The CON register (which contains the value in the CNST field) addresses the scratch pad.

Load SPAR

LS

3

This micro-order loads data into the scratch pad address register (SPAR). The default input for SPAR is IY[24-31]. Other inputs are possible using micro-orders in the RAND:GEN:REG0 field.

RAND:FIX—Fixed-point Random Micro-orders

The RAND:FIX portion of the microword contains the following fields:

- *CIB*—specifies the carry-in base. This field is set by the same micro-order that sets the RAND:FIX mode (XC or XZ). See the explanation at the beginning of the RAND section of this chapter and under "Carry-In Logic" in Chapter 2.
- *COVS*—controls the CARRY bit as well as OVR and OVK bits.
- *LOAD*—controls the loading of register and narrow or wide conditions in the IALU.
- *SPAD*—specifies scratch pad operations.

RAND:FIX:COVS—Carry, Overflow and Status

COVS field micro-orders load and set the Processor Status Register (PSR) and manipulate the CARRY register.

No Operation

N

0

No operation is performed.

Clear OVR

COVR

1

This micro-order resets PSR1 (OVR) to 0. (Once set, PSR1 remains 1 until COVR resets it.)

Clear OVK

COVK

2

This micro-order resets PSR0 (OVK) to 0.

Set OVK

SOVK

3

Set PSR0 (OVK) to 1.

Load the PSR

LPSR

4

Load the Processor Status Register from the ID Bus: ID[0-3] goes to PSR[0-3].

Load Overflow and Carry

LOVC

5

Set PSR1 (OVR) and CARRY from the results of the current ALU operation. For narrow operations (FLG3=0), OVR=OVR16 and CARRY=CRY16. For wide operations (FLG3=1), OVR=OVR0 and CARRY=CRY0. LOVC will cause an overflow trap if OVR=1 and OVK=1. An overflow trap will reset OVK to zero.

Load CARRY

LCRY

6

Load the CARRY register from carry-out of the integer ALU. For narrow operations (FLG3=0), CARRY=CRY16; for wide operations (FLG3=1), CARRY=CRY0.

Clear CARRY

CLRC

7

Reset the CARRY register to 0.

Load CARRY from R Bus

LDCY

8

Load the CARRY register from the R Bus. For narrow operations, CARRY=R16; for wide operations, CARRY=R0.

Enable ALC Functions

ALC

9

This micro-order enables the ALC skip and no-load logic and determines the ALC carry from bits 10 and 11 of the macroinstruction.

The effect of this micro-order on the CARRY register depends on micro-orders in the IY field. For IY:BR1 and IY:BL1, ALC uses ALC carry as the shift input, and loads CARRY with the shift out.

For other micro-orders in the IY field, CARRY is loaded with ALC carry.

Set the CARRY Register

SETC

A

Set the CARRY register to 1.

RAND:FIX:LOAD—Load Registers

The LOAD field inhibits loading of the register file, forces narrow operations in the ALU, and loads the transfer register.

No Op

N

0

No operation takes place.

Absolute Value

AV

1

Prevent the loading of negative values into the register file. If the FSGN test is true, then the value is not loaded into the register file.

Force Narrow Operations

NA

2

The IALU will operate on 16-bit values only, i.e., FLAG3=0.

Load Transfer Register

LT

3

Load the transfer register (TREG) from the CPM Bus: CPM[0-31] go to TREG[0-31].

RAND:FIX:SPAD—Scratch Pad Input Control

The SPAD field RAND micro-orders control the scratch pad, and the micro-orders in the field have the same meaning for the GEN, ATU, and FIX modes. The scratch pad is read when it is enabled onto the ID Bus by the micro-orders ID:SS and ID:SC. The scratch pad is loaded from the IY Bus or the CPM Bus, as selected by the IL field. It cannot be read and written on the same cycle.

No Operation

N

0

The scratch pad is not written to on this cycle; however, it may be read.

SPAR Addresses SPAD

WS

1

Load data into the scratch pad from the IY or CPM Bus, as selected by the IL field. The scratch pad address register (SPAR) addresses the scratch pad.

CON Addresses SPAD

WC

2

Load data into the scratch pad from the IY or CPM Bus, as selected by the IL field. The CON register (which contains the value in the CNST field) addresses the scratch pad.

Load SPAR

LS

3

Load data into the scratch pad address register (SPAR). The default input for SPAR is IY[24-31]. Other inputs are possible using micro-orders in the RAND:GEN:REG0 field.

RAND:FLT—Floating-Point Random Micro-orders

The RAND:FLT portion of the microword contains the following fields:

- *SGN*—controls the floating-point sign logic.
- *EXP*—controls the floating-point exponent logic.
- *SCNT*—specifies operations that control the hex-shifter shift count.

RAND:FLT:SGN—Floating-Point Sign

The floating-point sign is the value sourced to FD0.

No-op

SA

0

The sign is taken directly from the SA register. No operations are performed on it.

Sign Equal FA0

MOV

1

The sign is equal to FA0 (bit zero of the FA Bus).

Sign Equal FA0-

NEG

2

The sign is equal to the inverse of FA0.

Truncate and Invert

TRI

3

Truncate the bottom guard digit of the FS Bus if the Floating-Point Status Register round bit (FPSR8) is equal to zero. The bottom guard digit is FS[68-71] if FLAG2=1 or FS[23-39] if FLAG2=0. Load the SA register from FA0 and the SB register from FB0. The value loaded into SA is inverted if the SWAP bit in the floating-point state register is set. The FPSR must be set at least two cycles before this micro-order is executed.

This random is used during signed mantissa subtraction to determine the correct sign of the result.

Sign Equals 0

ZER

4

The new sign will be 0 (positive number).

Sign Equals SA EXOR SB

XOR

5

The sign equals SA XOR SB. This means that if the signs are the same, the sign will be positive (0), and if they are different, the sign will be negative (1). Thus this micro-order produces the correct sign for the multiplication or division of floating-point numbers.

Load SA and SB

LAB

6

The SA register is set to FA0 and the SB register to FB0. The previous value of SA becomes the new sign.

Truncate

TRN

7

Truncate the bottom guard digit of the FS Bus if the Floating-Point Status Register round bit (FPSR8) is equal to zero. The bottom guard digit is FS[68-71] if FLAG2=1 or FS[23-39] if FLAG2=0. Load the SA register from FA0 and the SB register from FB0. The FPSR must be set at least two cycles before this micro-order is executed. This random is used during signed mantissa addition to determine the correct sign of the result.

RAND:FLT:EXP—Floating-Point Exponent

The EXP field controls the exponent logic for floating-point numbers. The exponent is the value that is sourced to FD[1-7].

The micro-orders ACW, ACA, and ACN use the mantissa overflow (MOF). This is not the same as the MOF bit in the FPSR, which is a flag for macroinstructions. Mantissa overflow can be corrected by shifting the mantissa right by one hex digit between the mantissa ALU and the FD Bus; this shifts the overflow digit back into the mantissa proper, so that the most significant digit of the mantissa is 0001. The exponent is adjusted for this shift by adding in the MOF.

No-op

N

0

The exponent is the unaltered value in the Exponent Working Register (EWR).

Subtract 64

S64

0

The exponent is the value in the EWR, minus 64. In order for this micro-order to work you must code it in conjunction with FX:X64. The MV/10000 processor uses excess-64 notation internally. This micro-order is used to correct exponent addition:

```
(A+64) + (B+64) - 64 = (A+B) + 64
```

where A and B are the true exponent values.

Load FA

LAX

1

The exponent is the value from the FA Bus that is sourced to the Exponent ALU. This value is equal to 0,0,FA[1-7].

Subtract Normalize

SNM

2

The exponent is set to the value of the EWR minus MAG[0-3].

Subtract

SUB

3

The exponent is the value from the FA Bus minus the value from the FB Bus. This micro-order is normally used for a floating-point divide operation.

Correct EWR

ACW

4

Add the MOF to the exponent.

Add 64

A64

4

The exponent is the value in the EWR plus 64. In order for this micro-order to work, you must code it in conjunction with FX:X64. This micro-order is used to correct exponent subtraction:

```
(A+64) - (B+64) + 64 = (A-B) + 64
```

where A and B are the true exponent values.

Correct FA

ACA

5

Add the MOF to the FA source.

Correct and Normalize

ACN

6

The exponent is the exponent working register, plus MAG, plus the MOF. This micro-order normalizes floating-point numbers after an arithmetic operation. MAG is set to minus the number of leading hexadecimal zeros in the mantissa by using the SCNT:LZD random, and the exponent is added to the value in MAG. At the same time, the mantissa should be left-shifted in four-bit shifts to remove the zeros. The result is a number with the smallest possible exponent and no leading zeros in the mantissa (if the result is not zero). NOTE: MAG will be a negative value, which will produce the proper left shift in the hex shifter.

Add

ADD

7

Add the FA and FB sources. This micro-order is used for a floating-point multiply operation. After the mantissas are multiplied, the result can be normalized.

RAND:FLT:SCNT—Shift Count Control

The SCNT field controls the MAG register and, therefore, determines the size of the shift of the hex shifter. SCNT also sets the SWAP bit in the STATE register, which reverses the A and B outputs of the register file.

All of the micro-orders below except N, RST, and CMP will set the SWAP and X.GT.15 bits to 0. (Note that IPOP also clears these bits.) The CMP and RST micro-orders change the compare bits (STATE[2-3]).

Note that MAG must be loaded at least one cycle before the shift uses it.

No Load MAG

N

0

MAG and the swap bit are not loaded.

Restore STATE

RST

1

Load MAG[0-3], SWAP, X.GT.15, and the compare bits from the FA Bus.

Compare

CMP

2

Perform a prescale compare operation on the operands in preparation for signed mantissa addition or subtraction. If FA < FB, set the SWAP bit. If 'R' is coded in the FWR field, FR as the source for the working register when FA < FB; select FS when FA >= FB. For CMP to work correctly, a subtract operation must be coded on both the exponents and mantissas of FA and FB. This micro-order loads MAG with the absolute value of the exponent difference, which is used in the next cycle to prescale (right shift) the operand loaded in the working register.

Load Constant

LCN

3

Load a constant into MAG[0-3] from the IY field. The following constants are defined for the IY field when LCN is used:

R0 to R15	These are the appropriate values for right shifts from 0 to 15 hex digits. (1-15)
L0 to L15	These are the appropriate values for left shifts from 0 to 15 hex digits. (0,15-1)

These values in the IY field do *not* specify the direction of the shift. (The shift direction is specified by the FS field when the shift is desired.) The use of left and right magnitudes in the IY field is strictly for the convenience of the programmer. These codes are provided because the magnitude of the shift in the proper direction is not a straightforward mapping.

First Nibble Zero

FNZ

4

Set MAG to -1 if the first nibble of the mantissa is zero; otherwise, set MAG to 0. This micro-order is used for multiply normalization. Note that no mantissa overflow (MOF) is possible for a multiply operation.

Divide Prescale

DVP

5

Set MAG to 1 if there is a carry-out from the mantissa; otherwise, set MAG to 0.

Load Exponent Fbus

LEF

6

Load MAG[0-3] from EF[4-7] (the output of the Exponent ALU).

Leading Zero Detection

LZD

7

Detect leading zeros and mantissa overflow (MOF). If MOF occurs, the output of the mantissa ALU is shifted right by four bits. Do not code FOP:TAD while using the LZD random to detect leading zeros.

Because of timing, leading zero detection is done for a maximum of two hex digits during the cycle when the LZD random is coded. This value will be wrong if there are more than two leading zeros; however, if LZD is coded again for the cycle when the value is used, then the correct value will be calculated. Coding any other SCNT micro-orders except N and LZD after the LZD random will cancel any later correction attempt.

Integer ALU Micro-orders

The Integer ALU portion of the microword contains the following fields:

- *IA*—specifies the A output of the integer register file.
- *IB*—specifies the B output of the integer register file.
- *ID*—specifies the source for the ID Bus.
- *RS*—specifies the input sources for the ALU.
- *IOP*—controls the operation of the ALU.
- *IY*—specifies the source for the IY Bus.
- *IL*—controls the loading of the register file.

IA and IB—Integer Register File Addressing

The IA and IB fields control the addressing for the integer register file. Note that, when floating-point numbers are being manipulated, these same fields control the addressing for the floating-point register file. Each micro-order specifies a particular register. The following micro-orders can be used in the IA and IB fields:

Macroinstruction Accumulator 0

AC0

0

This is the programmer-visible Accumulator 0. Assembly code can access it. At IPOP, this register must contain the same value as Accumulator 0 in the AG register file.

Macroinstruction Accumulator 1

AC1

1

This is the programmer-visible Accumulator 1. Assembly code can access it. At IPOP, this register must contain the same value as Accumulator 1 in the AG register file.

Macroinstruction Accumulator 2

AC2

2

This is the programmer-visible Accumulator 2. Assembly code can access it. At IPOP, this register must contain the same value as Accumulator 2 in the AG register file.

Macroinstruction Accumulator 3

AC3

3

This is the programmer-visible Accumulator 3. Assembly code can access it. At IPOP, this register must contain the same value as Accumulator 3 in the AG register file.

Wide Frame Pointer

FP

4

Register 4 contains a copy of the Wide Frame Pointer, which is stored at page 0, address 10_{16}. By convention, these copies are not always the same. The register contains the valid copy.

Wide Stack Limit

SL

5

Register 5 contains a copy of the Wide Stack Limit for the current ring, which is stored at page 0, 14_{16}. By convention, these copies are always identical.

Wide Stack Base

SB

6

Register 6 contains a copy of the Wide Stack Base for the current ring, which is stored at page 0, 16_{16}. By convention, these copies are always identical.

Minus One

M1

7

Register 7 contains a constant -1 (all ones).

General Register 0

GR0

8

This is a general register for use by the microprogrammer. It has no assigned meaning.

General Register 1

GR1

9

This is a general register for use by the microprogrammer. It has no assigned meaning.

General Register 2

GR2

A

This is a general register for use by the microprogrammer. It has no assigned meaning.

General Register 3

GR3

B

This is a general register for use by the microprogrammer. It has no assigned meaning.

General Register 4

GR4

C

This is a general register for use by the microprogrammer. It has no assigned meaning.

General Register 5

GR5

D

This is a general register for use by the microprogrammer. It has no assigned meaning.

Register Addressed by ACSR

SRC

E

This micro-order takes the address for the register file from the Accumulator Source Register (ACSR).

Register Addressed by ACDR

DES

F

This micro-order takes the address for the register file from the Accumulator Destination Register (ACDR).

ID--ID Bus Source Control

SPAD Addressed by SPAR

SS

0

The scratch pad outputs to the ID Bus. The scratch pad address register (SPAR) addresses the scratch pad.

SPAD Addressed by CON

SC

1

The scratch pad outputs to the ID Bus. The CON register (which contains the value from the CNST field of the micro-order) addresses the scratch pad.

SPAR, ACD, and ACS to ID Bus

MS

2

The SPAR, ACD, and ACS registers source the ID Bus in the following manner:

```
ID[0-15] = -0- ; ID[16-31] = SPAR[0-7],ACD[0-3],ACS[0-3]
```

ACS to ID Bus

AS

3

The ACS register sources the ID Bus in the following manner:

```
ID[0-27] = 0 ; ID[28-31] = ACS[0-3]
```

Constant Register to ID Bus

CN

4

The constant register (CON) sources the ID Bus in the following manner:

```
ID[0-23] = 0 ; ID[24-31] = CON[0-7]
```

CPD Bus Register—PDR to ID Bus

PD

5

The PDR sources the ID Bus. This register is used to transfer data off the CPD Bus. Additionally, it can be used as a counter. (See TSEL:<CNT4 CNT8>.)

B Port to ID Bus

BR

6

The B output port of the integer ALU register file sources the ID Bus.

Zero

ZR

7

The ID Bus is forced to zero.

RS—ALU Input Multiplexer Control

The RS field controls the inputs to the ALU.

ID Bus and A Port

DA

0

The ID Bus is the input to the R side of the ALU; the A port of the register file is the input to the S side.

A Port and ID Bus

AD

1

The A port of the register file is the input to the R side of the ALU; the ID Bus is the input to the S side.

CPD Bus and A Port

CA

2

The CPD Bus is the input to the R side of the ALU; the A port of the register file is the input to the S side.

CPD Bus and ID Bus

CD

3

The CPD Bus is the input to the R side of the ALU; the ID Bus is the input to the S side.

IOP—ALU Control and Shift Magnitude

The IOP field has two separate functions: it controls the ALU and it determines the magnitude of the hex shift specified by the IY field. The codes for the hex shift will be found with the appropriate micro-orders in the IY field. The ALU control micro-orders determine combinations of the R and S inputs of the ALU and also control the Carry-In Base (CIB) polarity.

Logical AND

AND

0

The logical AND of the R and S inputs to the integer ALU.

Logical OR

OR

1

The logical OR of the R and S inputs to the integer ALU.

Logical AND with Complement

ANC

2

The logical AND of the R input and the complement of the S input.

Exclusive OR

XOR

3

The exclusive OR of the R and S inputs to the integer ALU.

Addition with Complement

CSR

4

The addition of the complemented R input to the S input with the CIB complemented (this is a twos complement subtract if CIB is zero):

```
R' + S + CIB'
```

Addition with Complemented Carry-in

CAD

5

The addition of the R and S inputs with the CIB complemented:

```
R + S + CIB'
```

Addition with Complement

SMR

6

The addition of the complemented R input to the S input with the uncomplemented CIB:

```
R' + S + CIB
```

Addition

ADD

7

The addition of the R and S inputs with the uncomplemented CIB:

```
R + S + CIB
```

IY --IY Bus Source

The IY field specifies the source for the IY Bus. As a result, it also controls the hex and bit shifters, as well as certain ALU output functions. The IY Bus can be wide (32 bits) or narrow (16 bits), depending on FLAG0. The effect of this is explained for each micro-order.

Append PSR

PSR

0

The Processor State Register (PSR) sources the most-significant bits of the Y Bus: Y[0-31] comes from PSR[0-3],F[4-31].

For narrow operations (FLAG0=0), the source data is sign extended if it goes to the address generator, SPAD, or the integer register file; for other destinations, the most-significant sixteen bits are one filled.

Pass the F Bus to the Y Bus

PASS

1

The current value of F (ALU output) passes unchanged to the Y Bus: F[0-31] goes to Y[0-31].

For narrow operations (FLAG0=0), the source data is sign extended if it goes to the address generator, SPAD, or the integer register file; for other destinations, the most-significant sixteen bits are one filled.

Edited Commercial Data

EDT

2

The edited translation of the least-significant byte on the A Bus goes to the Y Bus: Y[0-31] comes from (F[0-27],TRANS(A[24-31]=28-31)). This micro-order must be used with the TSEL:COM2 micro-order and the CNST field.

For narrow operations (FLAG0=0), the source data is sign extended if it goes to the address generator, SPAD, or the integer register file; for other destinations, the most-significant sixteen bits are one filled.

Swap Bytes

BSW

3

Interchange the two least significant bytes from the ALU output: Y[0-31] comes from F[0-15],F[24-31],F[16-23].

For narrow operations (FLAG0=0), the source data is sign extended if it goes to the address generator, SPAD, or the integer register file; for other destinations, the most-significant sixteen bits are one filled.

Bit Shift Left—One Filled

BL1

4

Shift the ALU output (F) left one bit. Shift in a one, unless RAND:<XC XZ>:COVS:ALC is coded. In that case, the least-significant bit will be the ALC carry, and the CARRY register will be loaded with the shifted-out bit.

For narrow operations (FLAG0=0), the bit-shifter output is sign extended if it goes to the address generator, SPAD, or the integer register file; for other destinations, the most-significant sixteen bits are one filled.

Bit Shift Left—Zero Filled

BL0

5

Shift the ALU output (F) left one bit. Shift in a zero, unless RAND:<XC XZ>:COVS:ALC is coded. In that case, the least-significant bit will be the ALC carry, and the CARRY register will be loaded with the shifted-out bit.

For narrow operations (FLAG0=0), the output of the bit shifter is sign extended if it goes to the address generator, SPAD, or the integer register file; for other destinations, the most-significant sixteen bits are one filled.

Bit Shift Right—One Filled

BR1

6

Shift the ALU output (F) right one bit. Shift in a one, unless RAND:<XC XZ>:COVS:ALC is coded. In that case, the most-significant bit will be the ALC carry, and the CARRY register will be loaded with the shifted-out bit.

For narrow operations (FLAG0=0), the bit-shifter output is sign extended if it goes to the address generator, SPAD, or the integer register file; for other destinations, the most-significant sixteen bits are one filled.

Bit Shift Right—Zero Filled

BR0

7

Shift the ALU output (F) right one bit. Shift in a one, unless RAND:<XC XZ>:COVS:ALC is coded. In that case, the most-significant bit will be the ALC carry, and the CARRY register will be loaded with the shifted-out bit.

For narrow operations (FLAG0=0), the shifter output is sign extended if it goes to the address generator, SPAD, or the integer register file; for other destinations, the most-significant sixteen bits are one filled.

Hex Shift Right—Zero Filled

HR0

8

The hex shifter shifts the data selected by ALU-input multiplexer R to the right in 4-bit increments and shifts zeros in from the left. The IOP field determines the magnitude of the shift. The IOP codes for a right shift are:

Mnemonic	**Value**	**Description**
R1	0	Hex shift right 1.
R2	1	Hex shift right 2.
R3	2	Hex shift right 3.
R4	3	Hex shift right 4.
R5	4	Hex shift right 5.
R6	5	Hex shift right 6.
R7	6	Hex shift right 7.
@R	7	Hex shift right (ACSR[1-3]+1). ACSR = X111 gives a result of zero.

FLAG0=0 → zero extended

Hex Shift Left—Zero Filled

HL0

9

The hex shifter shifts the data selected by ALU-input multiplexer R to the left in 4-bit increments and shifts zeros in from the right. The IOP field determines the magnitude of the shift. The IOP codes for a left shift are:

Mnemonic	Value	Description
@L	0	Hex shift left by ACSR[1-3]; ACSR = X000 gives a result of zero.
L1	1	Hex shift left 1.
L2	2	Hex shift left 2.
L3	3	Hex shift left 3.
L4	4	Hex shift left 4.
L5	5	Hex shift left 5.
L6	6	Hex shift left 6.
L7	7	Hex shift left 7.

FLAG0=0 → zero extended

Hex Rotate Right

HRT

A

The hex shifter rotates the data selected by ALU input multiplexer R to the right in 4-bit increments. The IOP field determines the magnitude of the shift. The IOP codes for rotation are:

Mnemonic	Value	Description
R1	0	Hex rotate right 1.
R2	1	Hex rotate right 2.
R3	2	Hex rotate right 3.
R4	3	Hex rotate right 4.

Mnemonic	Value	Description
R5	4	Hex rotate right 5.
R6	5	Hex rotate right 6.
R7	6	Hex rotate right 7.
@R	7	Hex rotate right (ACSR[1-3]+1). ACSR = X111 gives a result of zero.

FLAG0=0 → zero extended

Byte Sign Extension

BSX

B

Extend the sign bit of the least-significant byte on the R Bus: R24 is repeated 24 times, so that Y[0-31] comes from 24<R24>,R[24-31].

Word Zero Extension

WZX

C

The most-significant word on the Y Bus is zero filled; the least-significant word comes from the least-significant word of the R Bus. Y[0-31] equals 16<0>,R[16-31].

Word Sign Extension

WSX

E

Extend the sign bit of the least-significant word on the R Bus. Y[0-31] equals 16<R16>,R[16-31].

IL—Integer Register File Input

The IL field controls the input multiplexer for the integer register file and the scratch pad. It also enables loading of the register file. The input port address is the same as the B output port address, i.e., the IB field specifies it.

No Operation

N

0

This micro-order disables the multiplexer. Note that this micro-order is identical to NM. Different mnemonics are provided for the convenience of the microcoder, to indicate differences in intention.

Select CPM Data

NM

0

The multiplexer selects the CPM Bus, but the register file is not loaded. This micro-order allows the loading of CPM data into the scratch pad.

Select IY Data

NY

1

The multiplexer selects the IY Bus, but the register file is not loaded. This micro-order allows the loading of IY data into the scratch pad.

Select and Load CPM Data

M

2

The multiplexer selects the CPM Bus and loads the data into the register addressed by IB.

Select and Load IY Data

Y

3

The multiplexer selects the IY Bus and loads the data into the register addressed by IB.

Floating-Point ALU Micro-orders

The Floating-point Unit portion of the microword contains the following fields:

- *FR*—specifies the source of the FR Bus.
- *FS*—specifies the source of the FS Bus.
- *FOP*—controls the floating-point ALU operations.
- *FWR*—specifies the input to the working register.
- *FCW*—specifies the write address for the FPU register file.
- *FL*—specifies the input to the FPU register file.
- *FRG*—controls the loading of FPU registers.
- *FX*—controls the excess-64 exponent correction.

The FR, FS, FOP and FWR fields are also used as the system-wide CNST field. Therefore, use of the floating point unit generally prohibits use of the constant field and vice-versa.

FR—FR Bus Source

The FR Bus drives the R input of the mantissa ALU and can source the working register. The FR field determines the source for the FR Bus.

FA Bus

FA

0

FA[8-71] is the source for the FR Bus. You can use this micro-order to move values from the register file to the ALU.

Multiplier Partial Product

MP

1

The multiply ALU M[8-71] sources the FR Bus. This micro-order brings partial products to the mantissa ALU, where they are added into the partial sum during multiplication.

Round Bit

RB

2

The rounding logic calculates the rounding bit: either 31 or 63 (FLAG2=0 or 1). This bit represents either a truncation of the final result or an unbiased rounding, depending on whether FPSR8 is set. The bit is driven onto the FR Bus and may be added to the working register; the result is a correctly rounded mantissa. If the first four bits of the mantissa are zero, the FDI Bus is zeroed (true zero). The round bit is usually added to the result during the normalization cycle.

Divide Partial Remainder

DR

3

The DPR register (DPR[8-71]) sources the FR Bus. Note that passing data through the DPR register produces a 1-bit left shift. This shift is used during the division algorithm to move the current partial remainder into position for the next subtraction.

During a divide operation, both FCW and IB *must* be coded with the register that contains the divisor.

FS—FS Bus Source

The FS Bus sources the S input of the mantissa ALU. It can also source the working register. The sources to the FS Bus are controlled by the FS field. Because the FS Bus is the only output for the hex shifter, the FS field also includes the coding that determines whether the hex shifter shifts right or left. Note that the size of the shift is determined by MAG[0-3], which in turn has been determined by the RAND:FLT:SCNT field. FS[40-72] is zeroed for single-precision operations.

FB Bus

FB

0

The FB Bus sources the FS Bus. The FB Bus is the output bus for the B port of the register file.

Zero

ZR

1

The FS Bus is zeroed; no data is driven onto the bus at this time.

Right Shift

RS

2

The output of the working register to the FS Bus is shifted right. The value in MAG represents the number of hexadecimal digits shifted.

Left Shift

LS

3

The output of the working register to the FS Bus is shifted left. The value in MAG represents the number of hexadecimal digits shifted.

FOP—Mantissa Operations

The FOP field controls the operations of the mantissa ALU. The inputs to the ALU are the FR and FS buses. The output goes to the FF Bus.

Except during division, the mantissa ALU adds and subtracts unsigned numbers. A negative mantissa result is meaningful only for comparison purposes. During floating-point addition or subtraction, a prescale operation is performed to ensure that the smaller operand is on the FR Bus and the result of mantissa subtraction is meaningful.

The FF Bus sources the FD Bus either directly or right shifted by four bits.

Conditional Add

ADD

0

Add the values on FR and FS together if SA is equal to SB (i.e., if the signs are the same); otherwise, subtract FS from FR. A prescale operation must be performed on the FR operand to obtain a meaningful result. See the FLT:SCNT:CMP random description earlier in this chapter for details.

Conditional Subtract

SUB

1

Subtract FS from FR if SA is equal to SB (i.e., if the signs are the same); otherwise, add them. A prescale operation must be performed on the FR operand to obtain a meaningful result. See the FLT:SCNT:CMP random description for details.

Unconditional Add

TAD

2

Add FR to FS regardless of the signs of the numbers.

Unconditional Subtract

TSB

3

Subtract FS from FR regardless of the signs of the numbers. TSB is coded for prescale compare and divide operations. During divide operations, TSB will cause the ALU either to add or subtract, depending on the sign of the partial remainder from the previous operation.

FWR —Working Register Input

The FWR field controls the input multiplexer for the working register. Note that loading of the working register is controlled by micro-orders in the FRG field; FWR only selects the source to be loaded.

FS Bus

S

0

Select the FS Bus to source the working register.

FR Bus

R

1

Select the FR Bus to source the working register.

This micro-order can be overridden if the CMP micro-order is coded in the RAND:FLT:SCNT field. In this case, if the value on FR is greater than or equal to the value on FS (FR >= FS), then FS is the source for the working register.

Left Shift

Q

2

Shift the data from the working register left one bit. Put the value of the Q bit, from a divide operation, into the least-significant bit of the working register. (For single-precision, the least-significant bit is 39; for double-precision, 71.) The Q bit is derived from a subtraction or addition operation that produced a partial remainder. After a series of subtractions or additions, the bits that have been shifted into the working register will be the quotient from the complete divide operation.

During division cycles, TBS is coded in the FOP field. The division hardware will perform an addition or subtraction based upon the value of the partial remainder from the previous operation.

FD Bus

D

3

Select the FD Bus to source the working register.

FCW—Floating-Point Register Write Address

The FCW field specifies the write address for the floating-point register file. (The read addresses for the register file are specified by the IA and IB fields, the same as for the integer register file.) The input to the register file is the FDI Bus, and the FCW field specifies which register the data from the FDI Bus will go into.

The floating-point accumulators and general registers correspond in their addresses to the integer accumulators. This is useful for operations, such as integer division and multiplication, that use both the integer and floating-point ALUs.

The mnemonics listed below can also be used in the IA and IB fields when those fields are used to address the output ports of the floating-point register file.

FP Accumulator 0

FP0

0

FP0 addresses floating-point accumulator (FPAC) 0. This is the macroprogram accumulator.

FP Accumulator 1

FP1

1

FP1 addresses floating-point accumulator (FPAC) 1. This is the macroprogram accumulator.

FP Accumulator 2

FP2

2

FP2 addresses floating-point accumulator (FPAC) 2. This is the macroprogram accumulator.

FP Accumulator 3

FP3

3

FP3 addresses floating-point accumulator (FPAC) 3. This is the macroprogram accumulator.

Integer Halving Constant

IHC

4

IHC addresses a register that always contains a constant: EXP=14, Mantissa=.5. This constant is used for conversion from floating point to fixed point and for halving. (Note that the exponent is in excess-64 form, i.e., the actual decimal value stored is 78.)

Constant Zero

ZER

5

ZER addresses a register that contains a constant zero.

Constant Maximum Number

MAX

6

MAX addresses a register that contains the largest power of 10, minus 1, that will fit in a floating point mantissa: $(10^{16})-1 = 9999999999999999$.

Floating-Point General Register 6

FG6

7

FG6 addresses a general register the microprogrammer may use for any purpose. There is no restriction on this register such as there is on the FPAC registers.

Floating-Point General Register 0

FG0

8

FG0 addresses a general register that the microprogrammer may use for any purpose. There is no restriction on this register such as there is on the FPAC registers.

FG0 is the only general-purpose register that is saved in a context block when a page fault occurs. Use FG0 for memory to FPAC operations (e.g., FAMS).

Floating-Point General Register 1

FG1

9

FG1 addresses a general register that the microprogrammer may use for any purpose. There is no restriction on this register such as there is on the FPAC registers.

Floating-Point General Register 2

FG2

A

FG2 addresses a general register that the microprogrammer may use for any purpose. There is no restriction on this register such as there is on the FPAC registers.

Floating-Point General Register 3

FG3

B

FG3 addresses a general register that the microprogrammer may use for any purpose. There is no restriction on this register such as there is on the FPAC registers.

Floating-Point General Register 4

FG4

C

FG4 addresses a general register that the microprogrammer may use for any purpose. There is no restriction on this register such as there is on the FPAC registers.

Floating-Point General Register 5

FG5

D

FG5 addresses a general register the microprogrammer may use for any purpose. There is no restriction on this register such as there is on the FPAC registers.

Accumulator Source

SRC

E

When SRC is coded, the Accumulator Source Register (ACSR) addresses the floating-point register file.

Accumulator Destination

DES

F

When DES is coded, the Accumulator Destination Register (ACDR) addresses the floating-point register file.

FL—Register File Load Specifier

The FL field determines the source for the register specified in the FCW field. MV/10000 system buses are 32 bits wide, but the floating-point register file is 64 bits wide. Double-precision data must be loaded from the system in two 32-bit segments (most-significant followed by least-significant).

If the double-precision flag is not set (flag2=0), FDI[32-63] is forced to zero. Single-precision numbers use only the most significant half of a location in the register file.

No Load

N

0

No value is written to the register file. However, data is taken from the CPM Bus and driven onto the FDI Bus. CPM[0-31] goes to FDI[0-31] and FDI[32-63].

Load Lower Half

ML

1

ML loads the least-significant 32 bits of a location in the floating-point register file. CPM[0-31] are sourced onto FDI[32-63] and FDI[0-31], and the least significant portion of the register file (bits 32-63) is loaded from the FDI Bus.

Load Upper Half

MH

2

MH loads all 64 bits of a location in the floating-point register file. CPM[0-31] source these bits. For double-precision numbers (FLAG2=1), CPM[0-31] are sourced on FDI [0-31] and FDI [32-63]; for single-precision numbers (FLAG2=0), CPM[0-31] are sourced on FDI [0-31], while FDI[32-63] are set to zero. All 64 bits of the register file are loaded from the FDI Bus.

Note that the most significant half of double precision data must always be loaded first, because MH destroys the least significant bits.

Load From FD

D

3

D sources FD[0-63] onto all 64 bits of the FDI Bus and loads all 64 bits of a location in the floating-point register file from the FDI Bus. If the FR field is coded with RB (i.e., a rounding operation is being performed), the FDI Bus will be forced to zero if the top 4 bits and carry-out of the mantissa ALU are zero.

FRG—Floating-Point Register Load Control

FRG controls the loading of various floating-point registers.

No Operation

N

0

N produces no effect.

Update FPSR

UFS

1

The Z, N, OVF, and UNF bits of the FPSR are updated. Z and N are set to the current values from the FDI Bus; OVF and UNF are ORed with the current values from the exponent ALU, so that the values in FPSR represent an accumulated value since the last time the bits were cleared.

Care must be taken if only the Z and N flags are to be updated. To avoid causing invalid floating point faults, pass a valid exponent through the exponent ALU if a valid floating point calculation has not been performed.

Read LO

RLO

4

RLO reads LOW[0-63] to M[8-71]. The LOW register is part of the multiply ALU hardware, and its contents are calculated as follows (where X is the X register and MY is the byte selected by YSEL). This micro-order is primarily for diagnostic visibility.

```
MY * X[24-31] => LOW[0-7]   (These bits are indeterminate.)
MY * X[8-15]  => LOW[8-23]
MY * X[24-31] => LOW[24-39]
MY * X[40-47] => LOW[40-55]
            0 => LOW[56-63]
```

Load Guard Digits

LGD

5

Load the Divide Guard Digit (DGD) register. This register holds the least-significant digits of the dividend before it is transferred to the DPR register. During division setup, the prescaled dividend is transferred from the working register to the register file, so that it may be transferred to the FS Bus. The DGD register prevents loss of guard digits during this transfer.

Partial Multiply and Load Working Register

LWM

6

Step the multiply pipe;

1) Add the data in the HI and LO registers and source the result on the M Bus.

2) Multiply the x register by the byte in the Y register selected by YSEL.

3) Load intermediate result in the HI and LO registers.

4) Decrement YSEL.

5) Load the working register.

Normally, the multiply pipe will be stepped for each byte of the multiplier in Y. LWM must be invoked one cycle before the first partial product is added into the working register. To accumulate partial products, M should source the FR Bus, the working register should be right shifted one byte and sourced on the FS Bus, the mantissa ALU should perform addition, and the FD Bus should source the working register input bus.

Load Working Register

LWR

7

LWR loads the working register at the end of the cycle (i.e., after a value has been calculated by the Mantissa ALU).

Load Y

LY

8

Load the Y register and the YSEL counter. LY loads a new value in Y and restarts the multiplication process, while maintaining the old value in X. The YSEL counter is loaded with the value from the IY field. YSEL=0 points to the high byte of the Y register.

Load XY

LXY

9

LXY loads the X and Y multiplier registers and the YSEL count. This is the initial setup command for a multiply operation. The X and Y registers hold the multiplicand and multiplier, respectively. The YSEL counter, which is loaded from the IY field (inverted), specifies which byte of the multiplier will be used for the next partial product. YSEL=0 points to the high byte of the Y register.

