1.1 INTRODUCTION

The MASCOR/132 Data Processing System is a general-purpose system designed to achieve optimum cost/performance in a wide range of application environments.

The MASCOR/132 system is particularly suited to diversified environments; i.e., environments where a variety of applications (commercial, engineering, scientific, etc.) and a variety of operational modes (interactive, remote batch, local batch, etc.) are simultaneously present.

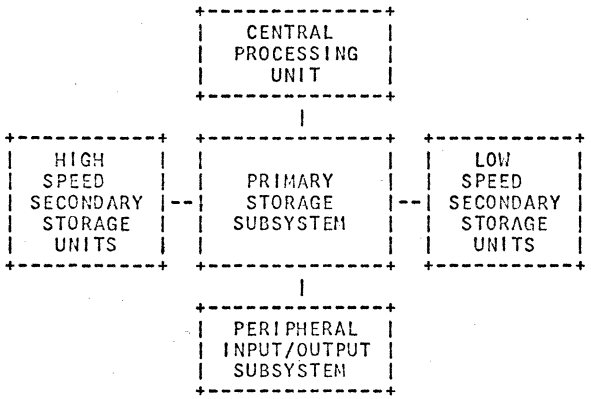


Figure 1.1.1. Gross Configuration of the MASCOR/132 System.

The gross configuration of the system is represented in Figure 1.1.1. The central component of the system is the Primary Storage Subsystem (PSS). It is the main memory for the Central Processing Unit (CPU), and it is the medium through which data can be moved to or from the High-Speed Secondary Storage Units, the Low-Speed Secondary Storage Units, and the Peripheral Input/Output Subsystem.

The primary storage subsystem and the high- and low-speed secondary storage unit form the basic storage system which will contain most of the data relevant to the system operation.

The central processing unit is the main processing unit of the MASCOR/132 system. It handles all major computational and processing functions. Under normal circumstances it controls the flow of data through the basic storage system.

One of the major design philosophies of the MASCOR/132 system, which is apparent in the organization of both the hardware and the software, is the distinction between storage devices on the one hand and input/output devices on the other. We classify monolithic and core memories, drums, disks, and other similar devices as storage devices. We classify line printers, card read and punch equipment, communication lines, terminals, microfilm output devices, removable disks, etc., as input/output devices.

The peripheral input/output subsystem consists of all such input/output devices in conjunction with a number of control units and of peripheral processing units required for the control of such devices and for the movement of data between the devices and the primary storage system.

1.2 PRIMARY STORAGE SUBSYSTEM

The Primary Storage Subsystem (PSS) is the central memory for the MASCOR/132. It is characterized by two major features:

a. It uses a buffered storage organization.

b. The data contained in it are referenced with a system virtual address by all units that have access to the PSS.

The buffered storage organization allows the subsystem to exhibit a low average access time at a cost comparable with the cost of considerably slower storage modules.

The uniform addressing by all units by means of a virtual address allows for simplicity of communication among different subsystems and allows efficient storage utilization. The PSS itself has the capability to translate such addresses into physical storage locations.

The 8-bit byte is the unit of addressing in the PSS. Data access however can be in multiple of bytes. The most frequently used multiple byte fields are the word (4 bytes) and the double word (8 bytes). Such multiple byte fields must consist of consecutive bytes whose first byte address is on the corresponding byte boundary.

1.2.1 Organization

The organization of the PSS is shown in Figure 1.2.1.1; the PSS includes three major components: 1) main storage modules, 2) the storage control unit, and 3) the buffer storage unit. The Main Storage Modules (MSM) have a relatively slow cycle time (typically within the 0.5 to 2 microsecond range). The MSMs can, however, within one cycle read or write multiple words in parallel. Such a group of words is referred to as a line, and consists typically of 16, 32, or 64 adjacent bytes. These lines are brought into and taken out of the buffers that are part of the PSS.

Each MSM line contains, in addition to the data words, an error correcting code The code is generated upon a write by the storage control unit and checked each time that is fetched. If a correctable error is found, correction will take place before the data are transferred to the requesting unit. All error occurrences are signalled so that the supervisor system can monitor and log such errors to allow knowledgeable preventive maintenance.

The Storage Control Unit (SCU) is responsible for coordinating the movement of data lines among the MSMs and the buffer storage unit. All transfers through the SCU are on a line basis.

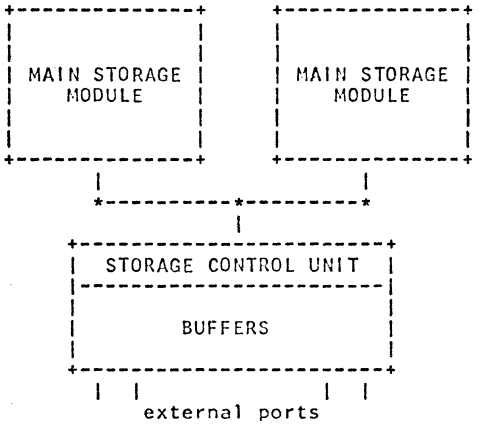


Figure 1.2.1.1. Primary Storage Subsystem.

The External Ports of the PSS are the interface with the outside world. Central processing units, peripheral processing units, and input/output channels are connected to the PSS through its external ports.

The external port interface allows for both data-transfer and control procedures. Not all ports, however, have identical capabilities. In particular, the data width may vary from one port to another.

The Buffer Storage Unit (BSU) provides fast acces to the most frequently used data. Whenever the PSS receives a request for a data item, it will determine if the line which includes the data item is in the buffer storage unit. In that case, it will satisfy the request immediately. If that is not the case, the SCU will transfer the corresponding line from the appropriate MSM to one of the buffer areas and

then satisfy the request. The line will be kept in the BSU until its location is required to bring a new line in. At that time, if no change to the data has been made, the buffer area containing the old line is made available; otherwise, the line will have to be rewritten into the appropriate MSM before the new line can be brought in.

If the frequency of finding the data in the buffers is sufficiently high, the net effect is that the requesting unit will see an average access time very close to the one of the buffers (Which typically will be 1/4 or 1/8 of the access time of the MSM). The fact that such an average will be obtained from a mixture of two quite different access times must be taken into account by all units connected to the PSS.

1.2.2 System Virtual Address

Each elementary addressable data item (byte) in the PSS is identified by a 40-bit System Virtual Address (SVA). An SVA is valid if it is associated with a location in a MSM. Locations in a MSM are identified by the Physical Storage Address (PSA). The transformation of SVAs to PSAs is referred to subsequently as mapping.

For proper operation of the PSS at most one SVA may correspond to a given PSA and vice versa, at most one PSA may correspond to a given SVA.

The SVA (see Figure 1.2.2.1) can be separated into a 16-bit segment identifier, 11-bit page identifier, a 5-bit paragraph identifier, and an 8-bit byte identifier.

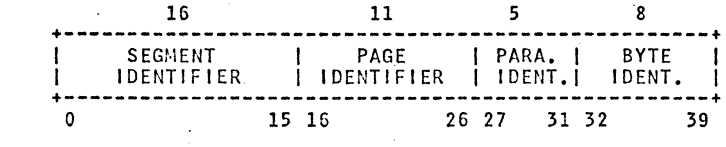


Figure 1.2.2.1. System Virtual Address.

The total system virtual address space is therefore viewed as consisting of 65536 segments, each of 2048 pages, each page consisting of 32 paragraphs, each of 256 bytes. Table 1.2.2.1 summarizes the units that define the address space.

Table 1.2.2.1. System Virtual Address Space.

	Segments	Pages	Paragraphs	Words	Bytes
Total Space	65536	2**27	2**32	2**38	2**40
Segment	1	2048	2**16 (64K)	2**22 (4M)	2**24 (16M)
Page	--	1	32	2048	8192
Paragraph	--	--	1	64	256
Word	--	--	--	1	4

Each element in the above hierarchy plays a certain role. The byte is the smallest addressable entity; the word is the standard addressable element. The paragraph is the smallest unit of mapping; i.e., PSAs always have the 8 low-order bits identical to the corresponding bits of the associated virtual address. The page is the standard unit of mapping; i.e., normally data elements within a page will have contiguous PSAs. The fact that a page can contain between 1 and 32 paragraphs (or 256 to 8192 bytes), allows tailoring of pages to natural boundaries of the program structure, without incurring waste due to large unused portions of pages.

The role of the segment is not related to the primary storage system per se, but to the organization of the CPU; we will discuss it in detail in Section 1.3.2.

A special role is assigned to SVAs whose first 16 bits (the segment identifier) are all zeros. Such addresses are interpreted by the PSS as actual PSAs. The use of such addresses is restricted to certain special situations.

1.2.3 Address Translation or Mapping

The translation from SVAs to PSAs is performed by the storage control unit through the use of the Primary Storage Directory (PSD). The PSD is a table residing in the Primary Storage Subsystem itself, addressed directly by PSAs (i.e., SVAs whose first 16 bits are all zeros). This table contains a number of Primary Storage Directory Entries (PSDE). Each PSDE provides the information required to map a page (or a subset of a page in units of paragraphs) into the corresponding physical storage addresses. In addition, such entries contain control information required for proper supervision of data transfer operations. The PSD is organized to allow fast lookup using coding technique implemented in the hardware of the PSS. The storage control unit does not, however, have to reference the PSD proper for each reference to the main storage modules. Within the SCU there is a High-Speed PSD which will contain the most frequently referenced directory entries and which uses associative memory organization to provide the necessary information in a large majority of the cases.

In review, we note that address translation is performed by the storage control unit. This implies that address translation is invoked only when the SCU itself is invoked--namely whenever a requested data item is not found in the buffers. This, in turn, implies that the buffer storage unit is organized totally on the basis of system virtual addresses; i.e., as long as data elements are present in a buffer area, no address translation is required. When address translation is required it may frequently be done in the High-Speed PSD.

1.3 CENTRAL PROCESSING UNIT

The Central Processing Unit (CPU) decodes the programmed instructions and manipulates data accordingly. The CPU of the MASCOR/132 is characterized by three major features:

a. It has 64 general-purpose registers.

b. It identifies data internally by means of a program virtual address distinct from the system virtual address.

c. It has an extensive instruction set which is able to handle a number of data formats in a variety of addressing modes and data lengths.

The large number of general-purpose registers allows a high degree of optimization in a variety of different data processing environments.

Addressing through the program virtual address allows distinct processes to reference the same data element in different ways.

The CPU is primarily word oriented. A full set of double-word operations is provided. It can also operate very efficiently on byte strings.

1.3.1 General-Purpose Registers

There are 64 32-bit General Purpose Registers (GPRs), numbered from 0 to 63. Each GPR can be used for any normal operational purpose--namely as a base register, an index register, a data register (for either logical, integer, floating-point, or decimal data) and as a condition register.

Arithmetic and logical operations can take place either between GPRs or between a GPR and data in storage. In the latter case, the result may either go to a register or be returned into storage. When the data items are of double word length then an even/odd register pair is used.

1.3.2 Program Virtual Address

CPU instructions reference data in storage through a Program Virutal Address (PVA). The PVA (see Figure 1.3.2.1) consists of a 2-bit field of zeros, a 6-bit segment number, and a 24-bit byte displacement.

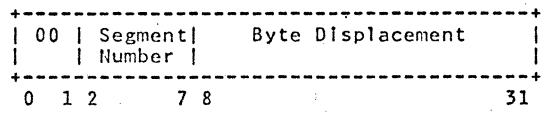


Figure 1.3.2.1. Program Virtual Address.

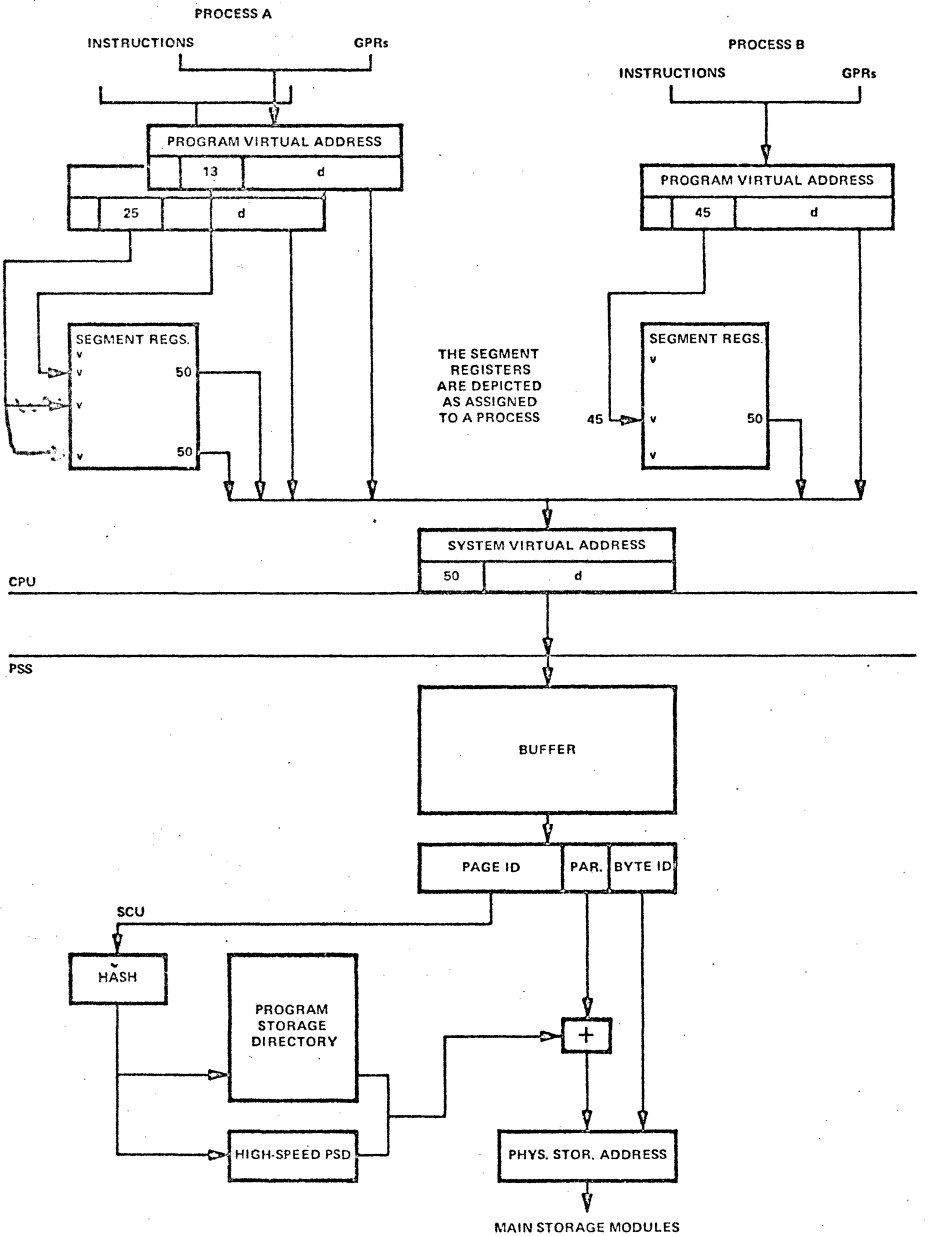
The segment number refers to one of 64 Segment Registers (SGRs). Several segment numbers may be used in a process which then will have the corresponding entries in the segment register marked as valid by the supervisory system (see Figure 1.3.4.1).

Each of the valid SGRs contains a 16-bit field (segment identifier field). The CPU translates a PVA into a SVA by replacing the high-order byte of the PVA with the segment identifier field of the SGR selected by the segment number.

Independent processes can refer to identical data by different segment numbers, provided that the segment registers are set appropriately. The byte displacement of the resulting SVA is identical to that of the PVA.

In order to understand the interplay of PVA and SVA, we will consider a simple example. (See also Figure 1.3.2.2). Let S be the 16-bit identifier associated with some data segment. Some portion of the addresses within that segment will be valid; i.e., there will be physical storage addresses associated with them. This is determined by the presence of entries in the primary storage directory with the appropriate information. A bank of segment registers values is associated with each process (such a bank would be represented in storage and loaded into the actual segment registers whenever that process is given control). Let's assume that process A wants to reference segment 50 as segment 13. The only thing which would have to be done is to set the segment identifier field of A's segment register 13 to the value 50. If some section of process A should address 50 as segment number 25, the appropriate value would be inserted also as the segment identifier field of A's segment register 25. If another process, say B, wants to access the same segment 50 as segment number 45, the segment identifier

field of B's segment register 45 would be set to 50. In none of the above assignments was any change to the primary storage directory involved.


PROCESS A
INSTRUCTIONS
GPRs
PROGRAM VIRTUAL ADDRESS
13
d
25
d
SEGMENT REGS.
v
v
50
v
v
50
PROCESS B
INSTRUCTIONS
GPRs
PROGRAM VIRTUAL ADDRESS
45
d
SEGMENT REGS.
v
45
v
50
v
THE SEGMENT REGISTERS ARE DEPICTED AS ASSIGNED TO A PROCESS
SYSTEM VIRTUAL ADDRESS
50
d
CPU
PSS
BUFFER
PAGE ID
PAR.
BYTE ID
SCU
HASH
PROGRAM STORAGE DIRECTORY
+
HIGH-SPEED PSD
PHYS. STOR. ADDRESS
MAIN STORAGE MODULES

Let us now consider the reverse situation in which, under the same circumstances described previously, it is desired to change the location in physical storage of some of the data of segment S. In order to do so, we would have to modify the primary storage directory accordingly. However, no change in any segment register would be required. In other words, the PVA/SVA organization totally decouples the control of access to data by the CPU processes from the control of placement of data in physical storage.

The scheme we have described is not unique in having the above described properties. However, our particular organization permits the CPU to access data in the buffer with the access speed of the buffer, while using virtual addressing rather than physical addressing.

1.3.3 Formation of Program Virtual Addresses

The PCU can form a program virtual address in a number of ways--namely a PVA is obtained:

a. directly from a GPR

b. by "address addition" (defined below) of a base GPR and an index GPR

c. by address addition of a base GPR and an immediate field in an instruction

d. directly from the instruction address register

e. by address addition of the instruction address and an immediate field in an instruction. The procedure of address addition involves two fields in a noncommutative order (see (Figure 1.3.3.1). The first field is obtained from either a general-purpose register (the base register) or the instruction address. The second field is obtained from either a general-purpose register (the index register) or by appropriate extension of the immediate field in an instruction.

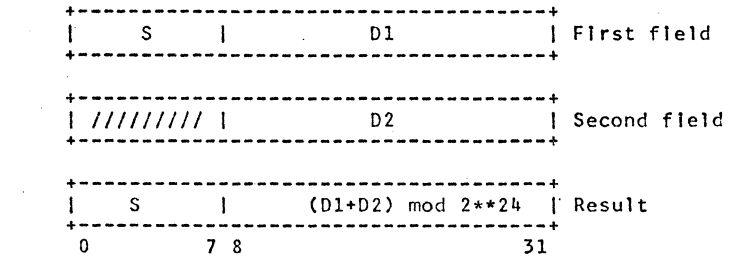


Figure 1.3.3.1. Address Addition.

The first 8 bits of the result containing the segment number are obtained from the corresponding bits of the first field without any change. The first field must already be in the form of a valid PVA. The lower-order 24 bits of the result are obtained by addition, modulo 2**24 of the corresponding bits of the two fields.

This procedure guarantees that no address computation of addresses using index or displacement values can result in accessing any other segment but the one referenced by the first field. Since any GPR can be used directly as a PVA, and since full arithmetic capability is available on any GPR, it is also possible, if desired, to compute addresses across segment numbers.

