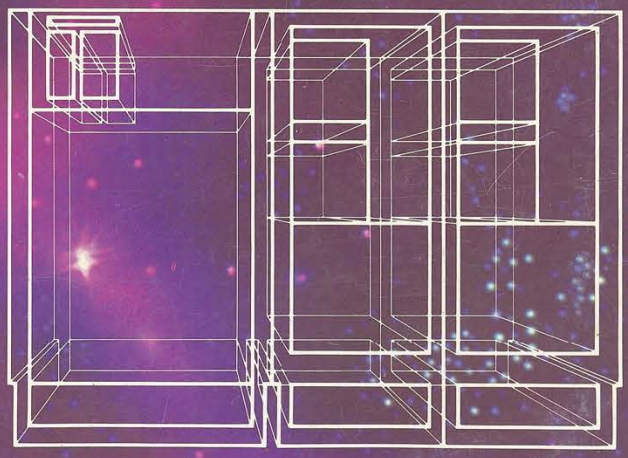
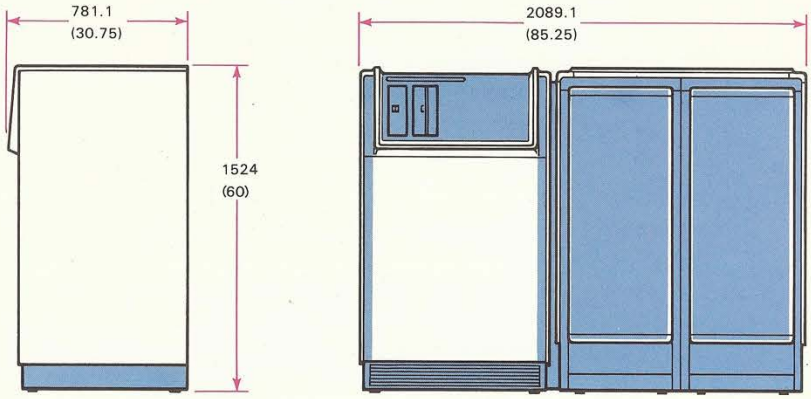
THE ECLIPSE MV/8000



Data General

Product Summary Series

MV/8000 Specifications



Side view **Front view**

Dimensions in millimeters
Inches in parentheses for reference

DG-06856

Dimensions:	**Width**	**Depth**	**Height**
Millimeters	2089.1	781.1	1524
Inches	82.5	30.75	60

Service Clearances:	**Front**	**Rear**	**Right**	**Left**
Millimeters	1219.2	1219.2	787	787
Inches	48	48	31	31

Weight:	**Empty**	**Fully Loaded**
Kilograms	359	585
Pounds	791	1289

Heat Output:	**Watts**	**BTU/hr**
Man Bay:	3,000	10,230
Expansion Bays:	220 each	750.2

Operating Environment:

Temperature (max)	32.2°C	90°F
Relative Humidity (max)	90%	
Altitude	3084 m	(10,000 ft)

	Areas	**Inches**	**cm**
Usable Vertical Rack Space Per Expansion Bay	25	43.75	111

Power Requirements: (Main Bay + Expansion Bays)

(Domestic)		
Voltage	120/208	
Hz	60±.5	
Amp per Phase	(24 + 20 + 20)	
	Main Bay	**Expansion Bay**
Phase	3Ø	1Ø

Startup Surge per Phase 60 Amps for 50 milliseconds

Power Requirements:

(Export)			
Voltage	220	220/380	240/415
Hz	50±.5	50±.5	50±.5
Amp per Phase	24+15	24+15	24+15
Phase	3	3	3

Startup Surge per Phase 60 Amps for 50 milliseconds

Cables: (User Supplied)

Primary Power - Main Bay	Conn
Domestic 60 Hz Export 50 Hz	L21-30R

Primary Power-Expansion Bays		
Domestic		L41-30R
Export		L6-15R
Power available	Each	Total (All bays all recp.)
Domestic 120 V	15	30.0
Export 220/240 V	15	14.25

Specifications and Performance

Word size	32 bits	
Instruction widths	16 - 80 bits	
Virtual address range	4 Gbytes (4,294,967,296 bytes)	
Physical address range	4 Mbytes (max.)	
Processor minor cycle time	110 ns	
Processor major cycle time	220 ns	
Memory transfer time	27.5 ns/byte 440 ns/block of 16 bytes	
Typical instruction execution times (32-bit)	32-bit fixed-point add	220ns
	32-bit floating-point:	
	Multiply	2200 ns
	Add	880 ns
	Store	440 ns
Memory transfer rate	36.4 Mbytes/s	
I/O-to-memory transfer rate	18.2 Mbytes/s	
CPU-to-memory transfer rate	18.2 Mbytes/s	

Configuration Specifications (Maximum -- cannot be reached simultaneously)

Main memory size	4 Mbytes
On-line storage capacity	6.648 Gbytes
DG/Disc storage subsystems	6
277-Mbyte disc drives	24
Fixed-head DG/disc subsystems	6
Fixed-head DG/disc drives	24
Terminals	128
Dual-mode tape drives	16
Data control units	4
Card readers	2
Incremental plotters	2

Operating System Parameters (Max.)

Users	128
Processes	255
Program size	512 Mbytes

Programming Languages Supported

ANSI FORTRAN 77	COBOL
ANSI BASIC	DG/L
ANSI PL/I	Extended BASIC
FORTRAN 5	RPG II

Utilities

SWAT	AZ-TEXT
MASM	RJE80
REPORT	HASP II
LINK	RCX70
INFOS II	MPL
SORT	TPMS
INFOS QUERY	Idea
DBMS	

ECLIPSE MV/8000™
Product Summary

DataGeneral

Table of Contents

1 Overview and Summary 3

Overview 3
- Hardware 3
- System I/O 4
- Software 4

Summary 5
- Instruction Set 5
- Architecture 5
- Hardware 6

2 Architectural Features 7

Design Goals 7
Virtual Memory 7
- Segments 8
- Rings 8

Resource Management 9
- Central Processing and Control 11
- System Control, Diagnostics, and Console Control 11
- Asynchronous Communications 12
- Synchronous Communications 12

Instruction Set 12
- New Functions 12
- Symmetrical Data Treatment 12

3 Hardware Features 13

Memory System 13
- Memory Modules 13
- Bank Controller 15
- System Cache 15

Central Processing System 17
- Address Translation Unit 17
- Processor 19

I/O Subsystem 20
- High-Speed I/O — The BMC 20
- Medium-Speed I/O — The Data Channel 20
- Low-Speed I/O — Programmed I/O 20
- Communications 22

4 The Operating System **23**

Design Objectives **23**
System Architecture **23**
Overview **23**
Memory Management **24**
Demand Paging **25**
Process and Task Management **25**
File Management **26**
I/O Management **26**
Programming Language and Utilities Support **26**
Programming Languages **26**
Utilities **27**

5 Instruction Set **29**

MV/8000 Addressing **29**
MV/8000 Data Formats **30**
Fixed-Point Data Formats **30**
Floating-Point Data Formats **30**
Commercial Data Formats **31**
Instruction Mnemonics and Formats **33**
Instruction Mnemonics **33**
ALC Format **33**
Summary of the MV/8000 Instruction Set **33**
Fixed-Point Instructions **33**
Floating-Point Instructions **34**
System Control Instructions **34**
Stack Instructions **35**
Program-Flow Instructions **35**
Commercial Instructions **35**
I/O Instructions **35**

6 Peripherals **37**

Disc Storage Subsystems **37**
Magnetic Tape Subsystems **39**
User Terminals **40**
Line Printers **41**

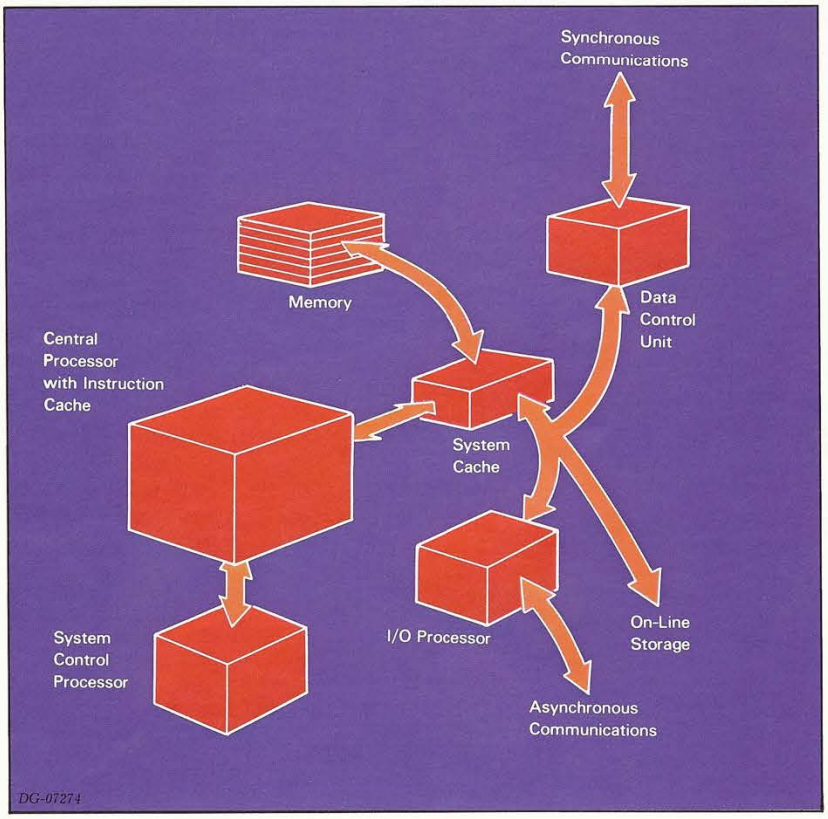


Figure 1.1 Block diagram of the ECLIPSE MV/8000 system

Chapter 1
Overview and Summary

Overview

The ECLIPSE MV/8000® system extends minicomputer technology in the area of mainframe computational characteristics. It is a sophisticated, state of the art, 32-bit data processing system that retains substantial hardware and mode-free software compatibility with previous 16-bit ECLIPSE® systems.

Figure 1.1 shows a diagram of the MV/8000 system.

Hardware

The MV/8000 hardware design achieves high throughput with the aid of several distinct processors distributed throughout the system. A system cache and an instruction cache further improve system performance.

Processors

Four separate processors are incorporated into the MV/8000 design:

- The central processing unit (CPU),
- The system control processor (SCP),
- The I/O processor (IOP),
- The optional data control unit (DCU).

These processors allow for efficient placement and management of resources at their point of use, increased system performance, and a minimum of data movement and system overhead.

All four processors have access to the system memory, and all but the CPU have their own local memories. To avoid confusion, we refer to system memory as *main memory* in this book.

The central processor is a 32-bit ECLIPSE CPU employing pipelining techniques and a 220-ns major cycle time (110-ns minor cycle time) for high-speed instruction execution. The CPU architecture is based on a segmented virtual memory and an associated ring protection mechanism. As a result, the MV/8000 computer supports user programs of 512 Mbytes (536,870,912 bytes); it also supports distributed operating system structures in a virtual address space of 4 Gbytes (4,294,967,296 bytes).

The system control processor (SCP) is a microNOVA® system, which oversees the entire MV/8000 system. This includes monitoring vital system parameters via the diagnostic scan bus, logging hardware errors, managing the operator's terminal, controlling system diagnostics, and loading the microcode control store at power up. The SCP contains 4 Kbytes of PROM and 32 Kbytes of RAM, along with its own 1.2-Mbyte diskette.

The I/O processor (IOP) is a 16-bit ECLIPSE CPU that controls all asynchronous communications for up to 128 user terminals plus card readers and plotters. Using its 64-Kbyte local memory and a direct link to main memory, it significantly reduces the load on the 32-bit central processor.

A fourth processor, the data control unit (DCU/200), is available for handling synchronous line protocols. By using its 8-Kbyte local memory and a direct link to main memory, the DCU handles communications supporting distributed processing environments.

Caches

A 16-Kbyte system cache acts as a high-speed look-ahead/look-behind buffer for main memory. It has separate CPU and I/O ports that operate without interfering with one another. The system cache is direct-mapped to main memory and employs write-back techniques to reduce the number of main memory write cycles.

The system cache transfers data between main memory and the central processor at a maximum rate of 18.2 Mbytes/second. It also transfers data concurrently between main memory and the I/O subsystem at the same maximum rate over a separate bus. The combination of busses yields a total cache-to-main-memory bandwidth of 36.4 Mbytes/second.

The CPU contains a separate 1-Kbyte instruction cache, which prefetches instructions during sequential execution. The instruction cache acts as a look ahead/look-behind buffer for the instruction stream.

System I/O

MV/8000 I/O is both electrically compatible and program compatible with 16-bit ECLIPSE I/O, thus supporting the family of standard Data General peripherals.

The I/O subsystem provides three levels of I/O:

- A high-speed burst multiplexor channel (BMC) providing transfer rates up to 16.16 Mbytes/second for high-speed devices such as discs;
- A data channel providing transfer rates up to 2.27 Mbytes/second for medium-speed devices such as tape units;
- Programmed I/O providing transfer rates of one or two bytes per I/O instruction.