Source FPSR

SFS

A

SFS sources the Floating-Point Status Register to the FA Bus, bits 0-15. SFS disables the output to FA from the register file.

Source STATE

SST

B

SST sources the floating-point STATE register to the FA Bus, bits 0-15. SST disables the A output of the register file.

Load FPSR

LFS

C

LFS loads the Floating-Point Status Register (FPSR) from the FDI Bus, bits 0-15.

Load STATE

LST

D

LST loads the floating-point STATE register from bits 0-15 of the FA Bus.

Initial Divide

IDV

E

IDV is the micro-order that begins a division procedure. At this point the Divide Guard Digit register should contain the least-significant bits of the dividend, while the most-significant bits should be in a location in the register file. IDV sources DGD onto the FR Bus bits 32-39 or 64-72 (FLAG2=0 or 1). Which bits are used depends on the precision flag (FLAG2=1, double precision; FLAG2=0, single precision). At the same time, the FR Bus should be sourced by the general register file for the most-significant bits. The FS Bus should be sourced with the divisor from the register file (address must be in IB and FCW fields) and the mantissa ALU should perform subtraction (FOP:TSB). At the end of the cycle, the DPR is loaded with the initial partial remainder.

Double Load

LWD

F

LWD loads the DPR and WR twice per cycle.

FX—Excess-64 Control

The FX field is used in conjunction with the RAND:FLT:EXP:<S64 and A64> micro-orders. These micro-orders correct excess-64 exponents after exponent addition or subtraction.

No Operation

0

N

No operation is performed. This micro-order should be used with the RAND:FLT:EXP:<N and ACW> micro-orders.

Excess-64 Conversion

X64

1

This micro-order sources 64 to the S input of the exponent ALU. That value can then be added to or subtracted from the Exponent Working Register. Adding and subtracting 64 corrects exponents after addition or subtraction.

End of Chapter

Chapter 4
Microprogramming Examples

This chapter gives examples of MV/10000 microcode. The microinstructions are presented as they are printed by the microassembler. The microfields are presented in the same order as in Chapters 2 and 3, except that the microassembler reverses the FL and FCW fields. Some microfield names have been abbreviated in the microword headers; the micro-orders for each field are those given in Chapter 3. Note that the "LABEL" field is a pseudo field for the microassembler. Table 4-1 lists the abbreviations found in the microword headers.

Table 4-1. Microword Header Abbreviations

Abbreviation	Field Name
OP	COP or UCOP
D	DSR
AG	AGB
ST	MEMS
CM	MEMC
CPM	CPMS
CPD	CPDS
R0	REG0 or ATU0 or COVS or SGN
R1	REG1 or ATU1 or LOAD or EXP
R2	SPAD or SCNT
W	FW

For illustration purposes, we have left many of the microcode fields blank. In a genuine microroutine, many of these fields would of course be used. For instance, in the examples of memory accesses, you would have to specify a source and destination for the CPM Bus.

Memory Accesses

Because synchronization between the memory and the CPU is automatic, you need not consider timing in memory accesses. You may start memory (MEMS micro-order) in one microinstruction and complete memory (MEMC micro-order) in any following microinstruction. (A memory complete need not follow the start immediately, so long as there is no other intervening memory start. However, you should not leave a memory start pending for long, as this blocks I/O traffic.) The examples below show typical sequences.

Read Operation

In the example below, the double-word (32-bit) read operation is started with the RD micro-order. This operation is completed on the next microinstruction with the R micro-order. Note that the CPM Bus is sourced by main memory (MM), as it will be for any read operation.

```
LABEL:    OP   TSEL   ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR  FS  FOP W FCW FL FRG

STARTRD:  ---- ------ -------- - --- AG0 B  PSB -- RD -- --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
RDCOMPLT: ---- ------ -------- - --- --- -- --- -- -- R  MM  --- GN ---- LT- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
```

Although the example above is for a double-word read, the same sequence would apply to single-word or byte reads.

Write Operation

In the following example, the word write (16-bit) operation is started with the WW micro-order and completed with the W micro-order.

```
LABEL:    OP   TSEL   ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR  FS  FOP W FCW FL FRG

STARTWR:  ---- ------ -------- - --- AR0 B  PSB -- WW -- --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
WRCOMPLT: ---- ------ -------- - ONE AG1 B  ADD -- -- W  AG  --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
```

Read and Modify Operation

A read and modify operation requires no special memory start or complete. It begins with a write start, followed by a read complete. The read complete will not release memory, which remains started until a write complete is coded.

In the following example, the first microinstruction starts memory for a word write. The second has a read complete. The third instruction actually completes the start with a write. This sequence is used when you want to read information and write it back, possibly modified, to the same location.

```
LABEL:    OP   TSEL   ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR  FS  FOP W FCW FL FRG

STARTRM:  ---- ------ -------- - --- --- -- --- -- WW -- --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
READ:     ---- ------ -------- - --- --- -- --- -- -- R  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
WRITE:    ---- ------ -------- - --- --- -- --- -- -- W  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
```

Execute Completion

Microcode reads instructions for the IP by starting memory with a read word and a RAND:ATU:ATU0:IPST or RAND:ATU:ATU0:ICAT instruction. The reference is then completed with a MEMC:X micro-order. If there is any other combination of micro-orders, the ATU will not check execute protection.

In the following example, RW is coded in the same microinstruction with IPST. The reference is completed in the next instruction with an X.

```
LABEL:     OP   TSEL   ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR  FS  FOP W FCW FL FRG

RDINST:    ---- ------ -------- - --- --- -- --- -- RW -- --- --- AT IPST --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
XCOMPLT:   ---- ------ -------- - --- --- -- --- -- -- X  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
```

Overlapped Write Operation

In this example, the WD micro-order starts a double-word write operation. This operation completes during the next microinstruction with the W micro-order. As the first write completes, a second is started. The third microinstruction completes the second start. For the first write, the most-significant bits of a register in the FPU source the CPM Bus; for the second write, the least-significant bits are the source.

```
LABEL:     OP   TSEL   ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR  FS  FOP W FCW FL FRG

ST1:       ---- ------ -------- - --- --- -- --- -- WD -- --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
ST2CMPL1:  ---- ------ -------- - --- --- -- --- -- WD W  HF  --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
CMPL2:     ---- ------ -------- - --- --- -- --- -- -- W  LF  --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
```

Overlapped Read and Write

The following sequence shows a series of reads overlapped with a series of writes.

```
LABEL:     OP   TSEL   ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR  FS  FOP W FCW FL FRG

RD1:       ---- ------ -------- - --- --- -- --- -- RW -- --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
RD2:       ---- ------ -------- - --- --- -- --- -- RW R  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
WR1:       ---- ------ -------- - --- --- -- --- -- WW R  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
WR2:       ---- ------ -------- - --- --- -- --- -- WW W  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
---------  ---- ------ -------- - --- --- -- --- -- -- W  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
```

The following sequence shows a series of writes overlapped with a series of reads. This sequence is not illegal, but should be avoided. When a read start is coded in the same instruction with a write complete, it causes memory to extend the microinstruction cycle.

```
LABEL:     OP   TSEL   ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR  FS  FOP W FCW FL FRG

WR1:       ---- ------ -------- - --- --- -- --- -- WW -- --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
WR2:       ---- ------ -------- - --- --- -- --- -- WW W  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
RD1:       ---- ------ -------- - --- --- -- --- -- RW W  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
RD2:       ---- ------ -------- - --- --- -- --- -- RW R  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
---------  ---- ------ -------- - --- --- -- --- -- -- R  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
```

Illegal Sequences

The following examples show illegal sequences of microinstructions. No memory start can occur before the previous memory start is completed.

In the first sequence, RW starts memory in the first microinstruction. However, the N in the MEMC [CM] field does not release memory. Therefore, the RW is illegal, because it tries to start memory before it has been released.

```
LABEL:      OP   TSEL   ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM  R0  R1  R2  IA  IB  ID RS IOP  IY  IL FR  FS  FOP W FCW FL FRG

RD1:       ---- ------ -------- - --- --- -- --- -- RW -- --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
RD2:       ---- ------ -------- - --- --- -- --- -- RW N  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
RDCOMPLT1:---- ------ -------- - --- --- -- --- -- -- R  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
RDCOMPLT2:---- ------ -------- - --- --- -- --- -- -- R  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
```

In the second example, the sequence begins with a WW memory start. However, in the next microinstruction, a read complete is coded. This is a read/modify sequence (see above). Memory is not released by the R micro-order, and, therefore, the WW coded in the same microinstruction is illegal.

```
LABEL:      OP   TSEL   ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM  R0  R1  R2  IA  IB  ID RS IOP  IY  IL FR  FS  FOP W FCW FL FRG

WRSTART1: ---- ------ -------- - --- --- -- --- -- WW -- --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
WRSTART2: ---- ------ -------- - --- --- -- --- -- WW R  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
WRCOMPLT1:---- ------ -------- - --- --- -- --- -- -- W  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
WRCOMPLT2:---- ------ -------- - --- --- -- --- -- -- W  --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
```

IPOP—Crossing Macroinstruction Boundaries

The IPOP procedure connects one macroinstruction routine to another. The object of IPOP is to get the IP to provide a starting microaddress based on its decoding of the next macroinstruction, and to properly set up the parameters for that instruction. IPOP is performed by coding a NAC:COP micro-order that pops an empty stack and specifies the Top of Stack (TOS) as the next address. At the same time, the microinstruction tries to perform an effective address calculation (EFA) for the next macroinstruction. (Of course, the next macroinstruction may not require an EFA, in which case the effort is wasted. However, no instruction cycles are lost, and if an EFA is needed, the calculations are already under way before the next microroutine starts.)

The following microinstruction implements IPOP. (For this to work, the stack must be empty.)

```
LABEL:      OP   TSEL   ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM  R0  R1  R2  IA  IB  ID RS IOP  IY  IL FR  FS  FOP W FCW FL FRG

--------- CRTN TRUE    -------- - --- --- D  EFA -- S@ -- --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
```

A) CRTN—When its condition is true, this micro-order has a next address of TOS and pops the microstack. The microstack must be empty.

B) TRUE—This forces CRTN to pop and take TOS as the next address. If a normal test is used, e.g., an ALU test, the false path of the microprogram must code an abort for the memory start.

C) D—This micro-order designates the displacement register as the source of the AGB Bus. The Address Generator will therefore try to construct a logical address from the displacement field of the next macroinstruction.

D) EFA—This micro-order causes the Address Generator to do an EFA calculation, based on the decoded macroinstruction from the IP, and using the displacement from that instruction.

E) S@—This micro-order, in conjunction with the EFA micro-order, attempts a memory start based on the IP decode information from the next macroinstruction. The IP also determines whether an actual memory start takes place, i.e., the S@ micro-order may not start memory.

Indirection Resolution

The following microroutine illustrates indirection resolution. The routine is called from a microinstruction that starts memory for a word write (WW). The address for the write is in AR1 in the AG. If the indirection bit is set in the address, the instruction does a jump to the indirecton resolution subroutine.

An important general rule for calling microroutines is that a memory abort should be coded following the return from a subroutine which attempted an EFA calculation (a bogus memory start will be pending as a result).

Calling Microinstruction

```
LABEL:     OP   TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM  R0  R1  R2  IA  IB  ID RS IOP  IY  IL FR  FS  FOP W FCW FL FRG

       ;Start a write to the address in AR1.
       ;Do a conditional jump to the indirection subroutine (IRES).  The indirection bit determines whether the jump occurs.
--------- CJSR INDR   IRES     - --- AR1 B  PSB N  WW -- --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
       ;The next microinstruction must complete the write.
```

Called Routine

The subroutine that does the indirection chaining aborts the original write reference, and begins a read instead. The RAND:ATU:ATU1:DF micro-order is coded, which ensures that indirection will not exceed 15 levels.

```
LABEL:     OP   TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM  R0  R1  R2  IA  IB  ID RS IOP  IY  IL FR  FS  FOP W FCW FL FRG

       ;Abort the write started in the calling routine.
       ;Start a read from the address in AR1.
       ;Code DF to ensure indirection depth protection.
       ;LEAP into the indirection loop.
IRES:    LEAP ------ IRES2     - --- AR1 B  PSB N  RD A  --- --- AT ---- DF  --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---

       ;This microinstruction is the top of the indirection loop.
       ;Abort the write started by IRES3.
       ;Start a read from the address in AR1.


       ;Code DF to ensure indirection depth protection.
IRES1:   LEAP ------ IRES2     - --- AR1 B  PSB N  RD A  --- --- AT ---- DF  --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---

       ;Complete the memory reference started in the previous microinstruction.
       ;Load the contents of the location addressed by AR1 into AR1.
IRES2:   LEAP ------ IRES3     - --- AR1 -- --- M  -- R  MM  --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---

       ;Start a memory write to the location addressed by AR1 (i.e., try the original memory reference again).
       ;If the indirect bit is not set, return to the calling routine.
       ;If the indirect bit is set, go to the top of the indirection loop.
IRES3:   CRTN NINDR  IRES1     - --- AR1 B  PSB N  WW -- --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---
```

Dispatching

The microsequencer can construct addresses using the dispatch register (see Chapter 2). The following example demonstrates dispatching in a microroutine with the implementation of an actual macroinstruction: WCOB (Wide Count Bits). WCOB counts the bits that are set in the source (ACS) accumulator and adds that count to the value in the destination (ACD) accumulator. The microroutine that implements WCOB uses a dispatch table to determine the number of 1-bits in each nibble of the source accumulator.

```
LABEL:     OP   TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM  R0  R1  R2  IA  IB  ID RS IOP  IY  IL FR  FS  FOP W FCW FL FRG

        ;This microinstruction is the initial instruction in the routine.  It is reached after IPOP by means of a starting
        ;microaddress (STUAD), generated by the IP decode RAMs.
        ;The SRC and AGA micro-orders drive the AG source register value onto the CPD Bus.  The GN and LD micro-orders
        ;(RAND:GEN:REG1:LD) load this value into the dispatch register (DSP[1-7] = CPD[24-31]).
        ;The PDR register is loaded automatically from the CPD Bus.

WCOB:      LEAP ------ WCOB1    - SRC --- -- --- -- -- -- --- AGA GN ---- LD  --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---


        ;This microinstruction provides the initial dispatch into the table.  The DSPA micro-order constructs a microaddress based on
        ;WCOBTAB (the dispatch table's location, which is found in the ADDRESS field).  The F micro-order specifies a 4-bit
        ;dispatch.  The AG micro-orders (DES, DES, C, ADD, and Y) add the CNST value (coded in the FA field) into the destination
        ;register in the AG.  The CPM:AG micro-order sources this result to the CPM Bus.  The IB:DES and IL:M micro-orders load this
        ;value into the destination register in the Integer ALU.
        ;
        ;The IOP:R1 and IY:HR0 micro-orders cause a 4-bit right shift.  The hex shifter is sourced by PDR via the ID Bus (PD) and the
        ;ALU R-input multiplexer (DA).  The output of the hex shifter goes to the IY Bus and then to the CPD Bus (CPD:IY), and the
        ;shifted value returns to the PDR.  Thus the new PDR value is the old value right shifted 4 bits and zero filled.  The PDR
        ;value is also ANDed with all 1's (IA:M1); the result can be tested by TSEL:FZR to see if the remaining bits in PDR are zero.

WCOB1:     DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  000 --- --- - --- -- ---
```

Dispatch Table

The dispatch table has one test instruction and fifteen nearly identical table entries. Each table entry performs a new dispatch based on the beginning of the table: WCOBTAB. Which table entry is executed depends on the DSP register, which holds the value of the current least-significant nibble in the PDR register. The CNST value equals the number of 1 bits in that nibble. For example, if the nibble is 0111, it dispatches to the seventh table entry; the CNST field for that entry contains 3_{10}. The CNST value is added into the value in the destination accumulator each time a table entry is executed.

```
LABEL:     OP   TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM  R0  R1  R2  IA  IB  ID RS IOP  IY  IL FR  FS  FOP W FCW FL FRG

        ;This microinstruction begins the table and is also the test to see whether the routine is complete.  Whenever the nibble
        ;being tested is 0, the table entries dispatch to location 0 in the table.  The TSEL:FRZ micro-order tests whether the
        ;remaining bits in the PDR register are equal to zero.  This test was set up by the M1 micro-order in the previous
        ;microinstruction.  If non-zero bits remain, control goes back to WCOB1, which dispatches into the table again.  If no bits
        ;remain, the routine is done and this microinstruction IPOPs.

WCOBTAB:  CRTN FZR    WCOB1    - --- --- D  EFA -- S@ -- --- --- -- ---- --- --- --- --- -- -- --- ---- -- --- --- --- - --- -- ---


        ;Each microinstruction in the table is identical to WCOB1,
  except for the CNST field, which holds a value equal to the
        ;number of 1's in the current nibble.

--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  001 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  001 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  002 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  001 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  002 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  002 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  003 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  001 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  002 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  002 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  003 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  002 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  003 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  003 --- --- - --- -- ---
--------- DSPA ------ WCOBTAB  F DES DES C  ADD Y   -- -- AG  IY  GN ---- LD  --- M1  DES PD DA R1  HR0  M  004 --- --- - --- -- ---
```

End of Chapter

Chapter 5
MV/10000 Microcode Macroassembler

The first section of this chapter outlines the macroassembler constructs. The second section provides examples of assembled microprogram segments.

The Macroassembler

The macroassembler makes your job easier by letting you code at a higher level than individual micro-orders. We use the following syntax conventions to describe the macroassembler:

Symbol	Name	Meaning
[ ]	Square brackets	Enclose optional elements.
< >	Angle brackets	Enclose non-terminal elements.
=	Equals	Precedes a data destination.
==	Double equals	Precedes a data source.
...	Ellipsis	Indicates repeatable element.
{ }	Braces	Indicate a choice of the enclosed elements. When several elements are listed on successive lines, this also indicates a choice among the elements.
::=	definition	"Is defined as"
□	Box	Encloses micro-order equivalents of macroassembler constructs. The micro-order codings are always presented immediately following the terminal construct from which they result.

Each macroassembler command consists of one or more constructs and produces a single microinstruction. A macroassembler command has the following syntax:

```
<macroassembler_construct>,...;
```

Comments have a percent sign (%) at the beginning of the line, or are enclosed in /* . . . */. The second type of comment delimiter can be nested to any level.

CPM Bus

```
CPM [ = <CPM_dest> ]... == <CPM_src>
```

```
<CPM_dest> ::=
```

MEM_WRITE	MEMC:W
EXECUTE_DATA	MEMC:A
FPSR	FRG:LFS
FP_LOW (<FPU_reg>)	FL:ML, FCW
FP_HIGH (<FPU_reg>)	FL:MH, FCW
ALU (<ALU_reg>)	IL:M, IB
TREG	{GEN:REG1,ATU:ATU1}:LT
SPAD (<literal>)	IL:{NM M},SPAD:WC,CNST
SPAD (SPAR)	IL:{NM M},SPAD:WS
AG (<AG_reg>)	AL:M, AB
AG_IF_TRUE (<AG_reg>)	AL:C, AB

```
<CPM_src> ::=
```

ALU (<ALU_reg>)	CPMS:IA, IA
CARRY_IN_ALU (<ALU_reg>)	CPMS:IA,IA,GEN:REG0:FCY
IY	CPMS:IY
AY	CPMS:AY
MEM_READ	CPMS:MM, MEMC:R
FP_HIGH (<FPU_reg>)	CPMS:HF, IA
FP_LOW (<FPU_reg>)	CPMS:LF, IA
FPSR	CPMS:HF, FRG:SFS
FP_STATE	CPMS:HF, FRG:SST
ALL_ONES	CPMS:N

Notes:

<FPU_reg> is a mnemonic specifying one of 16 FPU registers (see "Floating-Point ALU Micro-orders" in Chapter 3).

<ALU_reg> is a mnemonic specifying one of 16 ALU registers (see "Integer ALU Micro-orders" in Chapter 3).

In SPAD(<literal>), the literal is a mnemonic that specifies the scratch pad address.

CPD Bus

The two CPD Bus constructs are CPD and PDR. The CPD construct implies PDR should not be loaded; i.e,, the macroassembler codes {GEN:REG0 ATU:ATU0}:NPDR. The only exception to this is CPD==ZERO. The PDR construct does not code NPDR. In addition, the PDR construct can manipulate the reference bits. The only other difference between the two constructs is that CPD == ZERO codes CPDS:N and PDR == ZERO codes CPDS:ZER.

```
CPD [ = <CPD_dest> ]...  == <CPD_src>
```

```
<CPD_dest> ::=
```

FLAGS_INVERTED	GEN:REG0:LFLG
IO_CONTROLLER	ATU:ATU0:SIO
MICRO_STACK_INVERTED	NAC:PCPD
CONSOLE_DATA	GEN:REG0:CDW
CASE_DATA	GEN:REG1:LD
REF_MOD_BITS	ATU:ATU0:WRRM
SBR_INVERTED	ATU:ATU0:WSBR
TRANSLATION_CACHE	ATU:ATU0:OPTA
IP_STATE	ATU:ATU0:LIPS

```
<CPD_src> ::=
```

TREG	CPDS:TRG
LAR	CPDS:LAR
IP_STATE	CPDS:IPS
ATU_STATE	CPDS:ATS
SEQUENCER_STATE	CPDS:USS*
IO_DATA	CPDS:IOC
CONSOLE_DATA	CPDS:CDR
CONSOLE_INSTRUCTION	CPDS:CIR
PC	CPDS:PC
PC_OF_EXECUTION	CPDS:PCX
RETURN_PC	CPDS:PCN
IY	CPDS:IY
ATU_DIAGNOSTICS	CPDS:ATD
ZERO	CPDS:{ZER,N}
AG (<AG_reg>)	CPDS:AGA, AA

Notes:

* SEQUENCER_STATE sources MICRO_STACK and FLAGS in true sense, and CASE_DATA in inverted sense.

<AG_reg> is a mnemonic specifying one of 16 AG registers (see "Address Generator Micro-orders" in Chapter 3).

```
PDR  [ = CPD ] [ = <CPD_dest> ]... == [
<EIGHT_REF_BITS_ORed_WITH>
] <CPD_src>
```

`<CPD_dest> ::=` See the preceding section.

`<EIGHT_REF_BITS_ORed_WITH> ::=`	ATU:ATU0:RSRF

`<CPD_src> ::=` See the preceding section.

Memory Starts and Address Generator Operations

```
START <AG-op> [<adr_opt>] [ FOR ] <ref_type>
```

```
<AG_op> ::=

AY [ = <AG_dest> ]... == <AG_src>
```

```
<AG_dest> ::=
```

CRE	ATU:ATU0:LCRE
RESTORE_PC	ATU:ATU0:LIPS
LAR	ATU:ATU0:LLAR
PC	ATU:ATU0:IPST
ATU_STATE	ATU:ATU0:LATS
<CPM_dest>	See "CPM Bus," above.

```
<AG_src> ::=

PASS ( <AGB_src> )
<AGB_src> {+ -} A(<AG_reg>)
```

```
<AGB_src> ::=
```

CNST (<literal>)	AGB:C, CNST:<literal>
LAST_LA	AGB:L
B (<AG_reg>)	AGB:B, AB:<AG_reg>

Note:

In `CNST( <literal> )`, the literal is a hex number (00-FF).

`<AG_reg>` is a mnemonic specifying one of 16 AG registers (see "Address Generator Micro-orders" in Chapter 3).

```
<adr_opt> ::=

[ <byte/word> ]   [ IN_CURRENT_RING   {ATU,GEN}:REG1:AC   ]
```

```
<byte/word> ::=
```

WITH_BYTE_ADDRESSING	ATU:ATU0:BYTE
WITH_WORD_ADDRESSING	ATU:ATU0:WORD

```
<ref_type> ::=
```

WIDE_JUMP	ATU:ATU0:IPST, MEMS:RD
NARROW_JUMP	ATU:ATU0:IPST, MEMS:RW
IP_TRANSLATION	ATU:ATU0:ICAT, MEMS:RW
READ_DOUBLE	MEMS:RD
READ_WORD	MEMS:RW
READ_BYTE	MEMS:RB
WRITE_DOUBLE	MEMS:WD
WRITE_WORD	MEMS:WW
WRITE_BYTE	MEMS:WB
PER_IP_DECODE	MEMS:S@
PREVIOUS_REFERENCE	MEMS:S@
OBJECT_REFERENCE	ATU:ATU0:OPTA, MEMS:S@
DBL_WORD_ASSEMBLY	ATU:ATU0:CM0, MEMS:RW
CACHE_BLOCK_FLUSH	ATU:ATU0:CM0, MEMS:RD

```
START_EXECUTE

  ATU:ATU0:CM0, MEMS:WW, AA:0, AB:0, AGB:B, AOP:SUB
```

```
ATTEMPT_NEXT_EFA

  AGB:D, AOP:EFA, MEMS:S@
```

```
START_READ_FOR <PTE_level>
```

```
<PTE_level> ::=
```

FIRST_PAGE_TABLE_ENTRY	ATU:ATU0:RSBR, MEMS:RD
SECOND_PAGE_TABLE_ENTRY	ATU:ATU0:LPTA, MEMS:RD

Memory Completion

ABORT_MEMORY	MEMC:A
READ_MEMORY*	MEMC:R
COMPLETE_JUMP	MEMC:X
COMPLETE_FLUSH	MEMC:R
COMPLETE_TRANSLATE**	MEMC:X

Notes:

Normal reads and writes are handled by <CPM_dest>.

* READ_MEMORY does NOT source CPM.

** COMPLETE_TRANSLATE is for completion of ICATs.

ALU Operation Constructs

IY Bus

```
IY [ = <alu_dest>] ... == <alu_src>
```

```
<alu_dest> ::=
```

SPAR	GEN:SPAD:LS
SPAR_TABLE_OFFSET	GEN:REG0:SPY4, SPAD:LS
ABS_VALUE(<ALU_reg>)	FIX:LOAD:AV
PDR	CPDS:IY
CASE_DATA	CPDS:IY, GEN:REG1:LD*
<CPM_dest>	See "CPM Bus," above.

**Note:*

CASE_DATA will load PDR unless NO_LOAD_PDR is coded.

<ALU_reg> is a mnemonic specifying one of 16 ALU registers (see "Integer ALU Micro-orders" in Chapter 3).

```
<alu_src> ::=

<alu_op>
<alu_IY_op>  ( <alu_op> )
<hex_shift_op>  ( <shft_mag>, <IR_src> )
<extnd_op>  ( <IR_src> )
TRANSLATION_OF <ALU_reg> FOR <edit_type> IN_BOTTOM_NIBBLE_OF <alu_op>
```

Note:

<ALU_reg> is a mnemonic specifying one of 16 ALU registers (see "Integer ALU Micro-orders" in Chapter 3).

```
<alu_op> ::= <IS_src>  <iop>  <IR_src>
```

```
<IS_src> ::= A ( <ALU_reg> )
             <ID_src>
```

```
<ID_src> ::=
```

B (<ALU_reg>)	ID:BR
SPAD (<literal>)	ID:SC
SPAD (SPAR)	ID:SS
CNST (<literal>)	ID:CN
PDR	ID:PD
SRC_POINTER	ID:AS
MICROSTATE	ID:MS
ZERO	ID:ZR

Notes:

In `SPAD( <literal> )`, the literal is a mnemonic that specifies the scratch pad address.

In `CNST( <literal> )`, the literal is a hex number (00-FF).

<ALU_reg> is a mnemonic specifying one of 16 ALU registers (see "Integer ALU Micro-orders" in Chapter 3).

<iop> ::=

+	IOP:ADD
+1+	IOP:CAD, 'Add with carry in'
-	IOP:CSR
-1-	IOP:SMR, 'Subtract without carry'
AND	IOP:AND
XOR	IOP:XOR
OR	IOP:OR
NOT_AND	IOP:ANC, 'AND complement in reverse direction'

<IR_src> ::=

<IS_src>	See "ALU Test"
CPD_ZERO	RS:{CA, CD}, CPDS:N
AG(<AG_reg>)	RS:{CA, CD}, CPDS:AGA*
LAR	RS:{CA, CD}, CPDS:LAR*
TREG	RS:{CA, CD}, CPDS:TRG*
CPD	RS:{CA, CD}
PC_OF_EXECUTION	RS:{CA, CD}, CPDS:PCX**
RETURN_PC	RS:{CA, CD}, CPDS:PCN**
PC	RS:{CA, CD}, CPDS:PC**

Notes:

* AG, LAR, and TREG will cause PDR to load unless NO_LOAD_PDR is coded. CPD_ZERO will not load PDR.

** {ATU,GEN}:XTND is coded.

<AG_reg> is a mnemonic specifying one of 16 AG registers (see "Address Generator Micro-orders" in Chapter 3).

<alu_IY_op> ::=

BIT_SHIFT_RIGHT	IY:BR0
BIT_SHIFT_RIGHT_WITH_1	IY:BR1
BIT_SHIFT_LEFT	IY:BL0
BIT_SHIFT_LEFT_WITH_1	IY:BL1
BYTE_SWAP	IY:BSW
APPEND_PSR_TO	IY:PSR

<hex_shift_op> ::=

HEX_SHIFT_RIGHT	IY:HR0
HEX_SHIFT_LEFT	IY:HL0
HEX_ROTATE_RIGHT	IY:HRT

<shift_mag> ::=

{R1, R2, R3, R4, R5, R6, R7, R@, L1, L2, L3, L4, L5, L6, L7, L@}

<IR_src> ::= See "<alu_src>," above.

<extnd_op> ::=

WORD_SWAP	IY:HRT, IOP:R4
WORD_SIGN_EXTEND	IY:WSX
WORD_ZERO_EXTEND	IY:WZX
BYTE_SIGN_EXTEND	IY:BSX

<edit_type> ::=

SIGN_OVERPUNCH_BYTE	CNST:VSO
DIGIT	CNST:VDB
LOW_NIBBLE_DIGIT	CNST:VDL
HIGH_NIBBLE_DIGIT	CNST:VDH

ALU Test

This construct allows use of the ALU without sourcing the result to the IY Bus:

```
ALU_TEST == <alu_op>
```

<alu_op> ::= See "<alu_src>," above.

Loading SPAR

This construct loads SPAR with an address from the constant field:

```
SPAR == <literal>
```

```
GEN:REG0:SPCN, GEN:SPAD:LS, CNST:<literal>
```

Note:

`<literal>` is a hex number (00-FF).

Edit PROM

This construct specifies an edit PROM test generation only (See CNST: Commercial Translation):

```
TEST <ALU_reg> FOR <prom_test>
```

```
<prom_test> ::=
```

SIGN_OVERPUNCH_BYTE	CNST:VSO
DIGIT	CNST:VDB
LOW_NIBBLE_DIGIT	CNST:VDL
HIGH_NIBBLE_DIGIT	CNST:VDH
CHARACTER	CNST:VCB
SIGN	CNST:VSB
LOW_NIBBLE_SIGN	CNST:VSL
COMMERCIAL_SIGN	CNST:CSB
CPU_DEVICE	CNST:CPUD
IO_SKIP	CNST:SKPT
ION_FLAG_CHANGE	CNST:IONF

Note:

`<ALU_reg>` is a mnemonic specifying one of 16 ALU registers (see "Integer ALU Micro-orders" in Chapter 3).

ID Bus

This construct is for use of the ID Bus (when not required by an <alu_op>) and for loading the PSR:

```
ID [ = <ID_dest> ]...  == <ID_src>
```

```
<ID_dest> ::=
```

DES_POINTER	GEN:REG0:{LDAD,LDSD}
SRC_POINTER	GEN:REG0:{LDAS,LDSD}
PSR	FIX:COVS:LPSR

`<ID_src> ::=` See "`<alu_src>`," above.

IR

This construct specifies an IR (R Bus) source for the IR_SIGN test:

```
IR  == <IR_src>
```

`<IR_src> ::=` See "`<alu_src>`," above.

FPU Operations

FD Bus

```
FD [ = <FPU_dest> ]...  ==
      <FR_src><fop><FS_src>
                    [ TRUNCATED_IF_NOT_ROUNDING  FLT:SGN:{TRN TRI} ]
```

```
<FPU_dest> ::=
```

FPU (<FPU_reg>)	FL:D, FCW:<FP_reg>)
DPR	FRG:{LWD IDV})
WR	FRG:{LWR LWM LWD IDV})

`<FR_src> ::=`

ROUND_BIT	FR:RB
A(<FPU_reg>) [WITH_DGD]	FR:FA,IA,[FRG:IDV]
SELECTED_A(<FPU_reg>, <FPU_reg>)	FR:FA,IA,IB
DPR	FR:DR
MULTIPLIER	FR:MP

`<fop> ::=`

+	FOP:TAD
-	FOP:TSB
@+	FOP:ADD
@-	FOP:SUB

`<FS_src> ::=`

ZERO	FS:ZR
B(<FPU_reg>)	FS:FB,IB
PRESCALED_WR	FS:RS
WR_RIGHT	FS:RS
NORMALIZED_WR	FS:LS
WR_LEFT	FS:LS

Note:

`<FPU_reg>` is a mnemonic specifying one of 16 FPU registers (see "Floating-Point ALU Micro-orders" in Chapter 3).

FA and FB Buses

These constructs source FA and FB from the register file without sourcing on FR and FS:

1. FA == A(<FPU_reg>) (IA)
2. FB == B(<FPU_reg>) (IB)
3. FPU_B_SELECT_DURING_SECOND_HALF == B(< FP_reg>) (FCW)
   (This construct selects FB data during division cycles.)

Note:

`<FPU_reg>` is a mnemonic specifying one of 16 FPU registers (see "Floating-Point ALU Micro-orders" in Chapter 3).

WR ==

PRESCALE_OPERAND	FWR:R, RM:FL, FLT:SCNT:CMP
QUOTIENT	FWR:Q, FRG:LWD
A (<FPU_reg>)	FWR:R, FR:FA, IA
B (<FPU_reg>)	FWR:S, FS:FB, IB
ZERO	FWR:S, FS:ZER

Notes:

Loading WR from FD is specified with the `FD = WR ==` construct defined for <FPU_op>'s.

`<FPU_reg>` is a mnemonic specifying one of 16 FPU registers (see "Floating-Point ALU Micro-orders" in Chapter 3).

Sign and Exponent Control

`SIGN ==`

A_SIGN	FLT:SGN:{SA LAB TRN TRI}
FA0	FLT:SGN:MOV
NEG_FA0	FLT:SGN:NEG
ZERO	FLT:SGN:ZER
A_XOR_B	FLT:SGN:XOR

`LOAD_SIGNS`

```
FLT:SGN:{LAB TRN TRI}
```

`EXPONENT ==`

EWR	FLT:EXP:N
FA	FLT:EXP:LAX
EWR-64	FLT:EXP:S64,FX:X64
EWR+64	FLT:EXP:A64,FX:X64
EWR-MAG	FLT:EXP:SNM
FA-FB	FLT:EXP:SUB
EWR+MOF	FLT:EXP:ACW
FA+MOF	FLT:EXP:ACA
EWR+MAG+MOF	FLT:EXP:ACN
FA+FB	FLT:EXP:ADD

Shift Count

```
SHIFT_MAG ==
```

EXPONENT	FLT:SCNT:LEF
PRESCALE_COMPARE	FLT:SCNT:CMP
FIRST_NIBBLE_ZERO_DETECT	FLT:SCNT:FNZ
LEADING_ZERO_DETECT	FLT:SCNT:LZD
DIVIDE_PRESCALE_DETECT	FLT:SCNT:DVP
CNST (<literal>)	FLT:SCNT:LCN, IY:<literal>

Note:

<literal> is a hex number (0-F). Use a positive value for a right shift, or a twos complement value for a left shift. For example: E= right-shift 14 digits, or left-shift 2 digits. Use the symbols R0 through R15 and L0 through L15 to specify the literal.

```
ALLOW_SHIFT_MAG_CORRECTION
```

```
FLT:SCNT:LZD
```

```
ENABLE_MOF_CORRECTION
```

```
FLT:SCNT:LZD
```

Multiply Control

```
X == A(<FPU_reg>), Y == B(<FPU_reg>),  Y_SELECT == {0 to 7}
```

```
FRG:LXY, FR:FA, IA, FS:FB, IB, IY:<Y_select>
```

```
Y == B(<FPU_reg>),  Y_SELECT == {0 to 7}
```

```
FRG:LY, FS:FB, IB, IY:<Y_select>
```

Notes:

<Y_select> is a hex number (0-7) specifying one of eight bytes of the Y operand. Y_SELECT = 0 gives the most significant byte of Y.

The above two constructs must be coded in the order shown.

`<FPU_reg>` is a mnemonic specifying one of 16 FPU registers (see "Floating-Point ALU Micro-orders" in Chapter 3).

```
STEP_MULTIPLY_PIPE      FRG:LWM
```

FPU State

```
UPDATE_FPSR      FRG:UFS
```

```
RESTORE_STATE_WITH A(<FPU_reg>)      FRG:LST, IA, FLT:SCNT:RST
```

Note:

`<FPU_reg>` is a mnemonic specifying one of 16 FPU registers (see "Floating-Point ALU Micro-orders" in Chapter 3).