1.3.4 Data Access Control

Any segment that is accessed by a process has available a number of control fields which are used by the CPU to control access. These control fields are part of the segment register. The control mechanism not only assigns different access to different processes, but it also allows various levels of access control within a process. The full structure of each segment register is given in Figure 1.3.4.1.

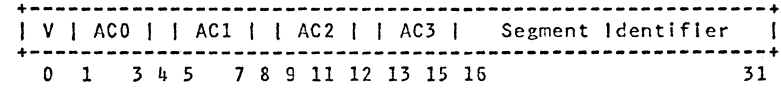


Figure 1.3.4.1. Segment Register.

Bit 0 indicates the validity of the register. The low-order 16 bits are the segment identifier field which we have discussed in Section 1.3.2. The five 3-bit Access Control Fields determine the access to the given segment. Only one of the five fields is active at any given time, as determined by the control vector (see Section 1.3.5). Each access control field is viewed as in Figure 1.3.4.2.

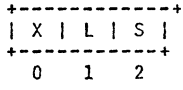


Figure 1.3.4.2. Access Control Field.

Each bit controls one mode of access to the data (0 allows access; 1 forbids it) as described in Table 1.3.4.1).

Table 1.3.4.1. Access Mode Control.

Bit	Access Mode
X	Instruction Fetch
L	Data Fetch
S	Data Store

This approach allows each process to have independently determined modes of access to each segment, as specified by the segment register used.

The five access control fields are used to determine different modes of access within the same process. This is useful in order to allow routines with different privileges to operate within a process; i.e., library, data entry and edit, or computational functions. The active access control field will be the same for all segment registers.

1.3.5 Program Status Block

The basic state of the CPU is determined by the segment registers, the general-purpose registers, and the three Program Status Registers (PSRs). The Program Status Block (PSB) consists of the three PSRs, plus the contents of GPR 0. The program status block represents all of the information that is switched whenever an interrupt condition is recognized.

The three PSRs are the Control Vector (CV), the Instruction Address Register (IAR), and the Iteration Counter (ITC).

The CV determines various modes of operation of the CPU and certain privileges.

In particular, as we said before, the CV will determine which of the five access control fields in the segment registers is active at any given time.

Part of the control vector is accessible in non-privileged mode and contains information related to unusual conditions in arithmetic operations.

The remainder of the control vector is inaccessible in non-privileged mode and it is used by the supervisory system to control input/output functions, timing and other privileged functions.

The instruction address register contains the program virtual address of the instruction in progress. Since instructions are always one word long and are always on word boundaries, the low-order 2 bits of the IAR are zero.

The iteration counter allows the CPU to interrupt and then properly resume certain types of instructions (mainly byte string operations).

1.3.6 Data and Instruction Formats

As a function of the operation involved, the CPU will interpret data according to a number of formats.

Addresses: Program virtual addresses (see Figure 1.3.2.1).

Logical: 32- and 64-bit strings. Operations bit by bit.

Unsigned Integers: 32- and 64-bit binary representation of unsigned (non_negative) integers; with provision for extended precision.

Signed Integers: 32- and 64-bit binary 2s complement representation of signed integers; with provision for extended precision.

Floating-Point Numbers: 32- and 64-bit quantities consisting of a sign bit, a 7-bit excess-64 hexadecimal exponent, and 24- or 56-bit hexadecimally normalized fraction.

Byte Strings: Byte strings consisting of from 0 to 2**24 contiguous bytes.

Decimal: 32- and 64-bit quantities consisting of a string of binary coded decimal digits in 10's complement representation; with provision for extended precision.

For each of these datatypes and lengths, appropriate load, store, and comparison operations are provided. Every data element's program virtual address must have boundaries consistent with the length of the data element.

All instructions are one word long and must reside on word boundaries. There are 4 basic instruction formats (Figure 1.3.6.1), namely the M, I, V and N formats.

In all formats, the fields identified by C contain the operation code. In M-, I-, and V-format instructions and in most N-format instructions, fields i and j identify general-purpose registers. The d and sd fields are interpreted as a byte displacement. The wd, extended to the right with two zeros, is interpreted as a word displacement. The k field is interpreted as either a general-purpose register identifier or an immediate 6-bit unsigned integer.

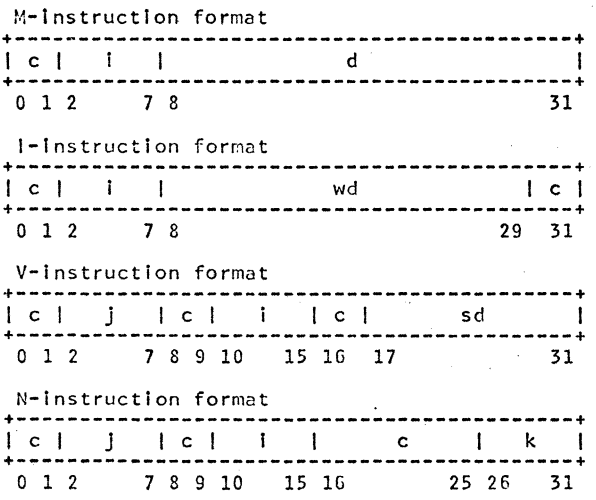


Figure 1.3.6.1. Instruction Formats.

There is only one M-format instruction which loads the byte displacement in GPR i.

This instruction provides fast loading of a register with displacement or other constants of up to 24 bit length.

There are 4 I-format instructions; the wd field is address added to the instruction address to form a new address.

There are 11 V-format instructions; in each of them the GPR identified by the j field is used as a base register to which the sd field is address added to form the address for load and store operations. These instructions allow direct referencing to data in segments that are pointed at by base registers.

Instructions in the N formats have typically three operands: Operand 1 is the result operand, while Operands 2 and 3 are the source operands. Table 1.3.6.1 gives the list of operand selector modes. The following notation is used:

Ri, Rj, Rk	GPR i, GPR j, GPR k
K	Immediate field K
@	Address addition
C(X)	Contents of location X

Table 1.3.6.1. Operand Selector Modes for Type N.

Mode	Operand 1	2	3
RRR Register to Register	Ri	Rj	Rk
RRK Register with Constant to Register	Ri	Rj	K
RSR Storage with Register to Register	Ri	C(Rj)	Rk
RSK Storage with Constant to Register	Ri	C(Rj)	K
RXR Indexed Storage with Register to Register	Ri	C(Rj@Rk)	Ri
RXK Displaced Storage with Register to Register	Ri	C(Rj@K)	Ri
XRR Register with Indexed Storage to Indexed Storage	C(Rj@Rk)	C(Rj@Rk)	Ri
XKR Register with Displaced Storage to Displaced Storage	C(Rj@K)	C(Rj@K)	Ri

The operands may be GPRs or references to storage. The third operand may also be an immediate 6-bit constand contained in the instruction itself. References to storage can be made using the contents of a GPR, by address addition of two GPRs, or by address addition of a GPR and the immediate field in the instruction.

Many instruction types allow multiple of these operand selector modes.

1.3.7 Non-Privileged Operations

The non-privileged operations include an extensive set of arithmetic and logical operations. The major characteristics are listed below:

a. Total parallelism between word and double word operations on logical, unsigned integers, signed integers, binary as well as decimal, and floating-point numbers is provided.

b. An extensive set of unsigned integer operations allows efficient coding of arbitrary precision arithmetic.

c. Both truncated and rounded floating-point arithmetic are available.

d. Decimal arithmetic using ten's complement arithmetic with convenient handling of multi-word operands.

e. Compare operations use a GPR to store a TRUE (all ones) or FALSE (all zeroes) result.

f. Byte string move and compare operations are designed to be particularly suited to text editing and pattern matching.

1.3.8 System Control Registers

The CPU has a number of System Control Registers (SCRs) used for the control of various functions. They are only accessible in privileged mode. Following is a list of the major functions associated with these registers.

Definition of Status Switching Save Areas

Two registers contain the pointers required upon interrupt for the storing of the old program status block and the fetching of the new one.

Interval Timing and Instruction Counting

Two interval timers allow for setting up times at which interrupts are to be generated. Two instruction counters allow an interrupt to be generated after executing a specified number of instructions.

Interrupt Masking

A set of masks permits control over the occurrence of internal and external events. Internal events can be generated under program control.

Logic Trace Capability

A register will hold when desired the PVA of the last branch instruction which caused a branch. Furthermore, the CPU can be set up to interrupt on any successful branch.

Address Reference Monitoring

A register can be set to any PVA, and it will cause an interrupt whenever that address is referenced for any reason.

Clock

The real-time clock is able to keep track of the time in the outside world with microsecond accuracy. It is connected to an external unit which may be common to many CPUs.

1.3.9 External Interface

In addition to manipulating the data in the PSS, the CPU has to be able to communicate with the various input/output subsystems. In order to do this the CPU can interface directly to the outside world through a basic external interface facility. Instructions and datapaths are provided here that allow the following functions:

1. Accepting single-bit signals with interrupt and masking capability.
2. Sending bit signals.
3. Accepting a full-word input.
4. Sending a full-word output.

Through the external interface the CPU will exchange control information with the high-speed secondary storage units, the standard I/O channels, and the peripheral

processing units. The actual transfer of data and programs is carried out by those units, through their ports to the primary storage system.

1.3.10 Interrupt System

The interrupt system allows various classes of interrupts to be handled separately. By distinguishing user-oriented service calls and program exceptions from system-oriented service calls and program exceptions, fast decoding and handling is provided for usages of the interrupt system as an adjunct to normal processing.

There are eight classes of interrupts.

1. Utility Service Call
2. System Service Call
3. Program Exception 0
4. Program Exception 1
5. Internal Signal
6. External Signal
7. Hardware Malfunction (Warning)
8. Power Off Warning

Whenever any of the above conditions occur, the active program status block is saved in a pre-defined area and a new PSB is loaded also from a pre-defined area.

The control vector may determine if certain interrupt conditions must be ignored.

The Utility Service Call (USC) instruction allows a non-privileged program to call upon another non-privileged program (with normally different access capability).

The System Service Call (SSC) instruction allows the non-privileged program to call upon a privileged program.

The Program Exception interrupts is invoked on any addressing, arithmetic, and similar exceptions. The two classes allow it to distinguish between such occurrences while in "user mode" and while in "system mode."

Internal Signals are those associated with the interval timers and instruction counters.

External Signals are those associated with the external interface.

Internal and External Signals are individually maskable.

The Hardware Malfunction interrupts are generated whenever such occurrences arise. They allow the distinction between corrected errors and uncorrectable errors.

1.4 HIGH-SPEED SECONDARY STORAGE UNIT

The role of the High-Speed Secondary Storage Unit (HSSSU) is to provide the system with a secondary storage unit with high access rate and high transfer rate. This facility is essential in a multiuser environment to allow many processes to be available within a low access time and provide both fast response time for interactive processes and efficient usage of system facilities by active processes.

Typical physical units making up the HSSSU are head-per-track disk and drum devices.

The HSSSU is organized into a number of modules, each module consisting of a number of sectors. Each module within a type of unit has the same number of sectors. Each sector contains a 32-byte key and a binary power of 256-byte data blocks. The unit of writing is the sector; the unit of reading is the data block.

A Typical HSSSU may consist of 64 modules, each with 64 sectors of 512 words for a total of approximately 2 million words. Typical transfer rate is in the neighborhood of 2 million words/sec.

The number of HSSSU's which can be attached to a primary storage subsystem is a function of the particular model of the PSS. A typical system would probably have two HSSSU.

Each HSSSU can be simultaneously connected to a number of PSSs for its data transfers and to a number of processors for its control. Only one such path is active for control purposes at any given time, but it can be switched under program control.

The HSSSU fetches its commands from a given primary storage subsystem and stores its status and the data identifying key in the same PSS. It can, however, transfer data to or from any PSS to which it is physically connected.

The commands to be executed are four words long and contain the required system virtual addresses for both the key and the data. A total of six commands are provided: four control commands and two data-transfer commands.

1.5 STANDARD I/O CHANNEL

The Standard I/O Channel (SIOC) is the basic interface to the outside world for attachment of medium speed devices (like disks) and buffered low-speed devices (line printers, card read/punch units, etc.). The data transfers executed by the SIOC are block or record oriented. Although each channel can support only one data transfer at a time, it can support a large number of simultaneous channel programs from a control point of view.

Each of these channel programs will control one subchannel and the data transfers of these may be sequentially interleaved on a block basis.

The SIOC presents to the outside world an interface compatible with that of many popular storage and I/O devices.

The SIOC has the capability of being simultaneously connected to a number of different primary storage subsystems for its data transfers and to a number of processors for its control. Only one such path is active for control purposes at any given time, but it can be switched under program control.

The SIOC fetches its commands from the controlling PSS and stores its status in the same PSS. It can, however, transfer data to or from any PSS to which it is physically connected. The commands to be executed are four words long.

A typical SIOC is able to support devices with a transfer rate of up to 4 million bytes/sec. The number of SIOCs which can be attached to a primary storage subsystem is a function of the particular model but will typically be in the range of 4 to 16.

1.6 LOW-SPEED SECONDARY STORAGE SUBSYSTEM

The Low-Speed Secondary Storage Subsystem (LSSSS) is supported by the standard I/O channel.

The devices supported on the LSSSS are principally magnetic drum, disk, or tape strip devices which are part of the resident storage facility of the system.

The data are organized in a way that the supervisory system can flexibly assign storage in a manner that is compatible with optimal usage of the data capacity of the particular devices. A hierarchical storage organization is envisaged where frequently accessed data, indexes, etc., will reside on higher-speed devices.

A typical LSSSS may consist of a number of disk drives each having the capability of storing 25 million bytes and a non-erasable, laser-perforated strip device capable of storing 100 billion bytes.

1.7 PERIPHERAL PROCESSING UNIT

In a system of the scope of the MASCOR/132 data processing system there are a number of functions which are enough specialized that they could not use the full capability of the central processing unit but, at the same time, are not specialized enough to justify special purpose hardware component. Such functions are handled by the Peripheral Processing Units (PPU).

Each PPU is a small general purpose processor with its own private memory. The amount of memory associated with each PPU varies according to the mode of use of that PPU.

Each PPU is typically connected to one or more control units which will enable the PPU to transmit and receive data and control signals to and from a number of input/output devices (including communication lines). In addition some control unit will allow the transfer of data between the primary storage subsystem and the PPU memory.

1.8 PERPHERAL INPUT/OUTPUT SUBSYSTEM

The Peripheral Input/Output Subsystem (PIOS) consists of peripheral processing units, channels, control units, devices and communication lines used by the system to communicate with the outside world.

The external interface of the standard I/O channel is such that there is a wide range of I/O devices which can be connected to it either directly or through an appropriate control unit. These include magnetic tape units, printers, card read/punch units, magnetic character readers, optical character readers, display units, audio response units, etc. Figure 1.8.1 gives an example of a simple configuration.

Communication lines, some types of slow speed I/O devices and, in general, devices not matching the external interface of the standard I/O channel is connected to the system through a Local Interface Unit (LIU) which is in turn controlled by a PPU. LIUs are located at the central site with the CPU and the PSS and have the capability for interfacing to the PSS.

Typically only relatively high speed communication lines is connected to a LIU (1200 to 9600 bauds). Remotely located terminals is connected through low or high speed lines to a Remote Interface Unit (RIU) which is also normally controlled by a PPU. Typically the RIUs is located away from the central system site and will communicate with it through relatively high speed communication lines connected to a central LIU. A RIU will also be able to control directly certain low speed devices. Figure 1.8.2 gives an example of a simple configuration.

LIUs and RIUs can be freely interconnected to provide an integrated multi-path network. The presence of the PPUs as controlling elements makes it possible to achieve a high degree of flexibility and to introduce in the system any required degree of redundancy and checking required to guarantee the reliability of the data transmission.

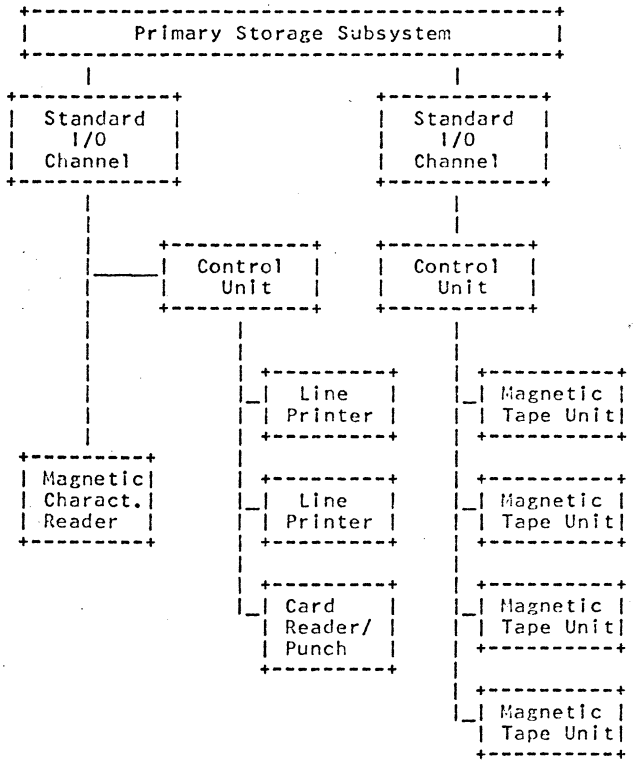


Figure 1.8.1. Example of I/O Device Set Up.

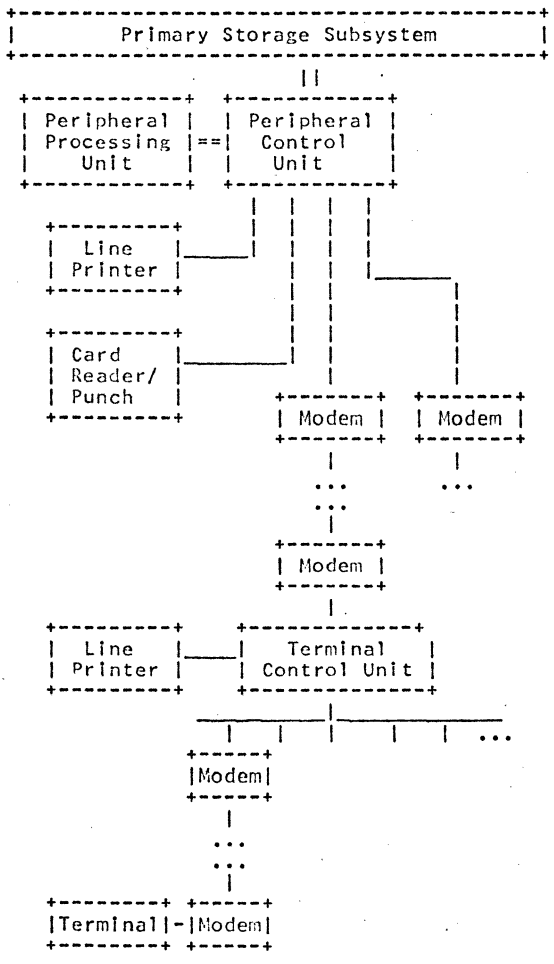


Figure 1.8.2. Example of Communication Set Up.

1.9 SYSTEM MONITOR UNIT

The MASCOR/132 system is designed to provide a high degree of availability and reliability.

To this end channels, control units, storage units and devices may be provided with multiple interconnection capabilities. It is therefore possible to configure a system in such a way that no single component failure will stop the system. In addition, most units have extensive error detection circuitry and are in some cases capable of performing single error correction

The supervision of the system configuration, the monitoring of detected errors and fault diagnosing and isolation is performed with the help of the System Monitor Unit (SMU) under the control of a PPU or a CPU.