Software

The Advanced Operating System/Virtual Storage (AOS/VS) extends the functions of Data General's Advanced Operating System (AOS) into the 32-bit virtual memory environment of the MV/8000 system.

Programming Languages
AOS/VS supports three 32-bit programming languages: ANSI PL/I, ANSI FORTRAN 77, and ANSI BASIC. It also supports several 16-bit languages such as COBOL, FORTRAN 5, DG/L® Systems Development Language, and RPG II. Together these provide a broad range of options to the programmer, from the simple sophistication of BASIC to the sophisticated data-handling abilities of PL/I and FORTRAN 77.

Utilities
AOS/VS supports a variety of utilities to assist the programming process. These include the SWAT® Debugger, which permits debugging in the programming language; the macroassembler and linking loader; and the Command Line Interpreter (CLI), which provides an interface to the user working at a system terminal. In addition, the operating system supports data management, transaction processing, and word processing software such as INFOS® II File Management System, DG/DBMS, TPMS, and AZ-TEXT® word processing software.

Summary

Instruction Set

- The 32-bit MV/8000 instruction set is a superset of the 16-bit ECLIPSE instruction set.
- The combined instruction set provides mode-free compatibility at the binary level with existing 16-bit programs plus the full flexibility of new 32-bit instructions.
- Sixteen-bit and 32-bit program development and debugging are included on one system.
- Fixed-point instructions manipulate 8-, 16-, and 32-bit data symmetrically without mode switching.
- Floating-point instructions use either of two floating-point rounding algorithms for maximum flexibility and accuracy.

Architecture

- Segmented memory, consisting of eight 512-Mbyte segments, provides efficient management of a 4-Gbyte logical address space. One- or two-level page table translations are possible for each segment.
- Eight hardware-implemented protection rings permit system functions to be embedded in the user's logical address space. This results in reduced overhead during the processing of operating system calls. All operating-system calls are inward, cross-ring calls, which aid in the construction of a layered operating system and in the production of reliable and modular software.
- Hardware-maintained page-referenced and page-modified bits record a reference or change to each 2-Kbyte page. This permits the use of sophisticated page replacement algorithms, thus reducing the page fault rate.
- The maximum contiguous memory necessary to support programs of any size is one page, because of the 2048-byte page size. This allows optimal performance for both large and small programs.
- The system control processor provides sophisticated and thorough control of the processor, system error logging, and diagnostics. It also implements the soft control console.

Hardware

- A dual-ported system cache acts as a 16-Kbyte direct-mapped buffer for main memory. The cache provides a high-performance memory system that segregates CPU and I/O memory accesses.
- A separate instruction cache acts as a 1-Kbyte direct-mapped buffer for the instruction stream. The instruction cache permits simultaneous fetch and decoding of one instruction with concurrent execution (including data fetching) of another. Up to four instructions can be in the pipeline.
- A high aggregate I/O bandwidth of 18.2 Mbytes/second helps eliminate I/O bottlenecks.
- A high main memory bandwidth of 36.4 Mbytes/second reduces CPU-I/O interference.
- A *sniffing* feature provides error detection and correction during memory refresh, thus increasing system reliability. Each memory location is checked every four seconds and corrected if necessary.
- Data channel and BMC I/O use existing data channel and BMC controllers, thus supporting the standard family of Data General peripherals. All peripheral diagnostics run on the MV/8000 system.
- An alterable control store, consisting of 4K x 75 bits of RAM, permits microcode updates to be implemented in a straightforward manner and the diagnostic instruction set to overlay the standard instruction set.
- A 4-line diagnostic scan bus (S bus) provides the system control processor with access to MV/8000 subsystems.
- System hardware errors are logged by the system control processor on a 1.2-Mbyte diskette, providing a record to aid in diagnosis of failing components.

Chapter 2
Architectural Features

Design Goals

The ECLIPSE MV/8000 design concepts specifically included the following goals:

- Mode-free compatibility with previous 16-bit ECLIPSE systems;
- Support of sophisticated data base structures in a large address space, using virtual memory techniques;
- Support of modular software structures independent of memory arrangements, using segmented address space techniques;
- Hardware-supported protection mechanisms to keep various software functions from impinging on one another;
- Multitasking resource management;
- Thirty-two-bit manipulation in a single operation;
- A symmetrical instruction set independent of data structure.

Virtual Memory

Virtual memory provides each system user with an address space much larger than the physical memory space on the machine. Thus, the address space seen by an MV/8000 user (the *logical address space*) is 4 Gbytes, with 4 Mbytes of physical memory (the *physical address space*). The large user address space makes it much easier to handle sophisticated data bases.

A virtual memory system moves the active portions of a program from disc to memory during the execution of the program, returning inactive portions of the program to the disc when more memory space is needed. This process, called *demand paging*, is transparent to the user. The MV/8000 demand-paging mechanism moves data in 2-Kbyte pages, an optimum size for efficient memory management.

When a program tries to use a location not currently represented in physical memory, a *page fault* occurs, invoking the demand-paging mechanism which retrieves the page from the disc.

Any new page must overwrite a page that is already in memory. To improve the effectiveness of the software page-replacement algorithm, the MV/8000 demand-paging hardware maintains *page-referenced* and *page-modified* bits for each page of physical memory. Use of these bits significantly lowers the page fault rate of a system.

Segments

A virtual memory system, by itself, will not support the modular software structures necessary for flexible handling of large and sophisticated data bases. Some division of the address space is necessary.

The MV/8000 logical address space is divided into eight units of 512 Mbytes each, called *segments*. These segments are independent, but connected by clearly defined protocols. Thus, each segment can be used for a different function; this makes management of the entire virtual memory efficient and reliable.

Rings

Software modularity also implies protection mechanisms. These mechanisms maintain the necessary degree of independence or interdependence among the modules. The MV/8000 memory system provides a hardware mechanism known as *protection rings* to support this function. If a program executing in one segment needs to alter or access the contents of another segment, the program must follow protocols established by the protection rings. This fast protection mechanism is transparent to the user.

Each segment in the MV/8000 virtual memory is surrounded by a protection ring, which is permanently bound to the segment. Thus, ring 0 is bound to and protects segment 0; ring 1 similarly is bound to and protects segment 1; and so forth through ring and segment 7.

The eight segments (with their associated rings) are hierarchically arranged: segment 0 has the greatest ability to alter or access the contents of other segments and is afforded the greatest protection by ring 0. Segment 7 has the least ability to alter or access other segments and is afforded the least protection by ring 7. Therefore, segment 0 contains the kernel of the operating system, while segment 7 is reserved for user programs. Intermediate segments are used for various operating system or user functions.

A cross-ring data reference is valid only if it originates from a segment with a number *lower* than or *equal* to the target segment. For example, an instruction in segment 4 can make a reference to data in segment 5, but not to data in segment 3.

A cross-ring subroutine (or system) call is valid only if it originates from a segment with a number *higher* than or *equal* to the target segment. In addition, the call cannot be made directly to the starting location of the subroutine, but must pass instead through a *gate* in the target segment that points to the start of the subroutine. The gate provides additional protection and ensures that instruction execution starts at the beginning of the subroutine. Gate checking is hardware supported and transparent to the user. The gate mechanism also offers a degree of program independence. For example, the caller of a routine may be provided with only a gate number. Then, the author of that routine may change its contents, independent of the caller. Linkage is through gates; it is not hard coded.

A subroutine return is valid only if it originates from a segment with a number *lower* than or *equal* to the target segment. Gates are not used in subroutine returns. Figure 2.1 illustrates these relationships.

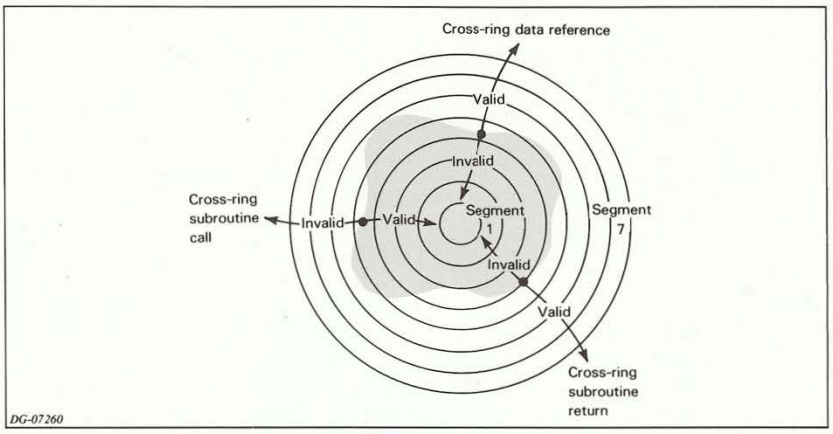


Figure 2.1 Cross-ring references

A large segmented and protected address space containing both the user program and the operating system provides several significant advantages. System calls become subroutine calls, thus eliminating time-consuming context switches. Overlays, with attendant complications, are unnecessary because of the size of the individual segments (512 Mbytes). User-related faults (e.g., floating-point faults, fixed-point overflow, stack overflow) can be handled within the current segment without involving the operating system.

Resource Management

A system designed to support multiple users requires careful resource management. An efficient way to do this is to off-load some functions that traditionally have been performed by the central processing unit (CPU). These include message concentration and networking as well as performance monitoring and self-checking.

The MV/8000 architecture uses the concept of *distributed intelligence* to implement these functions. As illustrated in Figure 2.2, this means that four functional groups are assigned separate processors:

- Central processing and control;
- System control, diagnostics, and console control;
- Asynchronous communications;
- Synchronous communications.

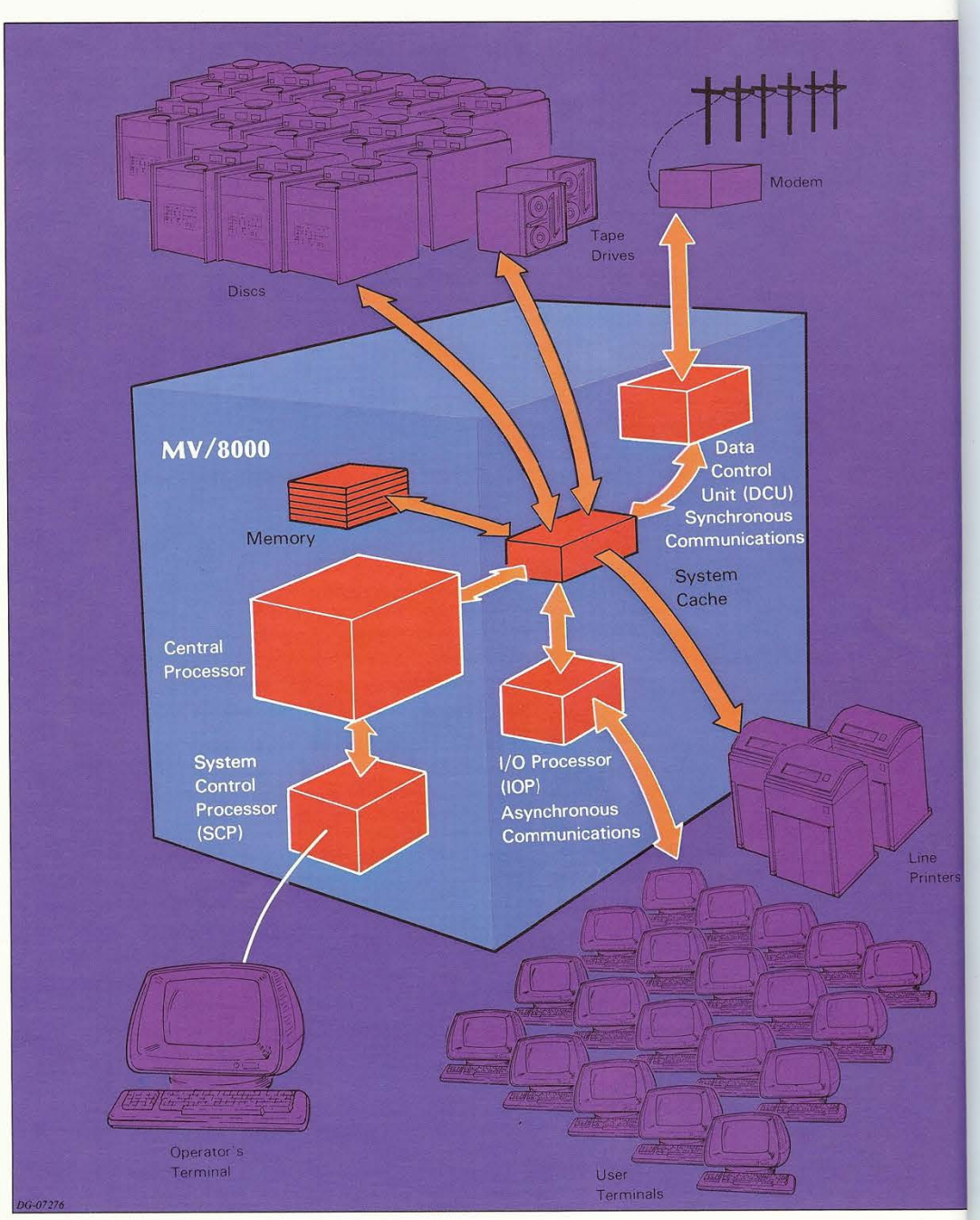


Figure 2.2 Distributed intelligence in the ECLIPSE MV/8000 system

Central Processing and Control

The central processing and control functions are handled by a 32-bit pipelined processor. See Chapter 3 for a more detailed description of this processor.