Divide Control

```
DGD == FS_GUARD_DIGITS      FRG:LGD
```

GEN Randoms

ACSR (SRC Register Pointer) Randoms

```
INCREMENT_SRC_POINTER          GEN:REG0:INCS
DECREMENT_SRC_POINTER          GEN:REG0:DECS
POINT_SRC_TO_E                 GEN:REG0:{FRCS,FRSD}
POINT_SRC_TO  <register>       GEN:REG0:{LDAS,LDSD},ID:CN,CNST:<reg>
```

Note:

`<register>` is a mnemonic specifying one of 16 registers.

ACDR (DES Register Pointer) Randoms

INCREMENT_DES_POINTER	GEN:REG0:INCD
DECREMENT_DES_POINTER	GEN:REG0:DECD
POINT_DES_TO_F	GEN:REG0:{FRCD,FRSD}
POINT_DES_TO <register>	GEN:REG0:{LDAD,LDSD},ID:CN,CNST:<reg>

Notes:

See the ALU (ID =) construct earlier in this chapter for loading SRC and DES from the ID Bus.

<register> is a hex number (0-F) or register mnemonic specifying one of 16 registers.

Flag Manipulation

These macros use the CNST field to specify flag manipulation.

```
MODIFY_FLAGS_0123 (<flg_mod>, <flg_mod>, <flg_mod>, <flg_mod>)
```

```
GEN:REG0:MFS0
```

```
<flg_mod> ::=
```

N	MFLG:N
SET	MFLG:S
CLEAR	MFLG:C
TOGGLE	MFLG:T

```
MODIFY_FLAGS_4567 (<flg_mod>, <flg_mod>, <flg_mod>, <flg_mod>)
```

```
GEN:REG0:MFS1
```

```
MODIFY_FLAGS_46_WITH_TEST ( <flg_tmod>, <flg_tmod> )
```

```
GEN:REG0:AF46
```

<flg_tmod> ::=

N	AFLG:N
SET	AFLG:S
CLEAR	AFLG:C
TOGGLE	AFLG:T
AND	AFLG:A
OR	AFLG:O
XOR	AFLG:X
LOAD	AFLG:L

```
MODIFY_FLAGS_57_WITH_TEST ( <flg_tmod>, <flg_tmod> )
```

```
GEN:REG0:AF57
```

Skips

```
SKIP ON <skp_condition>
```

<skp_condition> ::=

ALC_RESULT	FIX:COVS:ALC
FALSE_TEST	GEN:REG0:SKFT
ALU_CRY=1	GEN:REG0:WSKP, CNST:CRY
ALU_CRY=0	GEN:REG0:WSKP, CNST:NCRY
IS>=IR	GEN:REG0:WSKP, CNST:CRY
IS<IR	GEN:REG0:WSKP, CNST:NCRY
SIGNED_IS>=IR	GEN:REG0:WSKP, CNST:SGE
SIGNED_IS<IR	GEN:REG0:WSKP, CNST:NSGE
ALU=0	GEN:REG0:WSKP, CNST:FZR
ALU<>0	GEN:REG0:WSKP, CNST:NFZR

Miscellaneous Randoms (NPDR and XTND)

NO_LOAD_PDR	{GEN:REG0,ATU:ATU0}:NPDR
EXTEND_MICRO_CYCLE	{GEN:REG0, ATU:ATU0}:XTND

ATU Randoms

TURN_ATU_ON	ATU:ATU0:{AON, RSRF}
TURN_ATU_OFF	ATU:ATU0:AOFF
SET_IFLUSH	ATU:ATU0:IPFL
RESET_8_REF_BITS*	ATU:ATU0:RSRF
ENABLE_INTERRUPTS	ATU:ATU0:ION
DISABLE_INTERRUPTS	ATU:ATU0:IOFF
DISABLE_INTERRUPTS_ONE_INSTRUCTION**	ATU:ATU0:DISI
RESET_ESR	ATU:ATU0:LCRE, ATU1:AC
PURGE_THE_ATU_CACHE	ATU:ATU0:PRGA
DEFER_ON_FALSE_TEST	ATU:ATU1:DF
CPA ==OBJECT_PAGE_TABLE_ADDRESS	ATU:ATU0:OPTA
CPA ==LOW_ORDER_PAGE_TABLE_ADDRESS	ATU:ATU0:LPTA

Note:

* This random forces data on top of CPD<24-31>.

** Do not code during IPOP.

FIX Randoms

CARRY_IN_IS_CARRY	XC random mode: RM:FIX, CIB:C
CARRY == SIGN_OF(<IR_src>)	FIX:COVS:LDCY
CARRY == ZERO	FIX:COVS:CLRC
CARRY == ONE	FIX:COVS:SETC
CARRY == ALU_CRY	FIX:COVS:{LCRY, LOVC}
CARRY == ALC_CRY	FIX:COVS:ALC
CLEAR_OVR	FIX:COVS:COVR
UPDATE_OVR	FIX:COVS:LOVC
CLEAR_OVK	FIX:COVS:COVK
SET_OVK	SET_OVK
USE_16_BIT_TESTS	FIX:LOAD:NA

<IR_src> ::= See "<alu_src>," above.

Next Address Sequence

Conditional Address Generation

```
IF [NOT] <test-cond> GOTO  NAC:{CJMP CABT}  <page_address>[,
      POP_MICRO_STACK  NAC:CABT  ]
```

```
IF [NOT] <test_cond> <cnac_op> <page_address>
```

`<cnac_op> ::=`

CALL	NAC:CJSR
CASE_8_INTO	NAC:CDSP,DSR:E
CASE_4_INTO	NAC:CDSP,DSR:F
CASE_ATU_INTO	NAC:CDSP,DSR:A
RETURN_ELSE_GOTO	NAC:CRTN
RETURN_ELSE_POP_AND_GOTO	NAC:TWB
RESTORE_ELSE_GOTO	NAC:CRST

Note:

for a listing of `<test_cond>`, see "Test Definitions", below.

Unconditional Address Generation

```
<unac_op> <full_address>
```

`<unac_op> ::=`

GOTO	NAC:LEAP
CALL	NAC:LSR
CASE_8_INTO	NAC:DSPA,DSR:E
CASE_4_INTO	NAC:DSPA,DSR:F
CASE_ATU_INTO	NAC:DSPA,DSR:A
CALL_CASE_8_INTO	NAC:DSPR,DSR:E
CALL_CASE_4_INTO	NAC:DSPR,DSR:F
CALL_CASE_ATU_INTO	NAC:DSPR,DSR:A
POP_AND_GOTO	NAC:LPOP
PUSH	NAC:PUSH
PUSH_CPD_AND_GOTO	NAC:PCPD
GOTO_TOS_AND_PUSH	NAC:TPSH

Note:

`<full_address>` is a symbolic address.

Pseudo-unconditional Address Generation

RETURN	NAC:CRTN,TSEL:TRUE
POP_MICRO_STACK	NAC:CABT,TSEL:FALSE
RESTORE	NAC:CRST,TSEL:TRUE
RESTORE_WITH_EVEN_PARITY	NAC:CRST,TSEL:TRUE, GEN:REG0:EPAR

Test Definitions

Microsequencer Tests

Macro	TSEL Micro-order
TRUE	TRUE
FALSE	NTRUE
CPD<31>=1	CPD31
CPD<31>=0	NCPD31
MICRO_STACK_EMPTY	USMT
INTERRUPT_PENDING	INTR
IO_BUSY	IOB
INSTRUCTION_READY	IVLD
XCTED_INSTRUCTION	XCTF
ROUNDING	RND
FLAG0=1	FLG0
FLAG0=0	NFLG0
FLAG1=1	FLG1
FLAG1=0	NFLG1
FLAG2=1	FLG2
FLAG2=0	NFLG2
FLAG3=1	FLG3
FLAG3=0	NFLG3
FLAG4=1	FLG4
FLAG4=0	NFLG4
FLAG5=1	FLG5
FLAG5=0	NFLG5
FLAG6=1	FLG6
FLAG6=0	NFLG6
FLAG7=1	FLG7
FLAG7=0	NFLG7

ALU Test Conditions

Macro	TSEL Micro-order
IY<28>=1	Y28
IY<28>=0	NY28
IY<29>=1	Y29
IY<29>=0	NY29
IY<30>=1	Y30
IY<30>=0	NY30
IY<31>=1	Y31
IY<31>=0	NY31
ID<31>=1	D31
ID<31>=0	ND31
ID_SIGN=1	DSGN
ID_SIGN=0	NDSGN
SRC=DES	COMP
SRC<>DES	NCOMP
INTERRUPT_RESUME	IRES
SIGN_OVERPUNCH_BYTE	COM2
DIGIT	COM2
LOW_NIBBLE_DIGIT	COM2
HIGH_NIBBLE_DIGIT	COM2
CHARACTER	COM1
SIGN	COM1
LOW_NIBBLE_SIGN	COM1
COMMERCIAL_SIGN	COM1
CPU_DEVICE	IOT
IO_SKIP	IOT
ION_FLAG_CHANGE	IOT
ALU<31>=1	F31
ALU<31>=0	NF31
ALU_NIBBLE_CRY=1	CRY28
ALU_NIBBLE_CRY=0	NCRY28
IR_SIGN=1	RSGN
IR_SIGN=0	NRSGN
IY<0>=1	Y0
IY<0>=0	NY0
BYTE_COUNT+1=0	CNT8
BYTE_COUNT+1<>0	NCNT8
HEX_COUNT+1=0	CNT4
HEX_COUNT+1<>0	NCNT4
ALU_CRY=1	CRY
ALU_CRY=0	NCRY

The next seven tests assume that a twos-complement subtract has been performed on unsigned numbers during the previous cycle.

IS>=IR	CRY
IS<IR	NCRY
ALU_SIGN=1	FSGN
ALU_SIGN=0	NFSGN
OVERFLOW	OVF
ALU=0	FZR
ALU<>0	NFZR

The next four tests assume that a twos-complement subtract has been performed on signed numbers during the previous cycle.

SIGNED_IS>=IR	SGE
SIGNED_IS<IR	NSGE
CARRY=1	CRRY
CARRY=0	NCRRY

ATU Test Conditions

Macro	TSEL Micro-order
INDIRECT	INDR
RING=0	RNG0
RING<>0	NRNG0
INWARD_REFERENCE	RMAX
LA<ESR	LESR
LA>=ESR	NLESR
LA>CRE	GCRE
LA<=CRE	NGCRE
LA=CRE	ECRE
LA<>CRE	NECRE
LA<CRE	LCRE
LA>=CRE	NLCRE
ATU_ON	ATON
ATU_OFF	NATON
CACHE_BLOCK_X	CBLK
ATU_PURGING	PRGB
VALID_PTE	VPTE
VALID_SBR	VSBR

VALIDITY_BIT=1	VLD
VALIDITY_BIT=0	NVLD
PC_REL_INDEX	IXPC

FPU Test Conditions

Macro	TSEL Micro-order
FR=FS	UAEB
FR<>FS	NUAEB
FR<FS	UALB
FR>=FS	NUALB
SIGNED_FR=FS	SAEB
SIGNED_FR<>FS	NSAEB
SIGNED_FR>FS	SAGB
SIGNED_FR<=FS	NSAGB
SIGNED_FR<FS	SALB
SIGNED_FR>=FS	NSALB
MANTISSA_CRY=1	FCRY
MANTISSA_CRY=0	NFCRY
EXPONENT_CRY=1	ECRY
EXPONENT_CRY=0	NECRY
FF8=1	FF8
FF8=0	NFF8

Examples

This section contains an example of unassembled MV/10000 microcode macros and a number of examples of assembled microcode.

Unassembled Example

This example shows the unassembled instructions that produce a dispatch table. This is the same table that was used as an example in Chapter 4. The dispatch table will also appear among the assembled examples.

```
         % ***************************************************************

         %               BIT INSTRUCTION DISPATCH TABLES

         % ***************************************************************


         %       WCOBTAB - Used by WCOB, COB

         % Dispatch table is based on the number of bits set (which is
         % added to DES).
         % AG: CPM <- DES <- DES + CONST; Load DSP REG;
         % ALU: CPD <- PDR <- RSHIFT(PDR); DES <- CPM
         % F bus <- PDR AND M1 for FZR test

         % Location 0 of dispatch table checks for completion of instruction
WCOBTAB: ATTEMPT_NEXT_EFA,  IF ALU=0 RETURN_ELSE_GOTO WCOB1;

         AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
                 IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                         ALU_TEST == A(M1) AND PDR,
                                 CASE_4_INTO WCOBTAB;

         AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
                 IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                         ALU_TEST == A(M1) AND PDR,
                                 CASE_4_INTO WCOBTAB;

         AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
                 IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                         ALU_TEST == A(M1) AND PDR,
                                 CASE_4_INTO WCOBTAB;

         AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
                 IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                         ALU_TEST == A(M1) AND PDR,
                                 CASE_4_INTO WCOBTAB;

         AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
                 IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                         ALU_TEST == A(M1) AND PDR,
                                 CASE_4_INTO WCOBTAB;

         AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
                 IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                         ALU_TEST == A(M1) AND PDR,
                                 CASE_4_INTO WCOBTAB;
```

```
AY = AG(DES) = ALU(DES) == CNST(03) + A(DES),
        IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                ALU_TEST == A(M1) AND PDR,
                        CASE_4_INTO WCOBTAB;

AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
        IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                ALU_TEST == A(M1) AND PDR,
                        CASE_4_INTO WCOBTAB;

AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
        IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                ALU_TEST == A(M1) AND PDR,
                        CASE_4_INTO WCOBTAB;

AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
        IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                ALU_TEST == A(M1) AND PDR,
                        CASE_4_INTO WCOBTAB;

AY = AG(DES) = ALU(DES) == CNST(03) + A(DES),
        IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                ALU_TEST == A(M1) AND PDR,
                        CASE_4_INTO WCOBTAB;

AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
        IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                ALU_TEST == A(M1) AND PDR,
                        CASE_4_INTO WCOBTAB;

AY = AG(DES) = ALU(DES) == CNST(03) + A(DES),
        IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                ALU_TEST == A(M1) AND PDR,
                        CASE_4_INTO WCOBTAB;

AY = AG(DES) = ALU(DES) == CNST(03) + A(DES),
        IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                ALU_TEST == A(M1) AND PDR,
                        CASE_4_INTO WCOBTAB;

AY = AG(DES) = ALU(DES) == CNST(04) + A(DES),
        IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
                ALU_TEST == A(M1) AND PDR,
                        CASE_4_INTO WCOBTAB;
```