Whenever a unit detects an error for which the unit cannot guarantee correct recovery, the unit will stop at the earliest possible moment and then send a signal to the SMU. The SMU has the necessary interfaces and control circuitry so that under the control of its associated processor, it can:

1. retry the operation, if possible, to detect transient errors;

2. if the error is non-transient or not retriable, proceed to the isolation of the failing unit from the rest of the system;

3. reconfigure the system interconnections if necessary;

4. log out the state of the failing unit and then proceed under a combination of program and human control to the isolation of the unit failure;

5. after the unit has been repaired, reconfigure the system to reinclude the unit.

1.10 SYSTEM CONFIGURATIONS

The MASCOR/132 Data Processing System units can be interconnected in a number of ways to form systems suited for a wide range of requirements. In this section we will discuss a few examples of possible configurations, mainly to illustrate the overall structure of the system.

There are two distinct sections of a given system, namely the central site system consisting of all the units which must be in close proximity to each other and the communication network consisting of all units which can be remotely located. The communication network and the central site system meet at the local interface units (see Section 1.8) which are themselves centrally located.

Figure 1.10.1 illustrates a simple central site configuration including:

1 Primary Storage Subsystem
1 CPU
1 Drum
2 Channels
1 Disk Control Unit

(with the capability of handling up to 8 disk drives)

1 Tape Control Unit

(with the capability of handling up to 8 tape drives)

1 Unit Record Devices Control Unit with

2 printers
1 card read/punch

1 Peripheral Processing Unit
1 System Monitor Unit
1 Local Interface Unit with

1 plotter
2 high speed modems

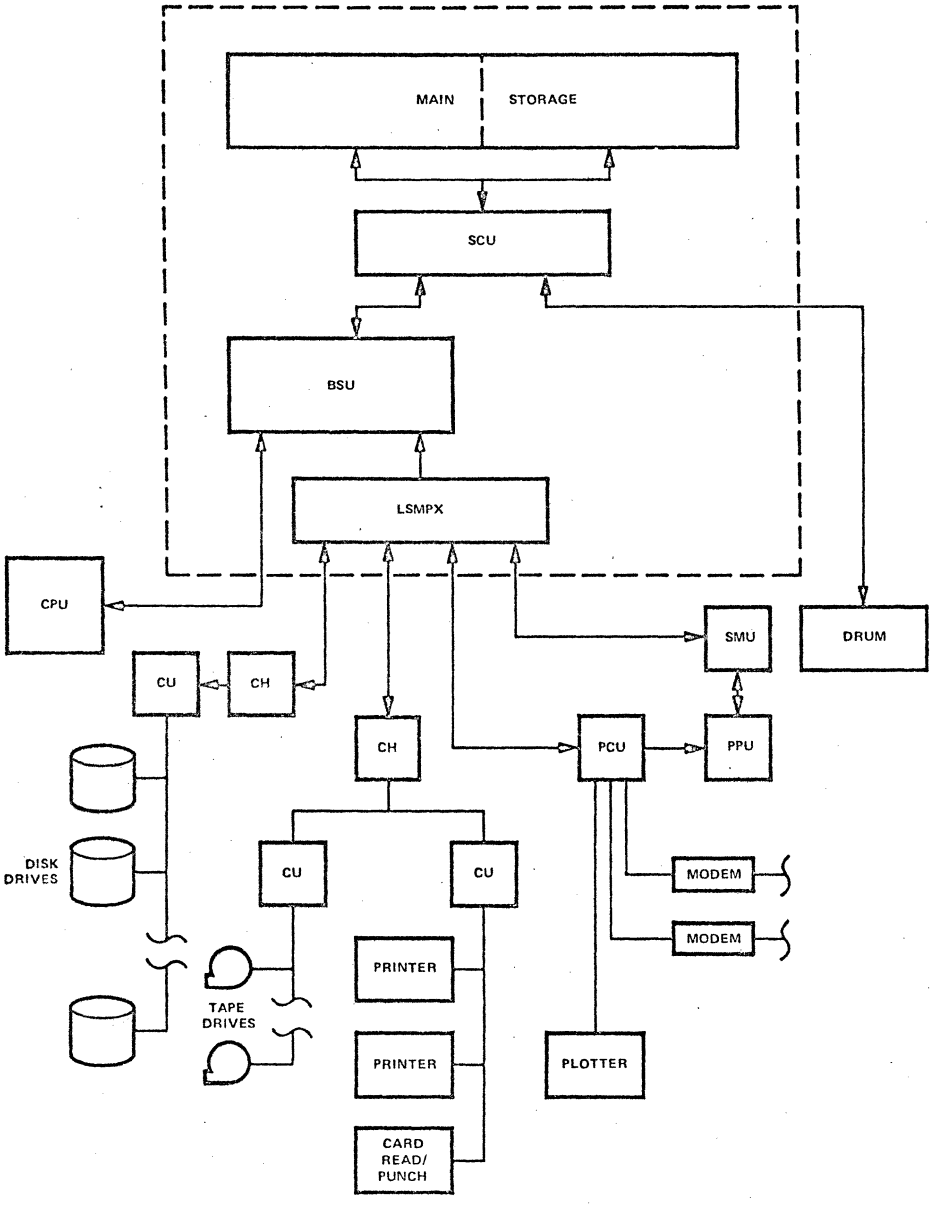
The above is probably the smallest practical configuration in a non-highly-specialized environment.

Figure 1.10.2 illustrates a large scale complex system configuration including:

2 Primary Storage Subsystems
4 CPU
6 Drums
8 Channels
6 Disk Control Units
1 Tape Control Unit
1 Unit Record Devices Control Unit
1 Optical Mass Storage Control Unit and Device
4 Peripheral Processing Units
2 Peripheral Control Units
2 System Monitor Units

The Main Purpose of Figure 1.10.2 is to show some of the multiple interconnection capabilities of the MASCOR/132 units. Note in particular that all secondary storage units are accessible from either primary storage subsystem.

Figure 1.10.3 gives an example of communication network indicating again the level of interconnection and the multipath capabilities.


MAIN
STORAGE
SCU
BSU
LSMPX
CPU
SMU
DRUM
CU
CH
CH
PCU
PPU
DISK
DRIVES
CU
CU
MODEM
MODEM
PRINTER
TAPE
DRIVES
PRINTER
PLOTTER
CARD
READ/
PUNCH

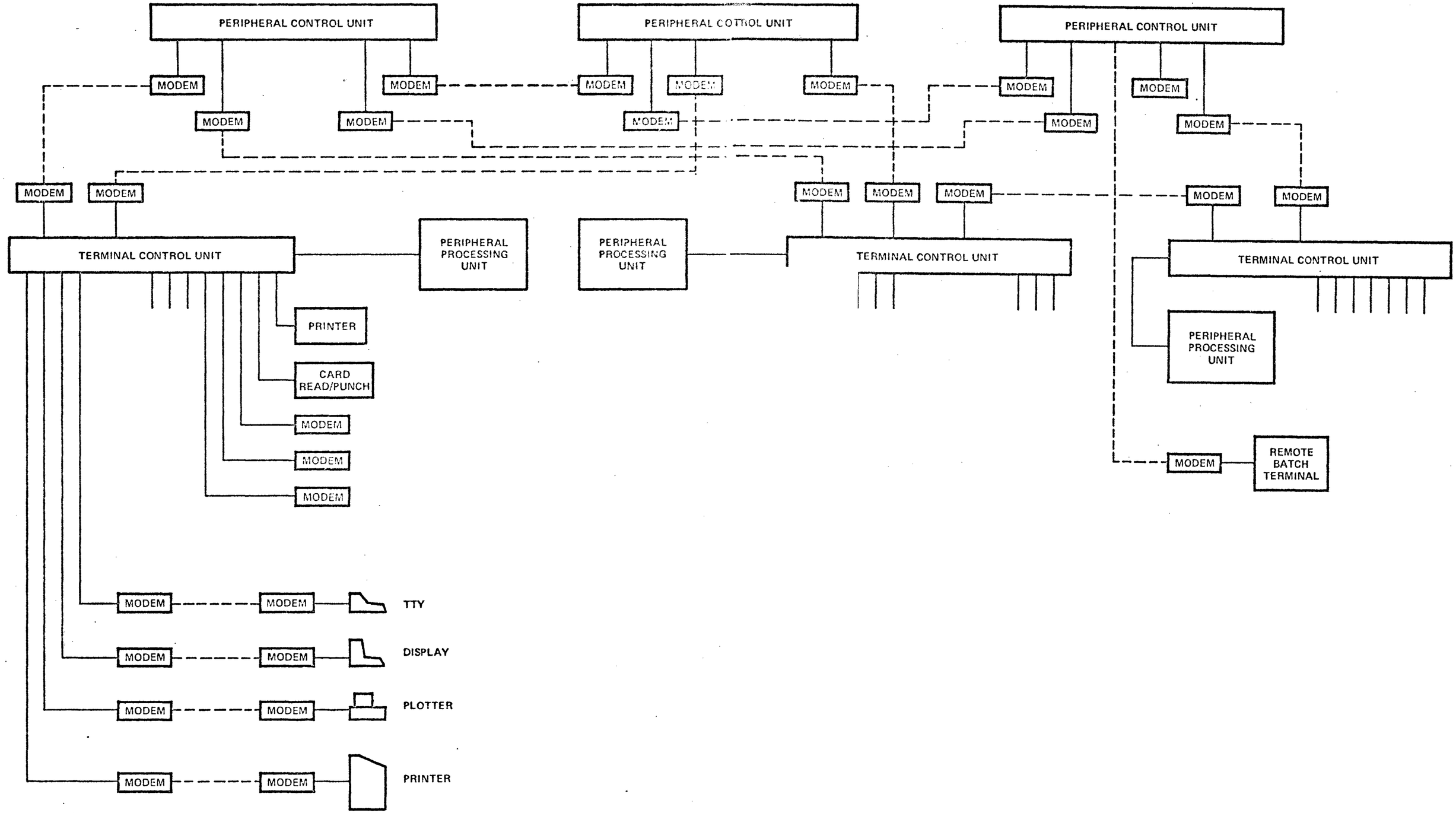

PERIPHERAL CONTROL UNIT
PERIPHERAL CONTROL UNIT
PERIPHERAL CONTROL UNIT
MODEM
TERMINAL CONTROL UNIT
PERIPHERAL PROCESSING UNIT
PERIPHERAL PROCESSING UNIT
TERMINAL CONTROL UNIT
TERMINAL CONTROL UNIT
PERIPHERAL PROCESSING UNIT
PRINTER
CARD READ/PUNCH
MODEM
MODEM
MODEM
MODEM
MODEM
REMOTE BATCH TERMINAL
TTY
DISPLAY
PLOTTER
PRINTER

Figure
VII-

2.1 INTRODUCTION

2.2 ORGANIZATION

2.2.1 Addressing of Data
2.2.2 Storage Control Unit

2.2.2.1 Organization of the PSD
2.2.2.2 Location of PSDE in the PSD
2.2.2.3 High Speed Directory

2.2.3 Buffers
2.2.4 External Ports

2.2.4.1 Command Register
2.2.4.2 System Virtual Address Register
2.2.4.3 Data Register
2.2.4.4 Byte Mask Register
2.2.4.5 Status Register

2.3 FUNCTIONAL SPECIFICATION

2.3.1 PSS Commands
2.3.2 Fundamental Procedures

2.3.2.1 Primary Storage Directory Search Procedure
2.3.2.2 Buffer Search Procedure
2.3.2.3 Release Entry Procedure

2.3.3 Data Commands

2.3.3.1 Fetch Command
2.3.3.2 I/O Fetch Command
2.3.3.3 Store Command
2.3.3.4 I/O Store Command
2.3.3.5 Fetch/Store Command
2.3.3.6 I/O Fetch/Store Command
2.3.3.7 Test and Set Lock Command
2.3.3.8 Reset Lock Command

2.3.4 Control Commands

2.3.4.1 Set Primary Storage Registers
2.3.4.2 Search Primary Storage Directory
2.3.4.3 Evaluate Hashing Function
2.3.4.4 Pause
2.3.4.5 Invalidate High Speed Directory

2.1 INTRODUCTION

The Primary Storage Subsystem (PSS) (see Figure 2.1.1) is designed to play the role of the common data storage facility in a data processing system consisting of a number of processing units. Typically, it will be connected to a variety of general or special purpose processors, input/output channels and peripheral units. The major components of PSS are:

1. The Main Storage Modules: The "main memory" of the system.
2. The External Ports: The interface between the PSS and the connected external subsystems. Through the external ports, data is transferred to and from the PSS, control commands are issued, and completion status is returned.
3. The Storage Control Unit: The address translation unit--since the main storage modules are addressed with the Physical Storage Address (PSA) while the PSS external ports use the System Virtual Address (SVA), the translation from SVA to PSA is done by the Storage Control Unit (SCU) through the Primary Storage Directory (PSD).
4. The Buffer Storage Unit: The colletion of all buffered storage modules in the PSS. Buffered storage modules are required for many reasons: to increase access performance, to match data transfer rates, to synchronize a channel or to meet other requirements. The Buffer Storage Unit is addressed with the System Virtual Address (SVA).

The next sections will describe the basic structure of the system and the basic characteristics of its major components.

It should be noted here that the detailed organization of the PSS is only of interest to the programmer developing code to run in privileged mode (see Chapter 3). The applications programmer may skip this chapter entirely, although he would probably find Sections 2.2.2.1 and 2.2.2.2 of some interest.

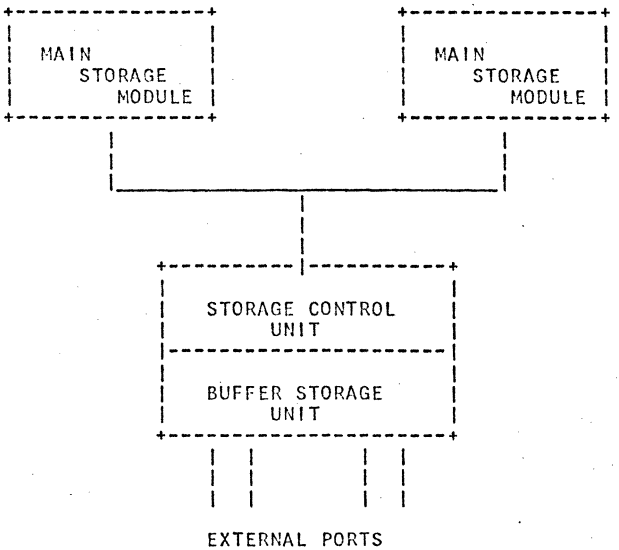
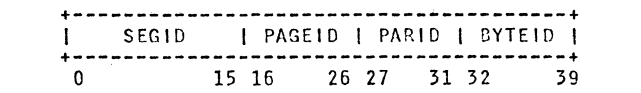


Figure 2.1.1 Primary Storage Subsystem

2.2 ORGANIZATION

2.2.1 Addressing of Data

Data in the PSS is addressed by the System Virtual Address (SVA). The SVA is a 40-bit address identifying a byte in the PSS. The format of the SVA is given in Figure 2.2.1.1.



BITS	FIELD NAME	FIELD SYMBOL
0-15	Segment Identifier	SEGID
16-26	Page Identifier	PAGEID
27-31	Paragraph Identifier	PARID
32-39	Byte Identifier	BYTEID

Figure 2.2.1.1 Format of the System Virtual Address

The SVA consists of a 16-bit segment identifier, a 11-bit page identifier, a 5-bit paragraph identifier and an 8-bit byte identifier. The total system virtual address space consists therefore of 65,536 segments, each consisting of 2,048 pages, each page consisting of 32 paragraphs of 256 bytes each. Table 2.2.1.1 gives a summary of the breakdown.

Table 2.2.1.1 System Virtual Address Space

	Segments	Pages	Paragraphs	Words	Bytes
Total Space	65536	2**27	2**32	2**38	2**40
Segment	1	2048	2**16 (64K)	2**22 (4M)	2**24 (16M)
Page	—	1	32	2048	8192
Paragraph	—	—	1	64	256
Word	—	—	—	1	4

The SVA is the only form of address by which data in the PSS can be referenced from the outside. Internally, however, the PSS addresses data through a 32-bit physical storage address (PSA). The PSA's are not available to the external PSS interface with the single exception that

If a SVA has a SEGID equal to zero, the SVA is interpreted as a PSA.

This implies that the first 2**24 bytes of data associated with the PSS are referenceable from the outside directly through their PSA.

For any given model of the PSS, there is a maximum installed PSA (PSAMAX) which, as its name indicates, is the maximum allowable value for a PSA.

2.2.2 The Storage Control Unit (SCU)

The main function of the Storage Control Unit (SCU) is to translate SVA into PSA. The correspondence between SVA's and PSA's in the general case (i.e., when SEGID¬=0) is established by the Primary Storage Directory (PSD).

2.2.2.1 Organization of the PSD

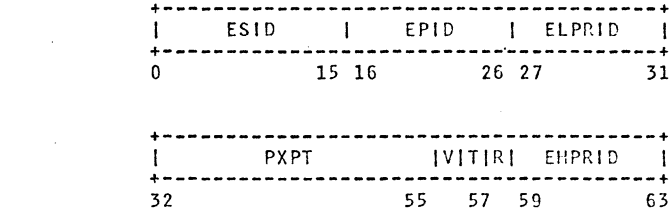
The PSD is a table contained in the PSS. It is addressable with a PSA whose value should not be greater than 2**24. (i.e., It can be addressed by SVA with SEGID = 0.)

There are two registers in the PSS which describe the PSD, namely, the Primary Storage Directory Origin Register (PSDOR) and the Primary Storage Directory Mask Register (PSDMR). Both registers are 16 bits wide.

The origin of the PSD is assumed to have a PSA which is a multiple of 256. By adding 8 zeroes to the right, the PSDOR will be assumed to contain the PSA of the PSD origin.

The length of the PSD is assured to be a multiple of 256 bytes. To insure proper operation of the PSD, the PSDMR must be loaded with a number of the form 2**K-1 for 0<=K<=16. The length of the PSD will be then determined as 2**K*256 bytes.

The PSD is viewed as consisting of a number of eight byte Primary Storage Directory Entries (PSDE) located on eight byte boundaries. The format of each PSDE is given in Figure 2.2.2.1.1.



BITS	FIELD NAME	FIELD SYMBOL
0-15	Entry Segment Identifier	ESID
16-26	Entry Page Identifier	EPID
27-31	Entry Low Paragraph Identifier	ELPRID
32-55	Physical Index Pointer	PXPT
56	Validity Control Flag	V
57	Transit Control Flag	T
58	Read Only Control Flag	R
59-63	Entry High Paragraph Identifier	EHPRID

Figure 2.2.2.1.1. Format of the Primary Storage Directory Entry.

A PSDE is said to be associated with an SVA (assumed in the format of Figure 2.2.1.1) if the following conditions hold

```
(V | T) = 1
ESID = SEGID
EPID = PAGEID
ELPRID <= PARID
EHPRID >= PARID
```

A PSDE is said to be valid if V = 1 and invalid if V = 0.

The first PSDE (in an order to be described later) which is associated with an SVA and which is also valid (if there are any) determines the PSA corresponding to the given SVA according to the algorithm

```
PSA:= (PXPT + PARID)||BYTEID
```

An SVA for which there is a corresponding PSA is said to be valid.

An entry is said to be not in use when both the values of V and T are zeroes.

2.2.2.2 Location of PSDE in the PSD

In order to facilitate their retrieval when necessary, a PSDE associated with a given SVA is located within the PSD according to the following fixed procedure.