System Control, Diagnostics, and Console Control

The MV/8000 system control, diagnostic, and console functions are handled by the system control processor (SCP), which includes a microNOVA CPU, a 32-Kbyte random access memory (RAM), a 4-Kbyte PROM, a separate operating system and operator's console, and a 1.2-Mbyte diskette drive.

The SCP acts as a system overseer. Portions of it are active during normal system operation and other portions are active during power-up and diagnostic operations.

The SCP uses a 4-line diagnostic and control bus (S bus) to detect the state of major subsystems, command remedial action, and control a unique diagnostic fault-resolution system.

System Control Functions

These functions include loading the microcode at system power up, logging hardware errors, and checking system integrity.

Loading Microcode. Because the MV/8000 control store is made up of alterable RAM, the microcode for the instruction set must be reloaded each time the machine is powered up. This is done automatically by the SCP. Updates to the microcode can be made at any time.

Hardware Error Logging. All hardware errors are logged automatically by the SCP. These include memory errors detected by the error checking and correction (ERCC) facility, cache parity errors, and other errors that can indicate hardware problems.

Checking System Integrity. During normal operations, the SCP monitors various system parameters such as air flow, air temperature, and power supply voltages. If any of these go out of acceptable range, that fact is logged and the system operator is notified.

Diagnostic Functions

These functions include automatic power-up diagnostics and operator-controlled system diagnostics.

During power up, the SCP performs diagnostics that check out basic system functions before loading the microcode. If problems are encountered, the operator is notified.

When system-wide diagnostics are required, the SCP program first checks the SCP itself, along with the interface between it and the rest of the system. Then, it loads a diagnostic instruction set into the MV/8000 microstore, overlaying the regular instruction set. Diagnostic tests use this diagnostic instruction set to test the MV/8000 hardware rapidly and thoroughly.

When searching for intermittent hardware problems, the operator can vary power supply voltages and system clock frequencies slightly in order to vary the stress on system components.

Console Functions
The SCP performs all the console functions traditionally performed by switches. The operator can load, examine, and modify memory or microcode locations by entering commands through the operator's console. The operator can also step through a program instruction by instruction, or microinstruction by microinstruction.

Asynchronous Communications

The concentration of messages required for asynchronous communications is handled by a separate I/O processor (IOP), which controls all user terminals. The IOP is a 16-bit ECLIPSE CPU with a 64-Kbyte local memory and data channel access to main memory.

Synchronous Communications

The protocol functions associated with synchronous communications are handled by a separate, optional data control unit (DCU/200). The DCU is a 16-bit NOVA® CPU with an 8-Kbyte local memory and data-channel access to main memory.

Instruction Set

The MV/8000 instruction set is a superset of the 16-bit ECLIPSE instruction set. The set includes both the 16-bit ECLIPSE instructions and the new MV/8000 instructions, which offer several new functions. The additions also provide symmetrical treatment of all data sizes.

New Functions

The new functions implemented in the MV/8000 instruction set include queue and linked list manipulation, field testing under mask, special instructions to control the memory management system, new program-flow instructions to implement subroutine calls more flexibly, and memory-to-accumulator fixed-point arithmetic instructions.

Symmetrical Data Treatment

The MV/8000 instruction set includes several extensions that increase its flexibility without destroying the basic simplicity of the 16-bit ECLIPSE instructions. The result is a large and flexible instruction set that is simple to learn and use.

All data manipulations can be applied to 16- or 32-bit data by using the appropriate instruction prefix. Similarly, another instruction prefix determines whether a 15-bit or 31-bit displacement is used for memory reference instructions. In each case, however, the function of the instruction is constant throughout the range of data and address types.

Chapter 3
Hardware Features

In this chapter, we discuss the hardware features of the ECLIPSE MV/8000 computer that significantly accelerate system performance.

Memory System

The MV/8000 memory system may be divided into three major components:

- The memory modules, which provide the main storage for the MV/8000 system;
- The bank controller, which selects the appropriate memory module and performs error checking and refresh operations;
- The system cache, which acts as a high-speed buffer between memory and the rest of the system.

Figure 3.1 illustrates the relationship between memory and the rest of the system.

Memory Modules

The MV/8000 memory system supports up to eight 256- or 512-Kbyte memory modules, for a maximum physical address space of 4 Mbytes. Each module contains 64K or 128K double words of 32 bits each. In addition, seven error checking and correcting (ERCC) bits are stored with each double word.

The physical addresses of the memory locations on each module are arranged so that four sequential double words in memory are located on four separate memory planes that operate concurrently. When access to one of these four double words is requested by the system cache, the memory module performs the selected read or write starting at plane 0, and then continues the operation through planes 1, 2, and 3.

Because the four planes operate concurrently, the memory operations overlap, making the transfer of the four double words very rapid. The transfer rate between the memory modules and the bank controller is 36.4 Mbytes per second, or 110 nanoseconds per four bytes.

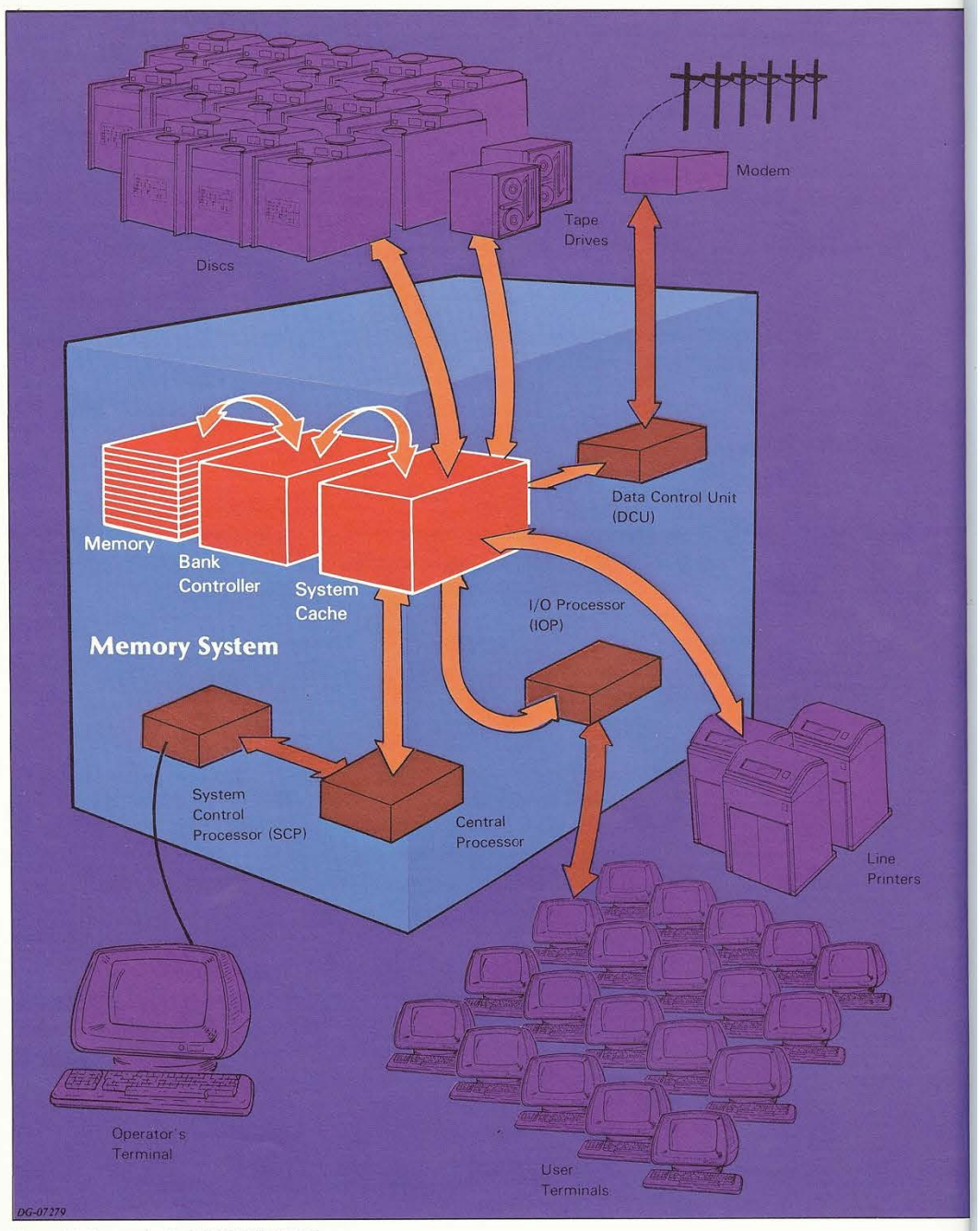


Figure 3.1 Memory in the ECLIPSE MV/8000 system

Bank Controller

The bank controller performs four functions: module selection, error checking and correction, parity generation and checking, memory refresh and sniffing.

Module Selection

When the system cache requests access to a memory location, the bank controller uses the physical address to select the proper memory module.

Error Checking and Correction

The bank controller calculates and appends seven ERCC bits to each double word it sends to a memory module. Each time the bank controller receives a double word from a memory module, it checks the ERCC code and corrects any single-bit errors in the double word before passing it on to the system cache. ERCC also detects multiple-bit errors.

All ERCC errors are logged by the system control processor. ERCC functions increase the reliability of MV/8000 memory operations and give early warning of memory problems.

Parity Functions

The bank controller adds one parity bit to each byte it sends to the system cache and checks the parity of each byte/parity bit combination it receives from the system cache. Parity generation and checking help maintain the integrity of the system by providing another level of error detection and reporting.

Memory Refresh and Sniffing

The bank controller performs the refresh operations required by the dynamic random access memory (RAM) modules. It accomplishes this by addressing all chips simultaneously, without reading or writing the addressed data. This function is performed on all of memory every two milliseconds.

Additionally, the bank controller performs an operation called *sniffing,* which checks for memory errors during the refresh operation. Sniffing verifies all memory locations, correcting all single-bit errors even if that memory location is not being used by a program. This prevents an unused area of memory from collecting single-bit errors and also prevents intermittent single-bit errors from becoming uncorrectable multiple-bit errors. Thus, sniffing reduces the possibility of encountering multiple-bit memory errors and increases system reliability.

The sniffing operation performs a read/correct/write operation on four double words during memory refresh. With this process, the four double words are read from a selected memory module, single-bit errors are corrected, and the double words are then written back into the memory module. The bank controller sniffs all memory locations for errors every four seconds.

System Cache

A 16-Kbyte direct-mapped system cache functions as a high-speed look-ahead/look-behind buffer for the main memory. It significantly reduces the average memory access time for both the central processing unit (CPU) and the I/O subsystem.

The system cache, like main memory, organizes its locations into blocks of 16 bytes each. The system cache contains 1024 of these blocks. See Figure 3.2 for a diagram of main memory/system cache mapping.

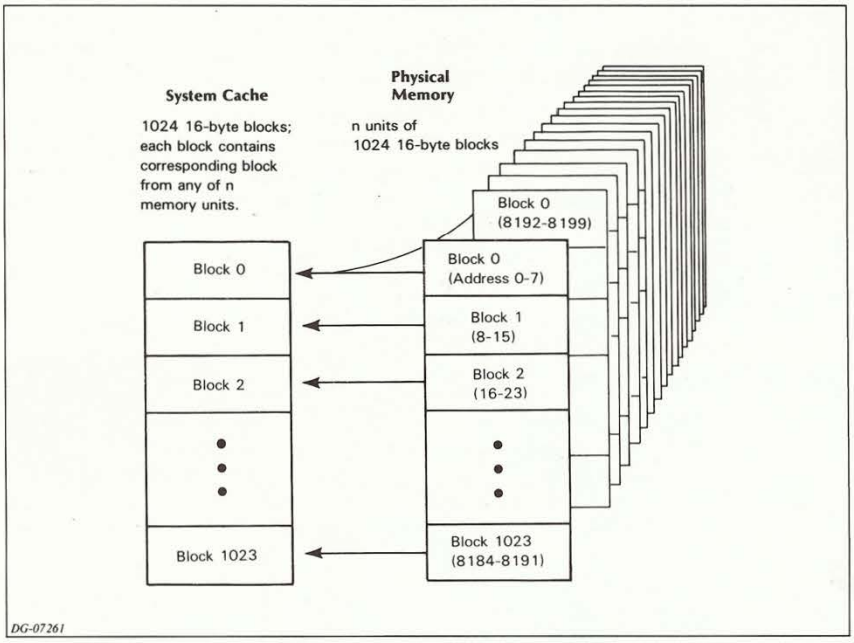


Figure 3.2 Main memory/system cache mapping

When a memory reference is made to a location not in the system cache, the cache retrieves the block of 16 bytes that contains the referenced data. These 16 bytes remain in the system cache block until a memory reference is made to another main memory location that maps into the same system cache block.

Data residing in the system cache can be modified by the program as necessary. The corresponding memory locations are not updated until the cache locations are required for a different block. Then they are updated only if the cache locations have been changed. This process is called *write back*. It reduces the total number of memory write cycles by ensuring that only necessary writes occur.