Assembled Examples

The following examples show typical MV/10000 microcode in assembled form. The assembler commands precede the assembled microword in each case.

```
Proprietary information of Data General Corporation

   55B  .EJECT;
   56B  .FT 1 "TABLES          Source File               Cycle 1        18-AUG-82 15:24:34 RGG"
   57B  ;
   58B
   59B  /*---------------------------------------------------------------------+
   60B* |                                                                     |
   61B* |      Dispatch table for WCOB instruction, which appears in          |
   62B* |      an example below.                                              |
   63B* |                                                                     |
   64B* +---------------------------------------------------------------------*/
   65B
   66B
   67B          %
   68B          % ************************************************************
   69B          %
   70B          %                BIT INSTRUCTION DISPATCH TABLES
   71B          %
   72B          % ************************************************************
   73B          %
   74B          %
   75B          %
   76B          %       WCOBTAB - Used by WCOB, COB
   77B          %
   78B          % Dispatch table is based on the number of bits set (which is
   79B          % added to DES).
   80B          % AG: CPM <- DES <- DES + CONST; Load DSP REG;
   81B          % ALU: CPD <- PDR <- RSHIFT(PDR); DES <- CPM
   82B          % F bus <- PDR AND M1 for FZR test
   83B          %
   84B          % Location 0 of dispatch table checks for completion of instruction
--0000--WCOBTAB: ATTEMPT_NEXT_EFA,  IF ALU=0 RETURN_ELSE_GOTO WCOB1;
         OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         CRTN  FZR   WCOB1              D  EFA    S@

--DFVs:  addr is WCOB1 (003B)
   86B
--0001--        AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
   88B                 IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
   89B                        ALU_TEST == A(M1) AND PDR,
   90B                               CASE_4_INTO WCOBTAB;
         OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN      LD      M1  DES PD DA AND HR0  M  01
                                                                                              R1
--DFVs:  addr is WCOBTAB (0000)
   91B
--0002--        AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
   93B                 IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
   94B                        ALU_TEST == A(M1) AND PDR,
   95B                               CASE_4_INTO WCOBTAB;
         OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN      LD      M1  DES PD DA AND HR0  M  01
                                                                                              R1
--DFVs:  addr is WCOBTAB (0000)
   96B
--0003--        AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
   98B                 IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
   99B                        ALU_TEST == A(M1) AND PDR,

SAMPLES         Instruction Set Microcode    Rev   1       09-DEC-82 10:47:39 RGG
TABLES          Source File                  Cycle 1       18-AUG-82 15:24:34 RGG
UASM    00.10.00                 0004 - 01
```

```
 1   .EJECT;
 2A  .TITLE "Widgeon Microcode: SAMPLES Code Group"
 3A  ;
 4B  .BEGIN;
 5B  .HD 1   "Proprietary information of Data General Corporation";
 6B  .HD 2   "";
 7B  .RADIX 16;
 8B  .FT 2 "SAMPLES         Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG"
 9B  ;
10B
11B

 ●
 ●
 ●

SAMPLES          Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG

UASM    00.10.00                    0001 - 01
```

```
Proprietary information of Data General Corporation

   12B  /*-------------------------------------------------------------------+
   13B* |                                                                    |
   14B* |       External definitions for Widgeon microcode samples.          |
   15B* |                                                                    |
   16B* |       Some of the samples reference routines that, for the sake    |
   17B* |       of brevity, are not worth including in the samples.  The     |
   18B* |       number of such references in the collection will be kept     |
   19B* |       to a minimum.                                                |
   20B* |                                                                    |
   21B* +-------------------------------------------------------------------*/
   22B
   23B  .EXTERNAL       NSTK_OVERFLOW,
   24B                  WSTK_OVERFLOW,
   25B                  PROTECTION_FAULT,
   26B                  PRIVILEGE_PROTECTION,
   27B                  RESTARTABLE_INTERRUPT;
   28B
   29B  .RADIX  16;

    ●
    ●
    ●
  SAMPLES          Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG

  UASM    00.10.00                    0002 - 01
```

```
Proprietary information of Data General Corporation

   30B  .EJECT;
   31B  .FT 1 "SAMPLES         Source File                   Cycle 1        18-AUG-82 15:24:34 RGG"
   32B  ;
   33B
   34B  /*-------------------------------------------------------------------+
   35B* |                                                                    |
   36B* |       Widgeon Microcode Samples                                    |
   37B* |                                                                    |
   38B* |       This collection of sample microcode is taken directly from   |
   39B* |       Widgeon sources.  Each selection is, as far as practical,    |
   40B* |       the code for an entire macro instruction.  Selections are    |
   41B* |       presented in order of increasing complexity.                 |
   42B* |                                                                    |
   43B* +-------------------------------------------------------------------*/
   44B
   45B  /*-------------------------------------------------------------------+
   46B* |                                                                    |
   47B* |       A note regarding style:                                      |
   48B* |                                                                    |
   49B* |       Having been drawn from the sources, the samples display      |
   50B* |       a variety of documentation and coding styles.  These         |
   51B* |       variations are preserved mainly to minimize the task of      |
   52B* |       compiling the samples.                                       |
   53B* |                                                                    |
   54B* +-------------------------------------------------------------------*/
    ●
    ●
    ●

SAMPLES          Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
SAMPLES          Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM    00.10.00                    0003 - 01
```

```
Proprietary information of Data General Corporation

   55B .EJECT;
   56B .FT 1 "TABLES          Source File                  Cycle 1        18-AUG-82 15:24:34 RGG"
   57B ;
   58B
   59B /*----------------------------------------------------------------------+
   60B* |                                                                       |
   61B* |       Dispatch table for WCOB instruction, which appears in           |
   62B* |       an example below.                                               |
   63B* |                                                                       |
   64B* +----------------------------------------------------------------------*/
   65B
   66B
   67B          %
   68B          % **************************************************************
   69B          %
   70B          %               BIT INSTRUCTION DISPATCH TABLES
   71B          %
   72B          % **************************************************************
   73B          %
   74B          %
   75B          %
   76B          %       WCOBTAB - Used by WCOB, COB
   77B          %
   78B          % Dispatch table is based on the number of bits set (which is
   79B          % added to DES).
   80B          % AG: CPM <- DES <- DES + CONST; Load DSP REG;
   81B          % ALU: CPD <- PDR <- RSHIFT(PDR); DES <- CPM
   82B          % F bus <- PDR AND M1 for FZR test
   83B          %
   84B          % Location 0 of dispatch table checks for completion of instruction
--0000--WCOBTAB: ATTEMPT_NEXT_EFA,  IF ALU=0 RETURN_ELSE_GOTO WCOB1;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  FZR   WCOB1                  D  EFA    S@

--DFVs:   addr is WCOB1 (003B)
   86B
--0001--        AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
   88B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
   89B                          ALU_TEST == A(M1) AND PDR,
   90B                                  CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1  DES PD DA AND HR0  M  01
                                                                                             R1
--DFVs:   addr is WCOBTAB (0000)
   91B
--0002--        AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
   93B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
   94B                          ALU_TEST == A(M1) AND PDR,
   95B                                  CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1  DES PD DA AND HR0  M  01
                                                                                             R1
--DFVs:   addr is WCOBTAB (0000)
   96B
--0003--        AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
   98B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
   99B                          ALU_TEST == A(M1) AND PDR,

SAMPLES         Instruction Set Microcode    Rev  1        09-DEC-82 10:47:39 RGG
TABLES          Source File                  Cycle 1       18-AUG-82 15:24:34 RGG
UASM    00.10.00                   0004 - 01
```

```
Proprietary information of Data General Corporation
  100B                             CASE_4_INTO WCOBTAB;
            OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA   IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
            DSPA         WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1   DES PD DA AND HR0  M  02
                                                                                                   R1
--DFVs:     addr is WCOBTAB (0000)
  101B
--0004--          AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
  103B                    IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  104B                            ALU_TEST == A(M1) AND PDR,
  105B                                    CASE_4_INTO WCOBTAB;
            OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA   IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
            DSPA         WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1   DES PD DA AND HR0  M  01
                                                                                                   R1
--DFVs:     addr is WCOBTAB (0000)
  106B
--0005--          AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
  108B                    IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  109B                            ALU_TEST == A(M1) AND PDR,
  110B                                    CASE_4_INTO WCOBTAB;
            OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA   IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
            DSPA         WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1   DES PD DA AND HR0  M  02
                                                                                                   R1
--DFVs:     addr is WCOBTAB (0000)
  111B
  112B
--0006--          AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
  114B                    IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  115B                            ALU_TEST == A(M1) AND PDR,
  116B                                    CASE_4_INTO WCOBTAB;
            OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA   IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
            DSPA         WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1   DES PD DA AND HR0  M  02
                                                                                                   R1
--DFVs:     addr is WCOBTAB (0000)
  117B
--0007--          AY = AG(DES) = ALU(DES) == CNST(03) + A(DES),
  119B                    IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  120B                            ALU_TEST == A(M1) AND PDR,
  121B                                    CASE_4_INTO WCOBTAB;
            OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA   IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
            DSPA         WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1   DES PD DA AND HR0  M  03
                                                                                                   R1
--DFVs:     addr is WCOBTAB (0000)
  122B
--0008--          AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
  124B                    IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  125B                            ALU_TEST == A(M1) AND PDR,
  126B                                    CASE_4_INTO WCOBTAB;
            OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA   IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
            DSPA         WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1   DES PD DA AND HR0  M  01
                                                                                                   R1
--DFVs:     addr is WCOBTAB (0000)
  127B
--0009--          AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
  129B                    IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  130B                            ALU_TEST == A(M1) AND PDR,


SAMPLES        Instruction Set Microcode    Rev   1       09-DEC-82 10:47:39 RGG
TABLES         Source File                  Cycle 1       18-AUG-82 15:24:34 RGG
UASM    00.10.00                 0005 - 01
```

```
Proprietary information of Data General Corporation

 131B                                   CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1  DES PD DA AND HR0  M  02
                                                                                                R1
--DFVs:   addr is WCOBTAB (0000)
 132B
--000A--        AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
 134B                   IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
 135B                           ALU_TEST == A(M1) AND PDR,
 136B                                   CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1  DES PD DA AND HR0  M  02
                                                                                                R1
--DFVs:   addr is WCOBTAB (0000)
 137B
--000B--        AY = AG(DES) = ALU(DES) == CNST(03) + A(DES),
 139B                   IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
 140B                           ALU_TEST == A(M1) AND PDR,
 141B                                   CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1  DES PD DA AND HR0  M  03
                                                                                                R1
--DFVs:   addr is WCOBTAB (0000)
 142B
--000C--        AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
 144B                   IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
 145B                           ALU_TEST == A(M1) AND PDR,
 146B                                   CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1  DES PD DA AND HR0  M  02
                                                                                                R1
--DFVs:   addr is WCOBTAB (0000)
 147B
--000D--        AY = AG(DES) = ALU(DES) == CNST(03) + A(DES),
 149B                   IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
 150B                           ALU_TEST == A(M1) AND PDR,
 151B                                   CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1  DES PD DA AND HR0  M  03
                                                                                                R1
--DFVs:   addr is WCOBTAB (0000)
 152B
--000E--        AY = AG(DES) = ALU(DES) == CNST(03) + A(DES),
 154B                   IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
 155B                           ALU_TEST == A(M1) AND PDR,
 156B                                   CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1  DES PD DA AND HR0  M  03
                                                                                                R1
--DFVs:   addr is WCOBTAB (0000)
 157B
--000F--        AY = AG(DES) = ALU(DES) == CNST(04) + A(DES),
 159B                   IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
 160B                           ALU_TEST == A(M1) AND PDR,


SAMPLES         Instruction Set Microcode      Rev   1         09-DEC-82 10:47:39 RGG
TABLES          Source File                    Cycle 1         18-AUG-82 15:24:34 RGG
UASM    00.10.00                   0006 - 01
```

```
Proprietary information of Data General Corporation

 161B                                   CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN       LD      M1  DES PD DA AND HR0  M  04
                                                                                                R1
--DFVs:   addr is WCOBTAB (0000)


   •
   •
   •


 SAMPLES         Instruction Set Microcode      Rev   1         09-DEC-82 10:47:39 RGG
 TABLES          Source File                    Cycle 1         18-AUG-82 15:24:34 RGG
 UASM    00.10.00                   0007 - 01
```

```
Proprietary information of Data General Corporation

  162B .EJECT;
  163B .FT 1 "MEM           Source File                   Cycle 1        18-AUG-82 15:24:34 RGG"
  164B ;
  165B
  166B /*-----------------------------------------------------------------------+
  167B* |                                                                       |
  168B* |      Memory references for the next macro instruction can be          |
  169B* |      started by the IP from decode information.  In these two         |
  170B* |      examples, the completion of an IP initiated memory               |
  171B* |      reference is shown.  The completion is generic, i.e. read        |
  172B* |      or write.  The start instigated by the IP specified the          |
  173B* |      exact type of transfer to perform.                               |
  174B* |                                                                       |
  175B* |      Also shown here is the attempt of the next EFA on behalf         |
  176B* |      of the next executing macro instruction.  This attempt must      |
  177B* |      be made in the last micro cycle of every macro instruction       |
  178B* |      The combination of the attempt and popping an empty micro        |
  179B* |      stack constitutes a macro instruction pop (IPOP).                |
  180B* |                                                                       |
  181B* +----------------------------------------------------------------------*/
  182B
  183B
  184B           %*********
  185B           %  Load and Store Instructions:  <<L X><W N> E ><LDA STA>
  186B           %
  187B           %  Perform load or store of AC pointed to by DES.  IPOP.
  188B           %*********
  189B
--0010--LWLDA:
--0010--XWLDA:
--0010--LNLDA:
--0010--XNLDA:
--0010--ELDA:
--0010--LDA:    CPM = AG(DES) = ALU(DES) == MEM_READ,  ATTEMPT_NEXT_EFA,  RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                 DES D  EFA M  S@ R  MM                      DES                M

--DFVs:
  196B
--0011--LWSTA:
--0011--XWSTA:
--0011--LNSTA:
--0011--XNSTA:
--0011--ESTA:
--0011--STA:    CPM = MEM_WRITE == ALU(DES),  ATTEMPT_NEXT_EFA,  RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                     D  EFA    S@ W  IA                  DES

--DFVs:
  203B
  204B
  205B           %*********
  206B           %  Jump Instructions:  WBR, LJMP, XJMP, EJMP
  207B           %
  208B           %  Complete IPST.  Go to IP_ALT WAIT.
  209B           %*********
  210B
--0012--WBR:
SAMPLES         Instruction Set Microcode     Rev  1        09-DEC-82 10:47:39 RGG
MEM             Source File                   Cycle 1       18-AUG-82 15:24:34 RGG
UASM    00.10.00                  0008 - 01
```

```
Proprietary information of Data General Corporation

--0012--LJMP:
--0012--XJMP:
--0012--EJMP:
--0012--JMP:    COMPLETE_JUMP,  GOTO WAIT;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          LEAP        WAIT                            W

--DFVs:   addr is WAIT (0FFF)
  216B
  217B
  218B           %*********
  219B           %  Jump Subroutine Instructions:  <L X E >JSR
  220B           %
  221B           %  Read PCN (Return PC) into PDR;  Complete IPST.
  222B           %
  223B           %  Move PDR to AC3, AG3; and IPOP.
  224B           %
  225B           %*********
  226B
--0013--LJSR:
--0013--XJSR:
--0013--EJSR:
--0013--JSR:    COMPLETE_JUMP,  PDR == RETURN_PC;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE                                 W      PCN

--DFVs:
  231B
--0014--        IY = AG(AG3) = ALU(AC3) == PDR AND A(M1),
  233B                  ATTEMPT_NEXT_EFA,  RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                 AG3 D  EFA M  S@    IY                  M1  AC3 PD AD AND PASS Y

--DFVs:

  •
  •
  •

SAMPLES         Instruction Set Microcode     Rev  1        09-DEC-82 10:47:39 RGG
MEM             Source File                   Cycle 1       18-AUG-82 15:24:34 RGG
UASM    00.10.00                  0009 - 01
```

```
Proprietary information of Data General Corporation

  234B  .EJECT;
  235B  .FT 1 "ALC             Source File                   Cycle 1        18-AUG-82 15:24:34 RGG"
  236B  ;
  237B
  238B  /*--------------------------------------------------------------------+
  239B* |                                                                     |
  240B* |      Some Nova ALC instructions illustrate the use of the ALU       |
  241B* |      for simple arithmetic.  The shift operation is used along      |
  242B* |      with the ALC opcode to provide the decode address.  Carry,     |
  243B* |      no-load and skip options are accelerated with hardware.        |
  244B* |                                                                     |
  245B* +--------------------------------------------------------------------*/
  246B
  247B
  248B          %*********
  249B          %  NOVA Arithmetic and Logical Instructions
  250B          %
  251B          %      EXECUTION TIME:         1 cycle no skip
  252B          %                              2 cycles skip, no EFA required
  253B          %
  254B          %  Perform ALU operation; then Pass, Shift, or Swap; Write result to AG
  255B          %      and ALU AC pointed to by DES.  Enable ALC skip and IPOP.
  256B          %*******
  257B
--0015--ADD:    IY = AG(DES) = ALU(DES) == B(DES) + A(SRC),
  259B                  CARRY == ALC_CRY,  SKIP_ON ALC_RESULT,
  260B                  ATTEMPT_NEXT_EFA,  RETURN;
        OP      TSEL  ADDRESS  D AA  AB  AG AOP AL  ST  CM  CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CRTN    TRUE                 DES D  EFA M  S@     IY       XZ ALC           SRC DES BR AD ADD PASS Y

--DFVs:
  261B
--0016--INC:    IY = AG(DES) = ALU(DES) == ZERO +1+ A(SRC),
  263B                  CARRY == ALC_CRY,  SKIP_ON ALC_RESULT,
  264B                  ATTEMPT_NEXT_EFA,  RETURN;
        OP      TSEL  ADDRESS  D AA  AB  AG AOP AL  ST  CM  CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CRTN    TRUE                 DES D  EFA M  S@     IY       XZ ALC           SRC DES ZR AD CAD PASS Y

--DFVs:
  265B
--0017--SUBL:   IY = AG(DES) = ALU(DES) == BIT_SHIFT_LEFT( B(DES) - A(SRC) ),
  267B                  CARRY == ALC_CRY,  SKIP_ON ALC_RESULT,
  268B                  ATTEMPT_NEXT_EFA,  RETURN;
        OP      TSEL  ADDRESS  D AA  AB  AG AOP AL  ST  CM  CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CRTN    TRUE                 DES D  EFA M  S@     IY       XZ ALC           SRC DES BR AD CSR BLO  Y

--DFVs:
  269B


SAMPLES          Instruction Set Microcode     Rev  1         09-DEC-82 10:47:39 RGG
ALC              Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM     00.10.00                  0010 - 01
```

```
Proprietary information of Data General Corporation

  270B  .EJECT;
  271B  .FT 1 "IMMEDIATE       Source File                   Cycle 1        18-AUG-82 15:24:34 RGG"
  272B  ;
  273B
  274B  /*-------------------------------------------------------------------+
  275B* |                                                                   |
  276B* |       A Load Effective Address instruction is nothing more        |
  277B* |       than an aborted memory reference.  The final contents of    |
  278B* |       the Logical Address Register are loaded, via the CPD bus    |
  279B* |       and ALU, into the required registers.                       |
  280B* |                                                                   |
  281B* |       The architecture specifies that the effective address is    |
  282B* |       checked for a ring crossing error.  This check will not     |
  283B* |       be performed by hardware because the memory operation       |
  284B* |       used to generate the address is aborted.  A micro test      |
  285B* |       is used to check validity.                                  |
  286B* |                                                                   |
  287B* |       This example also shows the use of a conditional IPOP.  A   |
  288B* |       memory abort operation is recommended following the failure |
  289B* |       of a conditional IPOP.                                      |
  290B* |                                                                   |
  291B* +-------------------------------------------------------------------*/
  292B
  293B
  294B          %============================================================
  295B          %       Load Effective Address:  LEF, ELEF, XLEF, LLEF
  296B          %
  297B          %  Load LAR into the AG and ALU DES registers. Abort Memory. If RMAX is
  298B          %       violated, then go to the RMAX Protection routine, else IPOP.
  299B          %  Load the RMAX fault code into GR0 and faulting address into
  300B          %       AR5 and go to the Protection routine.
  301B          %============================================================
  302B
--0018--LLEF:
--0018--XLEF:
--0018--ELEF:
--0018--LEF:    IY = AG(DES) = ALU(DES) == SPAD(BIT0) NOT_AND LAR,
  307B              EXTEND_MICRO_CYCLE,
  308B              ATTEMPT_NEXT_EFA,   ABORT_MEMORY,
  309B              IF NOT INWARD_REFERENCE RETURN_ELSE_GOTO RMAX_PROTECTION;
        OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CRTN NRMAX  RMAX_PRO       DES D  EFA M  S@ A  IY  LAR GN XTND                  DES SC CD ANC PASS Y  BIT0

--DFVs:   addr is RMAX_PROTECTION (0019)  const is BIT0 (0000)
  310B
--0019--RMAX_PROTECTION:
  312B          ABORT_MEMORY,
  313B          IY = ALU(GR0) == CNST(PRT_RMX) OR CPD_ZERO,
  314B              CPM = AG(AR5) == ALU(DES), GOTO PROTECTION_FAULT;
        OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        LEAP        PROTECTI       AR5        M     A  IA  N                    DES GR0 CN CD OR  PASS Y  PRT_RMX

--DFVs:   addr is PROTECTION_FAULT (0002 *EXT*)  const is PRT_RMX (0004)
  315B
  316B


SAMPLES         Instruction Set Microcode     Rev  1         09-DEC-82 10:47:39 RGG
IMMEDIATE       Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM     00.10.00                  0011 - 01
```

```
Proprietary information of Data General Corporation

 317B  /*-----------------------------------------------------------------------+
 318B* |                                                                        |
 319B* |       Instructions which load immediate data from the instruction      |
 320B* |       stream use an approach similar to the LEF instructions. In       |
 321B* |       their case, the immediate data has been loaded into the LAR      |
 322B* |       by the IP as specified by decode information, but no memory      |
 323B* |       reference has been initiated.                                    |
 324B* |                                                                        |
 325B* +-----------------------------------------------------------------------*/
 326B
 327B
 328B  %********
 329B  %  Long Add Immediates:  <W N >ADDI
 330B  %       DES  +  (Displacement)  ->  DES       (Displacement is in LAR)
 331B  %       The W and N types load overflow into OVR and CRY<0 16> into CARRY.
 332B  %********
 333B
--001A--ADDI:   IY = ALU(DES) = AG(DES) == B(DES) + LAR,  ATTEMPT_NEXT_EFA, RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                 DES D  EFA M  S@    IY  LAR                     DES BR CD ADD PASS Y

--DFVs:
 335B
--001B--WADDI:  % For the 32-bit Immediate
--001B--WNADI:  % For the 16-bit Immediate
--001B--NADDI:  IY = ALU(DES) = AG(DES) == A(DES) + LAR,
 339B                  CARRY == ALU_CRY,  UPDATE_OVR,
 340B                  ATTEMPT_NEXT_EFA,  RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                 DES D  EFA M  S@    IY  LAR XZ LOVC         DES DES    CA ADD PASS Y

--DFVs:
 341B
 342B

   •
   •
   •

SAMPLES         Instruction Set Microcode      Rev   1         09-DEC-82 10:47:39 RGG
IMMEDIATE       Source File                    Cycle 1         18-AUG-82 15:24:34 RGG
UASM     00.10.00                  0012 - 01
```

```
Proprietary information of Data General Corporation

 343B  /*-----------------------------------------------------------------------+
 344B* |                                                                        |
 345B* |       A short immediate field is derived from the source               |
 346B* |       accumulator bit field of the macro instruction.  Actual          |
 347B* |       values are 0 through 3, but implied values are 1 through 4.      |
 348B* |       The following instructions read an operand from a memory         |
 349B* |       location, subtract the implied immediate data from it, and       |
 350B* |       store the result back in the same memory location.               |
 351B* |                                                                        |
 352B* +-----------------------------------------------------------------------*/
 353B
 354B
 355B  %********
 356B  %  Short Subtract Immediate from Memory: <L X><W N >SBI
 357B  %
 358B  %       MEM  -  ([ACS] + 1)  ->  MEM
 359B  %
 360B  %  Read Memory operand into ALU (GR0).  Start same address to write back.
 361B  %       Go to XLSBI to complete the operation and perform the write.
 362B  %********
 363B
--001C--XWSBI:
--001C--LWSBI:  CPM = ALU(GR0) == MEM_READ,
 366B                  START AY == PASS(LAST_LA) FOR WRITE_DOUBLE,
 367B                  GOTO XLSBI;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          LEAP        XLSBI              L  PSB    WD R  MM                          GR0                M

--DFVs:   addr is XLSBI (001E)
--001D--XNSBI:
--001D--LNSBI:  CPM = ALU(GR0) == MEM_READ,
 370B                  START AY == PASS(LAST_LA) FOR WRITE_WORD,
 371B                  GOTO XLSBI;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          LEAP        XLSBI              L  PSB    WW R  MM                          GR0                M

--DFVs:   addr is XLSBI (001E)
 372B
--001E--XLSBI:  IY = MEM_WRITE == A(GR0) -1- SRC_POINTER,
 374B                  CARRY == ALU_CRY,  UPDATE_OVR,
 375B                  ATTEMPT_NEXT_EFA, RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                     D  EFA    S@ W  IY      XZ LOVC         GR0     AS DA SMR PASS

--DFVs:

   •
   •
   •

SAMPLES         Instruction Set Microcode      Rev   1         09-DEC-82 10:47:39 RGG
IMMEDIATE       Source File                    Cycle 1         18-AUG-82 15:24:34 RGG
UASM     00.10.00                  0013 - 01
```

```
Proprietary information of Data General Corporation

 .EJECT;
 .FT 1 "BYTE          Source File                   Cycle 1        18-AUG-82 15:24:34 RGG"
 ;

 /*------------------------------------------------------------------+
 |                                                                   |
 |       These selections from the byte microcode show the use of   |
 |       a conditional subroutine call, a memory start using the    |
 |       address generator, and a memory abort.  A copy of the PC   |
 |       of execution + 1 is moved to the address generator by first|
 |       loading PCX into PDR, and then subtracting -1 from it and  |
 |       loading the result in the AG register file via the CPM bus.|
 |                                                                   |
 +-----------------------------------------------------------------*/


 %*******************************************************************
 %
 %                   Byte   EFA    Instructions
 %                   ---------------------------
 %
 %       EFA calculations for a Byte address cannot be completely performed
 %       by the hardware.  The PC relative index case cannot be performed
 %       since the Displacement is a byte address and the PC is a word address.
 %       The AG converts the byte displacement and performs the other indexing.
 %
 %*******************************************************************


         %******
         %                  Subroutines to perform
         %             READ/WRITE PC Relative Byte Addresses
         %             -------------------------------------
         %       PC Relative Addressing must be handled separately
         %       since the IP cannot align a Byte displacement.
         %
         %  Abort the previous start and move the PC of the instruction
         %       plus 1 (PC of the DISP) to the AG (AR0).
         %  Form the PC relative address by adding the Displacement (LAST_LA)
         %       to the PC (AR0) and start for the Byte Read/Write.  Word
         %       addressing must be forced since the addresses have already
         %       been aligned to word addresses.  Return to the caller.
         %******

--001F--READ_PC_BYTE:
         IY = AG(AR0) == PDR - A(M1),  ABORT_MEMORY;
     OP   TSEL  ADDRESS  D AA  AB  AG AOP AL  ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
     CJMP FALSE              AR0      M    A   IY                        M1      PD AD CSR PASS

--DFVs:

--0020--         START AY == LAST_LA + A(AR0) WITH_WORD_ADDRESSING FOR READ_BYTE,
                     RETURN;
     OP   TSEL  ADDRESS  D AA  AB  AG AOP AL  ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
     CRTN TRUE             AR0      L  ADD    RB             AT WORD

--DFVs:

SAMPLES          Instruction Set Microcode      Rev  1        09-DEC-82 10:47:39 RGG
BYTE             Source File                    Cycle 1       18-AUG-82 15:24:34 RGG
UASM      00.10.00                    0014 - 01
```

```
Proprietary information of Data General Corporation

  426B
--0021--WRITE_PC_BYTE:
  428B          IY = AG(AR0) == PDR - A(M1),  ABORT_MEMORY;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE               AR0        M     A  IY                   M1      PD AD CSR PASS

--DFVs:
  429B
--0022--        START AY == LAST_LA + A(AR0) WITH_WORD_ADDRESSING FOR WRITE_BYTE,
  431B                 RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE             AR0      L  ADD    WB              AT WORD

--DFVs:
  432B
  433B
  434B  %********
  435B  %       Load Byte:   LDB, WLDB
  436B  %
  437B  %  Start the byte address in the SRC accumulator for a Read byte.
  438B  %       Finish the operation at LDA.
  439B  %********
  440B
--0023--WLDB:
--0023--LDB:    START AY == PASS(B(SRC)) FOR READ_BYTE,  GOTO LDA;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          LEAP        LDA                SRC B  PSB    RB

--DFVs:   addr is LDA (0010)
  443B
  444B
  445B  %********
  446B  %       Indexed Load/Store Byte:   LLDB, XLDB, ELDB
  447B  %
  448B  %  Check for PC relative Addressing before attempting complete of operation.
  449B  %       Get PC of instruction into PDR in case of PC relative addressing.
  450B  %       Subroutine to Start the correct address if index was PC relative.
  451B  %  Complete the operation and IPOP:
  452B  %       LDB:  Store the read byte into the AG and the ALU DES.
  453B  %********
  454B
--0024--LLDB:
--0024--XLDB:
--0024--ELDB:   PDR == PC_OF_EXECUTION,  IF PC_REL_INDEX CALL READ_PC_BYTE;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJSR  IXPC  READ_PC_                             PCX

--DFVs:   addr is READ_PC_BYTE (001F)
  458B
--0025--        CPM = AG(DES) = ALU(DES) == MEM_READ,  ATTEMPT_NEXT_EFA,  RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                DES D  EFA M  S@ R  MM                       DES                   M

--DFVs:
  460B
  461B


SAMPLES         Instruction Set Microcode     Rev    1        09-DEC-82 10:47:39 RGG
BYTE            Source File                   Cycle 1         18-AUG-82 15:24:34 RGG
UASM     00.10.00                 0015 - 01
```

```
Proprietary information of Data General Corporation

  462B  .EJECT;
  463B  .FT 1 "XCT              Source File               Cycle 1          18-AUG-82 15:24:34 RGG"
  464B  ;
  465B
  466B  /*-----------------------------------------------------------------+
  467B* |                                                                  |
  468B* |        The code for the XCT instruction address SPAD with a      |
  469B* |        constant, uses the flags to control sequencing and does   |
  470B* |        a word zero extend with the hex shifter.                  |
  471B* |                                                                  |
  472B* +-----------------------------------------------------------------*/
  473B
  474B
  475B  /*---------------------------------------------------------+
  476B* |                                                          |
  477B* |              XCT  Execute an AC's contents               |
  478B* |                                                          |
  479B* |       DES contains the opcode to be executed             |
  480B* |                                                          |
  481B* +---------------------------------------------------------*/
  482B
  483B  %        Enter here for ordinary XCT.  If restarting or resuming,
  484B  %        XCTed opcode must still be in DES.  Bit 0 of double word saved in
  485B  %        SPAD is cleared to indicate to interrupt handlers that saving
  486B  %        XCT opcode on wide stack is not required.  Start execute.
  487B
--0026--EXECUTE:
--0026--XCT:     GOTO &;  % Wait for XCTED_INSTRUCTION test to setup.
          OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          LEAP         &

--DFVs:   addr is & (0027)
  490B
--0027--         ID == SPAD( XCTOP ),  IF NOT XCTED_INSTRUCTION GOTO NORMAL_XCT;
          OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP NXCTF  NORMAL_X                                                        SC                   XCTOP

--DFVs:   addr is NORMAL_XCT (0029)  const is XCTOP (00C3)
  492B
  493B     %     If XCT was executed by a PBX, then the XCT should set Bit0 of
  494B     %     XCTOP since it was virtually executed by the PBX.
  495B
--0028--         IF ID_SIGN=1 GOTO EXECUTE_PBX;
          OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP   DSGN  EXECUTE_

--DFVs:   addr is EXECUTE_PBX (002D)
  497B
--0029--NORMAL_XCT:
  499B           IY = SPAD( XCTOP ) == WORD_ZERO_EXTEND (A(DES)),  START_EXECUTE;
          OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP   FALSE            AG0 AG0 B  SUB    WW              AT CM0      WC  DES         AD      WZX  NY XCTOP

--DFVs:   const is XCTOP (00C3)
  500B
  501B  %        Send instruction to the IP via the Cache.
  502B
--002A--         CPM = EXECUTE_DATA == ALU(DES),
SAMPLES          Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
XCT              Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM    00.10.00                   0016 - 01
```

```
Proprietary information of Data General Corporation

  504B                MODIFY_FLAGS_4567( CLEAR, N, N, N ),
  505B                     GOTO XCT_WAIT1;
           OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
           LEAP        XCT_WAIT                        A  IA      GN MFS1         DES
                                                                                                            C  N  N   N
--DFVs:    addr is XCT_WAIT1 (002B)
  506B
  507B  %         Wait 4 cycles before IPOPing.
  508B
--002B--XCT_WAIT1:  MODIFY_FLAGS_4567( TOGGLE, N, N, N ),
  510B                     IF FLAG4=0 GOTO XCT_WAIT1;
           OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
           CJMP  NFLG4 XCT_WAIT                                   GN MFS1
                                                                                                            T  N  N   N
--DFVs:    addr is XCT_WAIT1 (002B)
  511B
--002C--XCT_WAIT2:  MODIFY_FLAGS_4567( TOGGLE, N, N, N ),  ATTEMPT_NEXT_EFA,
  513B                     IF FLAG4=1 RETURN_ELSE_GOTO XCT_WAIT2;
           OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
           CRTN  FLG4  XCT_WAIT           D  EFA    S@            GN MFS1
                                                                                                            T  N  N   N
--DFVs:    addr is XCT_WAIT2 (002C)
  514B
  515B
  516B
  517B  %         Enter here from PBX or BKPT after PBX detected.  Opcode was
  518B  %         placed in DES.  Must set bit 0 before saving in SPAD to tell
  519B  %         interrupt handlers to save opcode on wide stack.  Start execute.
  520B  %         Push address for return from WAIT.
  521B
--002D--EXECUTE_PBX:
  523B            IY = ALU(GR5) == SPAD(BIT0) OR A(DES),  START_EXECUTE;
           OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
           CJMP  FALSE            AG0 AG0 B  SUB    WW            AT CM0          DES GR5 SC AD OR  PASS Y  BIT0

--DFVs:    const is BIT0 (0000)
  524B
  525B  %         Save executed opcode in SPAD.  Send instruction to IP.
  526B
--002E--          IY = SPAD(XCTOP) == ZERO OR A(GR5),
  528B                CPM = EXECUTE_DATA == ALU(GR5);
           OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
           CJMP  FALSE                                 A  IA      GN          WC  GR5     ZR AD OR  PASS NY XCTOP

--DFVs:    const is XCTOP (00C3)
  529B
--002F--          MODIFY_FLAGS_4567( SET, N, N, N ),   GOTO XCT_WAIT1;
           OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
           LEAP        XCT_WAIT                                   GN MFS1
                                                                                                            S  N  N   N
--DFVs:    addr is XCT_WAIT1 (002B)




SAMPLES          Instruction Set Microcode     Rev  1        09-DEC-82 10:47:39 RGG
XCT              Source File                   Cycle 1       18-AUG-82 15:24:34 RGG
UASM     00.10.00                  0017 - 01
```

```
Proprietary information of Data General Corporation

  531B  .EJECT;
  532B  .FT 1 "BIT             Source File                   Cycle 1        18-AUG-82 15:24:34 RGG"
  533B  ;
  534B
  535B  /*----------------------------------------------------------------------+
  536B* |                                                                       |
  537B* |       This selection from the bit microcode provides varied           |
  538B* |       examples of the use of the ALU hardware.  Also, this            |
  539B* |       code does a read-modify-write memory operation.  Note           |
  540B* |       the start for a write, and the subsequent read and write        |
  541B* |       completions.                                                    |
  542B* |                                                                       |
  543B* |       The subroutine WBITW is used to resolve indirection chains.     |
  544B* |       It requires three cycles per defer level because the word       |
  545B* |       pointer of the bit address is indirectable, while the final     |
  546B* |       bit address is formed by adding the bit offset to the           |
  547B* |       resolved word pointer.                                          |
  548B* |                                                                       |
  549B* +----------------------------------------------------------------------*/
  550B
  551B          %
  552B          %
  553B          %
  554B          %       WSZBO - Wide Skip on Zero Bit and set bit to One
  555B          %
  556B          %       5 cycles minimum
  557B          %     + 3 cycles for each level of indirection
  558B
  559B          % Skip if SRC=DES, AR0 <- RSHIFT(DES), D bus <- SRC
  560B
--0030--WSZBO:  IY = AG(AR0) == HEX_SHIFT_RIGHT(R1, A(DES)),
  562B          ID == B(SRC),
  563B          SPAR == BIT16,
  564B          IF SRC=DES GOTO WSZBOBIT;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  COMP  WSZBOBIT       AR0        M        IY      GN SPCN     LS  DES SRC BR AD     HR0      BIT16
                                                                                               R1
--DFVs:   addr is WSZBOBIT (0035)  const is BIT16 (0010)
  565B
  566B          % Call WBITW if indirect, LA(WW) <- SRC+AR0, GR1 <- DES AND 0F
  567B
--0031--        START AY == B(AR0) + A(SRC) FOR WRITE_WORD,
  569B          IY = SPAR_TABLE_OFFSET  == B(M1) AND A(DES),
  570B          IF ID_SIGN=1 CALL WBITW;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJSR  DSGN  WBITW      SRC AR0 B  ADD    WW             GN SPY4     LS  DES M1  BR AD AND PASS

--DFVs:   addr is WBITW (0036)
  571B
  572B          % GR0 <- CPM, SPAR <- GR1 OR BIT16
  573B
--0032--WSZBONRM: IY = ALU(GR0)   == SPAD(SPAR) AND A(M1),
  575B          CPM = TREG == MEM_READ;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE                                R  MM      GN      LT      M1  GR0 SS AD AND PASS Y

--DFVs:
  576B
SAMPLES         Instruction Set Microcode     Rev   1       09-DEC-82 10:47:39 RGG
BIT             Source File                   Cycle 1       18-AUG-82 15:24:34 RGG
UASM    00.10.00                   0018 - 01
```

```
Proprietary information of Data General Corporation

  577B          % Set WORD(BIT#)=1, Note, an unmodified copy exists in GR0 for testing.
  578B
--0033--        IY = MEM_WRITE == A(GR0) OR TREG;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE                                 W  IY  TRG                 GR0        CA OR  PASS

--DFVs:
  580B
  581B          % Macro skip if WORD(BIT#)=0 (GR0 * SPAD(SPAR)=0), else IPOP
  582B
--0034--        ATTEMPT_NEXT_EFA,
  584B          IY == A(GR0) AND TREG,
  585B          SKIP_ON ALU=0,
  586B          RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                     D  EFA    S@        TRG GN WSKP         GR0        CA AND PASS
                                                                                                           FZR
--DFVs:
  587B
  588B          % LA(RW) <- AR0, append CRE, GR1 <- DES AND 0F
  589B
--0035--WSZBOBIT: START AY == PASS(B(AR0)) IN_CURRENT_RING FOR WRITE_WORD,
  591B          IY = SPAR_TABLE_OFFSET  == B(M1) AND A(DES),
  592B          GOTO WSZBONRM;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          LEAP        WSZBONRM       AR0 B  PSB    WW            GN SPY4 AC  LS  DES M1  BR AD AND PASS

--DFVs:   addr is WSZBONRM (0032)
  593B
  594B
  595B          % *************************************************************
  596B          %
  597B          %
  598B          %
  599B          % WBITW - RESOLVES WIDE BIT INDIRECTION (MEMORY WRITE)
  600B          %
  601B          % Used by WBTO, WBTZ, WSZBO
  602B          %
  603B          % AR1 <- SRC (First time only), LA(RD) <- AR1, Abort assumed WW
  604B          % Use defer random to be sure indirection does not exceed 15 levels.
  605B
--0036--WBITW:  START AY = AG(AR1) == CNST(0) + A(SRC) FOR READ_DOUBLE,
  607B          ABORT_MEMORY,
  608B          DEFER_ON_FALSE_TEST,
  609B          GOTO WBITW2;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          LEAP        WBITW2     SRC AR1 C  ADD Y  RD A          AT      DF                                00

--DFVs:   addr is WBITW2 (0038)
  610B
  611B          % Continue resolving indirection, LA(RD) <- AR1% Abort WW
  612B
--0037--WBITW1: START AY == PASS(B(AR1)) FOR READ_DOUBLE,
  614B          ABORT_MEMORY,



SAMPLES         Instruction Set Microcode     Rev    1       09-DEC-82 10:47:39 RGG
BIT             Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM    00.10.00                   0019 - 01
```

```
Proprietary information of Data General Corporation

  615B          DEFER_ON_FALSE_TEST;
         OP     TSEL   ADDRESS  D AA   AB   AG AOP AL ST CM CPM CPD RM R0    R1  R2   IA   IB   ID RS IOP IY    IL FR FS FOP W FCW FL FRG X
         CJMP   FALSE                  AR1  B  PSB    RD A           AT      DF

--DFVs:
  616B
  617B          % AR1 <- CPM
  618B
--0038--WBITW2: CPM = AG(AR1) == MEM_READ;
         OP     TSEL   ADDRESS  D AA   AB   AG AOP AL ST CM CPM CPD RM R0    R1  R2   IA   IB   ID RS IOP IY    IL FR FS FOP W FCW FL FRG X
         CJMP   FALSE                  AR1         M      R  MM

--DFVs:
  620B
  621B          % Return if no indirection, LA(WW) <- AR1+AR0
  622B
--0039--        START AY == B(AR0) + A(AR1) FOR WRITE_WORD,
  624B          IF NOT INDIRECT RETURN_ELSE_GOTO WBITW1;
         OP     TSEL   ADDRESS  D AA   AB   AG AOP AL ST CM CPM CPD RM R0    R1  R2   IA   IB   ID RS IOP IY    IL FR FS FOP W FCW FL FRG X
         CRTN   NINDR  WBITW1     AR1  AR0  B  ADD    WW

--DFVs:  addr is WBITW1 (0037)
  625B
  626B

   •
   •
   •

SAMPLES         Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
BIT             Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM     00.10.00                0020 - 01
```

```
Proprietary information of Data General Corporation

  627B  /*-------------------------------------------------------------------+
  628B* |                                                                     |
  629B* |       Wide Count Bits uses a dispatch table to accelerate           |
  630B* |       counting the number of bits in an accumulator which are       |
  631B* |       set.                                                          |
  632B* |                                                                     |
  633B* +-------------------------------------------------------------------*/
  634B
  635B
  636B          %
  637B          %
  638B          %
  639B          %       WCOB - Wide Count Bits
  640B          %
  641B          %       Used by COB
  642B          %
  643B          %       3 cycles minimum (all zeroes)
  644B          %      18 cycles maximum
  645B          %
  646B          % Times:
  647B          %       2 cycles setup
  648B          %     + 2 cycles if nybble = 00   Repeat until remaining
  649B          %    or 1 cycle if nybble <> 00   result = 0 (by shifting)
  650B
  651B          % AG: CPD<- SRC, Load DSP REG, ALU: PDR <- CPD
  652B
--003A--WCOB:   PDR = CASE_DATA == AG(SRC);
         OP     TSEL   ADDRESS  D AA   AB   AG AOP AL ST CM CPM CPD RM R0    R1  R2   IA   IB   ID RS IOP IY    IL FR FS FOP W FCW FL FRG X
         CJMP   FALSE           SRC                             AGA GN       LD

--DFVs:
  654B
  655B          % AG: CPM <- DES <- DES + CONST, Load DSP REG
  656B          % ALU: CPD <- PDR <- RSHIFT(PDR), DES <- CPM
  657B          % F bus <- PDR AND M1 for FZR test
  658B          % Continue dispatching.
  659B
--003B--WCOB1:  AY = AG(DES) = ALU(DES) == CNST(0) + A(DES),
  661B          IY = CASE_DATA == HEX_SHIFT_RIGHT(R1, PDR),
  662B          ALU_TEST == A(M1) AND PDR,
  663B          CASE_4_INTO WCOBTAB;
         OP     TSEL   ADDRESS  D AA   AB   AG AOP AL ST CM CPM CPD RM R0    R1  R2   IA   IB   ID RS IOP IY    IL FR FS FOP W FCW FL FRG X
         DSPA          WCOBTAB  F DES  DES  C  ADD Y        AG  IY  GN       LD       M1   DES  PD DA AND HR0   M  00
                                                                                                     R1
--DFVs:  addr is WCOBTAB (0000)
  664B

SAMPLES         Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
BIT             Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM     00.10.00                0021 - 01
```

```
Proprietary information of Data General Corporation

  665B  .EJECT;
  666B  .FT 1 "STACK           Source File                   Cycle 1        18-AUG-82 15:24:34 RGG"
  667B  ;
  668B
  669B  /*-----------------------------------------------------------------------+
  670B* |                                                                        |
  671B* |       These two examples are corresponding narrow and wide stack       |
  672B* |       operations.  The narrow stack operation must first read          |
  673B* |       the stack parameters from memory.  The wide stack                |
  674B* |       parameters for the current ring are in dedicated ALU and AG      |
  675B* |       register file locations.                                         |
  676B* |                                                                        |
  677B* |       In order to read its stack parameters, the narrow stack          |
  678B* |       operation starts memory using an address generated from          |
  679B* |       the constant field.  Because the Logical Address bus is          |
  680B* |       forced to narrow conditions during the narrow stack              |
  681B* |       instruction, bits 1-3 of the address are forced to the           |
  682B* |       value of CRE.                                                    |
  683B* |                                                                        |
  684B* |       Both samples illustrate the use of the SRC and DES pointers      |
  685B* |       to address a range of accumulators in a loop.                    |
  686B* |                                                                        |
  687B* +-----------------------------------------------------------------------*/
  688B
  689B
  690B  /*----------------------------------------------------------------------+
  691B* |                                                                       |
  692B* |       Push Multiple Accumulators                                      |
  693B* |                                                                       |
  694B* |       PSH     acs,acd                                                 |
  695B* |                                                                       |
  696B* +----------------------------------------------------------------------*/
  697B
--003C--PSH:    START AY == PASS (CNST(NSP)) FOR READ_WORD;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE                    C  PSB    RW                                                       NSP

--DFVs:   const is NSP (0020)
  699B
  700B          %NSP-->AR0
  701B
--003D--        CPM = AG(AR0) == MEM_READ,
  703B                  START AY == PASS (CNST(NSL)) FOR READ_WORD;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE                AR0 C  PSB M  RW R  MM                                                 NSL

--DFVs:   const is NSL (0022)
  704B
  705B          %NSL-->GR1
  706B
--003E--        CPM = ALU(GR1) == MEM_READ,
  708B                  START AY = AG(AR0) == B(AR0) + A(ONE) FOR WRITE_WORD,
  709B                  DECREMENT_DES_POINTER, IF SRC<>DES CALL PSHL;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJSR  NCOMP PSHL       ONE AR0 B  ADD Y  WW R  MM      GN DECD             GR1                M

--DFVs:   addr is PSHL (0041)
  710B
SAMPLES         Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
STACK           Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM    00.10.00                   0022 - 01
```

```
Proprietary information of Data General Corporation

  711B          %NSL-TOP OF STACK-->TEST
  712B
--003F--PSHM:   CPM = MEM_WRITE == ALU(SRC),  PDR == AG(AR0),
  714B                  ALU_TEST == B(GR1) - CPD,
  715B                  START AY == PASS (CNST(NSP)) FOR WRITE_WORD;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE            AR0     C  PSB     WW W  IA  AGA                 SRC GR1 BR CD CSR         NSP

--DFVs:   const is NSP (0020)
  716B
--0040--NOTST:  IY = MEM_WRITE == PDR OR CPD_ZERO,  ATTEMPT_NEXT_EFA,
  718B                  IF IS>=IR RETURN_ELSE_GOTO NSTK_OVERFLOW;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  CRY   NSTK_OVE           D  EFA     S@ W  IY  N                           PD CD OR  PASS

--DFVs:   addr is NSTK_OVERFLOW (0000 *EXT*)
  719B
  720B          %DONE?  AR0+1-->AR0,LAR,WW   SRC-->CPM   [ACS]+1-->[ACS]
  721B
--0041--PSHL:   CPM = MEM_WRITE == ALU(SRC),
  723B                  START AY = AG(AR0) == B(AR0) + A(ONE) FOR WRITE_WORD,
  724B                  INCREMENT_SRC_POINTER,  IF SRC=DES RETURN_ELSE_GOTO PSHL;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  COMP  PSHL       ONE AR0 B  ADD Y  WW W  IA      GN INCS         SRC

--DFVs:   addr is PSHL (0041)
  725B
  726B
  727B  %*****
  728B  %       Wide Push Accumulators:  WPSH
  729B  %
  730B  %  Start a write for the push.
  731B  %  Write the Accumlator to the new TOS, and start a write for the next one.
  732B  %       Generate a test for overflow:  SL - SP.  Increment the SRC pointer
  733B  %       and compare it to the DES pointer for termination of the loop.
  734B  %  Update the SP from PDR and abort the pending write. IPOP if no overflow
  735B  %       occurred, else service the overflow.
  736B  %*****
--0042--WPSH:   START AY = AG(SP) == CNST(2) + A(SP) FOR WRITE_DOUBLE;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE            SP  SP  C  ADD Y  WD                                                    02

--DFVs:
  738B
--0043--WPSH_LOOP:
  740B          CPM = MEM_WRITE == ALU(SRC),  PDR == AG(SP),
  741B                  ALU_TEST == B(SL) - AG(SP),
  742B                  START AY = AG(SP) == CNST(2) + A(SP) FOR WRITE_DOUBLE,
  743B                  INCREMENT_SRC_POINTER, IF SRC<>DES GOTO WPSH_LOOP;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  NCOMP WPSH_LOO   SP  SP  C  ADD Y  WD W  IA  AGA GN INCS         SRC SL  BR CD CSR         02

--DFVs:   addr is WPSH_LOOP (0043)
  744B
--0044--WPSHT:  IY = AG(SP) == PDR AND A(M1),   ABORT_MEMORY,  ATTEMPT_NEXT_EFA,


SAMPLES         Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
STACK           Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM    00.10.00                   0023 - 01
```

```
Proprietary information of Data General Corporation

  746B                 IF IS>=IR RETURN_ELSE_GOTO WSTK_OVERFLOW;
         OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         CRTN  CRY   WSTK_OVE       SP  D  EFA M  S@ A  IY                       M1      PD AD AND PASS

--DFVs:  addr is WSTK_OVERFLOW (0001 *EXT*)

   •
   •
   •

 SAMPLES         Instruction Set Microcode      Rev  1        09-DEC-82 10:47:39 RGG
 STACK           Source File                    Cycle 1       18-AUG-82 15:24:34 RGG
 UASM     00.10.00                 0024 - 01
```

```
Proprietary information of Data General Corporation

  747B  .EJECT;
  748B  .FT 1 "PRIV            Source File                  Cycle 1        18-AUG-82 15:24:34 RGG"
  749B  ;
  750B
  751B  /*---------------------------------------------------------------------+
  752B* |                                                                      |
  753B* |       Privileged instructions are those which can be executed        |
  754B* |       only while the PC is in ring 0.  Microcode for these           |
  755B* |       instructions must confirm this by testing for CRE = 0.         |
  756B* |                                                                      |
  757B* |       This code restarts the IP at the next instruction to be        |
  758B* |       executed.  Specifying WIDE_JUMP will result in a double        |
  759B* |       word being fetched for the IP start, while specifying          |
  760B* |       NARROW_JUMP (or simply JUMP) will cause a single word to       |
  761B* |       fetched.  This is not to be confused with the width of the     |
  762B* |       address bus.  Rather, these will result in a wide or narrow    |
  763B* |       defer chain being resolved should indirection be specified.    |
  764B* |                                                                      |
  765B* +---------------------------------------------------------------------*/
  766B
  767B
  768B  /*------------------------------------------------------------------+
  769B* |                                                                   |
  770B* |                      PURGE THE ATU:    PATU                       |
  771B* |                                                                   |
  772B* |                                                                   |
  773B* |                                                                   |
  774B* +------------------------------------------------------------------*/
  775B
--0045--PATU:
  777B
  778B                % Take a Privilege Protection Fault if not in Ring 0.
  779B            IF RING<>0 GOTO PRIVILEGE_PROTECTION;
        OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CJMP  NRNG0 PRIVILEG

--DFVs:  addr is PRIVILEGE_PROTECTION (0003 *EXT*)
  780B
  781B                % Make sure a clean set of Validity bits are availble...
  782B                %also set CRE bits to zero for LSBRA
  783B
--0046--PATUW:  IF ATU_PURGING  GOTO PATUW,  PDR == PC,
  785B                  AY = CRE == B(AR3) - A(AR3);
        OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CJMP  PRGB  PATUW      AR3 AR3 B  SUB           PC  AT LCRE

--DFVs:  addr is PATUW (0046)
  786B
  787B                % ... then swap in the clean Validity bits.
--0047--        PURGE_THE_ATU_CACHE,  IY = AG(AR0) == A(M1) AND PDR;
        OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CJMP  FALSE              AR0        M        IY       AT PRGA         M1      PD DA AND PASS

--DFVs:
  789B
  790B                %   Restart the IP since Logical memory has been changed.


SAMPLES         Instruction Set Microcode      Rev  1        09-DEC-82 10:47:39 RGG
PRIV            Source File                    Cycle 1       18-AUG-82 15:24:34 RGG
UASM     00.10.00                 0025 - 01
```

```
Proprietary information of Data General Corporation

--0048--          START AY = PC == PASS (B(AR0)) FOR WIDE_JUMP,  GOTO JMP;