There is a pseudo randomizing function, H, which maps a 27_ bit field into another 21 bit field. This function is used to evaluate the nominal PSD location associated with an SVA (NOML(SVA)) according to the following algorithm.

```
NOML(SVA):=(H(SEGID||PAGEID)&(PSDMR||B'11111'))||B'000'
```

NOML(SVA) is the displacement relative to the origin of the PSD of a doubleword PSDE. NOML(SVA) identifies the first PSDE which is a candidate to be associated with the specified SVA. Successive candidates' displacements will be obtained by adding eight (modulo directory size) to the previous displacement.

2.2.2.3 High Speed Directory

For reasons of performance some models of the PSS will maintain a High Speed Directory (HSD). The HSD contains a high speed version of some or all of the valid Primary Storage Directory Entries.

2.2.3 Buffer Storage Unit

The Buffer Storage Unit (BSU) is the collection of all buffer storage modules in the PSS. Logically, it may be viewed as consisting of a set of N entries, numbered from 1 to N. The number N as well as the data length of each entry (may vary from entry to entry) are functions of the particular PSS model.

Each Buffer Entry has two parts: the Buffer Tag and the Buffer Data. The Buffer Tag identifies the System Virtual Address (SVA) and length of the Buffer Data. The Buffer Tag contains two registers and three control flags.

1. Buffer System Virtual Address Identifier (BSVAID): A 40_bit register denoting the SVA of the first byte of the Buffer Data.
2. Buffer Byte Mask Identifier (BBMID): A 40-bit register with (40-k) zeroes followed by k ones. It specifies the length (2**k bytes) of the Buffer Data which is always assumed on proper SVA boundary.
3. Control Flags:

   BV: Buffer Validity Control Flag

   BT: Buffer Transit Control Flag

   BR: Buffer Read Only Control Flag

A Buffer Entry is created as a byproduct of a Data Command (fetch, store, etc.). The values of the control flags BT and BR are copies of the T and the R bits from the corresponding PSD entry at the time of creation.

2.2.4 External Ports

The PSS interfaces to the outside through the External Ports. The number of ports associated with a PSS is a function of the particular model. Restrictions on the length of the transferred data (byte mask) and maximum transfer rate may also vary from port to port.

Each port has five registers associated with it as listed in Table 2.2.4.1.

Table 2.2.4.1 External Port Registers

Register Name	Register Symbols
Command Register	CR
System Virtual Address Register	SVAR
Data Register	DR
Byte Mask Register	BMR
Status Register	SR

The role of each of the registers is described in more detail in the following sections.

2.2.4.1 Command Register

The Command Register (CR) is five bits wide allowing for 32 different commands to be issued to the PSS. The list of commands will be given in Sections 2.3.3 and 2.3.4.

2.2.4.2 System Virtual Address Register

The System Virtual Address Register (SVAR) is 40 bits wide and will contain a SVA which for data transfer commands will be the SVA of the first byte of the data element. The SVA should be on the boundary corresponding to the length of the data element as selected by the BMR.

2.2.4.3 Data Register

The Data Register (DR), in the case of data transfer commands, will contain the value of the data element to be transferred. In the case of control commands it is used to exchange control information between the PSS and the requesting unit.

The length of the DR is a function of the particular PSS model and may be different from port to port. In any case, however, the DR is at least 4 bytes and at most 256 bytes.

2.2.4.4 Byte Mask Register

The Byte Mask Register (BMR) is 40 bits wide with (40-k) zeroes followed by k ones. It specifies the length (2**k bytes) of the data element to be transferred for data transfer commands. The lengths allowed are a function of the particular PSS model and may be different from port to port. In any case, however, the length should be equal to or smaller than the length of the Data Register.

2.2.4.5 Status Register

The Status Register (SR) is four bits wide and is used by the PSS to transfer back to the requesting unit information regarding the completion of the command. Table 2.2.4.5.1 gives the meaning of assigned codes.

Table 2.2.4.5.1 Status Codes

CODE	MEANING	SYMBOL
0	Normal Completion	COMP
1	Ready for Store	RFS
2	PSD entry Not Found	NDE
4	T bit in PSD is 1	T1
8	R bit in PSD is 1	R1

2.3 FUNCTIONAL SPECIFICATIONS

2.3.1 Primary Storage Subsystem Commands

There are two classes of commands that the PSS accepts through its external ports, namely, data commands and control commands.

Restrictions on the commands which can be issued by a port as well as the maximum allowable command rate are functions of particular PSS models.

I/O data commands are intended to be used only in the block data transfer procedures. Such procedures are discussed in Section 2.4

Functional specifications of the commands are presented as SPLIT (*note) programs in the following sections. The variable declaration parts, however, are omitted since most identifiers have been specified in Section 2.2.

2.3.2 Fundamental Procedures

There are several fundamental procedures which are part of many command executions and are presented in this section.

2.3.2.1 Primary Storage Directory Search Procedure

The Primary Storage Directory Search (PSDSEARCH) procedure accepts a SVA as parameter and searches the PSD to determine whether the SVA is valid. If it is valid, the corresponding PSDE is resulted. Otherwise, the procedure sets the status codes to "PSD Entry Not Found" (SR=2).

```
        procedure PSDSEARCH (SVA);
         I:=NOT'L(SVA);

START:   K:=(PSDOR||B'00000000')+I;
         PSDE:=PSD(K);
         If ((V=0)&(T=0)) then go to NEF;
         If (ESID||EPID~=SEGID||PAGEID) then go to NEXT;
         If (ELPRID>PARID) then go to NEXT;
```

(*note): SPLIT is a programming language developed by MASCOR.

```
        If (EMPRID<PARID) then go to NEXT;
        If (V=0) then go to NEF;
        return;

NEF:    SR:=2;
        return;

NEXT:   I:=(I+8)&(PSDMR||B'11111111');
        go to START;
        end
```

Remarks:
1. The functional algorithm NOM'L is specified in Section 2.2.2.2.
2. PSD(K) denotes the 8 byte entry in the PSD with starting byte address (PSA) of K.

2.3.2.2 Buffer Search Procedure

The Buffer Search (BUFFERSEARCH) Procedure accepts a SVA and a BM (byte mask) as parameters. The SVA denotes the first byte address and the BM specifies the length of the data to be searched. The search procedure produces a Buffer Entry if any part of the searched data is found in the Buffer Entry.0A no entry found (NEF) flag is set if no match is found.0

```
        procedure BUFFERSEARCH (SVA,BM);
        I:=1;

START:  BUFFER_ENTRY :=BUFFER(I);
        If (BV=0) then go to NEXT;
        If (XOR(SVA,BSVAID)&(¬BM)&(¬BBMID)=0) then return;

NEXT:   If (I=N) then begin NEF:=1; return end
        I:=I+1;
        go to START;
        end
```

Remarks:
1. BUFFER(I) denotes the Ith Buffer Entry in the Buffer. (Ref. 2.2.3 for definitions)
2. XOR(x,y) denotes the bit-by-bit exclusive-or function of x and y. The identity

$$XOR(SVA,BSVAID)\&(\neg BM)\&(\neg BBMID) = 0$$

Is true if and only if byte strings (SVA,MR) and (BSVAID,BBMID) are overlapping. Since byte strings are always in proper 2**n boundary, they are overlapping only if one is contained in another.

2.3.2.3 Release Entry Procedure

The Release Entry Procedure (RELEASE) accepts a Buffer Entry as parameter and results in storing the data part of the entry (Buffer Data) into the proper location of Main Storage Modules.

```
        procedure RELEASE (BUFFER_ENTRY);

START:  If (BR=1) then go to FINIS;
         call PSDSEARCH (BSVAID);
         If (SR=2) then begin SR:=0; go to FINIS end
         PSA:=(PXPT+PARID)||BYTEID;
         STORAGE(PSA,BBMID):=BUFFER_DATA;
         BV:=0;
FINIS:   BV:=0;
         return
```

Remarks:

1.0 BR is the Buffer Read Only Control Flag of the Buffer Entry. It value is copied from the R bit of the corresponding PSD entry at the time of creating the Buffer Entry.

2.0 STORAGE (PSA,BBMID) denotes the storage location in the Main Storage Modules with a starting byte address PSA and a length specified by Buffer Byte Mask Identifier (BBMID).

3.0 BUFFER_DATA is the data part of the BUFFER_ENTRY (Ref. 2.2.3).

2.3.3 Data Commands

There are eight data commands, namely

~~Return~~

FETCH
I/O FETCH
STORE
I/O STORE
FETCH/STORE
I/O FETCH/STORE
TEST AND SET LOCK
RESET LOCK

Commands are issued to the Primary Storage Subsystem through its External Ports (Ref. 2.2.4). The commands are specified by the Command Register (CR) with the System Virtual Address Register (SVAR), Data Register (DR) and Byte Mask Register (BMR) containing the parameters. The information regarding the completion of the command is returned in Status Register (SR).

In any of the following Data Commands, if the Segment Identifier (SEGID) of the SVA specified by SVAR is zero, the call PSDSEARCH statement is bypassed (Ref. 2.2.1) and the corresponding checking on T and R bit (if any) are also ignored.

2.3.3.1 Fetch Command

```
START:   call BUFFERSEARCH (SVAR,BMR);
         if (NEF=1) then go to SCU;
         if ((BT=1)|(BBMID<BMR)) then

           begin call RELEASE(BUFFER_ENTRY);

             go to START;

           end

         DR:=BUFFER_DATA(SVAR,BMR);
         return;

SCU:     call PSDSEARCH (SVAR);
         if (SR=2) then return;
         if (T=1) then begin SR:=4; return end
         PSA:=(PXPT+PARID)||BYTEID;
         DR:=STORAGE(PSA,BMR);
```

```
        return;
```

Remark:
BUFFER_DATA(SVAR,BMR) denotes the part of the Buffer Data In the Buffer Entry with System Virtual Address specified by SVAR and length specified by BMR (Ref. 2.2.3).

2.3.3.2 I/O Fetch Command

```
START:  call BUFFERSEARCH (SVAR,BMR);
        if (NEF=1) then go to SCU;
        if ((BT=0)&(BR=1)) then begin BV:=0; go to START end
        if ((BT=0)&(BR=0)) then BT:=1;
        if (BBMID<BMR) then

          begin call RELEASE (BUFFER_ENTRY);

            to to START;

          end

        DR:=BUFFER_DATA(SVAR,BMR);
        return;

SCU:    call PSDSEARCH(SVAR);
        if (SR=2) then return;
        PSA:=(PXPT+PARID)||BYTEID;
        DR:=STORAGE(PSA,BMR);
        return;
```

2.3.3.3 Store Command

```
START:  call BUFFERSEARCH(SVAR,BMR);
        if (NEF=1) then go to SCU;
        if((BT=1)|(BR=1)|(BBMID<BMR)) then

          begin call RELEASE(BUFFER_ENTRY);

            go to START

          end
```

```
          BUFFER_DATA(SVAR,BMR):=DR;
          return;

SCU:      call PSDSEARCH(SVAR);
          if (SR=2) then return;
          if (T=1) then begin SR:=4; return end
          if (R=1) then begin SR:=8; return end
          PSA:=(PXPT+PARID)||BYTEID;
          STORAGE(PSA,BMR):=DR;
          return;
```

2.3.3.4 I/O Store Command

```
START:    call BUFFERSEARCH(SVAR,BMR);
          if (NEF=1) then go to SCU;
          if((BR=1)|(BBMID<BMR)) then

            begin BV:=0;

              go to START;

            end

          if (BT=0) then BT:=1;
          BUFFER_DATA(SVAR,BMR):=DR;
          return;

SCU:      SCU: call PSDSEARCH(SVAR);
          if (SR=2) then return;
          if (R=1) then begin SR:=8; return end
          PSA:=(PXPT+PARID)||BYTEID;
          STORAGE(PSA,BMR):=DR;
          return
```

2.3.3.5 Fetch/Store Command

```
START:      call BUFFERSEARCH(SVAR,BMR);
            if (NEF=1) then go to SCU;
            if ((BT=1)|(BR=1)|(BBMID<BMR)) then

              begin call RELEASE(BUFFER_ENTRY);
```

```
            go to START;

          end

        DR:=BUFFER_DATA(SVAR,BMR);
        SR:=1;
        wait for STORE command;

BSTORE: call BUFFERSEARCH(SVAR,BMR);
        if (NEF=1) then go to SCUSTORE;
        if ((BT=1)|(BR=1)|(BBMID<BMR)) then

          begin call RELEASE(BUFFER_ENTRY);

            go to BSTORE;

          end

        BUFFER_DATA(SVAR,BMR):=DR;
        return;

SCU:    call PSDSEARCH(SVAR);
        if (SR=2) then return;
        if (T=1) then begin SR:=4; return end
        if (R=1) then begin SR:=8; return end
        PSA:=(PXPT+PARID)||BYTEID;
        DR:=STORAGE(PSA,BMR);
        SR:=1;
        wait for STORE command;
        go to BSTORE;

SCUSTORE: call PSDSEARCH(SVAR);
        if (SR=2) then return;
        if (T=1) then begin SR:=4; return end
        if (R=1) then begin SR:=8; return end
        PSA:=(PXPT+PARID)||BYTEID;
        STORAGE(PSA,BMR):=DR;
```

2.3.3.6 I/O Fetch/Store Command

```
START:  call BUFFERSEARCH(SVAR,BMR);
        if (NEF=1) then go to SCU;
        if ((BR=1)|(BBMID<BMR)) then

          begin call RELEASE(BUFFER_ENTRY);
```

```
            go to START;

          end

        If (BT=0) then BT:=1;
        DR:=BUFFER_DATA(SVAR,BMR);
        SR:=1;
        wait for I/O STORE command;

BSTORE: call BUFFERSEARCH(SVAR,BMR);
        If (NEF=1) then go to SCUSTORE;
        If ((BR=1)|(BBMID<BMR)) then

          begin call RELEASE(BUFFER_ENTRY);

            go to BSTORE;

          end

        If (BT=0) then BT:=1;
        BUFFER_DATA(SVAR,BMR):=DR;
        return;

SCU:    call PSDSEARCH(SVAR);
        if (SR=2) then return;
        if (R=1) then begin SR:=8; return end
        PSA:=(PXPT+PARID)||BYTEID;
        DR:=STORAGE(PSA,BMR);
        SR:=1;
        wait for I/O STORE command;
        go to BSTORE;

SCUSTORE: call PSDSEARCH(SVAR);
        If (SR=2) then return;
        If (R=1) then begin SR:=8; return end
        PSA:=(PXPT+PARID)||BYTEID;
        STORAGE(PSA,BMR):=DR;
        return;
```

2.3.3.7 Test and Set Lock Command

```
START:  call BUFFERSEARCH(SVAR,BMR);
        If (NEF=1) then go to SCU;
        If ((BT=1)|(BR=1)|(BBMID<BMR)) then

          begin call RELEASE(BUFFER_ENTRY);
```

```
                go to START;

              end

            X:=BUFFER_DATA(SVAR,BMR);
            if (X<0>=1) then go to BFETCH;
            BUFFER_DATA(SVAR,BMR):=DR;

BFETCH:     DR:=X;
            return;

SCU:        call PSDSEARCH(SVAR);
            if (SR=2) then return;
            if (T=1) then begin SR:=4; return end
            if (R=1) then begin SR:=8; return end
            PSA:=(PXPT+PARID)||BYTEID;
            X:=STORAGE(PSA,BMR);
            if (X<0>=1) then go to SCUFETCH;
            STORAGE(PSA,BMR):=DR;

SCUFETCH:   DR:=X;
            return;
```

2.3.3.8 Reset Lock Command

The functional procedure of Reset Lock Command is identical to that of Store Command (Ref. 2.3.3.3). Further restrictions on execution and sequencing are specified for both Test and Set Lock and Reset Lock commands; please refer to Section 2.3.5 for details.

2.3.4 Control Commands

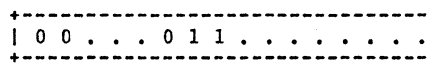
There are nine control commands, namely:

SET PRIMARY STORAGE REGISTERS
SEARCH PRIMARY STORAGE DIRECTORY
EVALUATE HASHING FUNCTION
PAUSE
INVALIDATE HIGH SPEED DIRECTORY
INVALIDATE HIGH SPEED DIRECTORY ENTRY
RELEASE PAGE
RELEASE SEGMENT
RELEASE BUFFER

2.3.4.1 Set Primary Storage Registers

```
START:   PSDOR:=SVAR<0-15>;
         PSDMR:=SVAR<16-31>;
         return;
```

Remark:
Bits 32-39 of SVAR are ignored. If bits 16-31 are not of the form



for some k, incorrect operation of the PSD may result. (Ref. 2.2.2.1 for the definitions of PSDOR and PSDMR).

2.3.4.2 Search Primary Storage Directory

```
         I:=NOML(SVAR);
START:   K:=(PSDOR||B'00000000')+I;
         PSDE:=PSD(K);
         if ((V=0)&(T=0)) then begin SR:=2; go to FINIS end
         if (ESID||EPID¬=SEGID||PAGEID) then go to NEXT;
         if (ELPRID>PARID) then go to NEXT;
         if (EHPRID<PARID) then go to NEXT;
FINIS:   DR:=B'0000000000000000'||K;
         return;
NEXT:    I:=(I+8)&(PSDMR||B'11111111');
         go to START;
```

Remark:
This command searches the first PSD entry associated with the given SVA, independently from the fact it is valid or not. If it finds one, it returns the PSA of that entry (B'0000000000000000'||K) in DR. If it does not find one, it returns the PSA of the next entry not in use. Such PSA may then be correctly used to make an entry associated with the given SVA. (Please compare with the PSDSEARCH procedure in Section 2.3.2.1.)

2.3.4.3 Evaluate Hashing Function

```
START: I:=H(SVAR<0-26>);
       DR:=B'0000000000000000'|| I ||B'000'
       return;
```

Remark:
The H function is the pseudo randomizing function which maps a 27 bit field into a 21 bit field and is used to evaluate the nominal PSD location associated with an SVA (see Section 2.2.2.2).

2.3.4.4 Pause

The Pause Command forces completion of all previous commands to that port before initiation of any of the commands following the Pause Command (see Section 2.3.5).

2.3.4.5 Invalidate High Speed Directory

The whole High Speed Directory is invalidated.

2.3.4.6 Invalidate High Speed Directory Entry

The High Speed Directory copy of the PSDE associated with the SVA specified by SVAR (if any) is invalidated.

2.3.4.7 Release Page

```
START: call BUFFERSEARCH(SVAR,BMR);
       if (NEF=1) then return;
       call RELEASE(BUFFER_ENTRY);
```

```
        go to START;
```

Remark:
SVAR contains the page address and BMR contains the page mask (K=13).

2.3.4.8 Release Segment

```
START: call BUFFERSEARCH(SVAR,BMR);
       If (NEF=1) then return;
       call RELEASE(BUFFER_ENTRY);
       go to START;
```

Remark:
SVAR contains the segment address and BMR contains the segment mask (K=24).

2.3.4.9 Release Buffer

```
       I:=1;
START: BUFFER_ENTRY:=BUFFER(I);
     call RELEASE(BUFFER_ENTRY);
       If (I=N) then return;
      I:=I+1;
       go to START;
```

Remark:
This command releases every entry in the Buffers.

2.3.5 Command Execution and Sequencing

Every PSS command which modifies data in the PSS will be executed in such a way that from the point of view of any port of the PSS the command has either not yet been started or has been completed. It should noted,however, the command fetch/store (or I/O fetch/store) is considered logically as two commands issued in sequence, i.e., a fetch command followed by a store command (or a I/O fetch followed by a I/O store).