In addition to the bus to memory, the system cache has two ports to the rest of the system — one for the CPU and the other for the I/O system. Each port has a bandwidth of 18.2 Mbytes/second, equal to half the bandwidth of the cache/memory bus. These ports can operate simultaneously except when they are contending for the cache/memory bus.

Central Processing System

The central processing system consists of the address translation unit (ATU), the instruction processor with its instruction cache, the microsequencer, and the arithmetic logic unit (ALU). Together, these components execute all user programs and translate all virtual memory references into physical addresses used by the system cache and memory. Figure 3.3 shows the processing system in relation to the rest of the system.

The processing system converts logical page addresses to physical page addresses using table references, called *page table translations*. A *one-level page table translation* uses only one page table reference and can translate any 1 Mbyte of addresses in a segment. A *two-level page table translation* uses two sequential page table references and can translate the full 512 Mbytes of a segment.

One-level translations take less time than two-level translations and require less physical memory. Therefore, programs that do not need more than 1 Mbyte of virtual memory gain a significant speed advantage by using one-level translations.

Address Translation Unit

The ATU accelerates the translation process by maintaining a table of 256 recently-used page translations and access rights. Because memory references tend to cluster in a few pages, any one memory reference is likely to find a translation entry in the ATU. As the cluster of pages that form the *working set* moves through the program's logical address space, the ATU updates the entries in its table of page address translations.

The ATU controls the page-modified and page-referenced bits used by page replacement algorithms. Use of these bits can lower the page fault rate for the system. (See the discussion of demand paging in Chapter 2.)

The ATU performs all the hardware checking required by the ring protection system, including checking access privileges and gate entry points. (See the discussion of rings in Chapter 2.)

Finally, the ATU can emulate the 16-bit ECLIPSE MAP. This permits AOS, all its utilities, and all 16-bit user programs created for previous 16-bit ECLIPSE systems to be run on the ECLIPSE MV/8000 system. Sixteen-bit MAP emulation and 32-bit ATU operations are mutually exclusive.

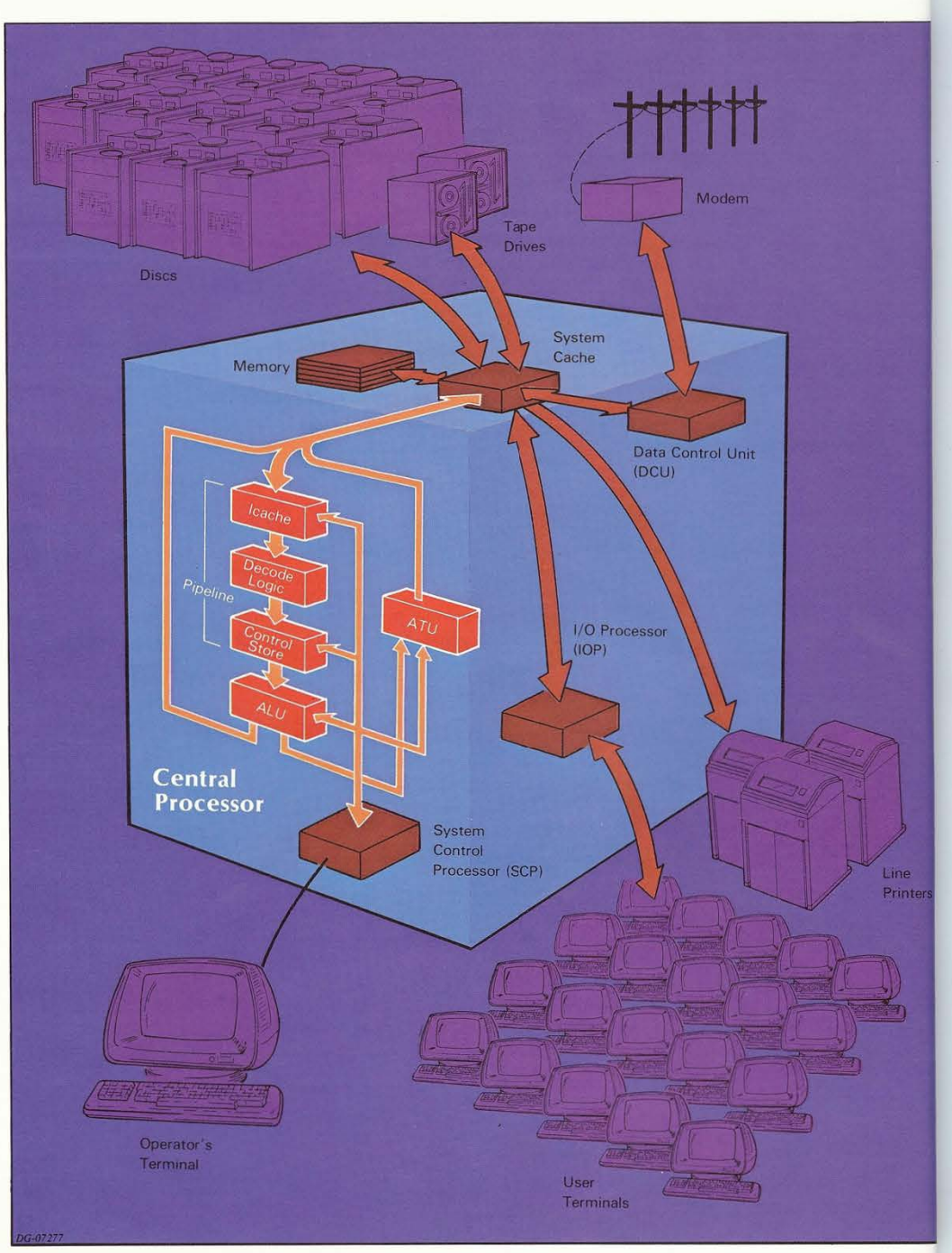


Figure 3.3 The processing system

Processor

The MV/8000 CPU takes advantage of *pipelining* techniques to enhance its performance. Pipelining involves simultaneous performance of the various steps in the instruction decoding and execution process. That is, while one instruction is being executed, the next one is being decoded, and the one after that is being fetched from the instruction cache. For sequential processing, this can substantially reduce the effective instruction execution time. The steps in the pipeline are as follows.

Step 1: The CPU's instruction processor fetches an instruction from the instruction cache.

Step 2: The decode logic in the instruction processor interprets the opcode of the instruction to obtain the starting microcode address.

Step 3: The microsequencer reads the specified microinstruction from the control store.

Step 4: The ALU executes the microinstruction.

Instruction Processor

The instruction processor retrieves and decodes instructions for subsequent execution by the microsequencer. This processor retrieves instructions from the system cache, placing them in an *instruction cache* for subsequent decoding.

The 1-Kbyte direct-mapped instruction cache functions as a high-speed, look-ahead/look-behind buffer for the instruction stream. It reduces the time the CPU is idle while waiting for an instruction to be fetched.

Figure 3.4 illustrates system cache/instruction cache mapping. When the program counter references an instruction that is not in the instruction cache, the instruction processor retrieves that location, as well as the locations following that one in the same system cache block.

Microsequencer

The microsequencer contains the RAMs that form the MV/8000 control store. The microprogrammed information (microcode) that forms the MV/8000's instruction set resides in this control store, which consists of 4K 75-bit microwords. Because the control store uses RAMs, the microcode must be reloaded from the system control processor diskette each time the system is powered up.

With alterable control store, updates to the microcode can be implemented with a replacement diskette. In addition, a diagnostic instruction set, which overlays the MV/8000 instruction set in the control store, makes possible fast and thorough diagnostics. The system control processor loads and uses the diagnostic instruction set when it runs system diagnostics.

Arithmetic Logic Unit

The ALU performs the arithmetic and logical manipulations for both fixed- and floating-point instructions. It contains four 32-bit fixed-point accumulators, four 64-bit floating-point accumulators, a 64-bit floating-point status register, four 32-bit stack management registers, and several internal registers not accessible to the programmer. The ALU operates with a basic cycle time of 110 nanoseconds; e.g., each partial product for a multiply instruction is calculated every 110 nanoseconds.

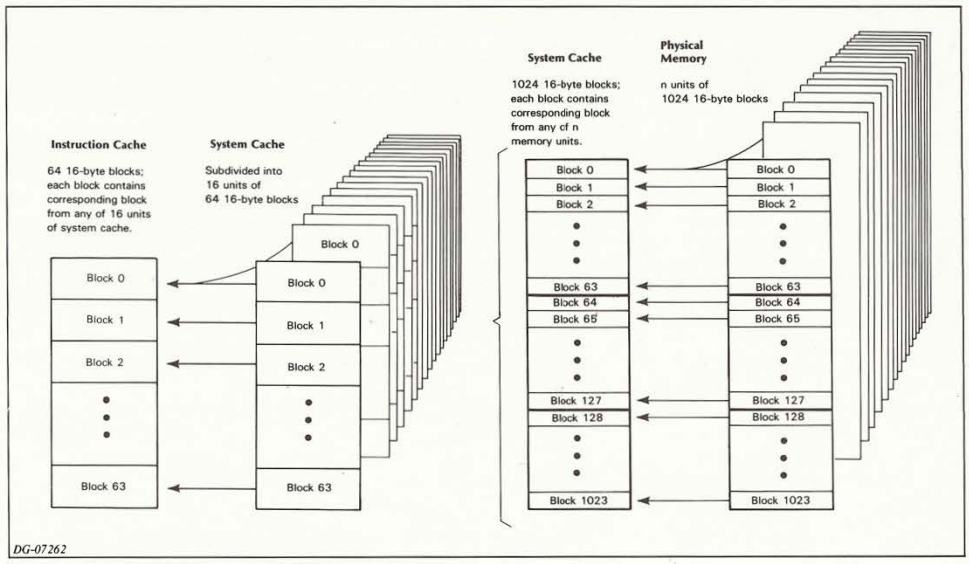


Figure 3.4 Memory/system cache/instruction cache mapping

I/O Subsystem

The MV/8000 I/O subsystem, which is compatible with existing Data General peripherals, handles all communications functions, using the separate I/O processor and the optional data control unit. Figure 3.5 shows the I/O subsystem in relation to the rest of the system.

High-Speed I/O — The BMC

The burst multiplexor channel (BMC) transfers data directly between the I/O port of the system cache and high-speed system peripherals such as discs. Data are transferred at a maximum rate of 16.16 Mbytes/second (input to memory) and 14.54 Mbytes/second (output from memory). Operation of the BMC does not affect the CPU except when the BMC is contending for the system cache/main memory bus.

Medium-Speed I/O — The Data Channel

The data channel transfers data between the I/O port of the system cache and medium-speed devices. Maximum transfer rates are 2.27 Mbytes/second (input) and 1.3 Mbytes/second (output). As long as the BMC and data channel together do not exceed the maximum transfer rate of the system cache I/O port (18.2 Mbytes/second), they operate without interfering with one another.

Low-Speed I/O — Programmed I/O

Programmed I/O (PIO) provides I/O transfers directly between the CPU and low-speed devices and between the CPU and the control registers of medium- and high-speed devices. Each I/O instruction executed moves one or two bytes of information.

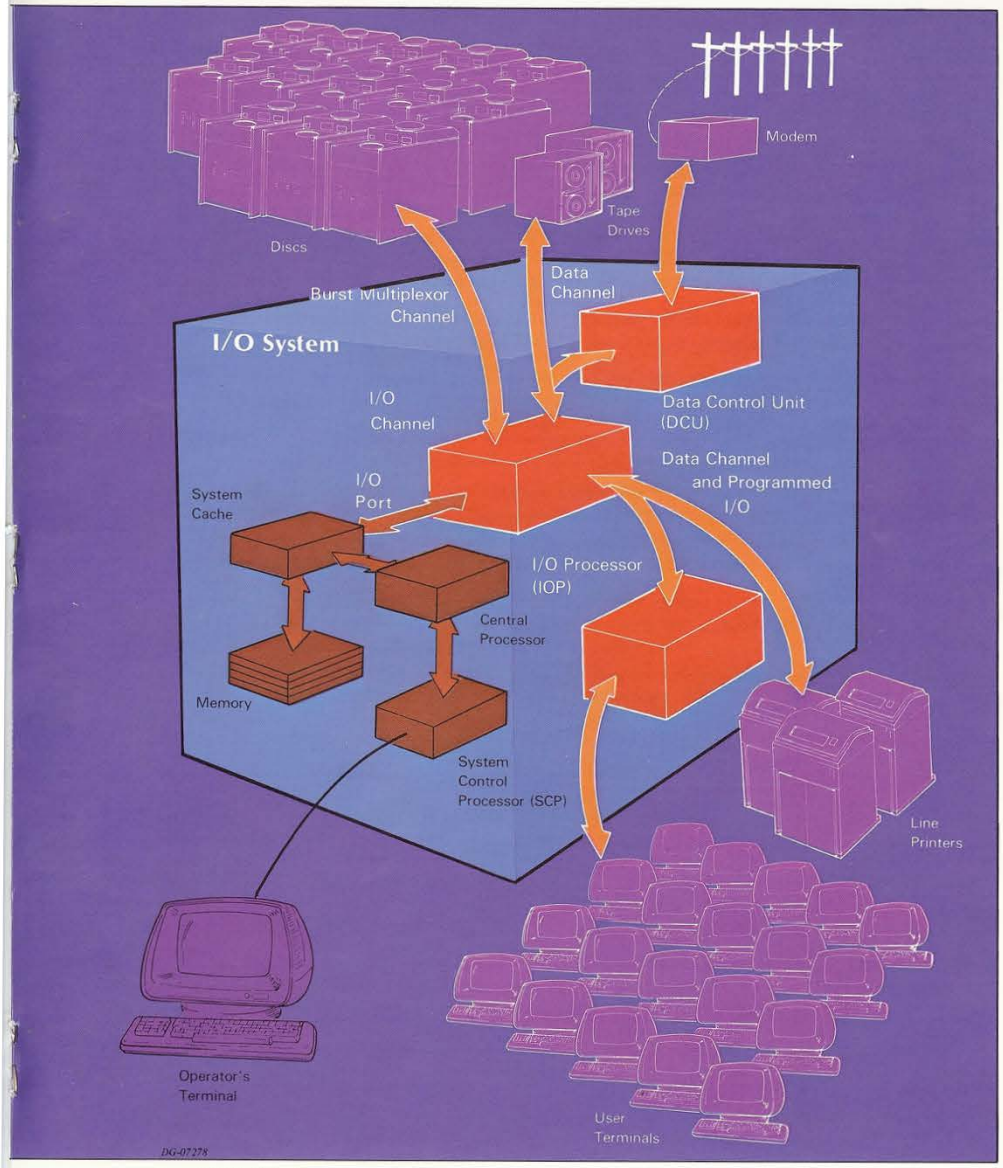


Figure 3.5 The I/O subsystem

Communications

The I/O processor and the data control unit handle all MV/8000 asynchronous and synchronous communications.