          OP      TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1   R2   IA   IB   ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          LEAP          JMP              AR0 B  PSB    RD              AT IPST

--DFVs:   addr is JMP (0012)

   •
   •
   •

SAMPLES          Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
PRIV             Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM     00.10.00                     0026 - 01
```

```
Proprietary information of Data General Corporation

  792B  .EJECT;
  793B  .FT 1 "FLPT             Source File                   Cycle 1        18-AUG-82 15:24:34 RGG"
  794B  ;
  795B
  796B  /*-------------------------------------------------------------------+
  797B* |                                                                    |
  798B* |       These selections from the floating point code illustrate     |
  799B* |       loads, stores, and addition.  Note that only one cycle       |
  800B* |       is required for single precision loads.  When the upper      |
  801B* |       32 bits of the floating point register file are loaded       |
  802B* |       with data from the CPM bus, the bottom 32 bits are loaded    |
  803B* |       with zeroes.                                                 |
  804B* |                                                                    |
  805B* |       Accumulator to accumulator code is also used for memory to   |
  806B* |       accumulator instructions by first loading the operand from   |
  807B* |       memory into a general register, loading the source pointer   |
  808B* |       with the general register number and entering the            |
  809B* |       accumulator to accumulator code.                             |
  810B* |                                                                    |
  811B* |                                                                    |
  812B* +-------------------------------------------------------------------*/
  813B
  814B

   •
   •
   •

SAMPLES          Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
FLPT             Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM     00.10.00                     0027 - 01
```

```
Proprietary information of Data General Corporation
  815B  /*----------------------------------------------------------------------+
  816B* |                                                                       |
  817B* |       Load Floating Point Single (Long Displacement)                  |
  818B* |                                                                       |
  819B* |       LFLDS   fpac,[@]displacement[,index]                            |
  820B* |       XFLDS   fpac,[@]displacement[,index]                            |
  821B* |       FLDS    fpac,[@]displacement[,index]                            |
  822B* |                                                                       |
  823B* |       Requires  1 cycle.                                              |
  824B* |                                                                       |
  825B* +----------------------------------------------------------------------*/
  826B
  827B  %       Read a double word from memory and place it in the high
  828B  %       order half of the destination FPAC.  Update the FPSR.
  829B  %       Return.
  830B
--0049--LFLDS:
--0049--XFLDS:
--0049--FLDS:   CPM = FP_HIGH(DES) == MEM_READ,
  834B          EXPONENT == FA, FA == A(DES), UPDATE_FPSR,
  835B          ATTEMPT_NEXT_EFA, RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                     D  EFA    S@ R  MM      FL
                                                                         LAX     DES                                   DES MH UFS
--DFVs:
  836B
  837B
  838B  /*----------------------------------------------------------------------+
  839B* |                                                                       |
  840B* |       Load Floating Point Double                                      |
  841B* |                                                                       |
  842B* |       LFLDD   fpac,[@]displacement[,index]                            |
  843B* |       XFLDD   fpac,[@]displacement[,index]                            |
  844B* |       FLDD    fpac,[@]displacement[,index]                            |
  845B* |                                                                       |
  846B* |       Cycles:   2                                                     |
  847B* |                                                                       |
  848B* +----------------------------------------------------------------------*/
  849B
  850B  %       Read a double word from memory and place it in the high
  851B  %       order half of the destination FPAC.  Update the FPSR.
  852B  %       Start memory for a read of the next double word.
  853B
--004A--LFLDD:
--004A--XFLDD:
--004A--FLDD:   CPM = FP_HIGH(DES) == MEM_READ,
  857B          EXPONENT == FA, FA == A(DES), UPDATE_FPSR,
  858B          START AY == LAST_LA + A(TWO) FOR READ_DOUBLE;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE            TWO     L  ADD    RD R  MM      FL
                                                                         LAX     DES                                   DES MH UFS
--DFVs:
  859B
  860B  %       Read a double word from memory and place it in the low order half
  861B  %       of the destination FPAC.
  862B
--004B--        CPM = FP_LOW(DES) == MEM_READ, FD == A(DES) + ZERO,

SAMPLES         Instruction Set Microcode     Rev  1        09-DEC-82 10:47:39 RGG
FLPT            Source File                   Cycle 1       18-AUG-82 15:24:34 RGG
UASM    00.10.00                   0028 - 01
```

```
Proprietary information of Data General Corporation
  864B          ATTEMPT_NEXT_EFA, RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                     D  EFA    S@ R  MM
                                                                                 DES                       FA ZR TAD   DES ML
--DFVs:
  865B
  866B

  •
  •
  •

SAMPLES         Instruction Set Microcode     Rev  1        09-DEC-82 10:47:39 RGG
FLPT            Source File                   Cycle 1       18-AUG-82 15:24:34 RGG
UASM    00.10.00                   0029 - 01
```

```
Proprietary information of Data General Corporation

  867B  /*----------------------------------------------------------------------+
  868B* |                                                                       |
  869B* |       Store Floating Point Single                                     |
  870B* |                                                                       |
  871B* |       LFSTS   fpac,[@]displacement[,index]                            |
  872B* |       XFSTS   fpac,[@]displacement[,index]                            |
  873B* |       FSTS    fpac,[@]displacement[,index]                            |
  874B* |                                                                       |
  875B* |       Cycles:   1                                                     |
  876B* |                                                                       |
  877B* +---------------------------------------------------------------------*/
  878B
  879B
  880B  %       Get a double word from the high order half of the
  881B  %       destination FPAC and write it to memory.
  882B
--004C--LFSTS:
--004C--XFSTS:
--004C--FSTS:   CPM = MEM_WRITE == FP_HIGH(DES), ATTEMPT_NEXT_EFA, RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                     D  EFA    S@ W  HF
                                                                             DES
--DFVs:
  886B
  887B
  888B  /*----------------------------------------------------------------------+
  889B* |                                                                       |
  890B* |       Store Floating Point Double                                     |
  891B* |                                                                       |
  892B* |       LFSTD    fpac,[@]displacement[,index]                           |
  893B* |       XFSTD    fpac,[@]displacement[,index]                           |
  894B* |       FSTD     fpac,[@]displacement[,index]                           |
  895B* |                                                                       |
  896B* |       Cycles:   2                                                     |
  897B* |                                                                       |
  898B* +---------------------------------------------------------------------*/
  899B
  900B
  901B  %       Read a double word from the high order half of the destination
  902B  %       FPAC and write it to memory.  Start memory for a write of the
  903B  %       next double word.
  904B
--004D--LFSTD:
--004D--XFSTD:
--004D--FSTD:   CPM = MEM_WRITE == FP_HIGH(DES),
  908B          START AY == LAST_LA + A(TWO) FOR WRITE_DOUBLE;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE            TWO      L  ADD    WD W  HF
                                                                             DES
--DFVs:
  909B
  910B  %       Write the low order half of the destination FPAC to memory.
  911B
--004E--        CPM = MEM_WRITE == FP_LOW(DES), ATTEMPT_NEXT_EFA, RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                     D  EFA    S@ W  LF
                                                                             DES
--DFVs:
SAMPLES         Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
FLPT            Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM     00.10.00                 0030 - 01
```

```
Proprietary information of Data General Corporation

  913B
  914B

  •
  •
  •

SAMPLES         Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
FLPT            Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM     00.10.00                 0031 - 01
```

```
Proprietary information of Data General Corporation

  915B  /*---------------------------------------------------------------------+
  916B* |                                                                      |
  917B* |       Add Single (Memory to FPAC)                                    |
  918B* |                                                                      |
  919B* |       LFAMS    fpac,[@]displacement[,index]                          |
  920B* |       XFAMS    fpac,[@]displacement[,index]                          |
  921B* |       FAMS     fpac,[@]displacement[,index]                          |
  922B* |                                                                      |
  923B* |       Cycles:    4                                                   |
  924B* |                                                                      |
  925B* +--------------------------------------------------------------------*/
  926B
  927B
  928B
  929B  %       Read a double word from memory and place it in the high order
  930B  %       half of a temporary register.  Place the number of the temporary
  931B  %       register in the source register pointer.  Go to code which
  932B  %       executes floating point add.
  933B
--004F--LFAMS:
--004F--XFAMS:
--004F--FAMS:   CPM = FP_HIGH(FG0) == MEM_READ, POINT_SRC_TO FG0, GOTO FAS;
            OP     TSEL   ADDRESS   D AA   AB   AG AOP AL ST CM CPM CPD RM R0    R1   R2   IA   IB   ID RS IOP IY    IL FR FS FOP W FCW FL FRG X
            LEAP          FAS                                R  MM      GN LDAS                      CN
                                                                                                                              FG0   FG0 MH
--DFVs:     addr is FAS (0052)
  937B
  938B
  939B  /*---------------------------------------------------------------------+
  940B* |                                                                      |
  941B* |       Add Double (Memory to FPAC) (Long Displacement)                |
  942B* |                                                                      |
  943B* |       LFAMD    fpac,[@]displacement[,index]                          |
  944B* |       XFAMD    fpac,[@]displacement[,index]                          |
  945B* |       FAMD     fpac,[@]displacement[,index]                          |
  946B* |                                                                      |
  947B* |       Cycles:    5                                                   |
  948B* |                                                                      |
  949B* +--------------------------------------------------------------------*/
  950B
  951B  %       Read a double word from memory and place it in the high order
  952B  %       half of a temporary register.  Start a memory read for the next
  953B  %       double word.  Place the number of the temporary register in the
  954B  %       source register pointer.
  955B
--0050--LFAMD:
--0050--XFAMD:
--0050--FAMD:   CPM = FP_HIGH(FG0) == MEM_READ,
  959B          START AY == LAST_LA + A(TWO) FOR READ_DOUBLE,
  960B          POINT_SRC_TO FG0;
            OP     TSEL   ADDRESS   D AA   AB   AG AOP AL ST CM CPM CPD RM R0    R1   R2   IA   IB   ID RS IOP IY    IL FR FS FOP W FCW FL FRG X
            CJMP   FALSE              TWO       L  ADD    RD R  MM      GN LDAS                      CN
                                                                                                                              FG0   FG0 MH
--DFVs:
  961B
  962B  %       Read a double word from memory and place it in the low order half
  963B  %       of the temporary register.  Go to code which executes floating
  964B  %       point add.
SAMPLES         Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
FLPT            Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM     00.10.00                 0032 - 01
```

```
Proprietary information of Data General Corporation

  965B
--0051--        CPM = FP_LOW(FG0) == MEM_READ, GOTO FAD;
         OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         LEAP         FAD                           R  MM
                                                                                                              FG0 ML
--DFVs:  addr is FAD (0052)
  967B
  968B
  969B  /*------------------------------------------------------------------+
  970B* |                                                                   |
  971B* |       Add Single (FPAC to FPAC)                                   |
  972B* |                                                                   |
  973B* |       FAS     facs,facd                                           |
  974B* |                                                                   |
  975B* |       Cycles:   3                                                 |
  976B* |                                                                   |
  977B* +------------------------------------------------------------------*/
  978B
  979B  /*------------------------------------------------------------------+
  980B* |                                                                   |
  981B* |       Add Double (FPAC to FPAC)                                   |
  982B* |                                                                   |
  983B* |       FAD     facs,facd                                           |
  984B* |                                                                   |
  985B* |       Cycles:   3                                                 |
  986B* |                                                                   |
  987B* +------------------------------------------------------------------*/
  988B
  989B  %       Compare source and destination mantissas and load signs.  Compare
  990B  %       exponents and load the shift magnitude register with the
  991B  %       difference.  If the exponents are equal, the working register is
  992B  %       loaded with the smaller mantissa.  If the exponents are not
  993B  %       equal, the working register is loaded with the mantissa
  994B  %       corresponding to the smaller exponent.  The swap bit is set if
  995B  %       the source mantissa is loaded into the working register.
  996B
--0052--FAD:
--0052--FAS:    FD == A(SRC) - B(DES), LOAD_SIGNS,
  999B          EXPONENT == FA-FB, WR == PRESCALE_OPERAND;
         OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         CJMP   FALSE                                           FL
                                                                   LAB   SUB CMP SRC DES                 FA FB TSB R          LWR
--DFVs:
 1000B
 1001B  %       Add with signs the right shifted mantissa in the working register
 1002B  %       to the mantissa selected by the swap bit.  Place the result in
 1003B  %       the working register.  Add mantissa overflow bit to the exponent
 1004B  %       selected by the swap bit.  Form the guard digits and load the
 1005B  %       signs as selected by the swap bit.  Detect leading zeroes in the
 1006B  %       result and load the shift magnitude register.
 1007B
--0053--        FD = WR == SELECTED_A(SRC,DES) @+ PRESCALED_WR TRUNCATED_IF_NOT_ROUNDING,
 1009B          EXPONENT == FA+MOF, LOAD_SIGNS, SHIFT_MAG == LEADING_ZERO_DETECT;
         OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0    R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         CJMP   FALSE                                           FL
                                                                   TRN   ACA LZD SRC DES                 FA RS ADD D          LWR
--DFVs:
 1010B
SAMPLES         Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
FLPT            Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM     00.10.00                0033 - 01
```

```
Proprietary information of Data General Corporation

 1011B  %       Add left shifted result in working register and round bit and
 1012B  %       allow correction for mantissa overflow.  Select sign in A
 1013B  %       register.  Adjust the exponent for normalization and
 1014B  %       mantissa overflow.  Store result in destination FPAC.  Update the
 1015B  %       FPSR.
 1016B
--0054--FRND:   FD = FPU(DES) == ROUND_BIT + NORMALIZED_WR,
 1018B          ENABLE_MOF_CORRECTION, ALLOW_SHIFT_MAG_CORRECTION,
 1019B          SIGN == A_SIGN,
 1020B          EXPONENT == EWR+MAG+MOF, UPDATE_FPSR, ATTEMPT_NEXT_EFA, RETURN;
         OP    TSEL   ADDRESS   D AA   AB   AG AOP AL  ST CM CPM CPD RM R0    R1   R2   IA   IB   ID RS IOP IY    IL FR FS FOP W FCW FL FRG X
         CRTN  TRUE                         D  EFA     S@            FL
                                                                        SA    ACN  LZD                                   RB LS TAD   DES D  UFS
--DFVs:
 1021B
 1022B
 1023B
 1024B
 1025B


SAMPLES         Instruction Set Microcode      Rev   1       09-DEC-82 10:47:39 RGG
FLPT            Source File                    Cycle 1       18-AUG-82 15:24:34 RGG
UASM     00.10.00                  0034 - 01
```

```
Proprietary information of Data General Corporation

 1026B  .EJECT;
 1027B  .FT 1 "WLMP             Source File                   Cycle 1        18-AUG-82 15:24:34 RGG"
 1028B  ;
 1029B
 1030B  /*-------------------------------------------------------------------+
 1031B* |                                                                   |
 1032B* |       The code for the load map instruction illustrates the      |
 1033B* |       communication protocal with the IO controller.  Note       |
 1034B* |       all IO commands are piped through TREG.  This is the       |
 1035B* |       only CPD source which will make timing to the IOC.         |
 1036B* |                                                                   |
 1037B* |       Although WMLP is a wide instruction, decode information    |
 1038B* |       starts the instruction as though it were narrow.  This is  |
 1039B* |       to handle a peculiarity of the instruction in a convenient |
 1040B* |       fashion (i.e. incrementing and sign extending the map slot |
 1041B* |       counter in the first cycle).  The ALU is placed in the wide|
 1042B* |       mode at the end of the first cycle.                        |
 1043B* |                                                                   |
 1044B* +-------------------------------------------------------------------*/
 1045B
 1046B
 1047B  /*-------------------------------------------------------------------+
 1048B* |                                                                   |
 1049B* |       Wide Load Map                                               |
 1050B* |                                                                   |
 1051B* |       WLMP                                                        |
 1052B* |                                                                   |
 1053B* |       FLAG0 must be cleared by decode.                            |
 1054B* |                                                                   |
 1055B* +-------------------------------------------------------------------*/
 1056B
--0055--WLMP:
 1058B
 1059B  %       Pre-increment and sign extend map slot counter to enter loop.
 1060B  %       Put ALU data paths in wide mode.  Take privilege protection
 1061B  %       fault if not in ring 0.
 1062B
 1063B          IY = TREG == ZERO +1+ A(AC1),
 1064B                  MODIFY_FLAGS_0123 (SET,N,N,N),
 1065B                  IF RING<>0 GOTO PRIVILEGE_PROTECTION;
         OP    TSEL   ADDRESS   D AA   AB   AG AOP AL  ST CM CPM CPD RM R0    R1   R2   IA   IB   ID RS IOP IY    IL FR FS FOP W FCW FL FRG X
         CJMP  NRNGO  PRIVILEG                                 IY      GN MFSO  LT        AC1       ZR AD CAD PASS
                                                                                                                     S  N  N   N
--DFVs:  addr is PRIVILEGE_PROTECTION (0003 *EXT*)
 1066B
 1067B  %       Decrement map slot counter. Check for interrupt BEFORE loading first
 1068B  %       Map slot.
 1069B
--0056--        IY = ALU(AC1) = AG(AG1) == A(M1) + TREG,
 1071B                  IF INTERRUPT_PENDING GOTO WLMP_INT;
         OP    TSEL   ADDRESS   D AA   AB   AG AOP AL  ST CM CPM CPD RM R0    R1   R2   IA   IB   ID RS IOP IY    IL FR FS FOP W FCW FL FRG X
         CJMP  INTR   WLMP_INT         AG1         M         IY  TRG                    M1   AC1     CA ADD PASS  Y

--DFVs:  addr is WLMP_INT (0063)
 1072B
 1073B
--0057--WLMP_LOOP:
 1075B
SAMPLES         Instruction Set Microcode      Rev   1       09-DEC-82 10:47:39 RGG
WLMP            Source File                    Cycle 1       18-AUG-82 15:24:34 RGG
UASM     00.10.00                  0035 - 01
```

```
Proprietary information of Data General Corporation

 1076B  %       Isolate the map slot number from AC0 and convert to an
 1077B  %       IOC register number by shifting left.  Start read for
 1078B  %       the first double word to load in the map.  If map slot
 1079B  %       counter is zero, there is nothing left to do.
 1080B
 1081B          IY = ALU(GR0) == BIT_SHIFT_LEFT (SPAD(WASHM17) NOT_AND A(AC0)),
 1082B          START AY == PASS (B(AG2)) FOR READ_DOUBLE,
 1083B          IF ALU=0 GOTO WLMP_DONE;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FZR   WLMP_DON       AG2 B  PSB     RD                            AC0 GR0 SC AD ANC BL0  Y  WASHM17

--DFVs:   addr is WLMP_DONE (0064)  const is WASHM17 (0031)
 1084B
 1085B  %       There are still more map slots to load.  Read the map data.
 1086B
--0058--        CPM = ALU(GR2) == MEM_READ;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE                                 R  MM                          GR2                 M

--DFVs:
 1088B
 1089B  %       Isolate high half of map data.
 1090B
--0059--        IY = ALU(GR1) == HEX_SHIFT_RIGHT (R4,A(GR2));
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE                                                            GR2 GR1    AD     HR0  Y
                                                                                               R4

--DFVs:
 1092B
 1093B  %       Form command to load high half of current map slot.
 1094B
--005A--        IY = TREG = ALU(GR0) == SPAD(IOCMD1) OR A(GR0);
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE                                    IY      GN      LT      GR0 GR0 SC AD OR  PASS Y  IOCMD1

--DFVs:   const is IOCMD1 (0079)
 1096B
 1097B  %       Send load command.
 1098B  %       Form data for high half of current map slot.
 1099B
--005B--        PDR = IO_CONTROLLER == TREG,
 1101B          IY = TREG == SPAD(IOCMD) OR A(GR1);
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE                                    IY  TRG AT SIO  LT      GR1     SC AD OR  PASS     IOCMD

--DFVs:   const is IOCMD (0078)
 1102B
 1103B  %       Send data for high half of map slot.
 1104B  %       Increment address part of command to write low half of map slot.
 1105B
--005C--        PDR = IO_CONTROLLER == TREG,
 1107B          IY = ALU(GR0) == ZERO +1+ A(GR0);
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE                                        TRG AT SIO          GR0 GR0 ZR AD CAD PASS Y

--DFVs:
 1108B
 1109B  %       Move command to load low half of current map slot to TREG.
SAMPLES         Instruction Set Microcode     Rev   1       09-DEC-82 10:47:39 RGG
WLMP            Source File                   Cycle 1       18-AUG-82 15:24:34 RGG
UASM    00.10.00                 0036 - 01
```

```
Proprietary information of Data General Corporation

 1110B  %        Wait until IO is finished.
 1111B
--005D--WLMP_HI_WAIT: CPM = TREG == ALU(GR0),
 1113B           IF IO_BUSY GOTO WLMP_HI_WAIT;
        OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CJMP   IOB   WLMP_HI_                           IA      GN      LT      GR0

--DFVs:   addr is WLMP_HI_WAIT (005D)
 1114B
 1115B  %        IO is done.  Increment map data pointer (by 2 since they are double
 1116B  %        words).  Send clear command.
 1117B
--005E--         AY = ALU(AC2) = AG(AG2) == B(AG2) + A(TWO),
 1119B               PDR = IO_CONTROLLER == ZERO;
        OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CJMP   FALSE            TWO AG2 B  ADD Y        AG  ZER AT SIO              AC2                M

--DFVs:
 1120B
 1121B  %        Form data for low half of map slot.
 1122B  %        Send command to load low half of current map slot.
 1123B
--005F--         IY = TREG == SPAD(IOCMD) OR A(GR2),
 1125B           PDR = IO_CONTROLLER == TREG;
        OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CJMP   FALSE                                    IY  TRG AT SIO  LT      GR2     SC AD OR  PASS    IOCMD

--DFVs:   const is IOCMD (0078)
 1126B
 1127B  %        Send data for low half of current map slot.  Increment
 1128B  %        map slot number.
 1129B
--0060--         PDR = IO_CONTROLLER == TREG,
 1131B           IY = ALU(AC0) = AG(AG0) == ZERO +1+ A(AC0);
        OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CJMP   FALSE                AG0        M        IY  TRG AT SIO          AC0 AC0 ZR AD CAD PASS Y

--DFVs:
 1132B
 1133B  %        Wait here until IO is finished.
 1134B
--0061--WLMP_LO_WAIT: IF IO_BUSY GOTO WLMP_LO_WAIT;
        OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CJMP   IOB   WLMP_LO_

--DFVs:   addr is WLMP_LO_WAIT (0061)
 1136B
 1137B  %        IO is done.  Send clear command and decrement slot counter.
 1138B  %        Loop if an interrupt is not pending.
 1139B
--0062--         CPD = IO_CONTROLLER == ZERO,
 1141B           IY = ALU(AC1) = AG(AG1) == A(M1) + B(AC1),
 1142B           IF NOT INTERRUPT_PENDING GOTO WLMP_LOOP;
        OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CJMP   NINTR WLMP_LOO       AG1        M        IY  N   AT SIO          M1  AC1 BR DA ADD PASS Y

--DFVs:   addr is WLMP_LOOP (0057)
 1143B
SAMPLES          Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
WLMP             Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM     00.10.00                 0037 - 01
```

```
Proprietary information of Data General Corporation


--0063--WLMP_INT:

      %       Interrupt is pending. Correct the map slot counter, load PCX into
      %       PDR, and honor the interrupt.

              IY = ALU(AC1) = AG(AG1) == B(AC1) - A(M1),
              PDR == PC_OF_EXECUTION, GOTO RESTARTABLE_INTERRUPT;
        OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        LEAP        RESTARTA       AG1          M        IY  PCX                   M1  AC1 BR AD CSR PASS Y

--DFVs:  addr is RESTARTABLE_INTERRUPT (0004 *EXT*)


--0064--WLMP_DONE:

      %       Normal termination of WLMP instruction.  Abort the pending
      %       read and IPOP.

              ABORT_MEMORY, ATTEMPT_NEXT_EFA, RETURN;
        OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CRTN  TRUE                     D  EFA    S@ A

--DFVs:







SAMPLES        Instruction Set Microcode     Rev   1         09-DEC-82 10:47:39 RGG
WLMP           Source File                   Cycle 1         18-AUG-82 15:24:34 RGG
UASM     00.10.00               0038 - 01
```

```
Proprietary information of Data General Corporation

        .EJECT;
        .FT 1 "IP_ALT          Source File                  Cycle 1         18-AUG-82 15:24:34 RGG"
        ;

        .LOC     0FFF;    %  WAIT
                 %================================================================
                 %       WAIT for the IP
                 %
                 %  Abort any outstanding memory starts and try to IPOP to
                 %       the next instruction.
                 %================================================================
--0FFF--WAIT:    ABORT_MEMORY,  ATTEMPT_NEXT_EFA,  RETURN;
        OP       TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CRTN     TRUE                     D  EFA    S@ A

--DFVs:


        .END;



SAMPLES        Instruction Set Microcode     Rev   1         09-DEC-82 10:47:39 RGG
IP_ALT         Source File                   Cycle 1         18-AUG-82 15:24:34 RGG
UASM     00.10.00               0039 - 01
```

```
SYMBOL                         QUALIFIER                      TYPE     VALUE
------                         ---------                      ----     -----
&                              .GLOBAL.                       PARSE     0003
+                              AOP1                           CONST     0001
                               ALU_OP                         CONST     0007
                               FP_OP                          CONST     0002
+1+                            ALU_OP                         CONST     0005
-                              AOP1                           CONST     0000
                               ALU_OP                         CONST     0004
                               FP_OP                          CONST     0003
-1-                            ALU_OP                         CONST     0006
.BEGIN                         .COMMAND.                      COMMAND   0000
.EJECT                         .COMMAND.                      COMMAND   0000
.END                           .COMMAND.                      COMMAND   0000
.EXTERNAL                      .COMMAND.                      COMMAND   0000
.FT                            .COMMAND.                      COMMAND   0000
.HD                            .COMMAND.                      COMMAND   0000
.LOC                           .COMMAND.                      COMMAND   0000
.RADIX                         .COMMAND.                      COMMAND   0000
.TITLE                         .COMMAND.                      COMMAND   0000
=                              AG_DEST                        PARSE     003A
                               CPM_DEST                       PARSE     0050
                               CPD_DEST                       PARSE     0067
                               ALU_DEST                       PARSE     00F8
                               FPU_DEST                       PARSE     0132
==                             AG_DEST                        CONST     0000
                               CPM_DEST                       CONST     0000
                               CPD_DEST                       CONST     0000
                               ALU_DEST                       CONST     0000
                               EQUAL                          CONST     0000
                               ID_DEST                        CONST     0000
                               FPU_DEST                       CONST     0000
@+                             FP_OP                          CONST     0000
A                              AGA                            PARSE     003F
                               IS                             PARSE     00F9
                               IR                             PARSE     0111
                               FR_SRC                         PARSE     0135
                               FA_SRC                         PARSE     0144
ABORT_MEMORY                   .GLOBAL.                       PARSE     0040
AC0                            IAB                            CONST     0000
AC1                            IAB                            CONST     0001
AC2                            IAB                            CONST     0002
AC3                            IAB                            CONST     0003
ADD                            .GLOBAL.                       LABEL     0015
ADDI                           .GLOBAL.                       LABEL     001A
AG                             AG_DEST                        PARSE     002E
                               CPM_DEST                       PARSE     0049
                               PDR_SRC                        PARSE     0069
                               ALU_DEST                       PARSE     00F1
                               IR                             PARSE     0119
AG0                            AAB                            CONST     0000
AG1                            AAB                            CONST     0001
AG2                            AAB                            CONST     0002
AG3                            AAB                            CONST     0003
ALC_CRY                        CRY_RAND                       CONST     0009
ALC_RESULT                     TSKIP                          PARSE     00CE
ALLOW_SHIFT_MAG_CORRECTION     .GLOBAL.                       PARSE     013F
ALU                            AG_DEST                        PARSE     002B
                               CPM_DEST                       PARSE     0046
                               CPM_SRC                        PARSE     0055
                               ALU_DEST                       PARSE     00EB
                            *** Symbol Table ***
UASM     00.10.00                0001 - 02
```

```
SYMBOL                          QUALIFIER                       TYPE      VALUE
------                          ---------                       ----      -----
ALU=0                           TEST                            CONST      007D
                                TSKIP                           CONST      0004
ALU_CRY                         CRY_RAND                        SET        0015
ALU_TEST                        .GLOBAL.                        PARSE      0122
AND                             ALU_OP                          CONST      0000
AR0                             AAB                             CONST      0008
AR1                             AAB                             CONST      0009
AR3                             AAB                             CONST      000B
AR5                             AAB                             CONST      000D
ATTEMPT_NEXT_EFA                .GLOBAL.                        PARSE      0026
ATU_PURGING                     TEST                            CONST      0019
AY                              .GLOBAL.                        PARSE      0028
                                AG_OP                           PARSE      0029
A_SIGN                          SGNR                            SET        0017
B                               AGB_SRC                         PARSE      003E
                                IS                              PARSE      00FA
                                IR                              PARSE      0112
                                ID_SRC                          PARSE      0129
                                FS_SRC                          PARSE      0138
BIT0                            .GLOBAL.                        LABEL      0000
BIT16                           .GLOBAL.                        LABEL      0010
BIT_SHIFT_LEFT                  IS                              PARSE      0105
CALL                            CONDITIONAL                     CONST      0001
CARRY                           .GLOBAL.                        PARSE      00E0
CASE_4_INTO                     .GLOBAL.                        PARSE      000E
CASE_DATA                       CPD_DEST                        PARSE      0062
                                ALU_DEST                        PARSE      00E9
CLEAR                           MF                              CONST      0002
CNST                            AGB_SRC                         PARSE      003C
                                IS                              PARSE      00FE
COMPLETE_JUMP                   .GLOBAL.                        PARSE      0042
CPD                             .GLOBAL.                        PARSE      005C
                                IR                              PARSE      011C
CPD_ZERO                        IR                              PARSE      011D
CPM                             .GLOBAL.                        PARSE      0045
CRE                             AG_DEST                         PARSE      0039
DECREMENT_DES_POINTER           .GLOBAL.                        PARSE      0071
DEFER_ON_FALSE_TEST             .GLOBAL.                        PARSE      00DB
DES                             AAB                             CONST      000F
                                IAB                             CONST      000F
                                FABC                            CONST      000F
EJMP                            .GLOBAL.                        LABEL      0012
EJSR                            .GLOBAL.                        LABEL      0013
ELDA                            .GLOBAL.                        LABEL      0010
ELDB                            .GLOBAL.                        LABEL      0024
ELEF                            .GLOBAL.                        LABEL      0018
ENABLE_MOF_CORRECTION           .GLOBAL.                        PARSE      0140
ESTA                            .GLOBAL.                        LABEL      0011
EWR+MAG+MOF                     EXPR                            CONST      0006
EXECUTE                         .GLOBAL.                        LABEL      0026
EXECUTE_DATA                    CPM_DEST                        PARSE      004F
EXECUTE_PBX                     .GLOBAL.                        LABEL      002D
EXPONENT                        .GLOBAL.                        PARSE      0154
EXTEND_MICRO_CYCLE              .GLOBAL.                        PARSE      00D1
FA                              .GLOBAL.                        PARSE      013A
                                EXPR                            CONST      0001
FA+MOF                          EXPR                            CONST      0005
FA-FB                           EXPR                            CONST      0003
FAD                             .GLOBAL.                        LABEL      0052
                            *** Symbol Table ***
UASM     00.10.00                 0002 - 02
```

```
SYMBOL                          QUALIFIER                       TYPE     VALUE
------                          ---------                       ----     -----
FAMD                            .GLOBAL.                        LABEL     0050
FAMS                            .GLOBAL.                        LABEL     004F
FAS                             .GLOBAL.                        LABEL     0052
FD                              .GLOBAL.                        PARSE     012E
FG0                             FABC                            CONST     0008
                                ALL_REGSS                       PARSE     0098
FLAG4=0                         TEST                            CONST     004C
FLAG4=1                         TEST                            CONST     000C
FLDD                            .GLOBAL.                        LABEL     004A
FLDS                            .GLOBAL.                        LABEL     0049
FOR                             MEM_STR                         PARSE     001D
FPU                             FPU_DEST                        PARSE     012F
FP_HIGH                         CPM_DEST                        PARSE     004C
                                CPM_SRC                         PARSE     0056
FP_LOW                          CPM_DEST                        PARSE     004B
                                CPM_SRC                         PARSE     0057
FRND                            .GLOBAL.                        LABEL     0054
FSTD                            .GLOBAL.                        LABEL     004D
FSTS                            .GLOBAL.                        LABEL     004C
GOTO                            CONDITIONAL                     SET       0012
                                .GLOBAL.                        PARSE     000B
GR0                             IAB                             CONST     0008
GR1                             IAB                             CONST     0009
GR2                             IAB                             CONST     000A
GR5                             IAB                             CONST     000D
HEX_SHIFT_RIGHT                 IS                              PARSE     0109
ID                              .GLOBAL.                        PARSE     0123
ID_SIGN=1                       TEST                            CONST     002D
IF                              .GLOBAL.                        PARSE     0001
INC                             .GLOBAL.                        LABEL     0016
INCREMENT_SRC_POINTER           .GLOBAL.                        PARSE     006C
INDIRECT                        TEST                            CONST     0050
INTERRUPT_PENDING               TEST                            CONST     0043
INWARD_REFERENCE                TEST                            CONST     0052
IN_CURRENT_RING                 MEM_STR                         PARSE     001C
IOCMD                           .GLOBAL.                        LABEL     0078
IOCMD1                          .GLOBAL.                        LABEL     0079
IO_BUSY                         TEST                            CONST     0044
IO_CONTROLLER                   CPD_DEST                        PARSE     005F
IS>=IR                          TEST                            CONST     003A
IY                              .GLOBAL.                        PARSE     00E7
JMP                             .GLOBAL.                        LABEL     0012
JSR                             .GLOBAL.                        LABEL     0013
LAR                             IR                              PARSE     011A
LAST_LA                         AGB_SRC                         PARSE     003D
LDA                             .GLOBAL.                        LABEL     0010
LDB                             .GLOBAL.                        LABEL     0023
LEADING_ZERO_DETECT             MAG_OP                          CONST     0007
LEF                             .GLOBAL.                        LABEL     0018
LFAMD                           .GLOBAL.                        LABEL     0050
LFAMS                           .GLOBAL.                        LABEL     004F
LFLDD                           .GLOBAL.                        LABEL     004A
LFLDS                           .GLOBAL.                        LABEL     0049
LFSTD                           .GLOBAL.                        LABEL     004D
LFSTS                           .GLOBAL.                        LABEL     004C
LJMP                            .GLOBAL.                        LABEL     0012
LJSR                            .GLOBAL.                        LABEL     0013
LLDB                            .GLOBAL.                        LABEL     0024
LLEF                            .GLOBAL.                        LABEL     0018
                             *** Symbol Table ***
UASM     00.10.00                 0003 - 02
```

```
SYMBOL                           QUALIFIER                        TYPE     VALUE
------                           ---------                        ----     -----
LNLDA                            .GLOBAL.                         LABEL     0010
LNSBI                            .GLOBAL.                         LABEL     001D
LNSTA                            .GLOBAL.                         LABEL     0011
LOAD_SIGNS                       .GLOBAL.                         PARSE     0153
LWLDA                            .GLOBAL.                         LABEL     0010
LWSBI                            .GLOBAL.                         LABEL     001C
LWSTA                            .GLOBAL.                         LABEL     0011
M1                               IAB                              CONST     0007
MEM_READ                         CPM_SRC                          PARSE     0053
MEM_WRITE                        CPM_DEST                         PARSE     004E
                                 ALU_DEST                         PARSE     00F6
MODIFY_FLAGS_0123                .GLOBAL.                         PARSE     00C8
MODIFY_FLAGS_4567                .GLOBAL.                         PARSE     00C9
N                                MF                               CONST     0000
NADDI                            .GLOBAL.                         LABEL     001B
NORMALIZED_WR                    FS_SRC                           CONST     0003
NORMAL_XCT                       .GLOBAL.                         LABEL     0029
NOT                              TEST                             PARSE     0002
NOTST                            .GLOBAL.                         LABEL     0040
NOT_AND                          ALU_OP                           CONST     0002
NSL                              .GLOBAL.                         CONST     0022
NSP                              .GLOBAL.                         CONST     0020
NSTK_OVERFLOW                    .GLOBAL.                         XTRNL     0000
ONE                              AAB                              CONST     0005
OR                               ALU_OP                           CONST     0001
PASS                             AGB_SRC                          PARSE     003B
PATU                             .GLOBAL.                         LABEL     0045
PATUW                            .GLOBAL.                         LABEL     0046
PC                               AG_DEST                          PARSE     002A
                                 PDR_SRC                          CONST     0006
PC_OF_EXECUTION                  PDR_SRC                          CONST     0005
PC_REL_INDEX                     TEST                             CONST     005F
PDR                              .GLOBAL.                         PARSE     005B
                                 ALU_DEST                         PARSE     00EA
                                 IS                               PARSE     00FB
                                 IR                               PARSE     0113
POINT_SRC_TO                     .GLOBAL.                         PARSE     006F
PRESCALED_WR                     FS_SRC                           CONST     0002
PRESCALE_OPERAND                 WR_SEL                           PARSE     014C
PRIVILEGE_PROTECTION             .GLOBAL.                         XTRNL     0003
PROTECTION_FAULT                 .GLOBAL.                         XTRNL     0002
PRT_RMX                          .GLOBAL.                         CONST     0004
PSH                              .GLOBAL.                         LABEL     003C
PSHL                             .GLOBAL.                         LABEL     0041
PSHM                             .GLOBAL.                         LABEL     003F
PURGE_THE_ATU_CACHE              .GLOBAL.                         PARSE     00DA
R1                               RSHIFT                           CONST     0000
R4                               RSHIFT                           CONST     0003
READ_BYTE                        MEM_STR                          CONST     0003
READ_DOUBLE                      MEM_STR                          CONST     0002
READ_PC_BYTE                     .GLOBAL.                         LABEL     001F
READ_WORD                        MEM_STR                          CONST     0001
RESTARTABLE_INTERRUPT            .GLOBAL.                         XTRNL     0004
RETURN                           .GLOBAL.                         PARSE     0007
RETURN_ELSE_GOTO                 CONDITIONAL                      CONST     0004
RETURN_PC                        PDR_SRC                          CONST     0004
RING<>0                          TEST                             CONST     0011
RMAX_PROTECTION                  .GLOBAL.                         LABEL     0019
ROUND_BIT                        FR_SRC                           CONST     0002
                              *** Symbol Table ***
UASM     00.10.00                  0004 - 02
```

```
            SYMBOL                           QUALIFIER                        TYPE      VALUE
            ------                           ---------                        ----      -----
            SELECTED_A                       FR_SRC                           PARSE      0134
            SET                              MF                               CONST      0001
            SHIFT_MAG                        .GLOBAL.                         PARSE      013D
            SIGN                             .GLOBAL.                         PARSE      0152
            SKIP_ON                          .GLOBAL.                         PARSE      00CC
            SL                               IAB                              CONST      0005
            SP                               AAB                              CONST      0004
            SPAD                             ALU_DEST                         PARSE      00EC
                                             IS                               PARSE      0100
                                             ID_SRC                           PARSE      012A
            SPAR                             .GLOBAL.                         PARSE      00CF
                                             R_SPAD                           PARSE      0101
            SPAR_TABLE_OFFSET                ALU_DEST                         PARSE      00EF
            SRC                              AAB                              CONST      000E
                                             IAB                              CONST      000E
                                             FABC                             CONST      000E
            SRC<>DES                         TEST                             CONST      006E
            SRC=DES                          TEST                             CONST      002E
            SRC_POINTER                      IR                               PARSE      0115
            STA                              .GLOBAL.                         LABEL      0011
            START                            .GLOBAL.                         PARSE      0019
            START_EXECUTE                    .GLOBAL.                         PARSE      0025
            SUBL                             .GLOBAL.                         LABEL      0017
            TOGGLE                           MF                               CONST      0003
            TREG                             CPM_DEST                         PARSE      0048
                                             PDR_SRC                          CONST      0002
                                             ALU_DEST                         PARSE      00F0
                                             IR                               PARSE      011B
            TRUNCATED_IF_NOT_ROUNDING        .GLOBAL.                         PARSE      0139
            TWO                              AAB                              CONST      0006
            UPDATE_FPSR                      .GLOBAL.                         PARSE      0148
            UPDATE_OVR                       .GLOBAL.                         PARSE      00E3
            WADDI                            .GLOBAL.                         LABEL      001B
            WAIT                             .GLOBAL.                         LABEL      0FFF
            WASHM17                          .GLOBAL.                         LABEL      0031
            WBITW                            .GLOBAL.                         LABEL      0036
            WBITW1                           .GLOBAL.                         LABEL      0037
            WBITW2                           .GLOBAL.                         LABEL      0038
            WBR                              .GLOBAL.                         LABEL      0012
            WCOB                             .GLOBAL.                         LABEL      003A
            WCOB1                            .GLOBAL.                         LABEL      003B
            WCOBTAB                          .GLOBAL.                         LABEL      0000
            WIDE_JUMP                        MEM_STR                          PARSE      001F
            WITH_WORD_ADDRESSING             MEM_STR                          PARSE      001B
            WLDB                             .GLOBAL.                         LABEL      0023
            WLMP                             .GLOBAL.                         LABEL      0055
            WLMP_DONE                        .GLOBAL.                         LABEL      0064
            WLMP_HI_WAIT                     .GLOBAL.                         LABEL      005D
            WLMP_INT                         .GLOBAL.                         LABEL      0063
            WLMP_LOOP                        .GLOBAL.                         LABEL      0057
            WLMP_LO_WAIT                     .GLOBAL.                         LABEL      0061
            WNADI                            .GLOBAL.                         LABEL      001B
            WORD_ZERO_EXTEND                 IS                               PARSE      010E
            WPSH                             .GLOBAL.                         LABEL      0042
            WPSHT                            .GLOBAL.                         LABEL      0044
            WPSH_LOOP                        .GLOBAL.                         LABEL      0043
            WR                               FPU_DEST                         PARSE      0131
                                             .GLOBAL.                         PARSE      014B
            WRITE_BYTE                       MEM_STR                          CONST      0007
                                         *** Symbol Table ***
            UASM     00.10.00                 0005 - 02
```

```
            SYMBOL                           QUALIFIER                        TYPE      VALUE
            ------                           ---------                        ----      -----
            WRITE_DOUBLE                     MEM_STR                          CONST      0006
            WRITE_PC_BYTE                    .GLOBAL.                         LABEL      0021
            WRITE_WORD                       MEM_STR                          CONST      0005
            WSTK_OVERFLOW                    .GLOBAL.                         XTRNL      0001
            WSZBO                            .GLOBAL.                         LABEL      0030
            WSZBOBIT                         .GLOBAL.                         LABEL      0035
            WSZBONRM                         .GLOBAL.                         LABEL      0032
            XCT                              .GLOBAL.                         LABEL      0026
            XCTED_INSTRUCTION                TEST                             CONST      0006
            XCTOP                            .GLOBAL.                         LABEL      00C3
            XCT_WAIT1                        .GLOBAL.                         LABEL      002B
            XCT_WAIT2                        .GLOBAL.                         LABEL      002C
            XFAMD                            .GLOBAL.                         LABEL      0050
            XFAMS                            .GLOBAL.                         LABEL      004F
            XFLDD                            .GLOBAL.                         LABEL      004A
            XFLDS                            .GLOBAL.                         LABEL      0049
            XFSTD                            .GLOBAL.                         LABEL      004D
            XFSTS                            .GLOBAL.                         LABEL      004C
            XJMP                             .GLOBAL.                         LABEL      0012
            XJSR                             .GLOBAL.                         LABEL      0013
            XLDB                             .GLOBAL.                         LABEL      0024
            XLEF                             .GLOBAL.                         LABEL      0018
            XLSBI                            .GLOBAL.                         LABEL      001E
            XNLDA                            .GLOBAL.                         LABEL      0010
            XNSBI                            .GLOBAL.                         LABEL      001D
            XNSTA                            .GLOBAL.                         LABEL      0011
            XWLDA                            .GLOBAL.                         LABEL      0010
            XWSBI                            .GLOBAL.                         LABEL      001C
            XWSTA                            .GLOBAL.                         LABEL      0011
            ZERO                             PDR_SRC                          CONST      000F
                                             CPD_SRC                          PARSE      006A
                                             IS                               PARSE      00FF
                                             FS_SRC                           CONST      0001
             •
             •
             •

                                         *** Symbol Table ***
            UASM     00.10.00                 0006 - 02
```

```
SYMBOL                   REFERENCES
------                   ----------
&:.GLOBAL.                 489-01
+:AOP1                      87-01      92-01      97-01     102-01     107-01     113-01
                           118-01     123-01     128-01     133-01     138-01     143-01
                           148-01     153-01     158-01     423-01     430-01     568-01
                           606-01     623-01     660-01     708-01     723-01     737-01
                           742-01     858-01     908-01     959-01    1118-01
+:ALU_OP                   258-01     334-01     338-01    1070-01    1141-01
+:FP_OP                    863-01    1017-01
+1+:ALU_OP                 262-01    1063-01    1107-01    1131-01
-:AOP1                     785-01
-:ALU_OP                   266-01     421-01     428-01     714-01     741-01    1150-01
-:FP_OP                    998-01
-1-:ALU_OP                 373-01
.BEGIN:.COMMAND.             4-01
.EJECT:.COMMAND.             1-01      30-01      55-01     162-01     234-01     270-01
                           376-01     462-01     531-01     665-01     747-01     792-01
                          1026-01    1160-01
.END:.COMMAND.            1174-01
.EXTERNAL:.COMMAND.         23-01
.FT:.COMMAND.                8-01      31-01      56-01     163-01     235-01     271-01
                           377-01     463-01     532-01     666-01     748-01     793-01
                          1027-01    1161-01
.HD:.COMMAND.                5-01       6-01
.LOC:.COMMAND.            1164-01    1164-01
.RADIX:.COMMAND.             7-01      29-01       7-01      29-01
.TITLE:.COMMAND.             2-01
=:AG_DEST                   87-01      87-01      92-01      92-01      97-01      97-01
                           102-01     102-01     107-01     107-01     113-01     113-01
                           118-01     118-01     123-01     123-01     128-01     128-01
                           133-01     133-01     138-01     138-01     143-01     143-01
                           148-01     148-01     153-01     153-01     158-01     158-01
                           606-01     660-01     660-01     708-01     723-01     737-01
                           742-01     785-01     791-01    1118-01    1118-01
=:CPM_DEST                 195-01     195-01     202-01     314-01     365-01     369-01
                           459-01     459-01     503-01     528-01     575-01     619-01
                           702-01     707-01     713-01     722-01     740-01     833-01
                           856-01     863-01     885-01     907-01     912-01     936-01
                           958-01     966-01    1087-01    1112-01
=:CPD_DEST                 653-01    1100-01    1106-01    1119-01    1125-01    1130-01
                          1140-01
=:ALU_DEST                  88-01      88-01      93-01      93-01      98-01      98-01
                           103-01     103-01     108-01     108-01     114-01     114-01
                           119-01     119-01     124-01     124-01     129-01     129-01
                           134-01     134-01     139-01     139-01     144-01     144-01
                           149-01     149-01     154-01     154-01     159-01     159-01
                           232-01     232-01     258-01     258-01     262-01     262-01
                           266-01     266-01     306-01     306-01     313-01     334-01
                           334-01     338-01     338-01     373-01     421-01     428-01
                           499-01     523-01     527-01     561-01     569-01     574-01
                           579-01     591-01     661-01     717-01     745-01     788-01
                          1063-01    1070-01    1070-01    1081-01    1091-01    1095-01