In addition the PSS guarantees that commands issued from a given port will be executed, as far as that port can determine, in strict order.

However, the PSS does not guarantee that any sequence of fetch, store and fetch/store commands (and their respective I/O commands) issued by one port will be seen by another port as having occurred in the same order.

For example, assume that two SVA's A and B contain the values a and b and the following commands would be executed by two ports:

Port 1	Port 2
A:=c	Y:=VALUE(B)
B:=d	X:=VALUE(A)
Y:=VALUE(B)	
X:=VALUE(A)	

For Port 1 the values of X and Y will always be c and d respectively. Port 2, however, may see all of the following combinations:

I	II	III	IV
Y=b	Y=b	Y=d	Y=d
X=a	X=c	X=a	X=c

While situations I, II and IV are consistent with the random placement of Port 2 commands with respect to Port 1 commands which still preserve relative ordering, situation III is not.

In order that the user of a port may eliminate the possibility of case III, three commands are provided, namely Test and Set Lock, Reset Lock and Pause.

The Test and Set Lock command will guarantee that

a. no access to the data word will be allowed between the testing of its first bit and the storing of the whole new value (if performed)
b. no command issued by the same port after the Test and Set Lock command will be initiated before the Test and Set Lock command itself is completed.

The Reset Lock command will guarantee that

a. all commands issued by the same port before the Reset Lock command will be completed before the Reset Lock command itself is initiated.

The Pause command will guarantee that all commands issued by the same port prior to the Pause command will be completed before any command issued to the same port following the Pause command will be initiated.

This implies that the following combinations of procedures from two distinct ports will not allow case III above to occur.

a. Port 1

```
L: Test and Set Lock R
   If (test_bit = 1) then go to L
   A:=c
   Reset Lock R
   B:=d
```

Port 2

```
L: Test and Set Lock R
   If (test_bit = 1) then go to L
   Y:=VALUE(B)
   X:=VALUE(A)
   Reset Lock R
```

b. Port 1

```
A:=c
PAUSE
B:=d
```

Port 2

```
Y:=VALUE(B)
PAUSE
X:=VALUE(A)
```

2.4 I/O OPERATIONAL PROCEDURES

2.4.1 Introduction

The Primary Storage Subsystem, like any other system, will respond meaningfully only if proper operational procedures are followed. In this section, normal procedures for various block transfer operations are presented.
Following are several general remarks.

a. It is usually required that for each block only one of the procedures is in progress at a time.
b. The term "block" is used to denote a block of data corresponding to an entry in the PSD which can be of size from one paragraph to one page.
c. In order to guarantee proper functioning of the PSD, it is necessary that every operation which modifies the contents of any valid PSD entry be followed by an Invalidate High Speed Directory Entry command. In some cases, it may be necessary to invalidate the whole High Speed Directory with the Invalidate High Speed Directory command if a procedure may have modified unknown PSDE's.
d. With minor modifications (on the updating PSD entries) the procedures can be used to transfer consecutive blocks up to the size of a segment.

2.4.2 Block Output Procedure

This procedure guarantees to output the latest copy and that further modifications to the data by non-I/O commands cannot be made without change of R to 0 in the PSD entry. The execution of the procedure may leave data in the Buffer Storage Unit with BT=1.

```
        procedure BLOCK_OUTPUT(BLOCK_SVA,BLOCK_LENGTH);
START:  Search Primary Storage Directory (BLOCK_SVA);
        PSDE:=PSD(K);
        If ((V=0)|(T=1)) then go to ERROR_EXIT;
        T:=1;
        R:=1;
        PSD(K):=PSDE;
        Invalidate High Speed Directory Entry (PSDE);
        loop on I from BLOCK_SVA to (BLOCK_SVA+BLOCK_LENGTH)
          I/O Fetch (I);
        T:=0;
        PSD(K):=PSDE;
        Invalidate High Speed Directory Entry (PSDE);
```

```
        V:=0;
        PSD(K):=PSDE;
        Invalidate High Speed Directory Entry;
        end
```

2.4.5 Block Scratch Procedure

This procedure guarantees no future normal reference of the block can be made after the Block Scratch operation. No data of the block will be left in the Buffer Storage Unit.

```
        procedure BLOCK_SCRATCH(BLOCK_SVA);
START:  Search Primary Storage Directory (BLOCK_SVA);
        PSDE:=PSD(K);
        if ((V=0)|(T=1)) then go to ERROR_EXIT;
        V:=0;
        PSD(K):=PSDE;
        Invalidate High Speed Directory Entry (PSDE);
        Release Page (BLOCK_SVA);
        end
```

2.4.6 Block Allocation Procedure

New entry in the PSD is entered corresponding to the newly allocated block.

```
        procedure BLOCK_ALLOCATION(BLOCK_SVA);
START:  Search Primary Storage Directory (BLOCK_SVA);
        PSDE:=PSD(K);
        if (V¬=0) then call BLOCK_SCRATCH(BLOCK_SVA);
        Search Primary Storage Directory (BLOCK_SVA);
        PSDE:=NEW_ENTRY;
        PSD(K):=PSDE;
        end
```

This procedure can be merged with Block Input Procedure if initialization is also provided with the allocation.

```
        end
```

2.4.3 Block Input Procedure

This procedure makes a new copy in the Primary Storage Subsystem. All future references are guaranteed to get this new copy. Further modification to the data by non_I/O commands cannot be made without change of R to 0 in the PSD entry. The executions of the procedure may leave data in the Buffer Storage Unit with BT=1 and BR=0.

```
        procedure BLOCK_INPUT(BLOCK_SVA,BLOCK_LENGTH);
START:  Search Primary Storage Directory (BLOCK-SVA);
        PSDE:=PSD(K);
        If ((V=0)|(T=1)) then go to ERROR_EXIT;
        T:=1;
        R:=0;
        PSD(K):=PSDE;
        Invalidate High Speed Directory Entry (PSDE);
        loop on I from BLOCK_SVA to (BLOCK_SVA+BLOCK_LENGTH)
          I/O Store (I);
        R:=1;
        T:=0;
        PSD(K):=PSDE;
        Invalidate High Speed Directory Entry (PSDE);
        end
```

2.4.4 Block Swap Procedure

This procedure guarantees to output the latest copy and that no further normal reference of the block can be made in the PSS. The execution of the procedure may leave data in the buffers with BT=1, but they may not be referenced by any non-I/O commands.

```
        procedure BLOCK_SWAP(BLOCK_SVA,BLOCK_LENGTH);
START:  Search Primary Storage Directory (BLOCK-SVA);
        PSDE:=PSD(K);
        If ((V=0)|(T=1)) then go to ERROR_EXIT;
        T:=1;
        R:=1;
        PSD(K):=PSDE;
        Invalidate High Speed Directory Entry (PSDE);
        loop on I from BLOCK_SVA to (BLOCK_SVA+BLOCK_LENGTH)
          I/O Fetch (I);
```

Chapter 3
CPU

INDEX

3.1 INTRODUCTION

The Central Processing Unit (CPU) is the unit which is responsible for most data processing functions (as distinct from simple data transfer functions) in the MASCOR/132 Data Processing System.

There can be more than one CPU associated with a given system.

We will attempt to introduce the various characteristics of the CPU in a logical sequence. In some instances, however, it is difficult to avoid references to material to be covered later. This is particularly true in discussing exceptional conditions; Section 3.10 will discuss and review the handling and meaning of such conditions.

3.2 GENERAL PURPOSE REGISTERS

There are 64 General Purpose Registers (GPR), numbered 0 to 63, each 32 bits wide.

The General Purpose Registers are the main operational registers of the CPU, i.e., they are used as operands by most operations. They are general purpose in the sense that they may be used to hold any type of operand:

- o program virtual addresses
- o indexing and displacement quantities
- o logical operands and condition codes
- o arithmetic operands of all types

3.2.1 General Purpose Register 0

Although GPR 0 can be used for any normal GPR operation, it plays a special role as part of the program status block: its contents are saved and restored in conjunction with any status switching operation.

GPR 0 is also referred to as the Link Address Register (LKAR).

3.3 PROGRAM VIRTUAL ADDRESS

The CPU references data through the 32_bit Program Virtual Address (PVA). Program virtual addresses have the format shown in Figure 3.3.1.

Bits	Field Name	Field Symbol
2-7	Program Segment Number	PSN
8-31	Byte Displacement	BD

Figure 3.3.1 Program Virtual Address

An attempt to reference, as a PVA, a field whose high order two bits are not zero will generate an invalid segment exception.

3.3.1 Use Of The Program Virtual Address

A PVA identifies a byte within the "address space" of a program. A PVA used to identify halfwords (2 bytes), words (4bytes) or doublewords (8bytes), is understood to represent the address of the first byte of the operand. Whenever referencing operands larger than a byte, certain low order bits of the PVA must be zero, as detailed in Table 3.3.1.1.

Table 3.3.1.1. PVA Constraints.

Operand Size	Zero PVA Bits
halfword	31
word	30,31
doubleword	29,30,31

In other words, operands must be aligned with the appropriate address boundaries. If such conditions are not satisfied, a boundary violation exception is generated.

3.3.2 Program Virtual Address Formation

Given any two 32_bit fields x and y, we define the procedure of address_addition, symbolized by @, as follows:*

x@y=x<0-7 >|((x<8 -31>+y<8 -31>)modulo 2**24)

as pictorially represented in Figure 3.3.2.1.

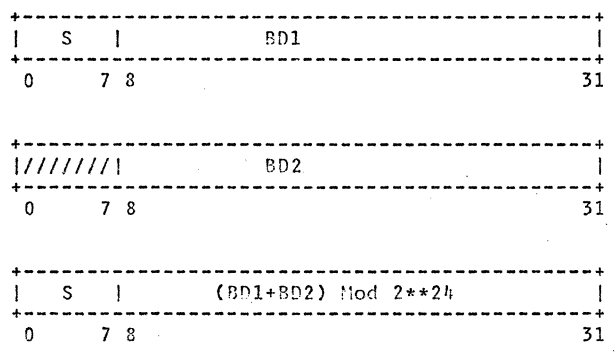


Figure 3.3.2.1 Address-Addition

In subsequent references we may refer to the first operand of address addition (here x) as the base and the second operand (here y) as the index or displacement.

Note that address_addition yields a correct program virtual address if and only if the x field already had the format of a correct PVA.

Note also that since the addition of the 24 low order bits is performed modulo 2**24, the resulting PVA may have a byte displacement lower than either BD1 or BD2. If, for example, BD2 is a string of 24 1's, the resulting byte displacement will be one less than BD1. Note also that bits 0-7 of the second operand are ignored, which means:

a. the program segment number of the resulting PVA is the same as the PSN of the x field;

---

*See Appendix A for notational conventions.

b. the y field could contain any information in its high order 8 bits without affecting the result. For example, y could itself be a valid PVA.

Program virtual addresses are obtained in one of the following ways:

a. directly from a GPR;

b. from address_addition of a "base" GPR and an "index" GPR;

c. from address_addition of a "base" GPR and an immediate field from the executing instruction;

d. directly from the instruction address register;

e. from address_addition of the instruction address register and an immediate field from the executing instruction.

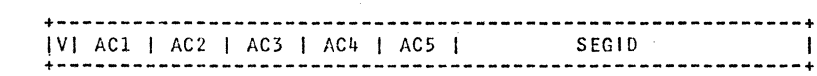
In cases c., d. and e. the immediate field is extended with zeroes to 32 bits before the address addition procedure.

3.4 DATA ACCESS CONTROL AND PVA TRANSLATION

The CPU controls access to data and translates program virtual addresses into system virtual addresses on a segment basis through the segment registers. Direct access to the segment registers is possible only through privileged operations.

3.4.1 Segment Registers

There are 64 Segment Registers (SGR) numberd 0 to 63. Each register is 32 bits wide. The format of each SGR is given in Figure 3.4.1.1.



Bits	Field Name	Field Symbol
0	Validity Control	V
1-3	Access Control 1	AC1
4-6	Access Control 2	AC2
7-9	Access Control 3	AC3
10-12	Access Control 4	AC4
13-15	Access Control 5	AC5
16-31	Segment Identifier	SEGID

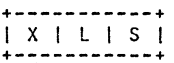
Figure 3.4.1.1 Segment Register Format

A segment register is valid if its V bit is 0 and invalid otherwise. Any attempt to access an invalid segment register in the process of PVA translation will generate an invalid segment exception.

3.4.2 Data Access Control

A field in the system control vector (see Section 3.5.1) selects which of the five control fields is active at any given time. The position of the active field is the same for all segment registers.

Each access control field is viewed as having the format of Figure 3.4.2.1.



Bit	Access Mode	Field Symbol
0	Instruction execution	X
1	Data load	L
2	Data store	S

Figure 3.4.2.1 Format Of The Access Control Field

Each of the three bits controls one mode of access as listed in Figure 3.4.2.1. A given mode of access to data in the given segment is allowed if the corresponding bit in the active control field is 0; it is not allowed otherwise.

Any attempt to access data in a non-allowed mode will generate an access violation exception.

3.4.3 Program Virtual Address Translation

In order to reference data in the primary storage system the CPU must translate the program virtual address into the corresponding system virtual address. The PVA is translated into a SVA by replacing the high order 8 bits of the PVA with the contents of the segment identifier field of the segment register that corresponds to the program segment number in the PVA (see Figure 3.4.3.1).

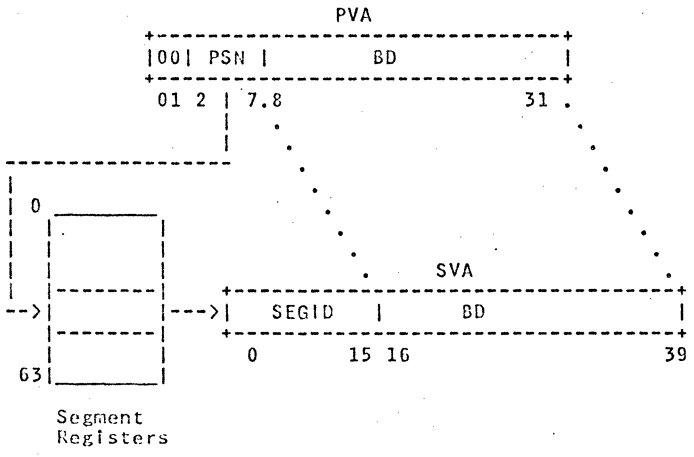


Figure 3.4.3.1 PVA Translation Procedure

The system virtual address is then presented to the primary storage system. The PSS will either return the data (or perform the store) using the Primary Storage Directory (PSD) or will notify the CPU of an exceptional condition (see Chapter 2 for details).

The following exceptions may be generated:

a. invalid system virtual address: there is no primary storage directory entry associated with the specified address.

b. data in transit: the T ("in-transit") control bit in the PSD entry is 1.

c. read only: the R (read-only) control bit in the PSD entry is 1 (this exception can only occur in store operations).

3.5 PROGRAM STATUS REGISTERS

There are three Program Status Registers (PSR), namely the Control Vector (CV), the Iteration Count Register (ICTR), and the Instruction Address Register (IAR). They control the program execution in the CPU.

The program status registers, together with the general purpose segment registers, typically contain the information in the CPU which is "local" to the executing program.

3.5.1 Control Vector

The Control Vector (CV) is a 32-bit register containing a collection of fields (mostly of just 1 bit) which control various CPU functions.

The Control Vector High Order Byte (CVB) is accessible through non-privileged operations; the whole control vector is accessible through privileged operations.

Table 3.5.1.1 gives the list of fields in the control vector. The meaning of each field is discussed in detail in the following sections.

Certain bit positions are left undefined. They are implemented in the hardware but have no function associated with them.

Table 3.5.1.1. Control Vector.

Bits	Field Name	Field Symbol
0	Arithmetic Exception Signal	Z
1	Arithmetic Carry	Y
2		
3		
4	Binary Integer Overflow Mask	BIOM
5	Decimal Integer Overflow Mask	DIOM
6	Floating Point Exponent Underflow Mask	FPUM
7	Floating Point Loss of Significance	FPSM
8	Address Match Mask	AMM
9	System Service Call Control	SSCC
10	Last Branch Address Control	LBAC
11	Successful Branch Mask	SBM
12		
13-15	Access Control Field Number	ACFN
16		
17		
18		
19		
20	Interval Timer 0 Activity	IT0A
21	Instruction Counter 0 Activity	IC0A
22	Interval Timer 1 Activity	IT1A
23	Instruction Counter 1 Activity	IC1A
24	Control Mode	CM
25	External Signal Mask	ESM
26	Internal Signal Mask	ISM
27	Program Exception Control	PXC
28	Power Off Warning Mask	POWM
29	Hardware Malfunction Mask	HMM
30	Hardware Malfunction Warning Mask	HMWM
31	Program Exception Mask	PXM

3.5.1.1 Arithmetic Exception Signal

The Arithmetic Exception Signal (Z) bit is set to 0 or 1 on certain arithmetic operation as determined by the non-occurrence or occurrence of a maskable binary or decimal exception. The Z bit is also used by the decimal arithmetic instructions to pass information regarding the possibility of overflow. Details regarding the generation and use of the Z-bit are given in Appendix B.

3.5.1.2 Arithmetic Carry

The Arithmetic Carry (Y) bit is set to the value of the carry upon each unsigned binary or unsigned decimal operation. Its setting is unaffected by signed or floating point operations.

In addition certain signed and unsigned integer operations (see Section 3.11) allow the value of the Y bit to be added in the low order bit position.

3.5.1.3 Arithmetic Exception Masks

There are four types of arithmetic exceptions which are maskable:

- o binary integer overflow
- o decimal integer overflow
- o floating point underflow
- o floating point loss of significance

Whenever one of these conditions occurs, and if the corresponding mask bit (BIOM, DIOM, FPUM, and FPSM, respectively) is 1, the exception is recognized and a program interrupt will occur.

When the corresponding mask bit is 0 then the exception is ignored and some additional action is taken (see Section 3.10).

3.5.1.4 Address Match Mask

The address match register (see Section 3.8) contains a program virtual address which is matched against any PVA referenced by the CPU. If a match occurs and the Address Match Mask (AMM) is 1, then an address match exception is recognized. Otherwise the match is ignored.

Address matching allows detailed programmable monitoring of data or instruction cells for debugging and maintenance purposes.

3.5.1.5 System Service Call Control

The System Service Call Control (SSCC) determines whether a system service call operation is allowed to proceed normally (SSCC=0) or not (SSCC=1).

If SSCC=1 any attempt to execute a SSC operation will generate a SSC violation exception.

This feature can be used to deny a subsystem user access to supervisory system facilities.

3.5.1.6 Last Branch Address Control

The Last Branch Address Control (LBAC) determines whether the last branch address register (see Section 3.8) is updated on every successful branch with the PVA of the branch instruction (LBAC=1) or not (LBAC=0).

Maintenance of a branch address register allows improved diagnostic capability in case of subsequent errors or exceptions.

3.5.1.7 Successful Branch Mask

The Successful Branch Mask (SBM) determines whether a successful branch exception is recognized (SBM=1) or not (SBM=0).

Branch exceptions allow the tracing of control flow of an executing program.

3.5.1.8 Access Control Field Number

The Access Control Field Number (ACFN) is an integer between 0 and 5 . The values 1 to 5 identify which of the five access control fields in the segment registers (see Section 3.4) is active.

If ACFN=0 then free access to any segment is assumed. If the ACFN is 6 or 7 no access to any segment will be allowed.