I/O Processor

The I/O processor (IOP) controls up to 128 asynchronous lines connected to data terminal equipment, card readers, and digital plotters. A MAP permits the IOP to map some of its logical address space into its 64-Kbyte local memory and the remainder of its logical address space into MV/8000 data channel address space. Using this feature, the IOP can quickly transfer data between its local memory and the system cache by moving data from one area of its logical address space to another. The IOP can change the data channel translations it needs by itself, giving it access to all main MV/8000 memory.

A cross-interrupt facility permits the IOP and the CPU to interrupt one another when necessary, reducing the need for either processor to monitor the other.

Data Control Unit

The optional data control unit (DCU) controls up to eight synchronous communications lines that provide interprocessor communications and other medium- and high-speed communications functions.

Up to 8 Kbytes of the DCU's logical address space can be mapped into its 8-Kbyte local memory. The rest of the logical address space is mapped into MV/8000 data channel address space. The DCU can transfer data between its local memory and the MV/8000 system cache by moving data from one area of its logical address space to another.

A cross-interrupt facility permits the CPU and DCU to interrupt one another when necessary, reducing the need for one processor to monitor the other.

Chapter 4
The Operating System

Design Objectives

Data General's Advanced Operating System/Virtual Storage (AOS/VS) provides the multiprogramming and virtual storage features required to match the strengths of the ECLIPSE MV/8000 system. AOS/VS was designed to meet the following objectives.

- Provide mode-free compatibility with 16-bit AOS, including the ability to run existing 16-bit AOS user programs concurrently with new 32-bit user programs;
- Fully exploit the MV/8000 32-bit, hardware-based, segmented virtual memory system;
- Support a large number of users, each with a very large address space, while keeping system overhead to a minimum;
- Simultaneously support timesharing, multiple batch stream, and real-time processes;
- Support sophisticated multiprogramming techniques, multitasking within processes, and intertask and interprocess communications;
- Support the latest versions of high-level programming languages, with optimization and native language debugging, and the full set of standard Data General languages and utilities.

System Architecture

Overview

AOS/VS is an evolutionary, state-of-the-art, 32-bit operating system built on the firm base of three years' experience with AOS — Data General's 16-bit multiprogramming operating system. The result is an operating system that maintains mode-free compatibility with 16-bit AOS, yet exceeds all requirements of the 32-bit environment.

Virtual memory techniques are used by AOS/VS to support up to 128 users, each with up to 512 Mbytes of programming area. Because AOS/VS makes full use of the MV/8000 memory management hardware and controls demand paging with a sophisticated page-fault-frequency paging algorithm, system overhead is unusually low for this type of operating system.

AOS/VS further reduces its overhead by distributing the operating system through both the hardware and the software. In the hardware, portions of the operating system control the I/O processor and the optional data control units (DCU/200). By distributing these

functions, the traditional conflict between an I/O-limited and a compute-bound system is eliminated. In the software, portions of the operating system are embedded in each user's address space, simplifying system calls and reducing the need for costly context switching.

Memory Management

Each user space in AOS/VS corresponds to the 4-Gbyte hardware-supported address space provided by the MV/8000 virtual memory system (see Chapter 2 for a discussion of the MV/8000 architecture). AOS/VS also supports the eight 512-Mbyte hardware-supported subdivisions, or *segments,* of this address space. Each segment contains a unit of the software — the user program, a user library module, or an operating system module.

Because the user program and the operating system reside within one large address space, a system call is accomplished using a subroutine call rather than context switching. Calls to library routines in another segment are also subroutine calls; they do not require any operating system intervention.

To protect the integrity of the software in the segments, each segment is permanently bound to a hardware-based protection ring. AOS/VS reinforces the ring protection features of the MV/8000 architecture by selecting gate entry points for the hardware-enforced gate arrays at each ring boundary. The hardware traps and the operating system reports all invalid ring-crossing attempts.

Operating System Modules

Figure 4.1 shows how AOS/VS assigns the function of each of the segments in a user's address space. Segments 0 through 3 contain most of the operating system. The kernel resides in segment 0 and is shared by all users. Segments 1 through 3 are unique to each user. The scheduler, memory manager, debugger, file system, and other operating system components are distributed among segments 0 through 3.

User Programs, Libraries, and Subroutines

Segments 4 through 6 support programming language libraries and user-written subroutines. These may contain code shared by different users, thus reducing the amount of physical memory used, which in turn reduces the system overhead required for paging. While the target segment for a subroutine call is selected during compiling, the contents of the target segment can be specified at execution time; this reduces the time devoted to recompiling programs.

User programs reside in segment 7, which provides 512 Mbytes of logical address space for user programs and data. Programs and data may be present in any proportion. Because this space is so large, overlays are unnecessary.

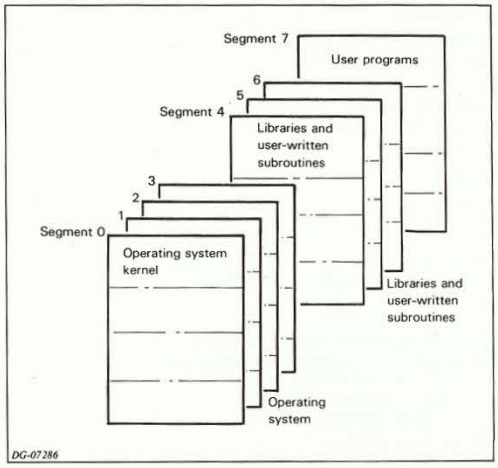


Figure 4.1 Structure of AOS/VS

Demand Paging

AOS/VS controls the hardware-based memory management system with a page-fault-frequency page replacement algorithm in the demand-paging system. This algorithm, which uses the hardware Page Referenced and Page Modified bits, dynamically allocates memory pages to the working set of each process, thus maintaining a constant page fault frequency. This insures that all processes equally share paging overhead.

Furthermore, any system load that forces the page-fault frequency beyond an operator-controlled threshold causes the lowest priority job to be swapped out; thus, the paging overhead is kept within a range that precludes thrashing.

Process and Task Management

AOS/VS is a process-oriented multiprogramming operating system. A process in AOS/VS is a collection of program tasks sharing up to 512 Mbytes of address space. Each process is assigned a set of privileges that determine how much memory and system resource time that process can use.

One process is created by another process — either a system process or a user process. In each case the parent process assigns the privileges of the offspring, thus imposing a "family-tree" hierarchy on all processes to support interprocess control. Parent processes can also block, unblock, and kill offspring processes.

The parent process decides whether an offspring process is permanently memory-resident, preemptible (usually resident but swappable if blocked or if a higher-priority process needs the space), or swappable. Resident and preemptible processes can have up to 256 levels of priority, while swappable processes can have up to 3 levels of priority. As many as 255 processes can be active on AOS/VS at any time.

A full set of system calls supports interprocess communications and synchronization. The interprocess communications facility includes the ability to spool messages until the destination process is ready to receive them.

Up to 32 tasks can be assigned to each process. Multitasking is facilitated on AOS/VS by a full set of system calls that control task priorities, intertask communications, synchronization, and timing.

File Management

AOS/VS provides a hierarchical file directory structure that includes complete file protection by user access. The user of the "family tree" that contains a file directly controls access to that file by assigning up to five different types of access to any other user or group of users.

Certain peripheral devices can be treated as files in AOS/VS. This device independence permits references to files in a program without specifying the device until the program is run.

I/O Management

AOS/VS supports dynamic, fixed-length, data sensitive, and variable-length records for all file I/O. The record type can be specified when creating a file and changed when opening it for I/O operations. In addition, block I/O is supported for discs and magnetic tape units.

AOS/VS supports a spool queue for each system I/O device, such as a line printer or asynchronous communication line. Spooling permits more efficient scheduling of the processors, reducing idle time of the system. Users may assign a priority to a queue entry that will affect the entry's ultimate position in the queue. The operator controls the limits of priority each user may assign.

Programming Language and Utilities Support

The programming languages and utilities provided by an operating system are tools of the users. AOS/VS provides excellent tools for its users by including versions of languages that meet the latest industry specifications, as well as standard languages and utilities that have been field proven through years of use with 16-bit AOS.

Programming Languages

AOS/VS supports implementations of FORTRAN 77, PL/I, and BASIC that meet or exceed the latest ANSI specification in each case. These languages use the full 32-bit data and addressing capability of the MV/8000 system and AOS/VS.

FORTRAN 77 and PL/I use compilers that share common code generators and optimizers, thus increasing the reliability and efficiency of both systems. They also use the SWAT debugger, which permits debugging in the programming language.

In addition, AOS/VS supports COBOL, FORTRAN 5, Extended BASIC, RPG II, Idea, and DG/L® System Development Language. These languages use compilers and runtime systems that have previously been used with 16-bit AOS.

Table 4.1 shows the languages supported by AOS/VS.

Language	Specification	Features
FORTRAN 77	ANSI FORTRAN X3.9-78	32-bit data structure; structured programming capability; character handling features; three levels of optimization; SWAT debugger.
PL/I	ANSI PL/I X3.74	32-bit data structure; on-line development environment for multiple users; three levels of optimization; SWAT debugger.
BASIC	ANSI BASIC X3.60-1978	Shareable interpreter; 32-bit data structure; includes significant extensions to the ANSI standard.
COBOL	ANSI '74	Screen-handling capabilities for interactive program execution; interactive debug module; virtual data capability.
FORTRAN 5	ANSI FORTRAN X3.9-1966	Global optimization; user sharable compiler; access to INFOS II file management system.
Extended BASIC		Shareable interpreter; message communications facilities.
RPG II		IBM compatible; supports INFOS II file management system.
Idea		Interactive data entry and inquiry/response capabilities; generates screen formats; uses INFOS II file management system.
DG/L Systems Development Language		ALGOL-like syntax for structured programming; global optimization; access to INFOS II file management system.

Table 4.1 Programming languages

Utilities

AOS/VS supports a wide variety of system utilities to simplify data base management, program development, and system management.

The full complement of AOS-standard data management, transaction processing, and word processing software is supported by AOS/VS. These include INFOS II, DG/DBMS, Interactive Query, TPMS, and AZ-TEXT software subsystems.

Program development utilities include the SWAT native language debugger for FORTRAN 77 and PL/I, MASM 32-bit macro assembler, AOS Macro Processor for Procedural Languages (AOS MPL), and a variety of text and file editors.

System management utilities include REPORT for monitoring resource use on the system, EXEC for managing consoles and batch streams on a timesharing system, and the Command Line Interpreter (CLI) for controlling the interaction between system users and the operating system.

Chapter 5
Instruction Set

This chapter describes the addressing and data formats used in the MV/8000 system, followed by a short summary of the MV/8000 instruction set. For complete information, see the *ECLIPSE MV/8000 Principles of Operation, Programmer's Reference Series* (DGC No. 014-000648).

MV/8000 Addressing

The MV/8000 instruction set contains memory reference instructions that can address bits, bytes, words, and double words. The types of addressing and the addressing range associated with each are shown in Table 5.1.

Addressing Type	Address Range Relative To Base Value
8-bit word	+127, −128 words
15-bit word or double word	± 16 Kwords ± 8 K double words
16-bit byte	± 32 Kbytes
31-bit word or double word	Entire address space
32-bit byte	Entire address space

Table 5.1 Addressing types and ranges

All of the MV/8000 memory reference instructions use three addressing modes: absolute, PC (program counter) relative, and AC (accumulator) relative. Each addressing mode uses a different source for the base value when calculating an address, adding that value to a displacement coded with the instruction.

Absolute Addressing. The base of the logical address space (location 0) is used as the base address. Therefore, the displacement itself is the logical address of the desired memory location.

PC Relative Addressing. The value in the program counter (the logical address of the word containing the displacement) is used as the base address.