                          1095-01    1101-01    1107-01    1124-01    1131-01    1131-01
                          1141-01    1141-01    1150-01    1150-01
=:FPU_DEST                1008-01    1017-01
==:AG_DEST                  87-01      92-01      97-01     102-01     107-01     113-01
                           118-01     123-01     128-01     133-01     138-01     143-01
                           148-01     153-01     158-01     366-01     370-01     423-01
                           430-01     442-01     568-01     590-01     606-01     613-01
                           623-01     660-01     698-01     703-01     708-01     715-01
                              *** Symbol Cross Reference ***
UASM     00.10.00                      0001 - 03
```

```
SYMBOL                REFERENCES
------                ----------
                       723-01    737-01    742-01    785-01    791-01    858-01
                       908-01    959-01   1082-01   1118-01
==:CPM_DEST            195-01    202-01    314-01    365-01    369-01    459-01
                       503-01    528-01    575-01    619-01    702-01    707-01
                       713-01    722-01    740-01    833-01    856-01    863-01
                       885-01    907-01    912-01    936-01    958-01    966-01
                      1087-01   1112-01
==:CPD_DEST            230-01    457-01    653-01    713-01    740-01    784-01
                      1100-01   1106-01   1119-01   1125-01   1130-01   1140-01
                      1151-01
==:ALU_DEST             88-01     93-01     98-01    103-01    108-01    114-01
                       119-01    124-01    129-01    134-01    139-01    144-01
                       149-01    154-01    159-01    232-01    258-01    262-01
                       266-01    306-01    313-01    334-01    338-01    373-01
                       421-01    428-01    499-01    523-01    527-01    561-01
                       569-01    574-01    579-01    584-01    591-01    661-01
                       717-01    745-01    788-01   1063-01   1070-01   1081-01
                      1091-01   1095-01   1101-01   1107-01   1124-01   1131-01
                      1141-01   1150-01
==:EQUAL                89-01     94-01     99-01    104-01    109-01    115-01
                       120-01    125-01    130-01    135-01    140-01    145-01
                       150-01    155-01    160-01    259-01    263-01    267-01
                       339-01    374-01    563-01    662-01    714-01    741-01
                       834-01    834-01    857-01    857-01    999-01    999-01
                      1009-01   1009-01   1019-01   1020-01
==:ID_DEST             491-01    562-01
==:FPU_DEST            863-01    998-01   1008-01   1017-01
@+:FP_OP              1008-01
A:AGA                   87-01     92-01     97-01    102-01    107-01    113-01
                       118-01    123-01    128-01    133-01    138-01    143-01
                       148-01    153-01    158-01    423-01    430-01    568-01
                       606-01    623-01    660-01    708-01    723-01    737-01
                       742-01    785-01    858-01    908-01    959-01   1118-01
A:IS                    89-01     94-01     99-01    104-01    109-01    115-01
                       120-01    125-01    130-01    135-01    140-01    145-01
                       150-01    155-01    160-01    338-01    373-01    579-01
                       584-01    662-01    788-01   1070-01   1141-01
A:IR                   232-01    258-01    262-01    266-01    421-01    428-01
                       499-01    523-01    527-01    561-01    569-01    574-01
                       591-01    745-01   1063-01   1081-01   1091-01   1095-01
                      1101-01   1107-01   1124-01   1131-01   1150-01
A:FR_SRC               863-01    998-01
A:FA_SRC               834-01    857-01
ABORT_MEMORY:.GLOBAL.            308-01    312-01    421-01    428-01    607-01
                       614-01    745-01   1159-01   1171-01
AC0:IAB               1081-01   1131-01   1131-01
AC1:IAB               1063-01   1070-01   1141-01   1141-01   1150-01   1150-01
AC2:IAB               1118-01
AC3:IAB                232-01
ADD:.GLOBAL.           258*01
ADDI:.GLOBAL.          334*01
AG:AG_DEST              87-01     92-01     97-01    102-01    107-01    113-01
                       118-01    123-01    128-01    133-01    138-01    143-01
                       148-01    153-01    158-01    606-01    660-01    708-01
                       723-01    737-01    742-01   1118-01
AG:CPM_DEST            195-01    314-01    459-01    619-01    702-01
AG:PDR_SRC             653-01    713-01    740-01
AG:ALU_DEST            232-01    258-01    262-01    266-01    306-01    334-01
                       338-01    421-01    428-01    561-01    745-01    788-01
                          *** Symbol Cross Reference ***
UASM     00.10.00                 0002 - 03
```

```
SYMBOL                REFERENCES
------                ----------
                       1070-01   1131-01   1141-01   1150-01
AG:IR                   741-01
AG0:AAB                1131-01
AG1:AAB                1070-01   1141-01   1150-01
AG2:AAB                1082-01   1118-01   1118-01
AG3:AAB                 232-01
ALC_CRY:CRY_RAND        259-01    263-01    267-01
ALC_RESULT:TSKIP        259-01    263-01    267-01
ALLOW_SHIFT_MAG_CORRECTION:.GLOBAL.        1018-01
ALU:AG_DEST              87-01     92-01     97-01    102-01    107-01    113-01
                        118-01    123-01    128-01    133-01    138-01    143-01
                        148-01    153-01    158-01    660-01   1118-01
ALU:CPM_DEST            195-01    365-01    369-01    459-01    707-01   1087-01
ALU:CPM_SRC             202-01    314-01    503-01    528-01    713-01    722-01
                        740-01   1112-01
ALU:ALU_DEST            232-01    258-01    262-01    266-01    306-01    313-01
                        334-01    338-01    523-01    574-01   1070-01   1081-01
                       1091-01   1095-01   1107-01   1131-01   1141-01   1150-01
ALU=0:TEST               85-01   1083-01
ALU=0:TSKIP             585-01
ALU_CRY:CRY_RAND        339-01    374-01
ALU_TEST:.GLOBAL.        89-01     94-01     99-01    104-01    109-01    115-01
                        120-01    125-01    130-01    135-01    140-01    145-01
                        150-01    155-01    160-01    662-01    714-01    741-01
AND:ALU_OP               89-01     94-01     99-01    104-01    109-01    115-01
                        120-01    125-01    130-01    135-01    140-01    145-01
                        150-01    155-01    160-01    232-01    569-01    574-01
                        584-01    591-01    662-01    745-01    788-01
AR0:AAB                 421-01    423-01    428-01    430-01    561-01    568-01
                        590-01    623-01    702-01    708-01    708-01    713-01
                        723-01    723-01    788-01    791-01
AR1:AAB                 606-01    613-01    619-01    623-01
AR3:AAB                 785-01    785-01
AR5:AAB                 314-01
ATTEMPT_NEXT_EFA:.GLOBAL.          85-01    195-01    202-01    233-01    260-01
                        264-01    268-01    308-01    334-01    340-01    375-01
                        459-01    512-01    583-01    717-01    745-01    835-01
                        864-01    885-01    912-01   1020-01   1159-01   1171-01
ATU_PURGING:TEST        784-01
AY:.GLOBAL.              87-01     92-01     97-01    102-01    107-01    113-01
                        118-01    123-01    128-01    133-01    138-01    143-01
                        148-01    153-01    158-01    660-01    785-01   1118-01
AY:AG_OP                366-01    370-01    423-01    430-01    442-01    568-01
                        590-01    606-01    613-01    623-01    698-01    703-01
                        708-01    715-01    723-01    737-01    742-01    791-01
                        858-01    908-01    959-01   1082-01
A_SIGN:SGNR            1019-01
B:AGB_SRC               442-01    568-01    590-01    613-01    623-01    708-01
                        723-01    785-01    791-01   1082-01   1118-01
B:IS                    258-01    266-01    334-01    569-01    591-01    714-01
                        741-01   1150-01
B:IR                   1141-01
B:ID_SRC                562-01
B:FS_SRC                998-01
BIT0:.GLOBAL.           306-01    523-01
BIT16:.GLOBAL.          563-01
BIT_SHIFT_LEFT:IS       266-01   1081-01
CALL:CONDITIONAL        457-01    570-01    709-01
CARRY:.GLOBAL.          259-01    263-01    267-01    339-01    374-01
                           *** Symbol Cross Reference ***
UASM     00.10.00                  0003 - 03
```

```
SYMBOL              REFERENCES
------              ----------
CASE_4_INTO:.GLOBAL.             90-01     95-01    100-01    105-01    110-01
                      116-01    121-01    126-01    131-01    136-01    141-01
                      146-01    151-01    156-01    161-01    663-01
CASE_DATA:CPD_DEST    653-01
CASE_DATA:ALU_DEST     88-01     93-01     98-01    103-01    108-01    114-01
                      119-01    124-01    129-01    134-01    139-01    144-01
                      149-01    154-01    159-01    661-01
CLEAR:MF              504-01
CNST:AGB_SRC           87-01     92-01     97-01    102-01    107-01    113-01
                      118-01    123-01    128-01    133-01    138-01    143-01
                      148-01    153-01    158-01    606-01    660-01    698-01
                      703-01    715-01    737-01    742-01
CNST:IS               313-01
COMPLETE_JUMP:.GLOBAL.          215-01    230-01
CPD:.GLOBAL.         1140-01
CPD:IR                714-01
CPD_ZERO:IR           313-01    717-01
CPM:.GLOBAL.          195-01    202-01    314-01    365-01    369-01    459-01
                      503-01    528-01    575-01    619-01    702-01    707-01
                      713-01    722-01    740-01    833-01    856-01    863-01
                      885-01    907-01    912-01    936-01    958-01    966-01
                     1087-01   1112-01
CRE:AG_DEST           785-01
DECREMENT_DES_POINTER:.GLOBAL.            709-01
DEFER_ON_FALSE_TEST:.GLOBAL.    608-01    615-01
DES:AAB                87-01     87-01     92-01     92-01     97-01     97-01
                      102-01    102-01    107-01    107-01    113-01    113-01
                      118-01    118-01    123-01    123-01    128-01    128-01
                      133-01    133-01    138-01    138-01    143-01    143-01
                      148-01    148-01    153-01    153-01    158-01    158-01
                      195-01    258-01    262-01    266-01    306-01    334-01
                      338-01    459-01    660-01    660-01
DES:IAB                87-01     92-01     97-01    102-01    107-01    113-01
                      118-01    123-01    128-01    133-01    138-01    143-01
                      148-01    153-01    158-01    195-01    202-01    258-01
                      258-01    262-01    266-01    266-01    306-01    314-01
                      334-01    334-01    338-01    338-01    459-01    499-01
                      503-01    523-01    561-01    569-01    591-01    660-01
DES:FABC              833-01    834-01    856-01    857-01    863-01    863-01
                      885-01    907-01    912-01    998-01   1008-01   1017-01
EJMP:.GLOBAL.         214*01
EJSR:.GLOBAL.         229*01
ELDA:.GLOBAL.         194*01
ELDB:.GLOBAL.         457*01
ELEF:.GLOBAL.         305*01
ENABLE_MOF_CORRECTION:.GLOBAL.           1018-01
ESTA:.GLOBAL.         201*01
EWR+MAG+MOF:EXPR     1020-01
EXECUTE:.GLOBAL.      488*01
EXECUTE_DATA:CPM_DEST           503-01    528-01
EXECUTE_PBX:.GLOBAL.            522*01    496-01
EXPONENT:.GLOBAL.     834-01    857-01    999-01   1009-01   1020-01
EXTEND_MICRO_CYCLE:.GLOBAL.     307-01
FA:.GLOBAL.           834-01    857-01
FA:EXPR               834-01    857-01
FA+MOF:EXPR          1009-01
FA-FB:EXPR            999-01
FAD:.GLOBAL.          997*01    966-01
FAMD:.GLOBAL.         958*01
                         *** Symbol Cross Reference ***
UASM    00.10.00                   0004 - 03
```

```
SYMBOL              REFERENCES
------              ----------
FAMS:.GLOBAL.        936*01
FAS:.GLOBAL.         998*01    936-01
FD:.GLOBAL.          863-01    998-01   1008-01   1017-01
FG0:FABC             936-01    958-01    966-01
FG0:ALL_REGSS        936-01    960-01
FLAG4=0:TEST         510-01
FLAG4=1:TEST         513-01
FLDD:.GLOBAL.        856*01
FLDS:.GLOBAL.        833*01
FOR:MEM_STR          366-01    370-01    423-01    430-01    442-01    568-01
                     590-01    606-01    613-01    623-01    698-01    703-01
                     708-01    715-01    723-01    737-01    742-01    791-01
                     858-01    908-01    959-01   1082-01
FPU:FPU_DEST        1017-01
FP_HIGH:CPM_DEST     833-01    856-01    936-01    958-01
FP_HIGH:CPM_SRC      885-01    907-01
FP_LOW:CPM_DEST      863-01    966-01
FP_LOW:CPM_SRC       912-01
FRND:.GLOBAL.       1017*01
FSTD:.GLOBAL.        907*01
FSTS:.GLOBAL.        885*01
GOTO:CONDITIONAL     491-01    496-01    510-01    564-01    743-01    779-01
                     784-01   1065-01   1071-01   1083-01   1113-01   1135-01
                    1142-01
GOTO:.GLOBAL.        215-01    314-01    367-01    371-01    442-01    489-01
                     505-01    530-01    592-01    609-01    791-01    936-01
                     966-01   1151-01
GR0:IAB              313-01    365-01    369-01    373-01    574-01    579-01
                     584-01   1081-01   1095-01   1095-01   1107-01   1107-01
                    1112-01
GR1:IAB              707-01    714-01   1091-01   1101-01
GR2:IAB             1087-01   1091-01   1124-01
GR5:IAB              523-01    527-01    528-01
HEX_SHIFT_RIGHT:IS    88-01     93-01     98-01    103-01    108-01    114-01
                     119-01    124-01    129-01    134-01    139-01    144-01
                     149-01    154-01    159-01    561-01    661-01   1091-01
ID:.GLOBAL.          491-01    562-01
ID_SIGN=1:TEST       496-01    570-01
IF:.GLOBAL.           85-01    309-01    457-01    491-01    496-01    510-01
                     513-01    564-01    570-01    624-01    709-01    718-01
                     724-01    743-01    746-01    779-01    784-01   1065-01
                    1071-01   1083-01   1113-01   1135-01   1142-01
INC:.GLOBAL.         262*01
INCREMENT_SRC_POINTER:.GLOBAL.           724-01    743-01
INDIRECT:TEST        624-01
INTERRUPT_PENDING:TEST        1071-01   1142-01
INWARD_REFERENCE:TEST          309-01
IN_CURRENT_RING:MEM_STR        590-01
IOCMD:.GLOBAL.      1101-01   1124-01
IOCMD1:.GLOBAL.     1095-01
IO_BUSY:TEST        1113-01   1135-01
IO_CONTROLLER:CPD_DEST        1100-01   1106-01   1119-01   1125-01   1130-01
                    1140-01
IS>=IR:TEST          718-01    746-01
IY:.GLOBAL.           88-01     93-01     98-01    103-01    108-01    114-01
                     119-01    124-01    129-01    134-01    139-01    144-01
                     149-01    154-01    159-01    232-01    258-01    262-01
                     266-01    306-01    313-01    334-01    338-01    373-01
                     421-01    428-01    499-01    523-01    527-01    561-01
                        *** Symbol Cross Reference ***
UASM    00.10.00                   0005 - 03
```

```
SYMBOL              REFERENCES
------              ----------
                      569-01    574-01    579-01    584-01    591-01    661-01
                      717-01    745-01    788-01   1063-01   1070-01   1081-01
                     1091-01   1095-01   1101-01   1107-01   1124-01   1131-01
                     1141-01   1150-01
JMP:.GLOBAL.          215*01    791-01
JSR:.GLOBAL.          230*01
LAR:IR                306-01    334-01    338-01
LAST_LA:AGB_SRC       366-01    370-01    423-01    430-01    858-01    908-01
                      959-01
LDA:.GLOBAL.          195*01    442-01
LDB:.GLOBAL.          442*01
LEADING_ZERO_DETECT:MAG_OP     1009-01
LEF:.GLOBAL.          306*01
LFAMD:.GLOBAL.        956*01
LFAMS:.GLOBAL.        934*01
LFLDD:.GLOBAL.        854*01
LFLDS:.GLOBAL.        831*01
LFSTD:.GLOBAL.        905*01
LFSTS:.GLOBAL.        883*01
LJMP:.GLOBAL.         212*01
LJSR:.GLOBAL.         227*01
LLDB:.GLOBAL.         455*01
LLEF:.GLOBAL.         303*01
LNLDA:.GLOBAL.        192*01
LNSBI:.GLOBAL.        369*01
LNSTA:.GLOBAL.        199*01
LOAD_SIGNS:.GLOBAL.   998-01   1009-01
LWLDA:.GLOBAL.        190*01
LWSBI:.GLOBAL.        365*01
LWSTA:.GLOBAL.        197*01
M1:IAB                 89-01     94-01     99-01    104-01    109-01    115-01
                      120-01    125-01    130-01    135-01    140-01    145-01
                      150-01    155-01    160-01    232-01    421-01    428-01
                      569-01    574-01    591-01    662-01    745-01    788-01
                     1070-01   1141-01   1150-01
MEM_READ:CPM_SRC      195-01    365-01    369-01    459-01    575-01    619-01
                      702-01    707-01    833-01    856-01    863-01    936-01
                      958-01    966-01   1087-01
MEM_WRITE:CPM_DEST    202-01    713-01    722-01    740-01    885-01    907-01
                      912-01
MEM_WRITE:ALU_DEST    373-01    579-01    717-01
MODIFY_FLAGS_0123:.GLOBAL.     1064-01
MODIFY_FLAGS_4567:.GLOBAL.      504-01    509-01    512-01    530-01
N:MF                  504-01    504-01    504-01    509-01    509-01    509-01
                      512-01    512-01    512-01    530-01    530-01    530-01
                     1064-01   1064-01   1064-01
NADDI:.GLOBAL.        338*01
NORMALIZED_WR:FS_SRC           1017-01
NORMAL_XCT:.GLOBAL.   498*01    491-01
NOT:TEST              309-01    491-01    624-01   1142-01
NOTST:.GLOBAL.        717*01
NOT_AND:ALU_OP        306-01   1081-01
NSL:.GLOBAL.          703-01
NSP:.GLOBAL.          698-01    715-01
NSTK_OVERFLOW:.GLOBAL.           23*01    718-01
ONE:AAB               708-01    723-01
OR:ALU_OP             313-01    523-01    527-01    579-01    717-01   1095-01
                     1101-01   1124-01
PASS:AGB_SRC          366-01    370-01    442-01    590-01    613-01    698-01
                         *** Symbol Cross Reference ***
UASM     00.10.00                 0006 - 03
```

```
SYMBOL                  REFERENCES
------                  ----------
                        703-01    715-01    791-01   1082-01
PATU:.GLOBAL.           776*01
PATUW:.GLOBAL.          784*01    784-01
PC:AG_DEST              791-01
PC:PDR_SRC              784-01
PC_OF_EXECUTION:PDR_SRC           457-01   1151-01
PC_REL_INDEX:TEST       457-01
PDR:.GLOBAL.            230-01    457-01    653-01    713-01    740-01    784-01
                       1100-01   1106-01   1119-01   1125-01   1130-01   1151-01
PDR:ALU_DEST             88-01     93-01     98-01    103-01    108-01    114-01
                        119-01    124-01    129-01    134-01    139-01    144-01
                        149-01    154-01    159-01
PDR:IS                  232-01    421-01    428-01    717-01    745-01
PDR:IR                   88-01     89-01     93-01     94-01     98-01     99-01
                        103-01    104-01    108-01    109-01    114-01    115-01
                        119-01    120-01    124-01    125-01    129-01    130-01
                        134-01    135-01    139-01    140-01    144-01    145-01
                        149-01    150-01    154-01    155-01    159-01    160-01
                        661-01    662-01    788-01
POINT_SRC_TO:.GLOBAL.             936-01    960-01
PRESCALED_WR:FS_SRC    1008-01
PRESCALE_OPERAND:WR_SEL           999-01
PRIVILEGE_PROTECTION:.GLOBAL.      26*01    779-01   1065-01
PROTECTION_FAULT:.GLOBAL.          25*01    314-01
PRT_RMX:.GLOBAL.        313-01
PSH:.GLOBAL.            698*01
PSHL:.GLOBAL.           722*01    709-01    724-01
PSHM:.GLOBAL.           713*01
PURGE_THE_ATU_CACHE:.GLOBAL.      788-01
R1:RSHIFT                88-01     93-01     98-01    103-01    108-01    114-01
                        119-01    124-01    129-01    134-01    139-01    144-01
                        149-01    154-01    159-01    561-01    661-01
R4:RSHIFT              1091-01
READ_BYTE:MEM_STR       423-01    442-01
READ_DOUBLE:MEM_STR     606-01    613-01    858-01    959-01   1082-01
READ_PC_BYTE:.GLOBAL.             420*01    457-01
READ_WORD:MEM_STR       698-01    703-01
RESTARTABLE_INTERRUPT:.GLOBAL.               27*01   1151-01
RETURN:.GLOBAL.         195-01    202-01    233-01    260-01    264-01    268-01
                        334-01    340-01    375-01    424-01    431-01    459-01
                        586-01    835-01    864-01    885-01    912-01   1020-01
                       1159-01   1171-01
RETURN_ELSE_GOTO:CONDITIONAL       85-01    309-01    513-01    624-01    718-01
                        724-01    746-01
RETURN_PC:PDR_SRC       230-01
RING<>0:TEST            779-01   1065-01
RMAX_PROTECTION:.GLOBAL.          311*01    309-01
ROUND_BIT:FR_SRC       1017-01
SELECTED_A:FR_SRC      1008-01
SET:MF                  530-01   1064-01
SHIFT_MAG:.GLOBAL.     1009-01
SIGN:.GLOBAL.          1019-01
SKIP_ON:.GLOBAL.        259-01    263-01    267-01    585-01
SL:IAB                  741-01
SP:AAB                  737-01    737-01    740-01    741-01    742-01    742-01
                        745-01
SPAD:ALU_DEST           499-01    527-01
SPAD:IS                 306-01    523-01    574-01   1081-01   1095-01   1101-01
                       1124-01
                            *** Symbol Cross Reference ***
UASM    00.10.00                        0007 - 03
```

```
SYMBOL              REFERENCES
------              ----------
SPAD:ID_SRC           491-01
SPAR:.GLOBAL.         563-01
SPAR:R_SPAD           574-01
SPAR_TABLE_OFFSET:ALU_DEST        569-01    591-01
SRC:AAB               442-01    568-01    606-01    653-01
SRC:IAB               258-01    262-01    266-01    562-01    713-01    722-01
                      740-01
SRC:FABC              998-01   1008-01
SRC<>DES:TEST         709-01    743-01
SRC=DES:TEST          564-01    724-01
SRC_POINTER:IR        373-01
STA:.GLOBAL.          202*01
START:.GLOBAL.        366-01    370-01    423-01    430-01    442-01    568-01
                      590-01    606-01    613-01    623-01    698-01    703-01
                      708-01    715-01    723-01    737-01    742-01    791-01
                      858-01    908-01    959-01   1082-01
START_EXECUTE:.GLOBAL.            499-01    523-01
SUBL:.GLOBAL.         266*01
TOGGLE:MF             509-01    512-01
TREG:CPM_DEST         575-01   1112-01
TREG:PDR_SRC         1100-01   1106-01   1125-01   1130-01
TREG:ALU_DEST        1063-01   1095-01   1101-01   1124-01
TREG:IR               579-01    584-01   1070-01
TRUNCATED_IF_NOT_ROUNDING:.GLOBAL.       1008-01
TWO:AAB               858-01    908-01    959-01   1118-01
UPDATE_FPSR:.GLOBAL.              834-01    857-01   1020-01
UPDATE_OVR:.GLOBAL.   339-01    374-01
WADDI:.GLOBAL.        336*01
WAIT:.GLOBAL.        1171*01    215-01
WASHM17:.GLOBAL.     1081-01
WBITW:.GLOBAL.        606*01    570-01
WBITW1:.GLOBAL.       613*01    624-01
WBITW2:.GLOBAL.       619*01    609-01
WBR:.GLOBAL.          211*01
WCOB:.GLOBAL.         653*01
WCOB1:.GLOBAL.        660*01     85-01
WCOBTAB:.GLOBAL.       85*01     90-01     95-01    100-01    105-01    110-01
                      116-01    121-01    126-01    131-01    136-01    141-01
                      146-01    151-01    156-01    161-01    663-01
WIDE_JUMP:MEM_STR     791-01
WITH_WORD_ADDRESSING:MEM_STR    423-01    430-01
WLDB:.GLOBAL.         441*01
WLMP:.GLOBAL.        1057*01
WLMP_DONE:.GLOBAL.   1154*01   1083-01
WLMP_HI_WAIT:.GLOBAL.          1112*01   1113-01
WLMP_INT:.GLOBAL.    1145*01   1071-01
WLMP_LOOP:.GLOBAL.   1074*01   1142-01
WLMP_LO_WAIT:.GLOBAL.          1135*01   1135-01
WNADI:.GLOBAL.        337*01
WORD_ZERO_EXTEND:IS   499-01
WPSH:.GLOBAL.         737*01
WPSHT:.GLOBAL.        745*01
WPSH_LOOP:.GLOBAL.    739*01    743-01
WR:FPU_DEST          1008-01
WR:.GLOBAL.           999-01
WRITE_BYTE:MEM_STR    430-01
WRITE_DOUBLE:MEM_STR              366-01    737-01    742-01    908-01
WRITE_PC_BYTE:.GLOBAL.            427*01
WRITE_WORD:MEM_STR    370-01    568-01    590-01    623-01    708-01    715-01
                        *** Symbol Cross Reference ***
UASM     00.10.00                  0008 - 03
```

```
               SYMBOL                  REFERENCES
               ------                  ----------
                                         723-01
               WSTK_OVERFLOW:.GLOBAL.              24*01      746-01
               WSZBO:.GLOBAL.            561*01
               WSZBOBIT:.GLOBAL.         590*01   564-01
               WSZBONRM:.GLOBAL.         574*01   592-01
               XCT:.GLOBAL.              489*01
               XCTED_INSTRUCTION:TEST             491-01
               XCTOP:.GLOBAL.            491-01   499-01   527-01
               XCT_WAIT1:.GLOBAL.        509*01   505-01   510-01   530-01
               XCT_WAIT2:.GLOBAL.        512*01   513-01
               XFAMD:.GLOBAL.            957*01
               XFAMS:.GLOBAL.            935*01
               XFLDD:.GLOBAL.            855*01
               XFLDS:.GLOBAL.            832*01
               XFSTD:.GLOBAL.            906*01
               XFSTS:.GLOBAL.            884*01
               XJMP:.GLOBAL.             213*01
               XJSR:.GLOBAL.             228*01
               XLDB:.GLOBAL.             456*01
               XLEF:.GLOBAL.             304*01
               XLSBI:.GLOBAL.            373*01   367-01   371-01
               XNLDA:.GLOBAL.            193*01
               XNSBI:.GLOBAL.            368*01
               XNSTA:.GLOBAL.            200*01
               XWLDA:.GLOBAL.            191*01
               XWSBI:.GLOBAL.            364*01
               XWSTA:.GLOBAL.            198*01
               ZERO:PDR_SRC             1119-01
               ZERO:CPD_SRC             1140-01
               ZERO:IS                   262-01   527-01  1063-01  1107-01  1131-01
               ZERO:FS_SRC               863-01

               •
               •
               •

                                        *** Symbol Cross Reference ***
               UASM     00.10.00                  0009 - 03
```

```
   1    .EJECT;
   2A   .TITLE "Widgeon Microcode: SAMPLES Code Group"
   3A   ;
   4B   .BEGIN;
   5B   .HD 1   "Proprietary information of Data General Corporation";
   6B   .HD 2   "";
   7B   .RADIX 16;
   8B   .FT 2 "SAMPLES          Instruction Set Microcode      Rev   1         09-DEC-82 10:47:39 RGG"
   9B   ;
  10B
  11B

   •
   •
   •

 SAMPLES          Instruction Set Microcode      Rev   1         09-DEC-82 10:47:39 RGG

 UASM     00.10.00                   0001 - 01
```

```
Proprietary information of Data General Corporation

   12B  /*-----------------------------------------------------------------------+
   13B* |                                                                        |
   14B* |      External definitions for Widgeon microcode samples.               |
   15B* |                                                                        |
   16B* |      Some of the samples reference routines that, for the sake         |
   17B* |      of brevity, are not worth including in the samples.  The          |
   18B* |      number of such references in the collection will be kept          |
   19B* |      to a minimum.                                                     |
   20B* |                                                                        |
   21B* +-----------------------------------------------------------------------*/
   22B
   23B  .EXTERNAL       NSTK_OVERFLOW,
   24B                  WSTK_OVERFLOW,
   25B                  PROTECTION_FAULT,
   26B                  PRIVILEGE_PROTECTION,
   27B                  RESTARTABLE_INTERRUPT;
   28B
   29B  .RADIX  16;

    •
    •
    •

   SAMPLES          Instruction Set Microcode      Rev   1         09-DEC-82 10:47:39 RGG

   UASM     00.10.00                   0002 - 01
```

```
Proprietary information of Data General Corporation

     30B  .EJECT;
     31B  .FT 1 "SAMPLES         Source File                    Cycle 1        18-AUG-82 15:24:34 RGG"
     32B  ;
     33B
     34B  /*------------------------------------------------------------------+
     35B* |                                                                   |
     36B* |       Widgeon Microcode Samples                                   |
     37B* |                                                                   |
     38B* |       This collection of sample microcode is taken directly from  |
     39B* |       Widgeon sources.  Each selection is, as far as practical,   |
     40B* |       the code for an entire macro instruction.  Selections are   |
     41B* |       presented in order of increasing complexity.                |
     42B* |                                                                   |
     43B* +------------------------------------------------------------------*/
     44B
     45B  /*------------------------------------------------------------------+
     46B* |                                                                   |
     47B* |       A note regarding style:                                     |
     48B* |                                                                   |
     49B* |       Having been drawn from the sources, the samples display     |
     50B* |       a variety of documentation and coding styles.  These        |
     51B* |       variations are preserved mainly to minimize the task of     |
     52B* |       compiling the samples.                                      |
     53B* |                                                                   |
     54B* +------------------------------------------------------------------*/

     •
     •
     •

   SAMPLES          Instruction Set Microcode      Rev   1        09-DEC-82 10:47:39 RGG
   SAMPLES          Source File                    Cycle 1        18-AUG-82 15:24:34 RGG
   UASM     00.10.00                    0003 - 01
```

```
Proprietary information of Data General Corporation

     55B  .EJECT;
     56B  .FT 1 "TABLES          Source File                    Cycle 1        18-AUG-82 15:24:34 RGG"
     57B  ;
     58B
     59B  /*------------------------------------------------------------------+
     60B* |                                                                   |
     61B* |       Dispatch table for WCOB instruction, which appears in       |
     62B* |       an example below.                                           |
     63B* |                                                                   |
     64B* +------------------------------------------------------------------*/
     65B
     66B
     67B          %
     68B          % ***************************************************************
     69B          %
     70B          %                 BIT INSTRUCTION DISPATCH TABLES
     71B          %
     72B          % ***************************************************************
     73B          %
     74B          %
     75B          %
     76B          %       WCOBTAB - Used by WCOB, COB
     77B          %
     78B          % Dispatch table is based on the number of bits set (which is
     79B          % added to DES).
     80B          % AG: CPM <- DES <- DES + CONST; Load DSP REG;
     81B          % ALU: CPD <- PDR <- RSHIFT(PDR); DES <- CPM
     82B          % F bus <- PDR AND M1 for FZR test
     83B          %
     84B          % Location 0 of dispatch table checks for completion of instruction
--0000--WCOBTAB: ATTEMPT_NEXT_EFA,  IF ALU=0 RETURN_ELSE_GOTO WCOB1;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  FZR   WCOB1              D  EFA    S@

--DFVs:   addr is WCOB1 (003B)
   86B
--0001--        AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
   88B                IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
   89B                      ALU_TEST == A(M1) AND PDR,
   90B                            CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN      LD      M1  DES PD DA AND HR0  M  01
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
   91B
--0002--        AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
   93B                IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
   94B                      ALU_TEST == A(M1) AND PDR,
   95B                            CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN      LD      M1  DES PD DA AND HR0  M  01
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
   96B
--0003--        AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
   98B                IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
   99B                      ALU_TEST == A(M1) AND PDR,

SAMPLES         Instruction Set Microcode      Rev   1        09-DEC-82 10:47:39 RGG
TABLES          Source File                    Cycle 1        18-AUG-82 15:24:34 RGG
UASM     00.10.00                    0004 - 01
```

```
Proprietary information of Data General Corporation
  100B                                     CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y         AG  IY  GN     LD      M1  DES PD DA AND HR0  M  02
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
  101B
--0004--        AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
  103B'                 IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  104B                          ALU_TEST == A(M1) AND PDR,
  105B                                     CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y         AG  IY  GN     LD      M1  DES PD DA AND HR0  M  01
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
  106B
--0005--        AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
  108B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  109B                          ALU_TEST == A(M1) AND PDR,
  110B                                     CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y         AG  IY  GN     LD      M1  DES PD DA AND HR0  M  02
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
  111B
  112B
--0006--        AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
  114B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  115B                          ALU_TEST == A(M1) AND PDR,
  116B                                     CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y         AG  IY  GN     LD      M1  DES PD DA AND HR0  M  02
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
  117B
--0007--        AY = AG(DES) = ALU(DES) == CNST(03) + A(DES),
  119B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  120B                          ALU_TEST == A(M1) AND PDR,
  121B                                     CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y         AG  IY  GN     LD      M1  DES PD DA AND HR0  M  03
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
  122B
--0008--        AY = AG(DES) = ALU(DES) == CNST(01) + A(DES),
  124B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  125B                          ALU_TEST == A(M1) AND PDR,
  126B                                     CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y         AG  IY  GN     LD      M1  DES PD DA AND HR0  M  01
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
  127B
--0009--        AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
  129B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  130B                          ALU_TEST == A(M1) AND PDR,


SAMPLES         Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
TABLES          Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM     00.10.00                   0005 - 01
```

```
Proprietary information of Data General Corporation

  131B                          CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN      LD      M1  DES PD DA AND HRO  M  02
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
  132B
--000A--        AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
  134B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  135B                          ALU_TEST == A(M1) AND PDR,
  136B                          CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN      LD      M1  DES PD DA AND HRO  M  02
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
  137B
--000B--        AY = AG(DES) = ALU(DES) == CNST(03) + A(DES),
  139B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  140B                          ALU_TEST == A(M1) AND PDR,
  141B                          CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN      LD      M1  DES PD DA AND HRO  M  03
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
  142B
--000C--        AY = AG(DES) = ALU(DES) == CNST(02) + A(DES),
  144B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  145B                          ALU_TEST == A(M1) AND PDR,
  146B                          CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN      LD      M1  DES PD DA AND HRO  M  02
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
  147B
--000D--        AY = AG(DES) = ALU(DES) == CNST(03) + A(DES),
  149B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  150B                          ALU_TEST == A(M1) AND PDR,
  151B                          CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN      LD      M1  DES PD DA AND HRO  M  03
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
  152B
--000E--        AY = AG(DES) = ALU(DES) == CNST(03) + A(DES),
  154B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  155B                          ALU_TEST == A(M1) AND PDR,
  156B                          CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN      LD      M1  DES PD DA AND HRO  M  03
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
  157B
--000F--        AY = AG(DES) = ALU(DES) == CNST(04) + A(DES),
  159B                  IY = PDR = CASE_DATA == HEX_SHIFT_RIGHT( R1, PDR),
  160B                          ALU_TEST == A(M1) AND PDR,


SAMPLES        Instruction Set Microcode     Rev   1       09-DEC-82 10:47:39 RGG
TABLES         Source File                   Cycle 1       18-AUG-82 15:24:34 RGG
UASM     00.10.00                  0006 - 01
```

```
Proprietary information of Data General Corporation

  161B                          CASE_4_INTO WCOBTAB;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          DSPA        WCOBTAB  F DES DES C  ADD Y        AG  IY  GN      LD      M1  DES PD DA AND HRO  M  04
                                                                                               R1
--DFVs:   addr is WCOBTAB (0000)
   •
   •
   •

SAMPLES        Instruction Set Microcode     Rev   1       09-DEC-82 10:47:39 RGG
TABLES         Source File                   Cycle 1       18-AUG-82 15:24:34 RGG
UASM     00.10.00                  0007 - 01
```

```
Proprietary information of Data General Corporation

 162B  .EJECT;
 163B  .FT 1 "MEM             Source File                   Cycle 1        18-AUG-82 15:24:34 RGG"
 164B  ;
 165B
 166B  /*------------------------------------------------------------------+
 167B* |                                                                  |
 168B* |       Memory references for the next macro instruction can be    |
 169B* |       started by the IP from decode information.  In these two   |
 170B* |       examples, the completion of an IP initiated memory         |
 171B* |       reference is shown.  The completion is generic, i.e. read  |
 172B* |       or write.  The start instigated by the IP specified the    |
 173B* |       exact type of transfer to perform.                         |
 174B* |                                                                  |
 175B* |       Also shown here is the attempt of the next EFA on behalf   |
 176B* |       of the next executing macro instruction.  This attempt must|
 177B* |       be made in the last micro cycle of every macro instruction |
 178B* |       The combination of the attempt and popping an empty micro  |
 179B* |       stack constitutes a macro instruction pop (IPOP).          |
 180B* |                                                                  |
 181B* +------------------------------------------------------------------*/
 182B
 183B
 184B           %********
 185B           %  Load and Store Instructions:  <<L X><W N> E ><LDA STA>
 186B           %
 187B           %  Perform load or store of AC pointed to by DES.  IPOP.
 188B           %********
 189B
--0010--LWLDA:
--0010--XWLDA:
--0010--LNLDA:
--0010--XNLDA:
--0010--ELDA:
--0010--LDA:    CPM = AG(DES) = ALU(DES) == MEM_READ,  ATTEMPT_NEXT_EFA,  RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL  ST  CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                DES D  EFA M  S@ R  MM                          DES                M

--DFVs:
  196B
--0011--LWSTA:
--0011--XWSTA:
--0011--LNSTA:
--0011--XNSTA:
--0011--ESTA:
--0011--STA:    CPM = MEM_WRITE == ALU(DES),  ATTEMPT_NEXT_EFA,  RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL  ST  CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                    D  EFA    S@ W  IA                      DES

--DFVs:
  203B
  204B
  205B          %********
  206B          %  Jump Instructions:  WBR, LJMP, XJMP, EJMP
  207B          %
  208B          %  Complete IPST.  Go to IP_ALT WAIT.
  209B          %********
  210B
--0012--WBR:
SAMPLES         Instruction Set Microcode     Rev  1        09-DEC-82 10:47:39 RGG
MEM             Source File                   Cycle 1       18-AUG-82 15:24:34 RGG
UASM    00.10.00                  0008 - 01
```

```
Proprietary information of Data General Corporation

--0012--LJMP:
--0012--XJMP:
--0012--EJMP:
--0012--JMP:    COMPLETE_JUMP,  GOTO WAIT;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL  ST  CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          LEAP        WAIT                             W

--DFVs:   addr is WAIT (0FFF)
  216B
  217B
  218B          %********
  219B          %  Jump Subroutine Instructions:  <L X E >JSR
  220B          %
  221B          %  Read PCN (Return PC) into PDR;  Complete IPST.
  222B          %
  223B          %  Move PDR to AC3, AG3; and IPOP.
  224B          %
  225B          %********
  226B
--0013--LJSR:
--0013--XJSR:
--0013--EJSR:
--0013--JSR:    COMPLETE_JUMP,  PDR == RETURN_PC;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL  ST  CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE                                  W      PCN

--DFVs:
  231B
--0014--        IY = AG(AG3) = ALU(AC3) == PDR AND A(M1),
  233B                ATTEMPT_NEXT_EFA,  RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL  ST  CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                AG3 D  EFA M  S@     IY                     M1  AC3 PD AD AND PASS Y

--DFVs:
  •
  •
  •
SAMPLES         Instruction Set Microcode     Rev  1        09-DEC-82 10:47:39 RGG
MEM             Source File                   Cycle 1       18-AUG-82 15:24:34 RGG
UASM    00.10,00                  0009 - 01
```

```
Proprietary information of Data General Corporation

  234B  .EJECT;
  235B  .FT 1 "ALC              Source File                   Cycle 1        18-AUG-82 15:24:34 RGG"
  236B  ;
  237B
  238B  /*---------------------------------------------------------------------+
  239B* |                                                                      |
  240B* |       Some Nova ALC instructions illustrate the use of the ALU       |
  241B* |       for simple arithmetic.  The shift operation is used along      |
  242B* |       with the ALC opcode to provide the decode address.  Carry,     |
  243B* |       no-load and skip options are accelerated with hardware.        |
  244B* |                                                                      |
  245B* +---------------------------------------------------------------------*/
  246B
  247B
  248B          %*********
  249B          %  NOVA Arithmetic and Logical Instructions
  250B          %
  251B          %       EXECUTION TIME:         1 cycle no skip
  252B          %                               2 cycles skip, no EFA required
  253B          %
  254B          %  Perform ALU operation; then Pass, Shift, or Swap; Write result to AG
  255B          %       and ALU AC pointed to by DES.  Enable ALC skip and IPOP.
  256B          %*******
  257B
--0015--ADD:    IY = AG(DES) = ALU(DES) == B(DES) + A(SRC),
  259B                  CARRY == ALC_CRY,  SKIP_ON ALC_RESULT,
  260B                  ATTEMPT_NEXT_EFA,  RETURN;
         OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         CRTN  TRUE                DES D  EFA M  S@    IY      XZ ALC           SRC DES BR AD ADD PASS Y

--DFVs:
  261B
--0016--INC:    IY = AG(DES) = ALU(DES) == ZERO +1+ A(SRC),
  263B                  CARRY == ALC_CRY,  SKIP_ON ALC_RESULT,
  264B                  ATTEMPT_NEXT_EFA,  RETURN;
         OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         CRTN  TRUE                DES D  EFA M  S@    IY      XZ ALC           SRC DES ZR AD CAD PASS Y

--DFVs:
  265B
--0017--SUBL:   IY = AG(DES) = ALU(DES) == BIT_SHIFT_LEFT( B(DES) - A(SRC) ),
  267B                  CARRY == ALC_CRY,  SKIP_ON ALC_RESULT,
  268B                  ATTEMPT_NEXT_EFA,  RETURN;
         OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         CRTN  TRUE                DES D  EFA M  S@    IY      XZ ALC           SRC DES BR AD CSR BLO  Y

--DFVs:
  269B




SAMPLES         Instruction Set Microcode      Rev  1        09-DEC-82 10:47:39 RGG
ALC             Source File                    Cycle 1       18-AUG-82 15:24:34 RGG
UASM     00.10.00                  0010 - 01
```

```
Proprietary information of Data General Corporation

  270B  .EJECT;
  271B  .FT 1 "IMMEDIATE       Source File                   Cycle 1        18-AUG-82 15:24:34 RGG"
  272B  ;
  273B
  274B  /*-----------------------------------------------------------------------+
  275B* |                                                                        |
  276B* |       A Load Effective Address instruction is nothing more             |
  277B* |       than an aborted memory reference.  The final contents of         |
  278B* |       the Logical Address Register are loaded, via the CPD bus         |
  279B* |       and ALU, into the required registers.                            |
  280B* |                                                                        |
  281B* |       The architecture specifies that the effective address is         |
  282B* |       checked for a ring crossing error.  This check will not          |
  283B* |       be performed by hardware because the memory operation            |
  284B* |       used to generate the address is aborted.  A micro test           |
  285B* |       is used to check validity.                                       |
  286B* |                                                                        |
  287B* |       This example also shows the use of a conditional IPOP.  A        |
  288B* |       memory abort operation is recommended following the failure      |
  289B* |       of a conditional IPOP.                                           |
  290B* |                                                                        |
  291B* +-----------------------------------------------------------------------*/
  292B
  293B
  294B          %===============================================================
  295B          %       Load Effective Address:  LEF, ELEF, XLEF, LLEF
  296B          %
  297B          %  Load LAR into the AG and ALU DES registers. Abort Memory. If RMAX is
  298B          %       violated, then go to the RMAX Protection routine, else IPOP.
  299B          %  Load the RMAX fault code into GR0 and faulting address into
  300B          %       AR5 and go to the Protection routine.
  301B          %===============================================================
  302B
--0018--LLEF:
--0018--XLEF:
--0018--ELEF:
--0018--LEF:    IY = AG(DES) = ALU(DES) == SPAD(BIT0) NOT_AND LAR,
  307B              EXTEND_MICRO_CYCLE,
  308B              ATTEMPT_NEXT_EFA,   ABORT_MEMORY,
  309B              IF NOT INWARD_REFERENCE RETURN_ELSE_GOTO RMAX_PROTECTION;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN NRMAX  RMAX_PRO       DES D  EFA M  S@ A  IY  LAR GN XTND             DES SC CD ANC PASS Y  BIT0

--DFVs:   addr is RMAX_PROTECTION (0019)  const is BIT0 (0000)
  310B
--0019--RMAX_PROTECTION:
  312B          ABORT_MEMORY,
  313B          IY = ALU(GR0) == CNST(PRT_RMX) OR CPD_ZERO,
  314B              CPM = AG(AR5) == ALU(DES), GOTO PROTECTION_FAULT;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          LEAP        PROTECTI       AR5        M     A  IA  N                   DES GR0 CN CD OR  PASS Y  PRT_RMX

--DFVs:   addr is PROTECTION_FAULT (0002 *EXT*)  const is PRT_RMX (0004)
  315B
  316B


SAMPLES         Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
IMMEDIATE       Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM    00.10.00                   0011 - 01
```

```
Proprietary information of Data General Corporation

       /*---------------------------------------------------------------------+
       |                                                                      |
       |       Instructions which load immediate data from the instruction    |
       |       stream use an approach similar to the LEF instructions. In     |
       |       their case, the immediate data has been loaded into the LAR    |
       |       by the IP as specified by decode information, but no memory    |
       |       reference has been initiated.                                  |
       |                                                                      |
       +---------------------------------------------------------------------*/


       %********
       %  Long Add Immediates:  <W N >ADDI
       %       DES  +  (Displacement)  ->  DES       (Displacement is in LAR)
       %       The W and N types load overflow into OVR and CRY<0 16> into CARRY.
       %********

--001A--ADDI:   IY = ALU(DES) = AG(DES) == B(DES) + LAR,  ATTEMPT_NEXT_EFA, RETURN;
         OP     TSEL  ADDRESS  D AA  AB  AG AOP AL  ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         CRTN   TRUE                 DES D  EFA M  S@    IY  LAR                      DES BR CD ADD PASS Y

--DFVs:

--001B--WADDI:  % For the 32-bit Immediate
--001B--WNADI:  % For the 16-bit Immediate
--001B--NADDI:  IY = ALU(DES) = AG(DES) == A(DES) + LAR,
                        CARRY == ALU_CRY,  UPDATE_OVR,
                        ATTEMPT_NEXT_EFA,  RETURN;
         OP     TSEL  ADDRESS  D AA  AB  AG AOP AL  ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         CRTN   TRUE                 DES D  EFA M  S@    IY  LAR XZ LOVC          DES DES    CA ADD PASS Y

--DFVs:


   •
   •
   •

SAMPLES          Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
IMMEDIATE        Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM    00.10.00                  0012 - 01
```

```
Proprietary information of Data General Corporation

       /*---------------------------------------------------------------------+
       |                                                                      |
       |       A short immediate field is derived from the source             |
       |       accumulator bit field of the macro instruction.  Actual        |
       |       values are 0 through 3, but implied values are 1 through 4.    |
       |       The following instructions read an operand from a memory       |
       |       location, subtract the implied immediate data from it, and     |
       |       store the result back in the same memory location.             |
       |                                                                      |
       +---------------------------------------------------------------------*/


       %********
       %  Short Subtract Immediate from Memory: <L X><W N >SBI
       %
       %       MEM  -  ([ACS] + 1)  ->  MEM
       %
       %  Read Memory operand into ALU (GR0).  Start same address to write back.
       %       Go to XLSBI to complete the operation and perform the write.
       %********

--001C--XWSBI:
--001C--LWSBI:  CPM = ALU(GR0) == MEM_READ,
                        START AY == PASS(LAST_LA) FOR WRITE_DOUBLE,
                        GOTO XLSBI;
         OP     TSEL  ADDRESS  D AA  AB  AG AOP AL  ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         LEAP         XLSBI             L  PSB   WD R  MM                         GR0                   M

--DFVs:   addr is XLSBI (001E)
--001D--XNSBI:
--001D--LNSBI:  CPM = ALU(GR0) == MEM_READ,
                        START AY == PASS(LAST_LA) FOR WRITE_WORD,
                        GOTO XLSBI;
         OP     TSEL  ADDRESS  D AA  AB  AG AOP AL  ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         LEAP         XLSBI             L  PSB   WW R  MM                         GR0                   M

--DFVs:   addr is XLSBI (001E)

--001E--XLSBI:  IY = MEM_WRITE == A(GR0) -1- SRC_POINTER,
                        CARRY == ALU_CRY,  UPDATE_OVR,
                        ATTEMPT_NEXT_EFA, RETURN;
         OP     TSEL  ADDRESS  D AA  AB  AG AOP AL  ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
         CRTN   TRUE                    D  EFA    S@ W  IY      XZ LOVC         GR0     AS DA SMR PASS

--DFVs:
   •
   •
   •


SAMPLES          Instruction Set Microcode     Rev   1        09-DEC-82 10:47:39 RGG
IMMEDIATE        Source File                   Cycle 1        18-AUG-82 15:24:34 RGG
UASM    00.10.00                  0013 - 01
```

```
Proprietary information of Data General Corporation

  376B  .EJECT;
  377B  .FT 1 "BYTE           Source File                Cycle 1        18-AUG-82 15:24:34 RGG"
  378B  ;
  379B
  380B  /*----------------------------------------------------------------------+
  381B* |                                                                       |
  382B* |       These selections from the byte microcode show the use of        |
  383B* |       a conditional subroutine call, a memory start using the         |
  384B* |       address generator, and a memory abort.  A copy of the PC        |
  385B* |       of execution + 1 is moved to the address generator by first     |
  386B* |       loading PCX into PDR, and then subtracting -1 from it and       |
  387B* |       loading the result in the AG register file via the CPM bus.     |
  388B* |                                                                       |
  389B* +----------------------------------------------------------------------*/
  390B
  391B
  392B  %*******************************************************************
  393B  %
  394B  %                   Byte   EFA    Instructions
  395B  %                   --------------------------
  396B  %
  397B  %       EFA calculations for a Byte address cannot be completely performed
  398B  %       by the hardware.  The PC relative index case cannot be performed
  399B  %       since the Displacement is a byte address and the PC is a word address.
  400B  %       The AG converts the byte displacement and performs the other indexing.
  401B  %
  402B  %*******************************************************************
  403B
  404B
  405B          %******
  406B          %                    Subroutines to perform
  407B          %              READ/WRITE PC Relative Byte Addresses
  408B          %              -------------------------------------
  409B          %       PC Relative Addressing must be handled separately
  410B          %       since the IP cannot align a Byte displacement.
  411B          %
  412B          %  Abort the previous start and move the PC of the instruction
  413B          %       plus 1 (PC of the DISP) to the AG (AR0).
  414B          %  Form the PC relative address by adding the Displacement (LAST_LA)
  415B          %       to the PC (AR0) and start for the Byte Read/Write.  Word
  416B          %       addressing must be forced since the addresses have already
  417B          %       been aligned to word addresses.  Return to the caller.
  418B          %******
  419B
--001F--READ_PC_BYTE:
  421B         IY = AG(AR0) == PDR - A(M1),  ABORT_MEMORY;
        OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CJMP   FALSE                AR0        M     A  IY                      M1      PD AD CSR PASS

--DFVs:
  422B
--0020--       START AY == LAST_LA + A(AR0) WITH_WORD_ADDRESSING FOR READ_BYTE,
  424B                 RETURN;
        OP     TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CRTN   TRUE             AR0     L  ADD    RB            AT WORD

--DFVs:
  425B
SAMPLES         Instruction Set Microcode      Rev  1        09-DEC-82 10:47:39 RGG
BYTE            Source File                    Cycle 1       18-AUG-82 15:24:34 RGG
UASM     00.10.00                 0014 - 01
```

```
Proprietary information of Data General Corporation

  426B
--0021--WRITE_PC_BYTE:
  428B          IY = AG(AR0) == PDR - A(M1),  ABORT_MEMORY;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL  ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJMP  FALSE              AR0        M      A  IY                      M1      PD AD CSR PASS

--DFVs:
  429B
--0022--        START AY == LAST_LA + A(AR0) WITH_WORD_ADDRESSING FOR WRITE_BYTE,
  431B                  RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL  ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE             AR0     L  ADD     WB            AT WORD

--DFVs:
  432B
  433B
  434B  %********
  435B  %       Load Byte:   LDB, WLDB
  436B  %
  437B  %  Start the byte address in the SRC accumulator for a Read byte.
  438B  %       Finish the operation at LDA.
  439B  %********
  440B
--0023--WLDB:
--0023--LDB:    START AY == PASS(B(SRC)) FOR READ_BYTE,  GOTO LDA;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL  ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          LEAP        LDA            SRC B  PSB     RB

--DFVs:   addr is LDA (0010)
  443B
  444B
  445B  %********
  446B  %       Indexed Load/Store Byte:   LLDB, XLDB, ELDB
  447B  %
  448B  %  Check for PC relative Addressing before attempting complete of operation.
  449B  %       Get PC of instruction into PDR in case of PC relative addressing.
  450B  %       Subroutine to Start the correct address if index was PC relative.
  451B  %  Complete the operation and IPOP:
  452B  %       LDB:  Store the read byte into the AG and the ALU DES.
  453B  %********
  454B
--0024--LLDB:
--0024--XLDB:
--0024--ELDB:   PDR == PC_OF_EXECUTION,  IF PC_REL_INDEX CALL READ_PC_BYTE;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL  ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CJSR  IXPC  READ_PC_                            PCX

--DFVs:   addr is READ_PC_BYTE (001F)
  458B
--0025--        CPM = AG(DES) = ALU(DES) == MEM_READ,  ATTEMPT_NEXT_EFA,  RETURN;
          OP    TSEL  ADDRESS  D AA  AB  AG AOP AL  ST CM CPM CPD RM RO   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
          CRTN  TRUE                 DES D  EFA M   S@ R  MM                        DES                  M

--DFVs:
  460B
  461B


SAMPLES         Instruction Set Microcode     Rev   1       09-DEC-82 10:47:39 RGG
BYTE            Source File                   Cycle 1       18-AUG-82 15:24:34 RGG
UASM    00.10.00                   0015 - 01
```

```
Proprietary information of Data General Corporation

  462B  .EJECT;
  463B  .FT 1 "XCT            Source File                  Cycle 1        18-AUG-82 15:24:34 RGG"
  464B  ;
  465B
  466B  /*-----------------------------------------------------------------------+
  467B* |                                                                        |
  468B* |       The code for the XCT instruction address SPAD with a              |
  469B* |       constant, uses the flags to control sequencing and does           |
  470B* |       a word zero extend with the hex shifter.                          |
  471B* |                                                                        |
  472B* +-----------------------------------------------------------------------*/
  473B
  474B
  475B  /*-------------------------------------------------------------+
  476B* |                                                              |
  477B* |              XCT  Execute an AC's contents                   |
  478B* |                                                              |
  479B* |       DES contains the opcode to be executed                 |
  480B* |                                                              |
  481B* +-------------------------------------------------------------*/
  482B
  483B  %       Enter here for ordinary XCT.  If restarting or resuming,
  484B  %       XCTed opcode must still be in DES.  Bit 0 of double word saved in
  485B  %       SPAD is cleared to indicate to interrupt handlers that saving
  486B  %       XCT opcode on wide stack is not required.  Start execute.
  487B
--0026--EXECUTE:
--0026--XCT:    GOTO &;  % Wait for XCTED_INSTRUCTION test to setup.
        OP      TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        LEAP          &

--DFVs:   addr is & (0027)
  490B
--0027--        ID == SPAD( XCTOP ),  IF NOT XCTED_INSTRUCTION GOTO NORMAL_XCT;
        OP      TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CJMP NXCTF  NORMAL_X                                                         SC                    XCTOP

--DFVs:   addr is NORMAL_XCT (0029)  const is XCTOP (00C3)
  492B
  493B     %    If XCT was executed by a PBX, then the XCT should set Bit0 of
  494B     %    XCTOP since it was virtually executed by the PBX.
  495B
--0028--        IF ID_SIGN=1 GOTO EXECUTE_PBX;
        OP      TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CJMP    DSGN  EXECUTE_

--DFVs:   addr is EXECUTE_PBX (002D)
  497B
--0029--NORMAL_XCT:
  499B          IY = SPAD( XCTOP ) == WORD_ZERO_EXTEND (A(DES)),  START_EXECUTE;
        OP      TSEL  ADDRESS  D AA  AB  AG AOP AL ST CM CPM CPD RM R0   R1  R2  IA  IB  ID RS IOP IY   IL FR FS FOP W FCW FL FRG X
        CJMP    FALSE           AG0 AG0 B  SUB    WW             AT CM0      WC  DES         AD      WZX  NY XCTOP

--DFVs:   const is XCTOP (00C3)
  500B
  501B  %       Send instruction to the IP via the Cache.
  502B
--002A--        CPM = EXECUTE_DATA == ALU(DES),
SAMPLES         Instruction Set Microcode     Rev   1       09-DEC-82 10:47:39 RGG
XCT             Source File                   Cycle 1       18-AUG-82 15:24:34 RGG
UASM     00.10.00                0016 - 01
```