3.5.1.9 Interval Timers and Instruction Counters Activity

The Interval Timers Activity fields (IT0A and IT1A) and the Instruction Counters Activity fields (IC0A and IC1A) determine whether the corresponding interval timers or instruction counters are active (bit=1) or inactive (bit=0) (see Section 3.8).

3.5.1.10 Control Mode

The Control Mode (CM) bit determines whether all valid operation codes are admissible (CM=0) or not (CM=1). If CM=0 the CPU is said to be running in privileged mode; otherwise the CPU is in non privileged mode.

If an operation code is subject to the above control it is so stated in its definition. Such operations are called privileged operations.

3.5.1.11 External Signal Mask

The External Signal Mask (ESM) determines whether the occurrence of a master external signal is recognized (ESM=1) or not (ESM=0).

3.5.1.12 Internal Signal Mask

The Internal Signal Mask (ISM) determines whether the occurrence of an internal signal is recognized (ISM=1) or not (ISM=0).

3.5.1.13 Program Exception Control

The Program Exception Control (PXC) determines which program exception old and new status areas will be used upon recognition of a program exception (see Section 3.10).

3.5.1.14 Power Off Warning Mask

The Power Off Warning Mask (POWM) determines whether the occurrence of a power off warning signal is recognized (POWM=1) or not (POWM=0).

3.5.1.15 Hardware Malfunction Mask

The Hardware Malfunction Mask (HWM) determines whether the occurrence of a hardware malfunction is recognized (HWM=1) or not (HWM=0).

Whenever a hardware malfunction is recognized the CPU will halt and a hardware malfunction signal to the system monitor unit will be generated (see Chapter 9).

Whenever a hardware malfunction is ignored the CPU will proceed in an unpredictable way.

3.5.1.16 Hardware Malfunction Warning Mask

The Hardware Malfunction Warning Mask (HWWM) determines whether the occurrence of a hardware malfunction warning signal is recognized (HWWM=1) or not (HWWM=0).

If the hardware malfunction warning signal is one of the hardware malfunction warning signals controlled by the Hardware Malfunction Warning Vector (HWWV) the corresponding bit is set and the actions described in Section 3.8.7 take place.

If the hardware malfunction warning signal is not one of the signals controlled by the HWWV and the HWWM is one, then a Hardware Malfunction Interrupt is generated. A Hardware Malfunction Warning Code (HWWC) is set as described in Section 3.10.3.

If the hardware malfunction warning signal is not one of the signals recognized by the HWWV and the HWWM is zero, then no further action takes place.

3.5.1.17 Program Exception Mask

The Program Exception Mask (PXM) determines whether the occurrence of any program exception is recognized (PXM=1) or not (PXM=0).

If PXM=0 and a program exception occurs the instruction may be completed in a unpredictable way.

3.5.2 Iteration Count Register

The Iteration Count Register (ITCR) is a 32-bit register which is used by all interruptible operations to determine the number of iterations to be performed. It contains a signed integer which is used to count and test for completion of such operations. Its use will be discussed in more detail in Section 3.11, in conjunction with the description of such operations.

3.5.3 Instruction Address Register

The Instruction Address Register (IAR) is a 32-bit register containing at any given time the program virtual address of the instruction which is being executed.

Since all instructions are one word long and must reside on word boundaries, the low order 2 bits (bits 30-31) of the IAR are always ignored and assumed to be zero.

The contents of the IAR, the instruction address (IA), is incremented by 4 at the completion of every instruction, with the exception of successful branch instructions. In the latter case the IAR is set to the program virtual address of the branch target.

3.6 DATA FORMATS

Depending upon the operations involved, the CPU will interpret data (either in the general purpose register or in primary storage) according to different formats.

A major classification of such formats is by length. Table 3.6.1 gives the list of all field types recognized by the CPU.

Table 3.6.1. Field Types.

Field Name	Length
Byte	8 bits
Halfword	16 bits
Word	32 bits
Doubleword	64 bits
Quadword	128 bits
Byte string	0 to 2**24 bytes
Word Group	0 to 2**22 words
Doubleword Group	0 to 2**21 doublewords

Quadword operands are only recognized in the GPR's.

3.6.1 Register Referencing

The registers may be viewed as a bank of 64 words, 32 doublewords, or 256 bytes.

When registers are addressed by instruction fields, two low order zero bits are appended to the register address so that all references will be on word boundaries. Doubleword references to registers require that the low order bit taken from the instruction field be zero, otherwise a register specification exception will occurr.

Similarly, quadword references require that the two low order bits of the register reference field of the instruction be zero.

When registers are addressed by the Iteration count register (ITCR) then the references are made directly to the byte boundaries of the registers. The contents of the ITCR is required to have two low order zeroes for word references and three low order zeroes for doubleword references, otherwise a register specification exception will occurr.

When registers are addressed by the contents of registers then the addressing is by byte position 0 to 255. The byte position is the register contents taken modulo 256.

3.6.2 Program Virtual Addresses

Whenever referencing a data element in storage via the program virtual address, the format of the PVA wil be assumed to be the one of Figure 3.3.1.

If the high order two bits are not zero an Invalid segment exception will be generated.

3.6.3 Data Move Operations

Operations are provided to load data from storage to the registers, store data from the registers into storage, transfer data between the registers, and move data between areas in storage. These operations will not affect the format of the data. Load, store, and transfer operations are provided for byte, halfword, word, and doubleword operands.

Move operations are provided for byte strings.

In addition operations are provided to iteratively load and store sequential bytes, words and doublewords, using the iteration counter.

3.6.4 Logical Data

Logical operations handle words and doublewords as bit strings of the corresponding length. Each bit in the string is handled as an independent binary digit.

Logical true may be represented in registers as a word of all ones and logical false as a word of all zeroes.

3.6.5 Binary Integers

There are two symmetric classes of operations handling respectively unsigned and signed binary integers. They accept word and doubleword operands. Some operations accept quadword operands. Unsigned binary integer operations treat their operands modulo 2*(operand-length). Signed binary integer operations treat their operands as binary "2's complement" representations of signed integers. In such representation all non-negative integers will have the high order bit (the "sign bit") equal to 0, while negative integers will have a sign bit value of 1.

The numbers n which can be represented in the two cases are as follows:

unsigned integers

word	0 <= n <= 2**32-1
doubleword	0 <= n <= 2**64-1
quadword	0 <= n <= 2**128-1

signed integers

word	-2**31 <= n <= 2**31-1
doubleword	-2**63 <= n <= 2**63-1
quadword	-2**127 <= n <= 2**127-1

These ranges are approximately equal to

unsigned integer

word	0 <= n < 6.2*10**9
doubleword	0 <= n < 1.8*10**19
quadword	0 <= n < 3.4*10**38

signed integer

word	-2.1*10**9 < n < 2.1*10**9
doubleword	-9.2*10**19 < n < 9.2*10**19
quadword	-1.7*10**38 < n < 1.7*10**38

3.6.6 Decimal Integers

There are two symmetric classes of operations handling respectively unsigned and signed decimal integer words and doublewords.

Unsigned decimal integer operations interpret their operands as 8 or 16 binary coded decimal digits and treat them modulo 10**8 and 10**16 respectively.

Signed decimal integer operations interpret their operands as " 10's complement " representations of signed integers. In such representation the high order four bits of the word are all zero (B'0000') for non-negative integers and have the pattern B'1001' (i.e., decimal 9) for negative integers. The numbers which can be represented in the two cases are as follows:

unsigned decimal integers

word	0 <= n <= 10**8-1
doubleword	0 <= n <= 10**16-1

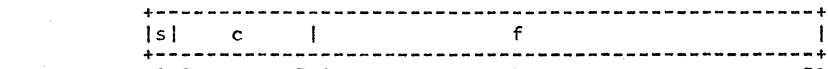
signed decimal integers

word	-(10**7-1) <= n <= 10**7-1
doubleword	-(10**15-1) <= n <= 10**15-1

A negative decimal zero is defined as a pattern of nine (B'1001') followed by only zeroes (B'0000'). This is an improper signed decimal pattern and will cause an exception when generated by signed decimal arithmetic instructions.

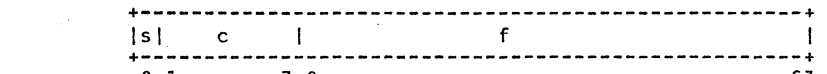
3.6.7 Floating Point Numbers

Floating point operations handle word and doubleword operands. The format of the data in the two cases is given in Figures 3.6.7.1 and 3.6.7.2.



Bits	Field Name
0	Sign bit
1-7	Hexadecimal characteristic
8-31	Hexadecimal fraction

Figure 3.6.7.1 Floating Point Word



Bits	Field Name
0	Sign bit
1-7	Hexadecimal characteristic
8-63	Hexadecimal fraction

Figure 3.6.7.2 Floating Point Doubleword

The sign bit is 0 for non-negative numbers and 1 for negative numbers. The characteristic is viewed as an "excess 64" binary integer. The fraction is interpreted as having the radix point immediately to its left. The number represented by a floating point word or doubleword has the value given by

```
r = ((-1)**s)*(.f)*(16**(c-64))
```

"True zero" is a pattern of all zeroes and has the numerical value zero. Data generated by the CPU (without creating an exception) will either be a true zero or will be in hexadecimally normalized form, i.e., the high order 4 bits of the fraction will not be all zeroes. If data elements not conforming to one of these two patterns are used in floating point operations, the operations may provide unexpected results. A negative (improper) zero has a sign bit of one and all other bits zero. In floating point only operations involving only zero and negative zero may generate another negative zero.

Apart from true zero, the magnitude (m) of a normalized floating point number is in the following ranges:

```
word        -2**260 <= m <= (2**252-2**228)
doubleword  -2**260 <= m <= (2**252-2**196)
```

This approximatly equivalent to a range of 10**-78 to 10**75.

3.6.8 Character Strings

Character strings are composed of 0 to 2**24 bytes. Character string operations use the iteration count register (ITCR) to specify the number of bytes to be operated on.

3.7 INSTRUCTION FORMATS

All instructions are one word (32-bits) in length. The information contained in the word is, however organized in a number of different ways. There are four major formats: M, I, V and N.

The fields which determine in each format the operation code will be marked with the letter "c".

3.7.1 M-Format Instructions

M-format instructions have the format of Figure 3.7.1.1.

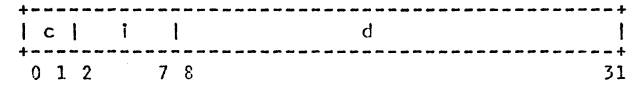


Figure 3.7.1.1 M-format Instructions

There is only one M-format instruction which uses as operands GPRi and the byte displacement d.

3.7.2 I-Format Instructions

I-format instructions have the format of Figure 3.7.2.1.

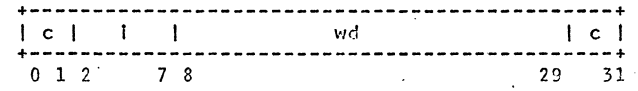


Figure 3.7.3.1 I-format Instructions

There are 4 I-format instructions. Their operands are GPRi and either IAQwd or C(IAQwd), i.e., they use relative addressing with respect to the instruction address.

3.7.3 V-Format Instructions

V-format instructions have the format of Figure 3.7.3.1.

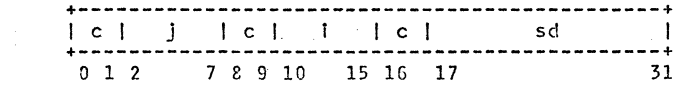


Figure 3.7.3.1 V-format Instructions

There are 11 V-format instructions. Their operands are GPRi and either GPRj@sd or C(GPRj@sd). The abbreviation V is due to the direct addressing capability provided here, similar to the original Von Neumann designs.

3.7.4 N-Format Instructions

N-format instructions have the format of Figure 3.7.4.1.

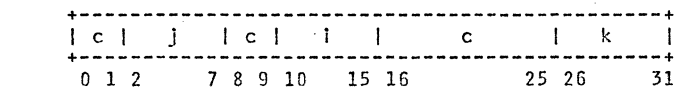


Figure 3.7.4.1 N-format Instructions

There are potentially 2560 N-format instructions. The operands used by these instructions vary, but the large majority of operation codes are associated with one of the eight basic operand selection modes listed in Table 3.7.4.1.

Table 3.7.4.1 Operand Selection Modes

Mode Code	Operand-1	Operand-2	Operand-3
RRR	Ri	Rj	Rk
RRK	Ri	Rj	K
RSR	Ri	C(Rj)	Rk
RSK	Ri	C(Rj)	K
RXR	Ri	C(Rj@Rk)	Ri
RXK	Ri	C(Rj@K)	Ri
XRR	C(Rj@Rk)	C(Rj@Rk)	Ri
XKR	C(Rj@K)	C(Rj@K)	Ri

The following associations between letters in the mode code and the corresponding operands are useful for mnemonic purposes:

R register used as operand or index

K immediate value of k field used as operand or index

S storage reference through base register only

X storage reference through base register and index (register or immediate).

3.8 SYSTEM CONTROL REGISTERS

The System Control Registers (SCR) are used to control a number of special functions, primarily those related to the interrupt system (see Section 3.10).

There are 64 system control registers, numbered 0 to 63. Each register is 32 bits wide. Some SCR are undefined; their use as sources will provide 32 zeroes; their use as targets will be equivalent to a null operation. There are also two auxiliary control registers which are not numbered.

Table 3.8.1 lists the system control registers. The sections following will describe their functions in detail.

Table 3.8.1 System Control Registers

Number	Name	Symbol
0	Old Status Area Origin	OSAO
1	New Status Area Origin	NSAO
2	Interval Timer 0	IT0
3	Instruction Counter 0	IC0
4	Interval Timer 1	IT1
5	Instruction Counter 1	IC1
6	Hardware Malfunction Warning Mask Vector	HMWMV
7	Hardware Malfunction Warning Vector	HMWV
8	Internal Signal Mask Vector	ISMV
9	Internal Signal Vector	ISV
10	Master External Signal Mask Vector, High Order	MESMVH
11	Master External Signal Vector, High Order	MESVH
12	Master External Signal Mask Vector, Low Order	MESMVL
13	Master External Signal Vector, Low Order	MESVL
14	Last Branch Address	LBA
15-63	-	
-	Address Match Register	AMR
-	Real Time Clock	RTC

3.8.1 Old and New Status Area Origins

The Old Status Area Origin (OSAO) and the New Status Area Origin (NSAO) contain the program virtual addresses of the first location of old status area and of the new status area respectively (see Section 3.10.1). Such addresses are assumed to be on 256 byte boundaries.

The high order two bits and the low order 8 bits are ignored and assumed to be zero.

3.8.2 Interval Timers

The Interval Timer 0 (IT0) and the Interval Timer 1 (IT1) are decremented by one in bit position 23 every microsecond if the corresponding activity control bits in the SCV (IT0A and IT1A) are 1. Whenever the value of an interval timer reaches zero the corresponding signal bit (IT0S, IT1S) in the internal signal vector is set to 1.

The low order 8 bits of each interval timer are ignored and assumed to be zero.

3.8.3 Instruction Counters

The Instruction Counter 0 (IC0) and the Instruction Counter 1 (IC1) are decremented by one in bit position 31 at the completion of every instruction if the corresponding activity control bits in the SCV (IC0A and IC1A) are 1. Whenever the value of an instruction counter reaches zero the corresponding signal bit (IC0S, IC1S) in the internal signal vector is set to 1.

3.8.4 Hardware Malfunction Warning Mask Vector and Hardware Malfunction Warning Vector

The Hardware Malfunction Warning Mask Vector (HMWMV) and the Hardware Malfunction Warning Vector (HMWV) are paired, i.e., bit k of the HMWMV is the mask bit for bit k of the HMWV.

When a hardware malfunction occurs which only requires a warning, a hardware malfunction warning is signalled. These warnings are controlled by the Hardware Malfunction Warning Mask (HMWM) in the Control Vector (CV) and in some instances by the HMWV. For this class a specific bit identifying the type of malfunction in the HMWV is set to one. If the corresponding bit in the HMWMV is also one, and the Hardware Malfunction Warning Mask (HMWM) is one, then a hardware malfunction warning interrupt is generated at the completion of the current instruction. The interrupt may also be generated when the HMWMV or the HMWM is set.

A Hardware Malfunction Warning Code (HMWC) of 0 (See Section 3.10.3) is associated with these interrupts.

3.8.5 Internal Signal Mask Vector and Internal Signal Vector

The Internal Signal Mask Vector (ISMV) and the Internal Signal Vector (ISV) are paired, i.e., bit k in the ISMV is the mask bit for bit k in the ISV. Whenever corresponding bits in the ISMV and ISV are both 1 an internal signal interrupt is recognized if the ISM bit in the SCV is also one.

The bits in the ISV are set to one by the occurrence of certain events (as listed below) or under program control.

The structure of both the ISMV and the ISV is given in Table 3.8.8.1.

Table 3.8.8.1 Structure of the ISMR and the ISV

Bit	Name	Symbols ISMV	ISV
0	Interval Timer 0	ITOM	ITOS
1	Instruction Counter 0	ICOM	ICOS
2	Interval Timer 1	IT1M	IT1S
3	Instruction Counter 1	IC1M	IC1S
4-31	Program Control		

3.8.6 Master External Signal Mask Vector and Master External Signal Vector

The Master External Signal Mask Vector (MESMV) and the Master External Signal Vector are paired, i.e., bit k of the MESMV is the mask bit for bit k of the MESV. Whenever a bit in the MESV is 1 and the corresponding bit in the MESMV is also 1, an external interrupt signal is generated.

Each bit in the MESV corresponds to one group of external input signal lines (see Section 3.9). Whenever there is a signal pending in the i-th external input signal line group while the corresponding bit position in the external signal input mask vector is also 1, the i-th bit of the MESV is set to 1.

Any attempt to load data into the MESV will be ignored; however the MESV may be indirectly affected by the execution of CONSLG instructions (see Section 3.9.1).

3.8.7 Last Branch Address

On every successful branch, the program virtual address of the instruction sequentially following the branch instruction is stored in the Last Branch Address (LBA) register if the LBAC bit in the SCV is zero. The instructions where the LBA may be set are the same as those that are subject to the Successful Branch exception and are indicated as such in Appendix B.

3.8.8 Address Match Register

The Address Match Register (AMR) contains a program virtual address which is checked against any primary storage reference. If a match is found an address match exception is caused if the AMM bit in the SCV is one. Otherwise, the match is ignored.

In the matching of the addresses, as many low order bits will be ignored as determined by the boundary appropriate fo the given primary storage reference request.

The address match exception allows interactive and selective control and supervision of program operation, when desired for program development, measurement, or debugging.

3.8.9 Real Time Clock

The Real Time Clock (RTC) is provided by a source outside the CPU. Its value, however, is always made readable to the CPU.

The real time clock is a 64 bit counter which is incremented by one in bit position 31 of the low order word every microsecond.

Attempts to load any value in the RTC will be ignored.

Bit	Name	Symbols ISMV	ISV
0	Interval Timer 0	ITOM	ITOS
1	Instruction Counter 0	ICOM	ICOS
2	Interval Timer 1	IT1M	IT1S
3	Instruction Counter 1	IC1M	IC1S
4-31	Program Control		

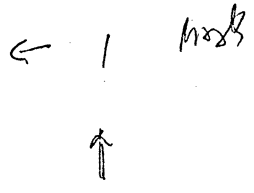
3.8.9 Master External Signal Mask Vector and Master External Signal Vector

The Master External Signal Mask Vector (MESMV) and the Master External Signal Vector are paired, i.e., bit k of the MESMV is the mask bit for bit k of the MESV. Whenever a bit in the MESV is 1 and the corresponding bit in the MESMV is also 1, an external interrupt signal is generated.