AC Relative Addressing. The value in one of two accumulators is used as the base address.

MV/8000 Data Formats

The MV/8000 system uses a variety of data formats to facilitate data handling.

Fixed-Point Data Formats

The MV/8000 fixed-point data formats support signed and unsigned 8-, 16-, and 32-bit integers. Eight-bit integers are supported in 32-bit registers by zero-extending the integers to 32 bits. Sixteen-bit integers are supported in 32-bit registers by sign-extending the integers to 32 bits. These formats are shown in Figure 5.1.

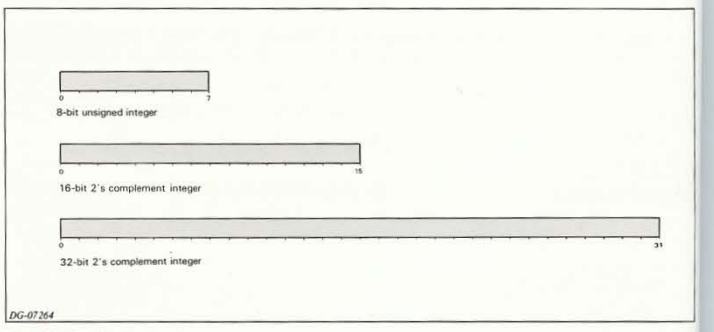


Figure 5.1 Fixed-point data formats

Floating-Point Data Formats

The MV/8000 floating-point data formats support single-precision (32-bit) and double-precision (64-bit) floating-point numbers without mode changing. These data formats can accommodate data representing up to 17 significant decimal digits within a range of 5.4×10^{-79} to $7.2 \times 10^{+75}$. These formats are shown in Figure 5.2.

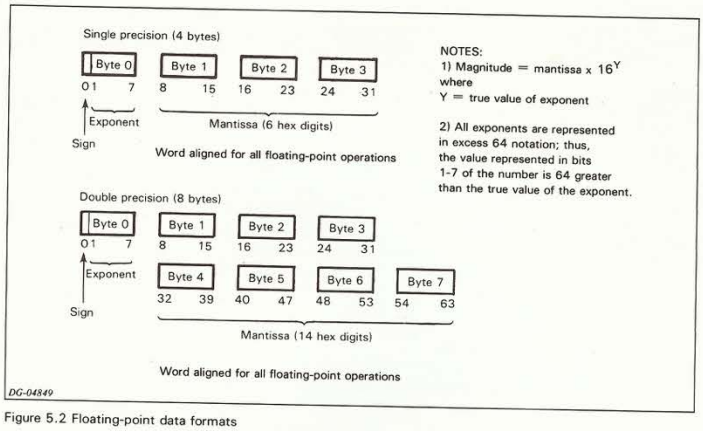


Figure 5.2 Floating-point data formats

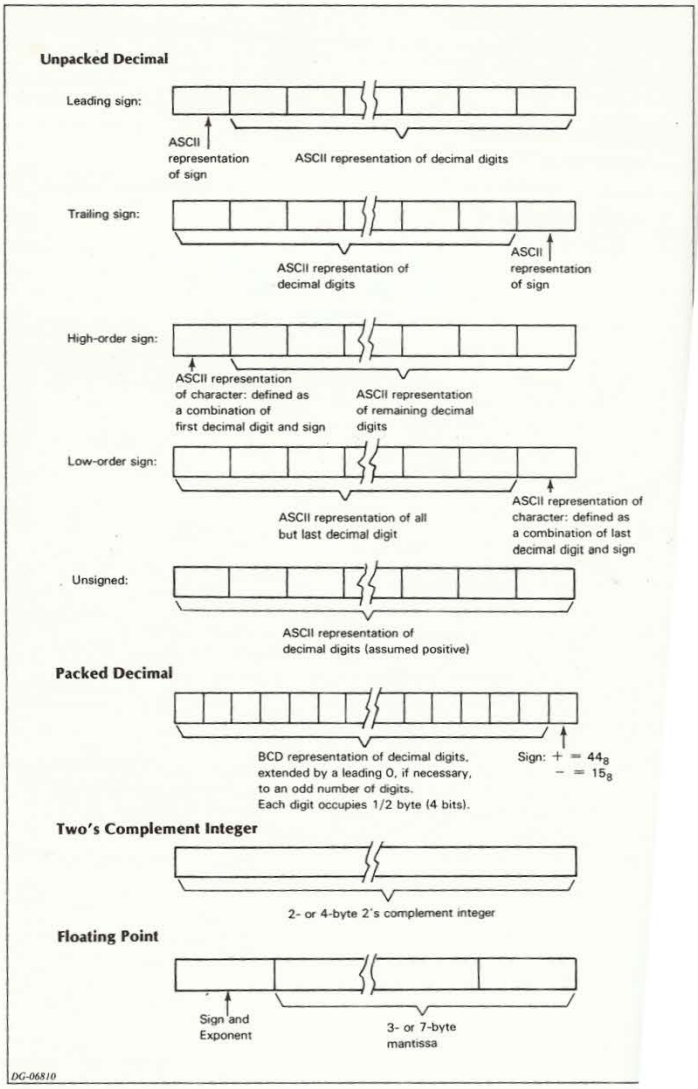


Figure 5.3 Commercial data types

Instruction Mnemonics and Formats

Several mnemonic conventions and instruction formats in the MV/8000 system are worthy of special note.

Instruction Mnemonics

Mnemonic prefixes are used in the MV/8000 system to distinguish between similar instructions that work with address displacements or data of different lengths:

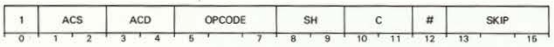
X Extended displacement (15 bits),
L Long displacement (31 bits),
N Narrow data (16 bits),
W Wide data (32 bits).

ALC Format

The MV/8000 instruction set includes the Arithmetic/Logic Class (ALC) instructions from the 16-bit ECLIPSE instruction set. These instructions perform functions generally classed as fixed-point operations (e.g., add, subtract); in addition, they accept mnemonic suffixes that perform the following operations:

- Preset carry,
- Shift a 16-bit word and carry left or right,
- Swap two bytes of a 16-bit word,
- Test for various conditions of the word or of carry,
- Prevent loading the resulting value into a specified destination accumulator.

The format for these instructions is:



Summary of the MV/8000 Instruction Set

The 32-bit MV/8000 instruction set is a powerful and logical extension of the 16-bit ECLIPSE instruction set. More than 250 instructions have been added to handle 32-bit addressing, 32-bit fixed-point operands, memory management functions, and queue and linked list structures. The instruction set supports 8-, 16-, 32-, and 64-bit data manipulation, together with a variety of operating system and high-level language functions.

The 16-bit ECLIPSE instruction set is an integral part of the MV/8000 instruction set. Thus, the ECLIPSE instruction set is available at all times without explicit mode switching. This simplifies upgrading and ensures continued ability to do program development for 16-bit ECLIPSE systems.

Fixed-Point Instructions

The 181 fixed-point instructions perform integer arithmetic, logical operations, compares, and 16-bit Arithmetic/Logic Class (ALC) operations. In addition, some fixed-point instructions manage bits and bytes. The fixed-point instructions manipulate 8-, 16- and 32-bit operands symmetrically: operations on one data size are identical to operations on other data sizes. Mixing data sizes requires no explicit function switching.

An example of a queue instruction is the **ENQH** *Enqueue Towards the Head* instruction. This instruction adds a data element to a queue, placing the new data element before a specified data element already in the queue. It then updates all the appropriate links and queue descriptors to reflect the change in the queue. The entire operation finishes before interrupts are enabled to ensure that an interrupt routine does not try to use an incomplete queue.

Stack Instructions

The 24 stack instructions define and manipulate stacks. The MV/8000 supports three types of stacks: the *wide stack*, used by programs incorporating 32-bit instructions; the *narrow stack*, used by programs incorporating 16-bit ECLIPSE instructions; and the *vector stack*, used by the I/O interrupt facility.

Each stack uses a stack pointer, frame pointer, stack limit, and stack base to define boundaries and limits of the stack. All of these boundaries and limits can be changed by the user to adjust the configuration of the stack to meet the user's needs.

An example of a stack instruction is the **SAVE** instruction. This instruction is used at the beginning of subroutines to save the state of the machine prior to entering the body of the subroutine. The *Save* instruction pushes a 5-word return block onto the stack and creates a stack frame, of size determined by the user, for storing values being passed to or from the subroutine. The frame pointer acts as a reference point for this storage area.

Program-Flow Instructions

The 83 program-flow instructions alter the normally sequential flow of instructions by placing a new value in the program counter. The new value may be determined by data coded along with the instruction or by the results of a test on the data. Both fixed-point and floating-point instructions are included in this category.

An example of a program-flow instruction is the **LDSP** *Dispatch (Long Displacement)* instruction. This instruction uses a table of addresses to transfer control to one of several possible routines. A number in an accumulator serves as a pointer into the table. The location pointed to contains the starting address of the desired routine.

Commercial Instructions

The 22 commercial instructions manipulate and convert between the commercial data types specified in the discussion of commercial data formats earlier in this chapter. In addition, they convert between decimal integers and floating-point numbers; evaluate the sign of a decimal number; and move, modify, and test strings of characters.

An example of a commercial instruction is the **WEDIT** *Wide Edit* instruction. This instruction invokes a group of subinstructions that can perform many different operations on a decimal number and its destination field, including leading zero suppression, floating fill characters, punctuation control, and insertion of text into the destination field. The instruction also performs operations on alphanumeric data.

I/O Instructions

The 20 I/O instructions control all system input and output. I/O instructions handle three types of I/O transfers: *programmed I/O* transfers data one byte or word at a time, with each transfer being controlled by a separate I/O instruction; *data channel I/O* requires I/O instructions to set up the parameters of the transfer, but the transfer itself,

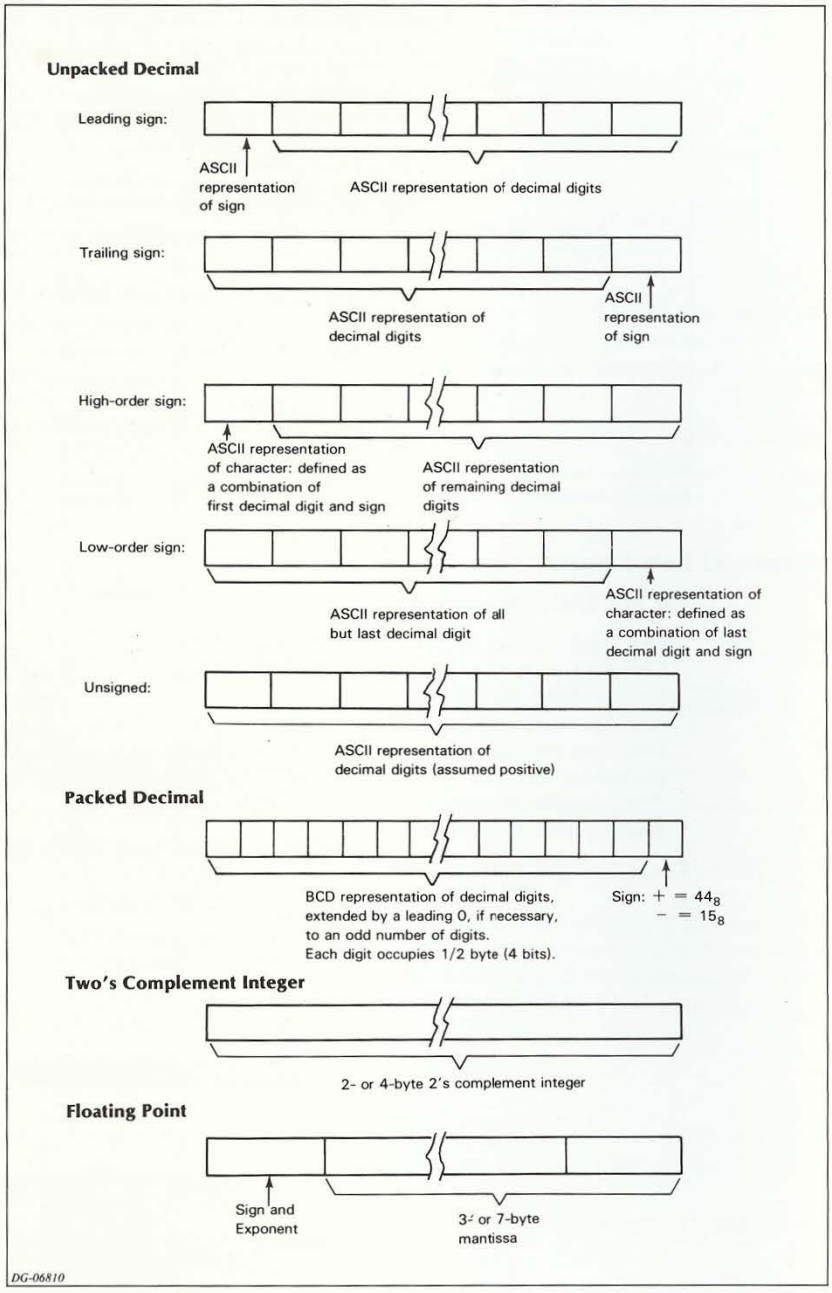


Figure 5.3 Commercial data types

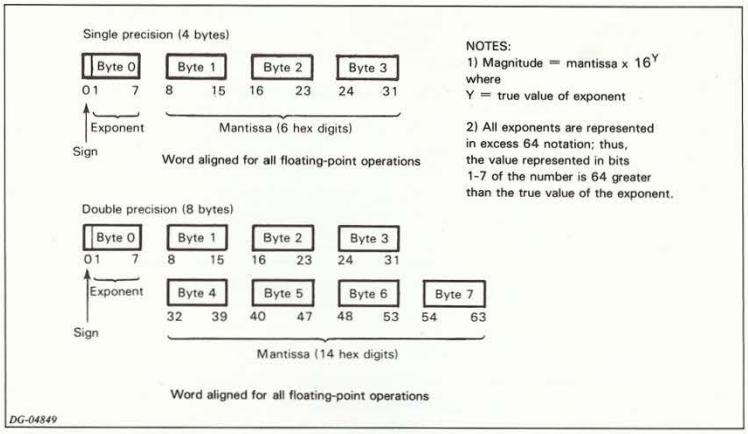


Figure 5.2 Floating-point data formats

Commercial Data Formats

The MV/8000 system can handle all standard commercial data types:

Unpacked decimal, leading sign;
Unpacked decimal, trailing sign;
Unpacked decimal, high-order sign;
Unpacked decimal, low-order sign;
Unpacked decimal, unsigned;
Packed decimal;
Two's complement integer, byte aligned;
Floating point, byte aligned.