End of Chapter

Appendix A
Page Faults

Page faults occur during Logical Address Translation (LAT). There are two possible causes for page faults:

- A referenced page is not in physical memory. This is indicated by the page table entry RESIDENT bit (bit 1) being 0.
- A referenced page requires a two-level page table when only a one-level page table has been defined. This is indicated when a Segment Base Register (SBR) LENGTH bit (bit 1) is 0, but the logical address bits 4-12 are not zero.

Page Table Entry:

0	1	2 4	5 12	13 31
V	R	ACC	RESERVED	PHYSICAL ADDRESS

Segment Base Register:

0	1	2	3	4 31
V	L	LEF	I/O	PHYSICAL ADDRESS

The LAT trap microcode must determine that the referenced address is nonresident, and then transfer control to the page-fault microcode.

The purpose of the page fault is to pass control to the operating system. The operating system may then bring the nonresident page into physical memory. Before the operating system receives control, microcode pushes a context block to preserve the current state of the hardware.

There are four types of context blocks, only two of which are now implemented. The type of block depends on the instruction executing when the page fault occurred. The following list shows the page-fault types.

1) Simple

2) Resumable

3) Floating Point (not implemented)

4) Decimal (not implemented)

Microcode pushes a *simple* context block when the macroinstruction will be restarted after its referenced page is brought into physical memory. A macroinstruction is restartable if the current microcycle is the first cycle of an instruction (i.e., one cycle after an IPOP) or inside an EFA. Restart may also be forced by setting SPAD location PF_RESTART to a nonzero value. An example of an instruction that can only restart is LDA. LDA references memory only during its first cycle.

Microcode pushes a *resumable* context block when a macroinstruction will resume with the microinstruction that was executing when the LAT trap occurred. Because most of the machine-visible state must be saved, a resumable context block is longer than a simple context block. Most instructions that reference memory (except for EFA references) are resumable.

Figure A-1 shows the context blocks.

WORD	CONTENTS			
0.	PSR, XCTOP		Simple Context Block	Resumable Context Block
2.	AC0			
4.	AC1			
6.	AC2			
8.	AC3			
10.	Carry, PCX			
12.	PC			
14.	LAR			
16.	<0-15>: ustack Size, <16-31>:Context Block TYPE			
18.	IP_STATE word (ION, XCTFLG, LPCX)			
20.	ATU_STATE word (See "ATU STATE" in Chapter 2)			
22.	ALU_STATE word (SPAR, ACD, ACS)			
24.	MSEQ_STATE word (TOS, FLGs, DSR)			
26.	PDR	ALU State		
28	TREG			
30.	GR0			
32.	GR1			
34.	GR2			
36.	GR3			
38.	GR4			
40.	GR5			
42.	GR6			
44.	GR7			
46	AG0	AG State		
48.	AG1			
50.	AG2			
52.	AG3			
54.	AR0			
56.	AR1			
58.	AR2			
60.	AR3			
62.	AR4			
64.	AR5			
66.	AR6			
68.	AR7			
70.	FG0 (64 bits)			
74.	Microstack Contents (variable-length; see word 16)			
	.			
	.			

Figure A-1. Context Blocks

After the context block is pushed, a fault code is returned in AC1:

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

Note: The multiple ERCC fault is currently a nonrecoverable condition.

End of Appendix

Appendix B
CPD Bus Legal Path Analysis

The following figure illustrates legal combinations of sources and destinations for the CPD Bus.

Typical Destinations

CPD Source	Into ALU*	PDR DSP CDW ATS/REF ATS/MOD	uSTK IPS ATS/ST	FLAGS**	IOC
TREG USS ZER/N LAR ATD	Type 1	Yes	Yes	Yes	Yes
CIR CDR ATS/STS***	Type 2	Yes	Yes	Yes	Yes
AGA	Type 3	Yes	Yes	Yes	Yes
IOC IPS	Type 4	Yes	Yes	Yes	Yes
ATS/REF*** ATS/MOD*** PCN PCX PC	Type 5	Yes Yes	Yes Yes	No No	No No

* *See the ALU path table (Appendix D) for these setups.*

** *FLAGS can be placed in the uSTK category, if they are guaranteed not to be tested next cycle.*

*** *The ATS source contains two different types of data: ATS/STS is the ATU Status bits and ATS/(REF MOD) is the Reference and Modify bits.*

End of Appendix

Appendix C
CPM Bus Legal Path Analysis

The following figure illustrates legal combinations of sources and destinations for the CPM Bus.

CPM Source	Typical Destinations: SPAD	ALU AG FPU CACHE	FPSR	TREG
ALU IY	*	*	*	*
ALU IA	Yes	Yes	Yes	Yes
FPU	Yes	Yes	Yes	Yes
FPSR	Yes	Yes	Yes	Yes
FP STATE	Yes	Yes	Yes	Yes
ALL_ONES	Yes	Yes	Yes	Yes
AG	Yes	Yes	Yes	Yes
MEMORY	Yes	Yes	Yes	Yes

** See ALU path analysis (Appendix D).*

End of Appendix

Appendix D
ALU Source and Destination Paths

The following table illustrates legal sources and destinations for the integer ALU. Sources are classified as follows:

Type	Sources
1	All ID Bus sources (except SPAD), ALU Register File, TREG, LAR, ZER/N, USS, ATD
2	CIR, CDR, ATS (Except REF and MOD bits)
3	AGA
4	IOC, SPAD, IPS
5	Any PC type, ATS (REF and MOD bits)*

	Destinations				
Sources	**AG RF FP RF FPSR uSTK IPS**	**SPAD ATS1**	**Cache**	**PDR TREG DSP CDW ATS2**	**ALU SPAR**
Arithmetics or Hex Shifts					
Type 1 (LAR)	Yes	Yes	Yes	Yes	Yes
Type 2 (CIR)	No	No	No	No	Yes
Type 3 (AG)	No	No	No	No	Yes
Type 4 (SPAD)	No	No	No	No	Yes
Logical					
Type 1 (LAR)					
Type 2 (CIR)	Yes	Yes	Yes	Yes	Yes
Type 3 (AG)	Yes	Yes	Yes	Yes	Yes
Type 4 (SPAD)	No	Yes	Yes	Yes	Yes
	No	No	Yes	Yes	Yes
EDIT Translate					
AREG only	Yes	Yes	Yes	Yes	Yes

* *Types 5 and 6 will not make any destination except PDR class CPD destinations without an EXTEND.*

Note: All types will make any desired destination, using any desired operation, if they are extended. Note that SPAD cannot be read and written in the same cycle under any circumstances.