Each bit in the MESV corresponds to one group of external input signal lines (see Section 3.9). Whenever there is a signal pending in the i-th external input signal line group while the corresponding bit position in the external signal input mask vector is also 1, the i-th bit of the MESV is set to 1.

Any attempt to load data into the MESV will be ignored.

The MESV is effected however by



3.9 EXTERNAL INTERFACE

The external interface consists of 64 External Signal Input Line Groups (ESILG) and four sets of registers: the External Signal Mask Registers (ESMR), the External Signal Output Registers (ESOR), the External Word Input Registers (EWIR), and the External Word Output Registers (EWOR).

The CPU uses these registers and line groups to communicate with the external world. Since such communication requirements vary drastically from one environment to another, the number of registers in each set actually installed will be a function of the particular system.

3.9.1 External Signal Input Line Groups and External Signal Mask Registers

There are 64 External Signal Input Line Groups (ESILG) numbered 0 to 63. Each group consists of 32 pairs of lines, the Signal Input Control Lines (SICL) and the Signal Input Acknowledge Lines (SIAL). Associated with each group there is a 32-bit External Signal Mask Vector (ESMV).

The SICL's are connected to external devices which control their state. Each SIAL is connected to the same device of the corresponding SICL, but it is controlled by the CPU.

Whenever (i) the k-th SICL in the i-th line group is set to 1, (ii) the corresponding SIAL is 0 and (iii) the k-th bit of the i-th external signal mask vector is 1, the i-th bit of the master external signal vector is set to 1.

Whenever a CONSLG (Control Signal Input Line Group) Instruction is issued to a given line group, all the SIAL's in that group for which (i) the corresponding SICL is on and, (ii) the corresponding bit in the mask provided by the instruction is also on, will be set to 1 .

Whenever the SICL is set to 0, the corresponding SIAL is also set to zero.

3.9.2 External Signal Output Registers

There are 64 External Signal Output Registers (ESOR) numbered 0 to 63. Each register is 32 bits wide.

Each bit of each ESOR can be connected to some external device, and it is used by the CPU to send appropriate signals to that device.

Associated with each bit of the ESOR there is a Signal Output Acknowledge Line (SOAL). Whenever a SOAL is set to 1 (by an external device) the corresponding bit of the ESOR is set to 0.

3.9.3 External Word Input Registers

There are 64 External Word Input Registers (EWIR) numbered 0 to 63. Each register is 32 bits wide.

Each EWIR can be controlled by an external source.

Associated with each register there is a Word Input Control Line (WICL) and a Word Input Acknowledge Line (WIAL).

An EWIR is valid if its WICL is 1 and its WIAL is 0, and invalid otherwise.

Whenever an INEXW (Input External Word) instruction is issued to a valid EWIR, the corresponding WIAL is set to 1.

Whenever a WICL is set to zero (by an external source), the corresponding WIAL is set to zero.

3.9.4 External Word Output Registers

There are 64 External Word Output Registers (EWOR) numbered 0 to 63. Each register is 32 bits wide.

Each EWOR can be connected to an external device.

Associated with each EWOR there is a Word Output Control Line (WOCL) and a Word Output Acknowledge Line (WOAL).

An EWOR is available if its WOCL and WOAL are both 0, and unavailable otherwise. Whenever an OUTEXW (Output External Word) instruction is issued to an available EWOR, the corresponding WOCL is set to 1 as soon as the signals on the EWOR outgoing lines are assured to be stable.

Whenever a WOAL is set to 1 (by an external device), the corresponding WOCL is set to 0.

3.10 INTERRUPT SYSTEM

There are a number of conditions, either generated by the CPU internally or produced by outside sources, which require that the program which was being executed when the condition occurred be stopped and a new program be initiated.

We call this procedure program status switching, and the events which generate it, interrupts.

The entity which gets switched is the Program Status Block (PSB) defined as the contents of the following registers:

Instruction Address Register
Control Vector
Iteration Count Register
Link Address Register

or, in other words, the three program status registers and the link address register.

There are 8 classes of interrupts:

Power Off Warning
Hardware Malfunction Warning
External Signal
Internal Signal
Program Exception 0
Program Exception 1
System Service Call
Utility Service Call

The last four classes are mutually exclusive, i.e. they cannot occur simultaneously; we will refer to them as a group as synchronous interrupts.

Whenever two or more interrupt conditions are simultaneously present, they will be honored by the hardware in the following order:

Synchronous Interrupt
Internal Signal
External Signal
Hardware Malfunction Warning
Power Off Warning

Note that the working of the interrupt procedure is such that the latest interrupt to be honored will be the one serviced first.

Note also that any settings of the mask bits in the control vector may modify the sequence and leave some interrupt conditions pending.

3.10.1 Old Status and New Status Areas

Associated with each interrupt class there are two quadword areas, the old status area and the new status area. Table 3.10.1.1 gives the organization of both areas.

Table 3.10.1.1 Old and New Status Areas

Word	Field Name	Field Symbols
0	Old (New) Instruction Address	OIA (NIA)
1	Old (New) Control Vector	OCV (NCV)
2	Old (New) Iteration Count	OITC (NITC)
3	Old (New) Link Address	OLA (NLA)

Upon the occurrence of an interrupt, the CPU will save the current PSB in the old status area, issue a pause command to the PSS, and use the data in the new status area to form a new current PSB.

The CPU will then proceed according to the new PSB.

The old and new status areas are located in the primary storage subsystem and their virtual addresses are computed from the contents of the OSAO and NSAO special control registers by adding a fixed displacement which is a function of the interrupt class as listed in Table 3.10.1.2.

Table 3.10.1.2 Status Area Displacements

Interrupt Class	Displacement
Power Off Warning	0 bytes
Hardware Malfunction Warning	32 bytes
External Signal Interrupt	64 bytes
Internal Signal Interrupt	96 bytes
Program Exception Interrupt 0	128 bytes
Program Exception Interrupt 1	160 bytes
System Service Call	192 bytes
Utility Service Call	224 bytes

3.10.2 Power Off Warning Interrupt

Whenever power is turned off, either to switch off the system or because of a power failure, a power off warning interrupt will occur if the POWM bit in the CV is one.

The POWM bit should be set to zero only when servicing a previous power off warning interrupt.

3.10.3 Hardware Malfunction Warning Interrupt

A Hardware Malfunction Warning Interrupt occurs when the HMWM bit of the CV is one and either of the two following conditions are true:

a. The Hardware Malfunction Warning Signal is controlled by the HMWV and corresponding bits of the HMWV and the HMWMV are one. These signals are generally due to events outside of the CPU and are identified by a Hardware Malfunction Warning Code (HMWC) of zero. The settings of the data reference code field (see below) and of the PVA field are undefined.

b. The Hardware Malfunction Warning Signal is not controlled by the HMWV. These signals are generally due to events within the CPU and are identified by a specific HMWC other than zero. A data reference code (see Table 3.10.6.1) and the value of the PVA will be available to further identify the hardware malfunction warning.

In conjunction with the program status switching the HMWC will be set into bits 0-29 of the word following the old status area for this interrupt class. Bits 30-31 of this word will be set to the data reference code and the word following will be set to the value of the PVA.

3.10.4 External Signal Interrupt

An external signal interrupt occurs whenever any two corresponding bits in the MESMV and the MESV are one, and the ESM bit of the CV is also one.

It should be noted that although normally such an interrupt will be triggered by a signal input control line being set to 1, it is possible for the interrupt to occur upon setting a new value in an ESMV, in the MESMV, or in the ESM.

3.10.5 Internal Signal Interrupt

An internal signal interrupt occurs whenever any two corresponding bits in the ISMV and ISV are one and the ISM bit of the CV is also one.

It should be noted that although normally such interrupt will be triggered by a bit in the ISV turning to one, it is possible for the interrupt to occur upon setting a new value in the ISMV or ISM.

3.10.6 Program Exception Interrupts

There are two types of program exceptions, namely, the CVB controlled exceptions and the PXM controlled exceptions. The recognition of the exceptions controlled by the leading byte of the Control Vector (CVB) is determined solely by the value of the corresponding mask bit in the CVB. If the exception is masked, the CPU will be able to proceed in a predictable and "correct" way.

The recognition of all other program exceptions is determined by the PXM bit in the system control vector. If the PXM bit is 0, the CPU will proceed in an unpredictable way.

If the program exception causes a status switch then the procedure is identical for both types of program exceptions. The PXC field in the SCV is used to select one of a pair of status areas. A code identifying the type of exception and, in some instances the offending PVA, are stored in the 2 words following the old status area.

Whenever meaningful, bits 30-31 of the code are set to a pattern identifying the type of data reference involved according to Table 3.10.6.1.

Table 3.10.6.1 Data Reference Code

Code in bits 30-31 of the exception identification	Type of Data Reference
00	Instruction Fetch
01	Data Load
10	Data Store
11	Data Load/Store

Table 3.10.6.2 gives the list of program exceptions, the corresponding bit of the code set into the high order 29 bits of the codeword, and if an address is stored into word following the code word. If no PVA is indicated the contents of the data reference code field and the PVA field are undefined.

The actions taken when a program exception occurs are further described below. In order to distinguish the three types of effects that a program exception may have on the instructions in progress we define three terms:

- The instruction is suppressed. The state of the CPU is not altered in any way other than by the program exception status switch itself. The instruction address is not updated and the instruction may be restarted after appropriate actions have been made by the routines handling the program exception.

- The instruction is completed. The instruction is carried out to a point where it leaves specific results in the CPU and the storage system. The instruction address is updated to point to the next instruction to be executed as if no program exception had occurred. These results are generally sufficient to allow proper handling of the exceptional condition and the continuation of the program. The detailed specification of the effect of such completed instructions is given in Appendix B with the description of the instructions themselves.

- The instruction is terminated. The state of the CPU may have been altered before the status switch takes place. The instruction address has not been updated. The instruction is in general not restartable without knowledge of the environment which causes the instruction to generate such an exception.

When program exceptions occur during the execution of instructions using the Iteration Counter (ITC), then interrupts that are indicated as suppressing or completing will allow the instruction to complete up to a point where the instruction is re_startable.

A number of program exceptions are further described below in Sections 3.10.6.1 to 3.10.6.10.

All others are specific to certain types of instructions and described in detail in the sections of Appendix B dealing with these instructions.

In Appendix B the exceptions ROD, DIT, ISVA, ISG, AVL, BVL, AMT are referred to as standard and not separately enumerated. All these, as well as the exceptions RSP, UOC, and SBR are described below.

Table 3.10.6.2 Program Exceptions

Exception Type	Symbol	Bit set in 1st location	Contents of 2nd location
Read Only Data	ROD	0	old PVA
Data In Transit	DIT	1	old PVA
Boundary Violation	BVL	2	old PVA
Access Violation	AVL	3	old PVA
Address Match	AMT	4	old PVA
Invalid Segment	ISG	5	old PVA
Invalid System Virtual Address	ISVA	6	old PVA
Undefined Operation Code	UOC	7	undefined
Privileged Operation Code	POC	8	undefined
Trap	TRAP	9	undefined
System Service Call Violation	SSCV	10	undefined
Register Specification	RSP	11	undefined
Successful Branch	SBR	12	undefined
Binary Integer Overflow	BIO	13	undefined
Binary Integer Divide	BID	14	undefined
Decimal Data Invalid	DDI	15	undefined
Decimal Integer Overflow	DIO	16	undefined
Floating Point Exponent Overflow	FPO	17	undefined
Floating Point Exponent Underflow	FPU	18	undefined
Floating Point Loss of Significance	FPS	19	undefined
Floating Point Divide	FPD	20	undefined

3.10.6.1 Read Only Data Exception (ROD)

This exception occurs when the Read Only Control Flag, R, of the Primary Storage Directory Entry (PSDE) associated with the given System Virtual Address (SVA) is one. See also Section 2.2.2.1 and 2.3.3.3.

This exception can occur with certain V-format instructions and with N-format instructions using XRR or XKR operand selection modes. The execution of the instructions will be suppressed.

The Instruction Address (IA) will not be updated and the instruction may be re_executed.

3.10.6.2 Data In Transit Exception (DIT)

This exception occurs when the Data In Transit Control Flag (T) of the Primary Storage Directory Entry (PSDE) associated with the given System Virtual Address (SVA) is one. See also Sections 2.2.2.1 and 2.3.2.1.

This exception can occur on any instruction reference; and on all storage references for data, i.e., most V-format instructions and N-format instructions using RSR, RSK, RXR, RXK, XRR, or XKR operand selection modes.

The execution of the instruction will be suppressed and the instruction may be re_executed.

3.10.6.3 Invalid System Virtual Address (ISVA)

This exception occurs when the Validity Control Flag, V, in the Primary Storage Directory Entry (PSDE) is one. See also Section 2.2.2.1 and 2.3.2.1.

This exception can occur on any instruction reference; and on all storage references for data, i.e., most V-format instructions and N-format instructions using RSR, RSK, RXR, RXK, XRR, and XKR operand selection modes.

The execution of the instruction will be suppressed and the instruction may be re_executed.

3.10.6.4 Invalid Segment Exception (ISG)

This exception will occur if a Program Virtual Address (PVA) refers to a non-existant Segment Register (SEGNO>63) or to a Segment Register which has a validity of one. See also Section 3.6.1.

This exception can occur on any instruction reference; and on all storage references for data, i.e., most V-format instructions and N-format instructions using RSR, RSK, RXR, RXK, XRR, and XKR operand selection modes.

The execution of the instruction will be suppressed. The contents of the base address registers referenced must be changed to allow uninterrupted re_execution.

3.10.6.5 Access Violation Exception (AVL)

This exception will occur if the type of access requested to a segment as: instruction execution, data load, or a data store, is not permitted due to the corresponding bit of the Access Control Field (ACn) currently active (n=ACFN) being

one. See Section 3.4.2.

This exception can occur on any instruction reference; or on data references to storage, i.e., most V-format instructions and N-format instructions having operand selection modes of RSR, RSK, RXR, RXK, XRR, and XKR.

The execution of the instruction is suppressed and the instruction is re_executable.

3.10.6.6 Boundary Violation Exception (BVL)

This exception will occur if the Program Virtual Address (PVA) does not have the appropriate number of low order zero bits for the length of the operand in storage referred to. See also Section 3.3.1.

This exception can occur with V-format and N-format instructions having operand selection modes of RSR, RSK, RXR, RXK, XRR, and XKR that refer to halfword, word, or doubleword operands.

The execution of the instruction will be suppressed. The contents of base address or indexing registers referenced must be changed to allow uninterrupted re_execution.

3.10.6.7 Register Specification (RSP)

This exception will occur if a doubleword operand is to be fetched or set into a General Purpose Register or System Control Register with an odd address (1,3,5,...) or if a quadword operand is to be fetched or set into a General Purpuse Register with an address which is not a multiple of four (1,2,3,5,6,...). See Section 3.2 and 3.8.

Instructions where this exception can occur are so described in Appendix B.

The execution of such instruction is suppressed. They will generally not be re_executable in an uninterruptable manner.

3.10.6.8 Undefined Operation Code Exception (UOC)

This exception occurs when the code field of the instruction has no function assigned. This is the case with one V-format and various N-format code fields. See Section 3.7.

All actions are suppressed.

3.10.6.9 Successful Branch Exception (SBR)

This exception will occur whenever the Successful Branch Mask bit (SBM) in the Control Vector is one and a branch instruction takes the indicated branch address.

Instructions subject to this interrupt are so described in Appendix B.

The instruction is completed and the instruction address is set to the next instruction to be executed.

If the Last Branch Address Control (LBAC) bit in the Control Vector (CV) was 0 then the Last Branch Address Register (LBA) will have been set.

3.10.6.10 Address Match Exception (AMT)

This exception will occur whenever a Program Virtual Address (PVA) used as an instruction or data references matches the address set into the Address Matching Register (AMR) and the Address Match Mask (AMM) is one.

This exception can occur on any instruction reference; or on data references to storage, i.e., most V-format instructions and N-format instructions having operand selection modes of RSR, RSK, RXR, RXK, XRR, and XKR.

The instruction is suppressed. The target adress of a Branch instruction will not cause an AMT exception, the fetching of the subsequent instruction however will cause the AMT exception.

3.10.7 System Service Call

A System Service Call interrupt occurs whenever the program executes a System Service Call (SSC) while the SSCC bit in the control vector is 0.

3.10.8 Utility Service Call

A Utility Service Call interrupt occurs whenever the program executes a Utility Service Call (USC).

3.11 Instructions and Functions

We will distinguish sixteen categories of functions which the instruction set of the MASCOR 132 supports. These categories are listed in table 3.11.1. In the subsequent sixteen sections which correspond to these categories all instruction types will be listed. A detailed description of every instruction is given in appendix B to this manual, which is also divided into sixteen corresponding sections.

Instructions of various formats may appear within a functional category. If the instruction is of the N-format then it may be available in a number of operand selection modes. To designate which operand selection modes are available for a given N-format instruction an operand selection class code is given with the format for every N-format instruction. The correspondence between the operand selection classes and operand selection modes is given in table 3.11.2 .

In the mnemonics for the instructions that follow certain conventions regarding the character identifying the data formats are used. These conventions are listed in table 3.11.3. Where more than one data format is relevant to the instruction the source data type identification will precede the result identification in the mnemonic.

Table 3.11.1 Categories of Instructions

1. Control instructions
2. Data move instructions
3. Logical operations
4. Compare instructions
5. Conditional branch instructions
6. Unsigned binary arithmetic
7. Signed binary arithmetic
8. Truncated floating point arithmetic
9. Rounded floating point arithmetic
10. Decimal arithmetic
11. Byte string operations
12. Shift instructions
13. Convert instructions
14. Privileged central processor control
15. Privileged primary storage system control
16. Privileged input/output interface control

Table 3.11.2 Operand Selection Classes

A	RRR,RRK,RSR,RSK,RXR,RXK,XRR,XKR
B	RRR,RSR,RXR,RXK,XRR,XKR
C	RRR,RRK,RSR,RSK,RXR,RXK
D	RRR,RRK,RSR,RSK
E	RRR,RSR,RXR,RXK
F	RRR,RRK
G	RRR,RSR
H	RXR,RXK
I	XRR,XKR
J	RRR
K	similar to RRR,RRK
L	similar to RRR
M	similar to RRK
N	similar to RSR
O	no operands

Table 3.11.3 Data Format Identification In Mnemonics

Unsigned binary integer	U
Signed binary integer	I
Truncated floating point	E
Rounded floating point	F
Decimal arithmetic, unsigned	TU
Decimal arithmetic, signed	TI
Single byte	B
Byte strings	S

3.11.1 Control Instructions

The control instructions allow a user program or system program to alter the flow of control, do address arithmetic, specify the number of iterations for subsequent iterative operations, and control those system states that are not within the privileged domain. The instructions are listed in table 3.1.11.1 .

Table 3.1.11.1 Control Instructions

Title	Mnemonic	Format
Branch unconditional	BRU	NL
Branch unconditional, indexed	BRUX	NK
Branch unconditional, relative	BRUR	I
Subroutine Call, External	CALLEX	NL
Subroutine Return, External	RETEX	NL
Subroutine Call, Internal	CALLIN	NL
Subroutine Call, Internal, Relative	CALLINR	I
System Service Call	SSC	NM
Utility Service Call	USC	NM
Address Add Byte	ADRAB	NF
Address Subtract Byte	ADRSB	NF
Address Add Halfword	ADRAH	NF
Address Subtract Halfword	ADRSH	NF
Address Add Word	ADRAW	NF
Address Subtract Word	ADRSW	NF
Address Add Doubleword	ADRAD	NF
Address Subtract Doubleword	ADRSD	NF
Address Add Direct	ADRAV	V
Address Add Relative	ADRAR	I
Test and Set Lock	TSETLOCK	NN
Reset Lock	RSETLOCK	NN
Set Iteration Count	SETITC	NK
Set Subtract Iteration Count	SETSITC	NK
Get Iteration Count	GETITC	NK
Set Control Vector Byte	SETCVB	NL
Get Control Vector Byte	GETCVB	NL
Trap	TRAP	NO
Pause	PAUSE	NO

3.11.2 Data Move Instructions

The data move instructions provide for the fetching of operands from storage, the transfer of operands between registers, and for storing the results from registers into storage.