Figure 5.3 shows the formats for these data types.

which can include many blocks of data, proceeds without intervention by the program; and *burst multiplexor channel I/O* operates similarly to data channel operations — I/O instructions are required only to set up the transfer of many blocks of data.

An example of an I/O instruction is the **VCT** *Vector on Interrupting Device Code* instruction. This instruction returns the device code of an interrupting device and uses that code as an index into a table of pointers to the appropriate interrupt handler. Depending on the user-specified mode of the instruction, it can also save the state of the machine by pushing various words onto the stack, create a new vector stack, and set up an interrupt priority structure.

Chapter 6
Peripherals

The MV/8000 system is designed to work in a heavily-loaded data processing environment, concurrently supporting timesharing and multiple batch stream processes. It is therefore dependent on fast, efficient, and reliable I/O.

Much of the MV/8000 internal architecture, in fact, is designed to improve I/O performance — for example, the I/O processor and data control unit both aid overall system performance by adding I/O capability. For more information on the MV/8000 I/O subsystem, see the discussion of the I/O subsystem in Chapter 3 and the block diagram on the inside back cover.

The same environment that requires an excellent subsystem also requires excellent peripherals: an active demand paging system requires large and fast discs; large and frequent file backups require fast and reliable tape drives; and a busy and varied user environment requires reliable and flexible user terminals.

Because the MV/8000 I/O subsystem is both program- and electrically-compatible with previous 16-bit ECLIPSEs, the full line of standard Data General field-proven peripherals is available to MV/8000 users. These peripherals provide the capability and reliability required by this system.

This chapter summarizes the capabilities of these peripherals. For complete information about peripherals for the MV/8000 system, consult your Data General salesperson.

Disc Storage Subsystems

Disc subsystems in the MV/8000 system support demand paging, virtual address spaces, and the file structure. Data General's line of peripherals includes large moving-head disc storage subsystems and fast fixed-head Winchester technology disc subsystems.

Figure 6.1 Disc storage subsystem

One example of the moving-head discs available is the 6122 DG/Disc Storage subsystem. This Data General-designed and -manufactured disc storage subsystem consists of two controller boards and up to four stand-alone disc drives. Maximum capacity of a 6122 subsystem is 1.1 billion bytes, using four removable-pack drives with a capacity of 277 million bytes each.

The MV/8000 system can support up to six 6122 subsystems, for a total of 6.6 billion bytes of on-line storage. The data transfer rate of each subsystem on the burst multiplexor channel (BMC) is 1.2 million bytes/second; average access time is 43.3 milliseconds.

An example of a fixed-head disc subsystem is the 6064 Fixed-Head DG/Disc subsystem. This high-performance disc uses Winchester technology to ensure reliability. Each subsystem consists of two controller boards and up to four drives with a capacity of 2 million bytes each. The MV/8000 system can support up to six 6064 subsystems. The data transfer rate of each subsystem on the burst multiplexor channel (BMC) is 910,000 bytes/second; average access time is 10.12 milliseconds.

Table 6.1 lists some of the disc storage subsystems available with the MV/8000 system.

Device and Model Number	Capacity	Typical Application*
6122 Moving-head disc subsystem	277 million bytes/drive	Data storage for large systems
6061-H Moving-head disc subsystem	190 million bytes/drive	Data storage for large systems
6060-H Moving-head disc subsystem	96 million bytes/drive	Data storage for medium to large systems
6066-H Fixed-head disc subsystem	4 million bytes/subsystem	High-speed data storage and retrieval
6064-H Fixed-head disc subsystem	2 million bytes/drive	High-speed data storage and retrieval
6097 Dual double-sided, double-density diskette	1.26 million bytes/drive	Data transfer

Table 6.1 Typical MV/8000 disc storage subsystems

*Each of these models can have up to four drives per subsystem.

Magnetic Tape Subsystems

Magnetic tape subsystems support file backup and transfer functions. Two magnetic tape subsystems are available with the MV/8000 system.

Figure 6.2 Magnetic tape subsystem

The 6026 Magnetic Tape subsystem is the MV/8000's standard dual-mode tape storage system, designed and manufactured by Data General. The 6026 supports both 1600-bpi phase-encoded and 800-bpi NRZI operation. Each 6026 subsystem consists of a controller board and up to eight cabinet-mounted tape transports.

Internal controller buffering of up to eight bytes of data minimizes channel overrun on systems with multiple high-performance storage devices. The data transfer rates over the data channel are 120,000 bytes/second at 1600 bpi and 60,000 bytes/second at 800 bpi.

The optional 6027 Magnetic Tape subsystem is a single-mode (800-bpi) tape storage system that is otherwise identical to the 6026 subsystem. It may be upgraded to a dual-mode system by adding a dual-mode tape transport.

User Terminals

User terminals support the interface between either the system users or the system operator and the system. All user terminals on the MV/8000 system connect to the I/O processor via asynchronous terminal interfaces (ATIs) or asynchronous modem interfaces (AMIs). The operator's terminal is controlled by the system control processor (SCP). Data General manufactures a variety of display and hardcopy terminals to serve all user needs.

Figure 6.3 User terminal

Two examples of the terminals available are the DASHER D100 and D200 display terminals. These are 96-character ASCII alphanumeric display terminals. The displays can provide reduced intensity, underscore, blink, and reverse video. Baud rates are selectable between 50 and 19,200 baud.

The DASHER D100 keyboard can transmit 96 upper- and lower-case symbols of the ASCII character set, 30 control characters, and 35 user-defined special function sequences. The D200 keyboard can transmit all of the above, plus 40 additional special function sequences.

Both terminals can be ordered with an optional interface for a slave printer. This interface also allows independent selection of transmit and receive baud rates.

Table 6.2 lists some of the terminals available with the MV/8000 system.

Device and Model Number	Typical Application/Special Feature
Display Terminals	
6107 DASHER D100 Display Terminal	User's or operator's terminal; selectable data rates to 9600 bps; 35 user-defined special function codes; blink, dim, underscore, reverse video; EIA printer interface for attaching DASHER TP1 or TP2 printer
6109 DASHER D200 Display Terminal	User's or operator's terminal; selectable data rates to 9600 bps; 75 user-defined special function codes; blink, dim, underscore, reverse video; EIA printer interface for attaching DASHER TP1 or TP2 printer
Printer Terminals	
6040 DASHER TP1 Terminal Printer	60-character/second terminal printer; 132-column, 5 x 7 dot matrix; EIA serial interface; 128 upper- and lower-case ASCII characters
6077 DASHER TP2 Terminal Printer	180-character/second logic-seeking, bidirectional terminal printer; 132-column, 7 x 9 dot matrix; selectable 6/8 lines/inch; expanded printing; horizontal and vertical tabs; EIA and 20-mA interfaces

Table 6.2 Typical MV/8000 user terminals

Line Printers

Line printers provide most of the hardcopy output for a large system. The line printers available with the MV/8000 system connect to the data channel for high-speed operation and low system overhead.

One line printer available is the 4245 DG Printer subsystem. This subsystem consists of a data channel controller board and a stand-alone printer. The printer is capable of printing 660 lines per minute of upper- and lower-case characters. Horizontal tabbing and vertical format are both program-controllable.

Table 6.3 lists the printers available with the MV/8000 system.

Device and Model Number	Printing Speed	Features*
4244 Line Printer Subsystem	900 lines per minute	136 columns, 64 ASCII upper-case characters
4245 Line Printer Subsystem	660 lines per minute	136 columns, 96 ASCII upper- and lower-case characters
4215 Line Printer Subsystem	600 lines per minute	136 columns, 64 ASCII upper-case characters
4216 Line Printer Subsystem	436 lines per minute	136 columns, 96 ASCII upper- and lower-case characters
4218 Line Printer Subsystem	300 lines per minute	136 columns, 64 ASCII upper-case characters
4219 Line Printer Subsystem	240 lines per minute	136 columns, 96 ASCII upper- and lower-case characters

Table 6.3 Typical MV/8000 line printer subsystems

*All of these line printer subsystems offer 6 or 8 lines/inch.

DataGeneral

Engineering Publications Comment Form

Please help us improve our future publications by answering the questions below. Use the space provided for your comments.

Title: _____

Document No. 014-000650-01

Yes	No		
☐	☐	Is this manual easy to read?	○ You (can, cannot) find things easily. ○ Other: ○ Language (is, is not) appropriate. ○ Technical terms (are, are not) defined as needed.
		In what ways do you find this manual useful?	○ Learning to use the equipment ○ To instruct a class. ○ As a reference ○ Other: ○ As an introduction to the product
☐	☐	Do the illustrations help you?	○ Visuals (are,are not) well designed. ○ Labels and captions (are,are not) clear. ○ 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.)	