End of Appendix

Appendix E
Page Zero Locations

By convention, an MV machine has a set of reserved storage locations in page 0 of segment 0 that are used by fault routines. This appendix lists the definitions for those locations.

```
.RADIX  16;
/* ----------------------------------------------------------------------+
|                                                                        |
|       PAGE ZERO LOCATION DEFINITIONS                                   |
|                                                                        |
|  The size of the pointer and whether it is indirectable is indicated   |
|       by a 16, 32, 16I, or 32I at the end of the comment.              |
|                                                                        |
+-------------------------------------------------------------------- */

.DEFINE  INTR_LEV       = 00;  % Current Level of Interrupt processing (A count)
.DEFINE  INTR_HDLR      = 01;  % Interrupt Handler address - 16I
.DEFINE  INTR_RTN       = 02;  % Interrupt Return address - 32
.DEFINE  SYC_HDLR       = 02;  % System Call Handler address - 16I (if ATU off)

.DEFINE  VSP            = 04;  % Vector Stack Pointer - 16
.DEFINE  VSL            = 06;  % Vector Stack Limit - 16
.DEFINE  VSF            = 07;  % Vector Stack Fault Handler address - 16I

.DEFINE  BKP_HDLR       = 08;  % Breakpoint Handler address - 32I
.DEFINE  WXOP_TBL       = 0A;  % Wide XOP Table base address - 32

.DEFINE  WSF            = 0C;  % Wide Stack Fault Handler address - 16I
.DEFINE  UIT_HDLR       = 0D;  % Unimplemented Instruction Handler address - 16
.DEFINE  WFP            = 10;  % Wide Stack Frame Pointer - 32
.DEFINE  WSP            = 12;  % Wide Stack Pointer - 32
.DEFINE  WSL            = 14;  % Wide Stack Limit - 32
.DEFINE  WSB            = 16;  % Wide Stack Base - 32

.DEFINE  PGF_HDLR       = 18;  % Page Fault Handler address - 32I
.DEFINE  CNTX_BLK       = 1A;  % Context Block Area Pointer - 32I

.DEFINE  GATE_TBL       = 1C;  % Gate Array base address - 32

.DEFINE  PRT_HDLR       = 1E;  % Protection Fault Handler address - 16I
.DEFINE  FIX_HDLR       = 1F;  % Fix Point Fault Handler address - 16I

.DEFINE  NSP            = 20;  % Narrow Stack Pointer - 16
.DEFINE  NFP            = 21;  % Narrow Frame Pointer - 16
.DEFINE  NSL            = 22;  % Narrow Stack Limit - 16
.DEFINE  NSF            = 23;  % Narrow Stack Fault Handler address - 16I

.DEFINE  NXOP_TBL       = 24;  % Narrow XOP Table base address - 16

.DEFINE  FLT_HDLR       = 25;  % Floating Point Fault Handler address - 16I
.DEFINE  COM_HDLR       = 26;  % Commercial Fault Handler address - 16I
.DEFINE  DERR_HDLR      = 27;  % Diagnostic Error Fault handler address - 16I

/* -------------------------------------------------------------------- */
```

End of Appendix

Appendix F
Fault Codes

This appendix lists the MV/10000 fault codes. These codes should be placed in macroaccumulator 1 after a fault occurs.

```
.RADIX  16;
%********
%   DEFINITION OF FAULT CODES
%       Protection Faults:
.DEFINE  PRT_RD      = 00;  % Read
.DEFINE  PRT_WR      = 01;  % Write
.DEFINE  PRT_EX      = 02;  % Execute
.DEFINE  PRT_VLD     = 03;  % Validity of SBR or PTE
.DEFINE  PRT_RMX     = 04;  % Inward Memory Reference (Ring Maximization)
.DEFINE  PRT_DFR     = 05;  % Defer level exceeded
.DEFINE  PRT_GATE    = 06;  % Illegal Gate
.DEFINE  PRT_CALL    = 07;  % Outward Call
.DEFINE  PRT_RTN     = 08;  % Inward Return
.DEFINE  PRT_PRV     = 09;  % Privileged Instruction
.DEFINE  PRT_IO      = 0A;  % IO Protection
.DEFINE  PRT_CB      = 0B;  % Invalid Context Block type
.DEFINE  PRT_IMI     = 0C;  % Invalid microinterrupt return block
%       Page Faults:
.DEFINE  PAG_PTD     = 01;  % Page Table Depth exceeded
.DEFINE  PAG_PTE     = 02;  % Nonresident Page Table
.DEFINE  PAG_REF     = 04;  % Nonresident Reference (object) page
%       Wide Stack Faults:
.DEFINE  STK_OVF     = 00;  % Any overflow except for WMSP,wide saves,or CALL
.DEFINE  STK_ABT     = 01;  % Underflow or overflow caused by WMSP,wide saves
.DEFINE  STK_ARG     = 02;  % Too many arguments on a CALL
.DEFINE  STK_UNF     = 03;  % Underflow
.DEFINE  STK_FOV     = 04;  % Overflow while pushing return block for
                            % fault or interrupt
%********

%********
%       Messages to the SCP:
.DEFINE  SCP_HALT    = 00;  % The machine has entered the HALT loop
.DEFINE  SCP_IPRT    = 01;  % Infinite Protection Fault loop detected
.DEFINE  SCP_IPGF    = 02;  % Infinite Page Fault loop detected
.DEFINE  SCP_LCS     = 03;  % Load Control Store request
.DEFINE  SCP_IORST   = 04;  % An IORST is being performed
.DEFINE  SCP_INTR    = 05;  % A trap to the SCP has occured
%********
```

End of Appendix

Appendix G
Exceptions

```
.RADIX  16;
%********
%       Definition of TRAP addresses:
.DEFINE  TRAP_PROT       = 04;     % Protection
.DEFINE  TRAP_LAT        = 0C;     % Long Address Translation
.DEFINE  TRAP_CBXR       = 14;     % Read Cache Block Crossing
.DEFINE  TRAP_CBXW       = 1C;     % Write Cache Block Crossing
.DEFINE  TRAP_FXERR      = 24;     % Fix Point Error (Overflow)
.DEFINE  TRAP_FLERR      = 2C;     % Floating Point Error
.DEFINE  TRAP_UNUSED     = 34;     % Unused
.DEFINE  TRAP_SCP        = 3C;     % System Console Processer
%********

%********
%       Definition of IP Alternate Addresses:
.DEFINE  IP_WAIT         = 0FFF;  % Wait for IP
.DEFINE  IP_USKP         = 0FFE;  % Microcode-forced Skip
.DEFINE  IP_HLT          = 0FFC;  % Halt
.DEFINE  IP_IFLUSH       = 0FFA;  % IP Pipeline Flush
.DEFINE  IP_INTRT        = 0FF8;  % Interrupt
.DEFINE  IP_@JMP         = 0FF4;  % Indirect Jump Reference
.DEFINE  IP_@MRF         = 0FF2;  % Indirect Memory Reference
.DEFINE  IP_ICAT         = 0FF0;  % ICache Translation
%********
```

End of Appendix

Appendix H
Scratch Pad Addresses

The scratch pad in the integer ALU contains reserved locations that are used as save areas or to hold constants used by microroutines. This appendix lists the SPAD reserved locations.

```
.FT 3 SCRATCH PAD;
%*********************************************************************
%              SCRATCH PAD ADDRESSES
%*********************************************************************
.RADIX 10;
.WORD 32;
.WIDTH 4;
.LENGTH 256;
.DESTINATION  SCRATCH_PAD;
.SFIELD spadcontenth (0-15);
.SFIELD spadcontentl(16-31);
.DFIELD (spadcontenth), (spadcontentl);
.LIST SAME, 16;
.RADIX 16;
% Used in BTZ,BTO,SNB,SZBO,SZB
% LOCed to hardware address mux for WSKBO, WSKBZ
BIT0:     8000  0000;
BIT1:     4000  0000;
BIT2:     2000  0000;
BIT3:     1000  0000;
BIT4:     0800  0000;
BIT5:     0400  0000;
BIT6:     0200  0000;
BIT7:     0100  0000;
BIT8:     0080  0000;
BIT9:     0040  0000;
BIT10:    0020  0000;
BIT11:    0010  0000;
BIT12:    0008  0000;
BIT13:    0004  0000;
BIT14:    0002  0000;
BIT15:    0001  0000;
BIT16:    0000  8000;
BIT17:    0000  4000;
BIT18:    0000  2000;
BIT19:    0000  1000;
BIT20:    0000  0800;
BIT21:    0000  0400;
BIT22:    0000  0200;
BIT23:    0000  0100;
BIT24:    0000  0080;
BIT25:    0000  0040;
```

```
BIT26:    0000  0020;
BIT27:    0000  0010;
BIT28:    0000  0008;
BIT29:    0000  0004;
BIT30:    0000  0002;
BIT31:    0000  0001;
% Table of mask constants for WASH
WASHM0:  08000  0000;
WASHM1:  0C000  0000;
BRNGMSK:        % Byte ring mask
WASHM2:  0E000  0000;
WASHM3:  0F000  0000;
WASHM4:  0F800  0000;
WASHM5:  0FC00  0000;
WASHM6:  0FE00  0000;
WASHM7:  0FF00  0000;
WASHM8:  0FF80  0000;
WASHM9:  0FFC0  0000;
WASHM10: 0FFE0  0000;
WASHM11: 0FFF0  0000;
WASHM12: 0FFF8  0000;
WASHM13: 0FFFC  0000;
WASHM14: 0FFFE  0000;
WDMSK:
WASHM15: 0FFFF  0000;
WASHM16: 0FFFF 08000;
WASHM17: 0FFFF 0C000;
WASHM18: 0FFFF 0E000;
WASHM19: 0FFFF 0F000;
WASHM20: 0FFFF 0F800;
WASHM21: 0FFFF 0FC00;
WASHM22: 0FFFF 0FE00;
WASHM23: 0FFFF 0FF00;
WASHM24: 0FFFF 0FF80;
WASHM25: 0FFFF 0FFC0;
WASHM26: 0FFFF 0FFE0;
WASHM27: 0FFFF 0FFF0;
WASHM28: 0FFFF 0FFF8;
WASHM29: 0FFFF 0FFFC;
WASHM30: 0FFFF 0FFFE;
WASHM31: 0FFFF 0FFFF;
```

```
/*  This group of elements is location-locked so that it will not be loaded
    and verified by SCP/OS.  If the size or location of this group must
    be changed, the MAKE_WIDGEON macro must be changed.    */
%         Save locations for console primitives.
%         Must be aligned for SPAD table offset computations.
AC0SV:    0000 0000;     % Save area for ALU register file.
AC1SV:    0000 0000;
AC2SV:    0000 0000;
AC3SV:    0000 0000;
FPSV:     0000 0000;
SLSV:     0000 0000;
SBSV:     0000 0000;
M1SV:    0FFFF 0FFFF;
GR0SV:    0000 0000;
GR1SV:    0000 0000;
GR2SV:    0000 0000;
GR3SV:    0000 0000;
GR4SV:    0000 0000;
GR5SV:    0000 0000;
GR6SV:    0000 0000;
GR7SV:    0000 0000;
AG0SV:    0000 0000;     % Save area for AG register file.
AG1SV:    0000 0000;
AG2SV:    0000 0000;
AG3SV:    0000 0000;
SPSV:     0000 0000;
ONESV:    0000 0001;
TWOSV:    0000 0002;
LATSV:    0000 0000;
AR0SV:    0000 0000;
AR1SV:    0000 0000;
AR2SV:    0000 0000;
AR3SV:    0000 0000;
AR4SV:    0000 0000;
AR5SV:    0000 0000;
AR6SV:    0000 0000;
AR7SV:    0000 0000;
SBR0CP:   0000 0000;     % Shadow copies of SBRs.  Needed for SCP examine SBR
SBR1CP:   0000 0000;     % primitive.
SBR2CP:   0000 0000;
SBR3CP:   0000 0000;
SBR4CP:   0000 0000;
SBR5CP:   0000 0000;
SBR6CP:   0000 0000;
SBR7CP:   0000 0000;
PDRSV:    0000 0000;     % PDR
TRGSV:    0000 0000;     % TREG
LLASV:    0000 0000;     % Last LA
FLDSV:    0000 0000;     % Flags and dispatch register   USEQ STATE
SDSSV:    0000 0000;     % SPAR and DES/SRC pointers     ALU  STATE
ATUSV:    0000 0000;     % MemStart, RefMod, Faultcode   ATU  STATE
IPSSV:    0000 0000;     % Reserved for IP State word
PCXSV:    0000 0000;     % Reserved for PC of executing instruction
/*      Do not change these SCP save locations without fixing
                  the MAKE_WIDGEON macro.                      */
```

```
%******************
%    Constants
%******************
% Table of constants for I/O
SNIO:      0001  0000;
SDIA:      0001  0100;
SDOA:      0001  0200;
SDIB:      0001  0300;
SDOB:      0001  0400;
SDIC:      0001  0500;
SDOC:      0001  0600;
SSKP:      0001  0700;
IOCMD:     0003  0000; % I/O Transfer Command Constant used by DIX and DOX
IOCMD1:    0001  8000; % I/O Transfer Command Constant for CIO (Type 1)
DESMSK:    0000  3000; % Used by PIO to mask the DES field
PORT_MSK:0000    7000; % Used by PIO to mask the Port field
HI9:       0000  9000; % Mask used by the interrupt code for Type 1
IO_OP:     0000  6000; % Used by PIO to construct a NOVA I/O opcode
XVCT_OP:0FFFF 0C709; % XVCT's opcode used by interrupt code
INTA_OP: 0001    733F; % Interrupt Acknowledge Opcode used by XVCT
DMSKO_OP:0001    743F; % Device Mask Out Opcode used by XVCT
PMSKO_OP:0001 0FFC1; % Port Mask Out Opcode for use by XVCT
IOCNT:     0000  1000; % I/O Time Out Count
RNGMSK: 07000    0000; % Ring bit Mask
M12:      0FFFF 0FFF4; % CONSTANT -12
CHAR:      2020  2020; % A Double Word of Spaces
EXPM:      7F00  0000; % Mask for floating-point exponent
EXP4:      4400  0000; % Floating-point exponent of value 4
EXP8:      4800  0000; % Floating point exponent of value 8
M280:      0000  0280; % Mask for ATU State word in WDPOP
M680:      0000  0680; % Mask for ATU State word in WDPOP
ZNBIT:     0300  0000; % Mask for Z and N FPSR bits
UOBIT:     6000  0000; % Mask for UNF and OVF FPSR bits
UDBIT:     3000  0000; % Mask for UNF and DVZ FPSR bits
ODBIT:     5000  0000; % Mask for OVF and DVZ FPSR bits
% The following constants are used to convert floating-point
% integers in the range 0 to 10**16-1 into a fraction for
% conversion to decimal format.  If the correct factor is used,
% subsequent multiplications by 10 will yield at the most
% one leading zero.
STITM1:   4019  9999;     % 1/10
          9999  9999;
STITM2:   3F28 0F5C2;     % 1/10**2
          8F5C  28F6;
STITM3:   3E41  8937;     % 1/10**3
          4BC6 0A7F0;
STITM4:   3D68 0DB8B;     % 1/10**4
         0AC71  0CB3;
STITM6:   3C10 0C6F7;     % 1/10**6
         0A0B5 0ED8D;
STITM7:   3B1A 0D7F2;     % 1/10**7
          9ABC 0AF48;
STITM8:   3A2A 0F31D;     % 1/10**8
         0C461  1874;
STITM9:   3944 0B82F;     % 1/10**9
         0A09B  5A53;
STITM10: 386D 0F37F;      % 1/10**10
          675E 0F6EB;
STITM12: 3711   9799;     % 1/10**12
          812D. 0EA11;
STITM13: 361C   25C2;     % 1/10**13
          6849  7682;
STITM14: 352D   0937;     % 1/10**14
          0D42  5736;
STITM15: 3448   0EBE;     % 1/10**15
          7B9D  5857;
```

```
%  LOC'ed so constants can be added without moving the Scratch locations
.LOC  0C0;
%**************************************************
%    Scratch Locations and Register Extensions
%**************************************************
DEFAULT_PORT:
         0000  0000; % Port number to be used for NOVA I/O Opcodes
PORT_NUM:0000  0000; % Port number to be used by the DIX/DOX routines
                     %   It is always reset to DEFAULT_PORT upon completion.
FPPC:    0000  0000; % PC of last floating point instruction with an error.
XCTOP:   0000  0000; % Op code of the last XCT'ed instruction
PROTST:  0000  0000; % Used to detect protection faults within prot. faults
AC_UPDATE: 0000 0000;  % Used by prot. faults
TREGSAVE:  0000 0000; % Temporary register used during the LAT routine
%==================================================
%  Reserved for Page Fault Context Save
PF_RESTART: 0  0000;    % Flag to force instruction restart if Page Faulted.
PF_FLAG: 0000  0000;    % Page Fault Lock set to detect Recursive faults.
PF_CODE: 0000  0000;    % Reason for Page Fault
PF_TYPE: 0000  0000;    % Type of context block to be pushed
PF_TREG: 0000  0000;    % Single Register:  TREG, PDR, and LAR
PF_PDR:  0000  0000;
PF_LAR:  0000  0000;
PF_MSEQ: 0000  0000;    % Microsequencer State word: Fault uPC, FLAGs, and DSP
PF_ALU:  0000  0000;    % ALU state word:  SPAR, ACDR, ACSR
PF_IP:   0000  0000;    % IP state word:  Length and other state.
PF_ATU:  0000  0000;    % ATU state word:  Start type and other state.
PF_AG0:  0000  0000;    % AG Reg File variables:
PF_AG1:  0000  0000;
PF_AG2:  0000  0000;
PF_AG3:  0000  0000;
PF_AR0:  0000  0000;
PF_AR1:  0000  0000;
PF_AR2:  0000  0000;
PF_AR3:  0000  0000;
PF_AR4:  0000  0000;
PF_AR5:  0000  0000;
PF_AR6:  0000  0000;    % Mapped by SRC = E
PF_AR7:  0000  0000;    % Mapped by DES = F
PF_GR0:  0000  0000;    % Holds GR0 to allow calculations on the ALU
%==================================================
.LOC    0FC;
% Locked to last four locations for SCP
MODEL:   0000  MODEL_NUM;
MNREV:   0000  MINOR_REV;
MJREV:   0000  MAJOR_REV;
MEMSZ:   0000  000F;      % Memory size placed here by the SCP
```

End of Appendix

Index

Note: Page numbers in bold type (e.g., **1-5**) indicate definitions of terms or other key information.

AA Bus 2-3, 2-6
ACDR register 2-12
ACSR register 2-12
Address Generator 1-3, 2-31
  ALU 2-36
  buses 2-32
  micro-orders 3-24
Address Generator bus 2-33
Address Generator Bus, sources 2-35
Address Generator register file 2-33
  addressing 2-34
Address Translation Cache 2-37
Address Translation Unit 1-4, 2-36
  diagnostic register 2-41
  dispatch 2-42
  state 2-40
  tests 3-9
AG, see Address Generator
AGB, see Address Generator Bus
ALU, IALU 2-15
ATU
  see Address Translation Unit
AY bus 2-33

Bit shifter 2-18
Bus control micro-orders 3-34

CARRY 2-16
Carry-in base 2-16
Carry-in logic 2-16
CIB, see Carry-in base
Clocks and Timing 2-1
Commercial edit PROMs 2-17
Commercial test PROMs 2-17
Commercial tests 2-17
CON register 2-12
Context block A-2
Control store 1-1
CPA Bus 1-4
CPD Bus 1-4, 2-3, 2-43, B-1
CPD Bus register 2-19
CPM Bus 1-4, C-1
Crossbar network 2-5

DISP Bus 2-32
Dispatch multiplexer 2-5
Dispatch register 2-5
Divide guard digit register 2-29
Divide hardware 2-28
Divide Partial Remainder register 2-29
DPR, see Divide Partial Remainder register

Exception definitions G-1
Excess-64 conversion 3-99
Exponent ALU 2-31
Exponent logic 2-30
Exponent working register 2-30

FA Bus 2-21
Fault codes F-1
FB Bus 2-21
Flags 2-6
Floating-point ALU micro-orders 3-86
Floating-point divide hardware 2-28
Floating-point multiply hardware 2-26
Floating-point register file 2-22
Floating-point STATE register 2-24
Floating-Point Status Register 2-22
Floating-point tests 3-18
Floating-Point Unit 1-4, 2-19
  buses 2-21
  multiply hardware 2-26
FPSR, see Floating-Point Status Register
FPU
  see Floating-Point Unit
FR Bus 2-21
FS Bus 2-21

Hex shifter
  FPU 2-24
  IALU 2-14

I/O protocols 2-46
IALU
  see Integer ALU
ID Bus 2-11
Indirection protection 2-39
Instruction Processor 1-3, 2-43
  state 2-44
Integer ALU 1-4, 2-8, D-1
  tests 3-12
  width of operations 2-8
Integer ALU micro-orders 3-71
Integer register file 2-10
Interrupts 2-45
IP
  see Instruction Processor

LA Bus 1-4
LA bus 2-32
LAT, see Logical Address Translation
Leading Zero Detector 2-25
Logical Address Translation 2-38, A-1
LZD, see Leading Zero Detector

M Bus 2-21
Macroassembler
  ALU operation constructs 5-6
  CPD Bus constructs 5-3
  CPM Bus constructs 5-2
  FA and FB Bus constructs 5-12
  flag constructs 5-16
  FPU operation constructs 5-11
  ID Bus constructs 5-11
  memory completion constructs 5-6
  memory start constructs 5-4
  syntax 5-1
Macroinstruction 1-1
MAG register 2-24
Mantissa ALU 2-26
Mantissa logic 2-21
Memory control micro-orders 3-30
Micro-order 1-1
Microassembler 1-1
Microcode 1-1
Microfield 1-1
Microfields
  AA 3-24
  AB 3-24
  AGB 3-28
  AL 3-29
  AOP 3-29
  CPDS 3-35
  CPMS 3-34
  FCW 3-90
  FL 3-92
  FOP 3-88
  FR 3-86
  FRG 3-95
  FS 3-87
  FWR 3-89
  IA 3-71
  IB 3-71
  ID 3-75
  IL 3-84
  IOP 3-77
  IY 3-79
  MEMC 3-32
  MEMS 3-31
  NAC 3-2
  NAC:COP 3-4
  NAC:DSR 3-23
  NAC:TSEL 3-6
  NAC:UCOP 3-20
  RAND:ATU:ATU0 3-50
  RAND:ATU:ATU1 3-58
  RAND:ATU:SPAD 3-59
  RAND:FIX:COVS 3-60
  RAND:FIX:LOAD 3-63
  RAND:FIX:SPAD 3-63
  RAND:FLT:EXP 3-66
  RAND:FLT:SCNT 3-69
  RAND:FLT:SGN 3-65
  RAND:GEN:REG0 3-40
  RAND:GEN:REG1 3-48
  RAND:GEN:SPAD 3-49
  RS 3-77
Microinstruction 1-1
Microinstruction register 2-2

Microorders
0 3-99
A 3-23, 3-33
A64 3-68
AC 3-48, 3-58
AC0 3-71
AC1 3-72
AC2 3-72
AC3 3-72
ACA 3-68
ACN 3-68
ACW 3-68
AD 3-77
ADD 3-29, 3-69, 3-79, 3-88
AF46 3-45
AF57 3-45
AG 3-34
AG0 3-24
AG1 3-25
AG2 3-25
AG3 3-25
AGA 3-39
ALC 3-62
ANC 3-78
AND 3-78
AOFF 3-57
AON 3-57
AR0 3-26
AR1 3-26
AR2 3-26
AR3 3-27
AR4 3-27
AR5 3-27
AS 3-75
ATD 3-38
ATON 3-11
ATS 3-38
AV 3-63
B 3-28
BL0 3-81
BL1 3-81
BR 3-76
BR0 3-82
BR1 3-81
BSW 3-80
BSX 3-84
BYTE 3-56
C 3-28, 3-30
CA 3-77
CABT 3-5
CAD 3-79
CBLK 3-10
CD 3-77
CDR 3-39
CDSP 3-5
CDW 3-41
CIR 3-38
CIRV 3-8
CJMP 3-5
CJSR 3-5
CLRC 3-61
CM0 3-50
CMP 3-69
CN 3-76
CNT4 3-17
CNT8 3-17
COM1 3-14
COM2 3-15
COMP 3-14
COVK 3-60
COVR 3-60
CPD31 3-7
CRRY 3-18
CRST 3-6
CRTN 3-6
CRY 3-17
CRY28 3-16
CSR 3-78
D 3-28, 3-90, 3-95
D31 3-13
DA 3-77
DECD 3-42
DECS 3-41
DES 3-27, 3-74, 3-94
DF 3-58
DISI 3-55
DR 3-87
DSGN 3-13
DSPA 3-21
DSPR 3-22
DVP 3-70
E 3-24
ECRE 3-10
ECRY 3-20
EDT 3-80
EFA 3-29
EPAR 3-47
F 3-23
F31 3-16
FA 3-86
FB 3-87
FCRY 3-20
FCY 3-42
FG0 3-92
FG1 3-92
FG2 3-93
FG3 3-93
FG4 3-93
FG5 3-93
FG6 3-92

Microorders, cont.
FLG[0-7] 3-9
FNZ 3-70
FP 3-72
FP0 3-91
FP1 3-91
FP2 3-91
FP3 3-91
FRCD 3-42
FRCS 3-41
FRSD 3-44
FSGN 3-17
FZR 3-18
GCRE 3-10
GR0 3-73
GR1 3-73
GR2 3-73
GR3 3-74
GR4 3-74
GR5 3-74
HF 3-35
HL0 3-83
HR0 3-82
HRT 3-83
IA 3-35
ICAT 3-56
IDV 3-98
IHC 3-91
INCD 3-42
INCS 3-41
INDR 3-9
INTR 3-7
IOB 3-8
IOC 3-38
IOEN 3-12
IOFF 3-55
ION 3-55
IOT 3-15
IPS 3-37
IPST 3-8, 3-56
IRES 3-14
IVLD 3-12
IXPC 3-12
IY 3-34, 3-36
L 3-28
LAB 3-66
LAR 3-38
LAT 3-26
LATS 3-51
LAX 3-67
LCN 3-70
LCRE 3-10, 3-50
LCRY 3-61
LD 3-48
LDAD 3-42
LDAS 3-41
LDCY 3-62
LDSD 3-44
LEAP 3-21
LEF 3-70
LESR 3-10
LF 3-35
LFLG 3-46
LFS 3-98
LGD 3-96
LIPS 3-54
LLAR 3-53
LOVC 3-61
LPOP 3-22
LPSR 3-61
LPTA 3-53
LS 3-49, 3-59, 3-64, 3-88
LSR 3-21
LST 3-98
LT 3-48, 3-58, 3-63
LWD 3-98
LWM 3-96
LWR 3-97
LXY 3-97
LY 3-97
LZD 3-71
M 3-30, 3-85
M1 3-73
MAX 3-92
MFS0 3-44
MFS1 3-44
MH 3-94
ML 3-94
MM 3-34
MOV 3-65
MP 3-86
MS 3-75
N 3-30, 3-31, 3-33, 3-34, 3-36, 3-40, 3-48, 3-49, 3-50, 3-58, 3-59, 3-60, 3-63, 3-64, 3-67, 3-69, 3-85, 3-94, 3-95
NA 3-63
NEG 3-65
NM 3-85
NPDR 3-47, 3-57
NY 3-85
ONE 3-25
OPTA 3-54
OR 3-78
OVF 3-18
PASS 3-80
PC 3-37
PCN 3-37
PCPD 3-22
PCX 3-37
PD 3-76

Microorders, cont.
PRGA 3-54
PRGB 3-11
PSB 3-29
PSR 3-79
PUSH 3-22
Q 3-90
R 3-33, 3-90
RB 3-31, 3-87
RD 3-31
RLO 3-96
RMAX 3-9
RND 3-8
RNG0 3-9
RS 3-88
RSBR 3-52
RSGN 3-16
RSRF 3-53
RST 3-69
RW 3-31
S 3-89
S64 3-67
S@ 3-32
SA 3-65
SAEB 3-19
SAGB 3-20
SALB 3-19
SAMP 3-2
SB 3-73
SC 3-75
SETC 3-62
SFS 3-97
SGE 3-18
SIO 3-56
SKFT 3-46
SL 3-72
SMR 3-79
SNM 3-67
SOVK 3-61
SP 3-25
SPCN 3-47
SPY4 3-47
SRC 3-27, 3-74, 3-93
SS 3-75
SST 3-97
SUB 3-29, 3-67, 3-89
TAD 3-89
TPSH 3-22
TRG 3-36
TRI 3-65
TRN 3-66
TRUE 3-7
TSB 3-89
TWB 3-6
TWO 3-26
UAEB 3-19
UAGB 3-19
UALB 3-19
UFS 3-95
USMT 3-7
USS 3-36
VLD 3-11
VPTE 3-11
VSBR 3-11
W 3-33
WB 3-32
WC 3-49, 3-59, 3-64
WD 3-32
WORD 3-57
WRRM 3-53
WS 3-49, 3-59, 3-64
WSBR 3-52
WSKP 3-43
WSX 3-84
WW 3-32
WZX 3-84
X 3-33
X64 3-99
XCTF 3-8
XOR 3-66, 3-78
XTND 3-48, 3-57
Y 3-30, 3-85
Y0 3-16
Y28 3-12
Y29 3-13
Y30 3-13
Y31 3-13
ZER 3-39, 3-66, 3-92
ZR 3-76, 3-88
Microprogram counters 2-3
Microprogramming 1-2
examples 4-1
Microroutine 1-1
Microsequencer 1-3, 2-2
micro-orders 3-2
tests 2-7, 3-7
Microstack 2-3
Microstack Input Multiplexer 2-3
Microword format 3-1
Multiply ALU 2-28
Multiply hardware 2-26
MV/10000 computer 1-1
architecture 2-1
buses 1-4
operation 2-1
subsystems 1-2

NAC, see Next Address Control
Narrow and wide operations 2-8
Next Address Control 3-2

Page fault A-1
Page table addressing logic 2-38
Page zero locations E-1
PDR, see CPD Bus register
Processor Status Register 2-18
PSR, see Processor Status Register

RA multiplexer, see RAM Address multiplexer
RAM Address multiplexer 2-5
Random micro-orders 3-39
Read/write/execute protection 2-40
Referenced/modified RAM 2-38
Register file
  AG 2-33
  floating-point 2-22
  IALU 2-10
Register File In multiplexer 2-36
RFIN multiplexer, see Register File In multiplexer
Ring protection 2-39

SA and SB registers 2-31
SCP
  see System Control Processor
Scratch pad 2-12, H-1
Sign logic 2-30, 2-31
SPAD, see Scratch pad
SPAR register 2-12
Starting microaddress 2-5
STUAD, see Starting microaddress
System Control Processor 1-2, 2-7

Terminology 1-1
Tests 2-7
  Address Translation'Unit 3-9
  floating-point 3-18
  integer ALU 3-12
  microsequencer 3-7
Top of stack register 2-4, 2-5
TOS, see Top of stack register
Transfer register 2-13

Validity RAM 2-38

WCS
  see Writable Control Store
Working register 2-26
Writable Control Store 1-2, 2-2

X and Y registers 2-27

YSEL counter 2-28

Data General

Comment Form

Please help us improve our future publications by answering the questions below. Use the space provided for your comments.

Title: ______________________

Document No. ______________________

Yes	No			
☐	☐	Is this manual easy to read?	O You (can, cannot) find things easily. O Language (is, is not) appropriate. O Technical terms (are, are not) defined as needed.	O Other:
		In what ways do you find this manual useful?	O Learning to use the equipment O As a reference O As an introduction to the product	O To instruct a class. O Other:
☐	☐	Do the illustrations help you?	O Visuals (are,are not) well designed. O Labels and captions (are,are not) clear. O Other:	
☐	☐	Does the manual tell you all you need to know? What additional information would you like?		
☐	☐	Is the information accurate? (If not please specify with page number and paragraph.)		

CUT ALONG DOTTED LINE

FOLD FOLD

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

DataGeneral

ATTN: Technical Publications
62 Alexander Drive
Research Triangle Park, NC 27709

CUT ALONG DOTTED LINE

Data General

TP________

TIPS ORDER FORM
Technical Information & Publications Service

BILL TO:	SHIP TO: (if different)
COMPANY NAME________	COMPANY NAME________
ADDRESS ________	ADDRESS ________
CITY________	CITY________
STATE________ ZIP ________	STATE________ ZIP ________
ATTN: ________	ATTN: ________

QTY	MODEL #	DESCRIPTION	UNIT PRICE	LINE DISC	TOTAL PRICE

(Additional items can be included on second order form) [Minimum order is $50.00]

Tax Exempt #________
or Sales Tax (if applicable)

TOTAL	
Sales Tax	
Shipping	
TOTAL	

METHOD OF PAYMENT

☐ Check or money order enclosed
For orders less than $100.00

☐ Charge my ☐ Visa ☐ MasterCard
Acc't No.________ Expiration Date________

☐ Purchase Order Number:________

SHIP VIA

☐ DGC will select best way (U.P.S or Postal)

☐ Other:
☐ U.P.S. Blue Label
☐ Air Freight
☐ Other ________
____

NOTE: ORDERS LESS THAN $100, INCLUDE $5.00 FOR SHIPPING AND HANDLING.

Person to contact about this order ________ Phone ________ Extension ________

Mail Orders to:

Data General Corporation
Attn: Educational Services/TIPS F019
4400 Computer Drive
Westboro, MA 01580
Tel. (617) 366-8911 ext. 4032

____
Buyer's Authorized Signature Date
(agrees to terms & conditions on reverse side)

____
Title

____
DGC Sales Representative (If Known) Badge #

DISCOUNTS APPLY TO MAIL ORDERS ONLY

012-1780

educational services

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE
TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form shown on the reverse hereof which is accepted by DGC.

1. **PRICES**
   Prices for DGC publications will be as stated in the Educational Services Literature Catalog in effect at the time DGC accepts Buyer's order or as specified on an authorized DGC quotation in force at the time of receipt by DGC of the Order Form shown on the reverse hereof. Prices are exclusive of all excise, sales, use or similar taxes and, therefore are subject to an increase equal in amount to any tax DGC may be required to collect or pay on the sale, license or delivery of the materials provided hereunder.

2. **PAYMENT**
   Terms are net cash on or prior to delivery except where satisfactory open account credit is established, in which case terms are net thirty (30) days from date of invoice.

3. **SHIPMENT**
   Shipment will be made F.O.B. Point of Origin. DGC normally ships either by UPS or U.S. Mail or other appropriate method depending upon weight, unless Customer designates a specific method and/or carrier on the Order Form. In any case, DGC assumes no liability with regard to loss, damage or delay during shipment.

4. **TERM**
   Upon execution by Buyer and acceptance by DGC, this agreement shall continue to remain in effect until terminated by either party upon thirty (30) days prior written notice. It is the intent of the parties to leave this Agreement in effect so that all subsequent orders for DGC publications will be governed by the terms and conditions of this Agreement.

5. **CUSTOMER CERTIFICATION**
   Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub-licensee of the software which is the subject matter of the publication(s) ordered hereunder.

6. **DATA AND PROPRIETARY RIGHTS**
   Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

7. **DISCLAIMER OF WARRANTY**
   DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS SUPPLIED HEREUNDER.

8. **LIMITATIONS OF LIABILITY**
   IN NO EVENT SHALL DGC BE LIABLE FOR (I) ANY COSTS, DAMAGES OR EXPENSES ARISING OUT OF OR IN CONNECTION WITH ANY CLAIM BY ANY PERSON THAT USE OF THE PUBLICATION OF INFORMATION CONTAINED THEREIN INFRINGES ANY COPYIGHT OR TRADE SECRET RIGHT OR (II) ANY INCIDENTIAL, SPECIAL, DIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOSS OF DATA, PROGRAMS OR LOST PROFITS.

9. **GENERAL**
   A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer.

DISCOUNT SCHEDULES

DISCOUNTS APPLY TO MAIL ORDERS ONLY.

LINE ITEM DISCOUNT

5-14 manuals of the same part number - 20%
15 or more manuals of the same part number - 30%

DISCOUNTS APPLY TO PRICES SHOWN IN THE CURRENT TIPS CATALOG ONLY.

Data General

TIPS ORDERING PROCEDURE:

Technical literature may be ordered through the Customer Education Service's Technical Information and Publications Service (TIPS).

1. Turn to the TIPS Order Form.
2. Fill in the requested information. If you need more space to list the items you are ordering, use an additional form. Transfer the subtotal from any additional sheet to the space marked "subtotal" on the form.
3. Do not forget to include your MAIL ORDER ONLY discount. (See discount schedules on the back of the TIPS Order Form.)
4. Total your order. (MINIMUM ORDER/CHARGE after discounts of $50.00.)

   If your order totals less than 100.00, enclose a certified check or money order for the total (include sales tax, or your tax exempt number, if applicable) plus $5.00 for shipping and handling.
5. Please indicate on the Order Form if you have any special shipping requirements. Unless specified, orders are normally shipped U.P.S.
6. Read carefully the terms and conditions of the TIPS program on the reverse side of the Order Form.
7. Sign on the line provided on the form and enclose with payment. Mail to:

   TIPS
   Educational Services – M.S. F019
   Data General Corporation
   4400 Computer Drive
   Westboro, MA 01580
8. We'll take care of the rest!

educational services

CUT ALONG DOTTED LINE

Data General

users group

Installation Membership Form

Name ______ Position ______ Date ______

Company, Organization or School ______

Address ______ City ______ State ______ Zip ______

Telephone: Area Code ______ No. ______ Ext. ______

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
MV/Series ECLIPSE®		
Commercial ECLIPSE		
Scientific ECLIPSE		
Array Processors		
CS Series		
NOVA®4 Family		
Other NOVAs		
microNOVA® Family		
MPT Family		

Other ______
(Specify) ______
______

3. Software

☐ AOS	☐ RDOS
☐ AOS/VS	☐ DOS
☐ AOS/RT32	☐ RTOS
☐ MP/OS	☐ Other
☐ MP/AOS	

Specify ______

4. Languages

☐ ALGOL	☐ BASIC
☐ DG/L	☐ Assembler
☐ COBOL	☐ FORTRAN 77
☐ Interactive COBOL	☐ FORTRAN 5
	☐ RPG II

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communication

☐ HASP	☐ X.25
☐ HASP II	☐ SAM
☐ RJE80	☐ CAM
☐ RCX 70	☐ XODIAC™
☐ RSTCP	☐ DG/SNA
☐ 4025	☐ 3270
	☐ Other

Specify ______
______

7. Application Description

○ ______
______
______
______

8. Purchase

From whom was your machine(s) purchased?

☐ **Data General Corp.**
☐ Other
Specify ______
______

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

FOLD FOLD

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

DataGeneral

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581

CUT ALONG DOTTED LINE

Data General users group

Installation Membership Form

Name ______________ Position ______________ Date ______

Company, Organization or School ______________

Address ______________ City ______ State ______ Zip ______

Telephone: Area Code ______ No. ______ Ext. ______

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
MV/Series ECLIPSE®		
Commercial ECLIPSE		
Scientific ECLIPSE		
Array Processors		
CS Series		
NOVA®4 Family		
Other NOVAs		
microNOVA® Family		
MPT Family		

Other ______
(Specify) ______

3. Software

- ☐ AOS
- ☐ AOS/VS
- ☐ AOS/RT32
- ☐ MP/OS
- ☐ MP/AOS
- ☐ RDOS
- ☐ DOS
- ☐ RTOS
- ☐ Other

Specify ______

4. Languages

- ☐ ALGOL
- ☐ DG/L
- ☐ COBOL
- ☐ Interactive COBOL
- ☐ BASIC
- ☐ Assembler
- ☐ FORTRAN 77
- ☐ FORTRAN 5
- ☐ RPG II

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communication

- ☐ HASP
- ☐ HASP II
- ☐ RJE80
- ☐ RCX 70
- ☐ RSTCP
- ☐ 4025
- ☐ X.25
- ☐ SAM
- ☐ CAM
- ☐ XODIAC™
- ☐ DG/SNA
- ☐ 3270
- ☐ Other

Specify ______

7. Application Description

○ ______

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify ______

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

FOLD FOLD

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

DataGeneral

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581