Name: _____ Title: _____

Company: _____ Division: _____

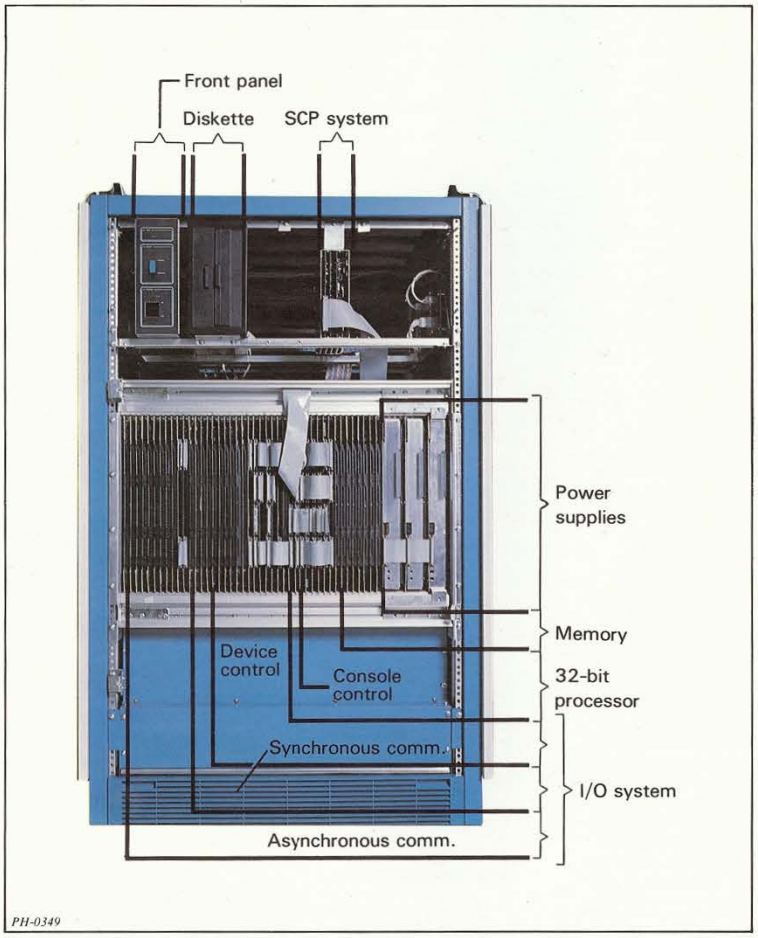
Address: _____ City: _____

State: _____ Zip: _____ Telephone: _____ Date: _____

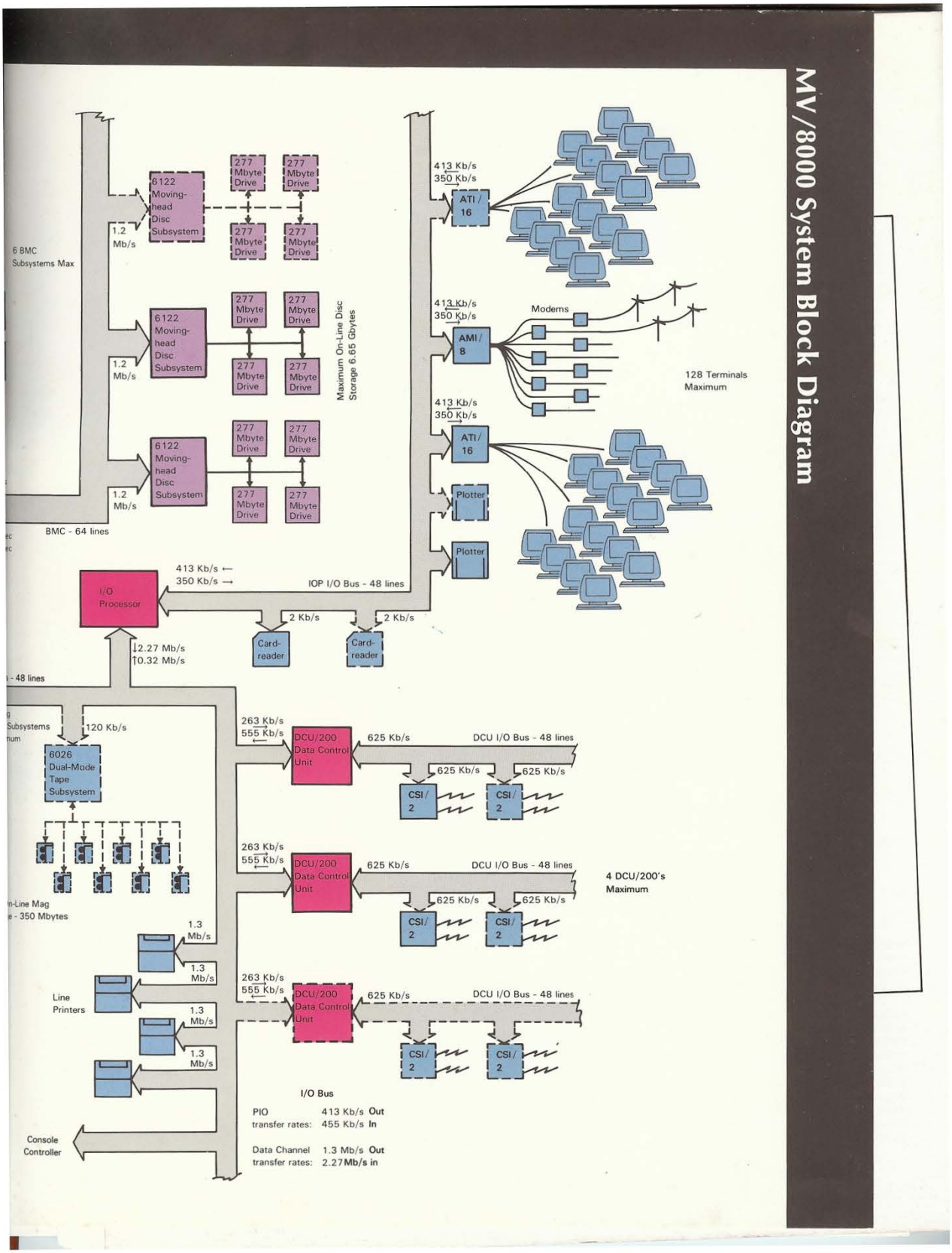
DG-06895

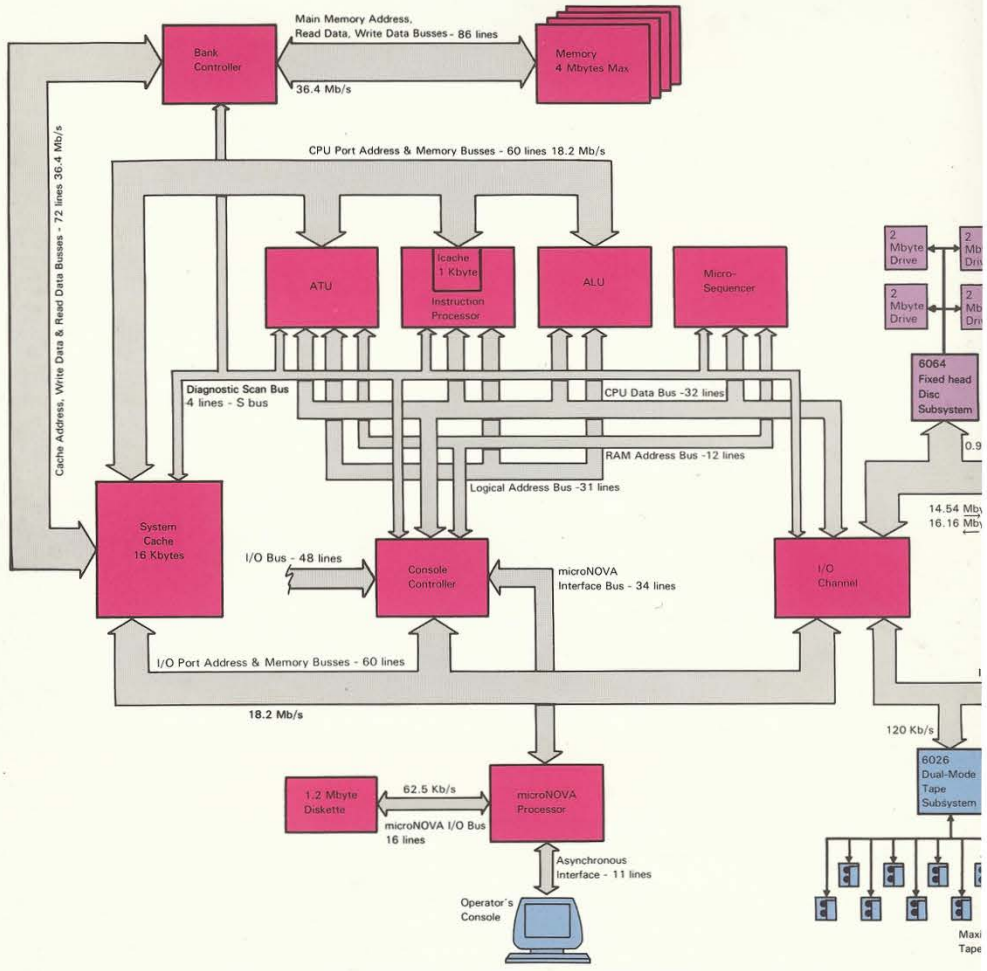
DataGeneral

Data General Corporation, Westboro, Massachusetts 01580


Front panel
Diskette
SCP system
Power supplies
Memory
Device control
Console control
32-bit processor
Synchronous comm.
I/O system
Asynchronous comm.
PH-0349

MV/8000 System Block Diagram




Main Memory Address,
Read Data, Write Data Busses - 86 lines
Bank Controller
Memory
4 Mbytes Max
36.4 Mb/s
Cache Address, Write Data & Read Data Busses - 72 lines 36.4 Mb/s
CPU Port Address & Memory Busses - 60 lines 18.2 Mb/s
ATU
Icache
1 Kbyte
Instruction Processor
ALU
Micro-
Sequencer
2 Mbyte Drive
2 Mbyte Drive
6064
Fixed head
Disc
Subsystem
Diagnostic Scan Bus
4 lines - S bus
CPU Data Bus -32 lines
RAM Address Bus -12 lines
Logical Address Bus -31 lines
System
Cache
16 Kbytes
I/O Bus - 48 lines
Console
Controller
microNOVA
Interface Bus - 34 lines
I/O
Channel
I/O Port Address & Memory Busses - 60 lines
18.2 Mb/s
120 Kb/s
6026
Dual-Mode
Tape
Subsystem
1.2 Mbyte
Diskette
62.5 Kb/s
microNOVA I/O Bus
16 lines
microNOVA
Processor
Asynchronous
Interface - 11 lines
Operator's
Console

DG-07319

DG OFFICES

SALES AND SERVICE OFFICES

Alabama: Birmingham
Arizona: Phoenix, Tucson
Arkansas: Little Rock
California: El Segundo, Fresno, Palo Alto, Sacramento, San Diego, San Francisco, Santa Ana, Santa Barbara, Van Nuys
Colorado: Englewood
Connecticut: North Branford
Florida: Ft. Lauderdale, Orlando, Tampa
Georgia: Norcross
Idaho: Boise
Illinois: Peoria, Schaumburg
Indiana: Indianapolis
Kentucky: Louisville
Louisiana: Baton Rouge
Maryland: Baltimore
Massachusetts: Springfield, Wellesley, Worcester
Michigan: Southfield
Minnesota: Richfield
Missouri: Kansas City, St. Louis
Nevada: Las Vegas
New Hampshire: Nashua
New Jersey: Cherry Hill, Wayne
New Mexico: Albuquerque
New York: Buffalo, Latham, Melville, Newfield, New York, Rochester, Syracuse, White Plains
North Carolina: Charlotte, Greensboro
Ohio: Columbus, Dayton, Brooklyn Heights
Oklahoma: Oklahoma City, Tulsa
Oregon: Portland
Pennsylvania: Blue Bell, Carnegie
Rhode Island: Rumford
South Carolina: Columbia
Tennessee: Knoxville, Memphis
Texas: Austin, Dallas, El Paso, Ft. Worth, Houston
Utah: Salt Lake City
Virginia: McLean, Norfolk, Richmond, Salem
Washington: Kirkland
West Virginia: Charleston
Wisconsin: West Allis

Australia: Melbourne, Victoria
France: Le Plessis Robinson
Italy: Milan, Padua, Rome
The Netherlands: Rijswijk
New Zealand: Auckland, Wellington
Sweden: Gothenburg, Malmoe, Stockholm
Switzerland: Lausanne, Zurich
United Kingdom: Birmingham, Dublin, Glasgow, London, Manchester
West Germany: Filderstadt, Frankfurt, Hamburg, Munich, Ratingen, Rodelheim

MANUFACTURER'S REPRESENTATIVES & DISTRIBUTORS

Argentina: Buenos Aires
Costa Rica: San Jose
Ecuador: Quito
Egypt: Cairo
Finland: Helsinki
Greece: Athens
Hong Kong: Hong Kong
India: Bombay
Indonesia: Jakarta
Iran: Tehran
Israel: Givatayim
Japan: Tokyo
Jordan: Amman
Korea: Seoul
Kuwait: Kuwait
Lebanon: Beirut
Malaysia: Kuala Lumpur
Mexico: Mexico City
Nicaragua: Managua
Nigeria: Lagos, Ibadan
Norway: Oslo
Peru: Lima
Philippine Islands: Manila
Puerto Rico: Hato Rey
Saudi Arabia: Riyadh
Singapore: Singapore
South Africa: Johannesburg, Pretoria
Spain: Barcelona, Bilbao, Madrid, San Sebastian, Valencia
Taiwan: Taipei
Thailand: Bangkok
Uruguay: Montevideo
Venezuela: Maracaibo

ADMINISTRATION, MANUFACTURING RESEARCH AND DEVELOPMENT

Massachusetts: Cambridge, Framingham, Westboro, Southboro
Maine: Westbrook
New Hampshire: Portsmouth
California: Anaheim, Sunnyvale
North Carolina: Research Triangle Park, Johnston County

Hong Kong: Kowloon, Tai Po
Thailand: Bangkok

DG-04976

Data General

Data General Corporation, Westboro, Massachusetts 01580

An example of fixed-point manipulation is the **LWDIV** *Wide Divide From Memory (Long Displacement)* instruction. This instruction divides a 32-bit integer in an accumulator by a 32-bit integer in a memory location, placing the quotient in the accumulator. Along with the instruction, the programmer specifies the address of the memory word and the number of the accumulator. An overflow flag indicates out-of-range quotients.

Floating-Point Instructions

The 87 floating-point instructions manipulate single-precision (32-bit) and double-precision (64-bit) floating-point numbers. Single- and double-precision numbers are handled interchangeably without the need for explicit conversion instructions.

To increase the accuracy of floating-point operations, one or two hex guard digits are used during the execution of most floating-point instructions. The result is then truncated or rounded, depending on the state of a flag controlled by the programmer.

An example of a floating-point operation is the **LFMMD** *Multiply Double (FPAC By Memory) (Long Displacement)* instruction. This instruction multiplies a 64-bit floating-point number in a specified floating-point accumulator (FPAC) by a 64-bit floating-point number in a specified memory location. The result is rounded or truncated and placed in the FPAC.

System Control Instructions

The following groups of instructions are used primarily by the operating system. They significantly enhance operating system performance by executing complex system operations in one instruction.

Privileged Instructions

The nine privileged instructions perform various operating system memory management functions. These instructions can only be executed in segment 0, where the kernel of the operating system resides. Privileged instructions flush the address translation unit, manipulate and test the page-modified and page-referenced bits, change the segment base registers, and control vectoring on an interrupt.

An example of a privileged instruction is the **PATU** *Purge the ATU* instruction. This instruction purges the entire address translation unit of all entries.

Queue Instructions

The 35 queue instructions manipulate priority-based queue structures and linked lists. Queues in the MV/8000 consist of individual data elements, each containing a forward link and a backward link, as well as user information in a form determined by the programmer. Each queue also consists of a queue descriptor, which contains pointers to the head and the tail of the queue.

Queue instructions add, remove, or search for elements of a queue or linked list. Enqueue and dequeue instructions are not interruptible, thus ensuring that a queue will not be used while in a transition state. Because search queue instructions can take some time to execute, they are interruptible during portions of their execution. The queue instructions, while used by various portions of the operating system, are available to all users.