Instructions are provided for operand lengths of 8 bytes (doublewords), 4 bytes (words), 2 bytes (halfwords), and individual bytes.

Three types of instructions use the iteration counter to provide loading and storing of multiple sequential words or double words.

Three byte instructions (LPB, SPB, MPB) view the 64 general purpose registers as a contiguous field of 256 bytes, addressed through a register to allow convenient construction of complex character sequences.

A load immediate instruction (LM) allows setting a register with up to 24 bits from the instruction itself. All these instructions are listed in Table 3.11.2.1. Additional instructions are provided for moving byte strings within storage, which may also be helpful when moving other data types. These are discussed in section 3.11.11 (Bytestring instructions) hereafter.

Table 3.11.2.1 Data Move Instructions

Title	Mnemonic		Format
Load word/double word	L	W/D	NE
Store word/double word	S	W/D	NI
Load direct word/double word	LV	W/D	V
Store direct word/double word	SV	W/D	V
Store word/double word in vector	SVEC	W/D	NN
Load double word group	LDG		NN
Store double word group	SDG		NN
Load half word left	LHL		NE
Load half word right	LHR		NE
Store half word left	SHL		NI
Store half word right	SHR		NI
Load direct half word right	LVHR		V
Store direct half word right	SVHR		V
Insert half word right	INSHR		NE
Insert half word left	INSHL		NE
Insert direct half word left	INSVHL		V
Load byte zero	LB0		NE
Load byte three	LB3		NE
Store byte zero	SB0		NI
Store byte three	SB3		NI
Load direct byte three	LVB3		V
Store direct byte three	SVB3		V
Insert byte zero	INSB0		NE
Insert byte one	INSB1		NE
Insert byte two	INSB2		NE
Insert byte three	INSB3		NE
Insert direct byte zero	INSVB0		V
Load register position with byte	LPB		NN
Move register position byte	MPB		NN
Store register position byte	SPB		NN
Load Immediate	LM		M

3.11.3 Logical Operations

A large set of logical operations, that operate on strings of 32 bits (words) or 64 bits (doublewords) is provided.

The following Table 3.11.3.1 lists these operations with their functions on individual bits. The bit complement function is accomplished by the Or Complement operation with an immediate operand of zero. All possible bit patterns of two operands may be generated by executing a single instruction with the exception of Complement (A and B) and Complement (A or B) which require two instructions.

An additional two instruction types allow the bitwise merging of one operand into another according to a bit mask provided in the third operand, these operations are available only on single word operands.

Two instruction allow setting of the sign bit in words or doublewords as a convenient means for building bitstrings for further logical operations and tests.

Table 3.1.11.3 Logical Operations

Title	Mnemonic	Format		Function	
				A :	0011
				B :	0101
And	AND	W/D	NA	A&B	0001
And Complement	ANC	W/D	NA	A&¬B	0010
Inverse And Complement	IANC	W/D	NA	¬A&B	0100
Or	OR	W/D	NA	A|B	0111
Or Complement	ORC	W/D	NA	A|¬B	1011
Inverse Or Complement	IORC	W/D	NA	¬A|B	1101
Exclusive Or	XOR	W/D	NA	(A|B)&¬(A&B)	0110
Exclusive Or Complement	XORC	W/D	NA	(A&B)|¬(A|B)	1001
Logical Merge	MRG		ND		
Inverse Merge	IMRG		ND		
Set Leading bit to one	SETSIGN	W/D	NB		
Set Leading bit to zero	RSETSIGN	W/D	NB		

3.11.4 Compare Instructions

A set of compare instructions is provided for the testing of greater than, greater than or equal, equal, not equal, less than and less than or equal conditions applied to unsigned and signed binary or decimal integers, floating point numbers and the value zero of both single and double word lengths, as well as for bytes. In byte comparison operation bytes are always treated as unsigned quantities.

For the testing of equal to and not equal, two additional tests are provided, one which allows testing a part word and zero by using a mask, and one that tests a byte in storage with zero. The result of all these comparison operations is a full word of ones or a full word of zeroes, set in a specified general purpose register, depending on whether the result of the comparison is true or false.

These results may be combined with other comparison results by using the logical operations described in the preceding section.

All these comparison operations types are listed in Table 3.11.4.1.

Additional comparison instructions exist for bytestrings, which are described in that section.

Table 3.11.4.1 Compare Instructions

Title	Mnemonic		Format
Compare greater than unsigned	CGTU	W/D	ND
Compare greater than integer	CGTI	W/D	ND
Compare greater than floating point	CGTF	W/D	NG
Compare greater than zero	CGTZ	W/D	NH
Compare greater than byte	CGTB		ND
Compare greater or equal to unsigned	CGEU	W/D	ND
Compare greater or equal to integer	CGEI	W/D	ND
Compare greater or equal to floating pt.	CGEF	W/D	NG
Compare greater or equal to zero	CGEZ	W/D	NH
Compare greater or equal to byte	CGEB		ND
Compare equal	CEQ	W/D	ND
Compare equal to zero	CEQZ	W/D	NH
Compare equal to zero part word	CEQZP	W/D	NG
Compare equal to byte	CEQB		ND
Compare equal to zero byte	CEQZB		NH
Compare not equal	CNE	W/D	ND
Compare not equal to zero	CNEZ	W/D	NH
Compare not equal to zero part word	CNEZP	W/D	NG
Compare not equal to byte	CNEB		ND
Compare not equal to zero byte	CNEZB		NH
Compare less than or equal to unsigned	CLEU	W/D	ND
Compare less than or equal to integer	CLEI	W/D	ND
Compare less than or equal to float. pt.	CLEF	W/D	NG
Compare less than or equal to zero	CLEZ	W/D	NH
Compare less than or equal to byte	CLEB		ND
Compare less than unsigned	CLTU	W/D	ND
Compare less than integer	CLTI	W/D	ND
Compare less than floating point	CLTF	W/D	NG
Compare less than zero	CLTZ	W/D	NH
Compare less than byte	CLTB		ND

3.11.5 Conditional Branch Instruction

A set of conditional branch instructions that matches the compare instructions of the preceding section is provided, which on finding that the result of the comparison is true, will branch to a specified instruction in storage or if the result of the comparison is false proceed to the next instruction. Two additional instructions that permit branching relative to the instruction address provide simplified handling of sign dependent tests. These instructions are listed according to function in Table 3.11.5.1 .

Table 3.11.5.1 Conditional Branch Instructions

Title	Mnemonic		Format
Branch if greater than unsigned	BGTU	W/D	ND
Branch if greater than integer	BGTI	W/D	ND
Branch if greater than floating point	BGTF	W/D	NG
Branch if greater than zero	BGTZ	W/D	NH
Branch if greater than byte	BGTB		ND
Branch if greater or equal to unsigned	BGEU	W/D	ND
Branch if greater or equal to integer	BGEI	W/D	ND
Branch if greater or equal to floating point	BGEF	W/D	NG
Branch if greater or equal to zero	BGEZ	W/D	NH
Branch if greater or equal to zero relative	BGEZR		I
Branch if greater or equal to byte	BGEB		ND
Branch if equal to word/double word	BEQ	W/D	ND
Branch if equal to zero	BEQZ	W/D	NH
Branch if equal to zero part word	BEQZP	W/D	NG
Branch if equal to byte	BEQB		ND
Branch if equal to zero byte	BEQZB		NH
Branch if not equal to word/double word	BNE	W/D	ND
Branch if not equal to zero	BNEZ	W/D	NH
Branch if not equal to zero part word	BNEZP	W/D	NG
Branch if not equal to byte	BNEB		ND
Branch if not equal to zero byte	BNEZB		NH
Branch if less than or equal to unsigned	BLEU	W/D	ND
Branch if less than or equal to integer	BLEI	W/D	ND
Branch if less than or equal to floating point	BLEF	W/D	NG
Branch if less than or equal to zero	BLEZ	W/D	NH
Branch if less than or equal to byte	BLEB		ND
Branch if less than unsigned	BLTU	W/D	ND
Branch if less than integer	BLTI	W/D	ND
Branch if less than floating point	BLTF	W/D	NG
Branch if less than zero	BLTZ	W/D	NH
Branch if less than zero relative	BLTZR		I
Branch if less than byte	BLTB		ND

3.11.6 Unsigned Binary Arithmetic

The four basic operations of arithmetic: add, subtract, multiply, quotient, and also remainder of division, together with the inverse operation for the non-commutative operations: inverse subtract, inverse quotient, inverse remainder are provided.

These eight operations are available in a complete set of operand selection modes, allowing register to register, storage to register and register to storage operational modes.

For each of the four basic operations extended instruction types provide facilities for arithmetic of high precision. These are available in a more limited number of operand selection modes. Each instruction type is implemented for both single and double word operands.

Unsigned arithmetic is equivalent to modulo (2 to the power 32 or 64) arithmetic. None of the operations can cause overflow and hence the Z-bit will always be left undefined. The carry bit, Y, has an important function in these instructions because it provides a convenient means of linking multiple operations together to provide very fast high precision addition and subtraction. The available operations are listed in table 3.11.6.

Table 3.11.6 Unsigned Binary Arithmetic Instructions

Title	Mnemonic		Format
Unsigned Add	ADDU	W/D	NA
Unsigned Add with Carry	ADDYU	W/D	NC
Unsigned Subtract	SUBU	W/D	NA
Unsigned Subtract with Carry	SUBYU	W/D	NC
Inverse Unsigned Substract	ISUBU	W/D	NA
Unsigned Multiply	MULU	W/D	NA
Unsigned Multiply Extended	MULXU	W/D	NJ
Unsigned Quotient	QUOU	W/D	NA
Inverse Unsigned Quotient	IQUOU	W/D	NA
Unsigned Remainder	REMU	W/D	NA
Inverse Unsigned Remainder	IREMU	W/D	NA
Unsigned Division Extended	DIVXU	W/D	NJ

3.11.7 Integer Binary Arithmetic

The four basic operations of arithmetic : add, subtract, multiply, quotient, and also remainder of division, together with the inverse operation for the non-commutative operations: inverse subtract, inverse quotient, inverse remainder are provided.

These eight operations are available in a complete set of operand selection modes, allowing register to register, storage to register and register to storage operational modes.

For the operations add, subtract, and multiply extended instruction types provide facilities for arithmetic of high precision. These are available in a more limited number of operand selection modes. Each instruction type is implemented for both single and double word operands.

Integer arithmetic provides for both positive and negative 31 bit and 63 bit quantities. Negative quantities are represented in two's complement form and as such will have the leading bit one. Operations that cause the 31 bit or 63 bit field to be exceeded will cause a binary integer overflow exception, or if the exception is masked, the Z bit to be set to one. The carry bit, Y, may also be set by these instructions but its setting is undefined.

Table 3.11.7.1 Integer Binary Arithmetic Instructions

Title	Mnemonic		Format
Integer Add	ADDI	W/D	NA
Integer Add with Carry	ADDYI	W/D	NC
Integer Subtract	SUBI	W/D	NA
Integer Subtract with Carry	SUBYI	W/D	NC
Inverse Integer Subtract	ISUBI	W/D	NA
Integer Multiply	MULI	W/D	NA
Integer Multiply Extended	MULXI	W/D	NJ
Integer Quotient	QUOI	W/D	NA
Inverse Integer Quotient	IQUOI	W/D	NA
Integer Remainder	REMI	W/D	NA
Inverse Integer Remainder	IREMI	W/D	NA
Absolute Value Integer	ABSI	W/D	NB
Negate Integer	NEGI	W/D	NB

3.11.8 Truncated Floating Point Arithmetic

The four basic operations of arithmetic are provided, together with inverse operations for the non-commutative operations. Both single and double word operands are provided for and a large set of operand selection modes are

provided.

In order to achieve proper sign control the negate operation is provided. The absolute value of a floating point operand can be obtained by using the RSETSIGN instruction.

All operations expect and return normalized floating point numbers, or true zero. Operations on unnormalized numbers or on negative zero may entail a loss of precision, but all instruction except negate (NEG) will still produce a normalized or true zero result. All internal intermediate results are carried out to at least 7 hexadecimal digits for single word (6 hexadecimal digit fraction) operands and to at least 15 hexadecimal digits for double word (14 hexadecimal digit fraction) operands. The final results however are truncated again to 6 or 14 hexadecimal digits.

Table 3.11.8.1 lists the available instructions.

Table 3.11.8.1 Truncated Floating Point Instructions

Title	Mnemonic	Format
Add floating	ADDE W/D	NB
Subtract floating	SUBE W/D	NB
Inverse subtract floating	ISUBE W/D	NB
Multiply floating	MULE W/D	NB
Quotient floating	QUOE W/D	NB
Inverse quotient floating	IQUOE W/D	NB
Negate floating	NEGE W/D	NB

3.11.9 Rounded Floating Point Arithmetic

An identical set of instructions for the four basic arithmetic operations exists which provides rounded results. Rounding is achieved by taking the high order bit of the hexadecimal digit that otherwise would be truncated, and adding the value of that bit to the low order position of the absolute value of the result.

An additional bit of significance is carried in the intermediate calculations to insure correctness of the leading bit of the guard digit.

A separate operation exists to shorten a double word floating point number to a single word floating point number by rounding.

All these operations are listed in Table 3.11.9.1.

Table 3.11.9.1 Rounded Floating Point Instructions

Title	Mnemonic	Format
Add floating and round	ADDF W/D	NB
Subtract floating and round	SUBF W/D	NB
Inverse subtract floating and round	ISUBF W/D	NB
Multiply floating and round	MULF W/D	NB
Quotient floating and round	QUOF W/D	NB
Inverse quotient floating round	IQUOF W/D	NB
Round floating	ROUNDF	NJ

3.11.10 Decimal Arithmetic

The operations add and subtract for decimal operands are directly implemented. The unsigned instruction can handle up to 16 digits in a doubleword whereas the signed instructions can handle a sign and up to 15 digits in a doubleword. Through the use of the carry bit, Y, these operations are easily combined to allow handling of decimal operands of any length.

A decimal digit occupies four bits and may have the values from zero (B'0000') to nine (B'1001'). Negative decimal operands are represented in tens complement notation and hence will have the leading digit equal to nine (B'1001'). Signed decimal numbers must have a proper sign digit in the first position and the Signed Decimal Number of -0 (B'1001 0000 ...') is not valid. Invalid decimal patterns will not be generated by decimal instruction without causing an exception.

These decimal integer instructions are listed in Table 3.11.10.1 .

Other operations on decimal operands can be carried out by using the decimal and binary shift instructions, the unsigned and signed compare instructions, or the appropriate binary instructions using the conversion instructions to and from binary.

Table 3.11.10.1 Decimal Arithmetic Instructions

Title	Mnemonic		Format
Add base ten unsigned	ADDTU	W/D	NC
Add with carry base ten unsigned	ADDYTU	W/D	NC
Add base ten signed	ADDTI	W/D	NC
Add with carry base ten signed	ADDYTI	W/D	NC
Subtract base ten unsigned	SUBTU	W/D	NC
Subtract with carry base ten unsigned	SUBYTU	W/D	NC
Subtract base ten signed	SUBTI	W/D	NC
Subtract with carry base ten signed	SUBYTI	W/D	NC

3.11.11 Byte String Operations

Instructions that manipulate byte strings which may be up to an entire segment long (0 to 2**24) bytes are a feature of the MASCOR 132. The length of the strings to be operated on is always set into the iteration counter before the instructions can be used. The iteration counter can be loaded with values up to +/- (2**31-1) .

The instructions are all interruptible and the iteration counter will provide information on how far the instruction had progressed when interrupted or terminated. Sequences of compare instructions may be executed with a single setting of the iteration counter. Byte strings may be moved in memory both in forward and reverse order to avoid overlaying of strings on themselves. These operations may also be useful for moving data vectors of other data types. Byte strings may also be loaded or stored from registers, in that case the 64 registers are viewed as a byte addressed field of 256 bytes length.

Compare instructions exist both for the comparison of bytestrings to each other and for the searching for specific bytes in byte strings. In the first case only four hardware functions are provided since the remaining two are symmetrical to two of the provided ones. To manipulate individual bytes many instructions are provided in the preceding sections.

All byte string operations are listed in Table 3.11.11.1.

Table 3.11.11.1 Byte String Instructions

Title	Mnemonic	Format
Move byte string lower in storage	MSL	NN
Move byte string higher in storage	MSH	NN
Load byte string into registers	LS	NN
Store byte string from registers	SS	NN
Store byte into bytestring	SBS	NN
Compare byte string equal to byte string	CEQS	NN
Compare byte string not equal to byte string	CNES	NN
Compare byte string less than or equal to byte string	CLES	NN
Compare byte string less than byte string	CLTS	NN
Compare byte greater than byte string	CGTSB	NN
Compare byte greater than or equal to byte string	CGESB	NN
Compare byte equal to byte string	CEQSB	NN
Compare byte not equal to byte string	CNESB	NN
Compare byte less than or equal to byte string	CLESB	NN
Compare byte less than byte string	CLTSB	NN

3.11.12 Shift Instructions

Shift instruction which will fetch the contents of a register or storage cell and deposit it shifted by 0 to 63 bit positions are provided.

All shift instructions operate on both single and double word operands and the amount of shift is always given in terms of bit positions to be shifted. The shift count is always taken modulo 64.

Four types of shift are provided: unsigned or logical shifts, integer shifts and decimal shifts which provide overflow indication or sign extension, and circular shifts.

Six bit count instruction allow searching for consecutive zeros, ones, or bits equal to the first bit encountered in a word or doubleword when scanning its bits form the left or right hand side.

The instructions are listed in Table 3.11.12.1.

Table 3.11.12.1 Shift Instructions

Title	Mnemonic	Format	
Shift unsigned left	SUL	W/D	ND
Shift unsigned right	SUR	W/D	ND
Shift integer left	SIL	W/D	ND
Shift integer right	SIR	W/D	ND
Shift circular left	SCL	W/D	ND
Shift decimal left	STL	W/D	ND
Shift decimal right	STR	W/D	ND
Count left bits equal zero	CTLEQZ	W/D	NG
Count right bits equal zero	CTREQZ	W/D	NG
Count left bits not equal zero	CTLNEZ	W/D	NG
Count right bits not equal zero	CTRNEZ	W/D	NG
Count left bits equal	CTLEQ	W/D	NG
Count right bits equal	CTREQ	W/D	NG

3.11.13 Convert Instructions

A complete set of instructions to convert single and double word floating point values to unsigned or signed binary integers and vice versa is provided. This includes rounded operations to preserve the maximum precision possible when converting to floating point. All floating point numbers generated will be properly normalized or true zero.

Both unsigned and signed decimal values may be converted to their binary counterparts using single word precision.

Table 3.11.13.1 Convert Instructions

Title	Mnemonic	Format	
Convert unsigned to floating truncated	CVRUE	W/D	NJ
Convert unsigned to floating rounded	CVRUF	W/D	NJ
Convert unsigned to decimal	CVRUT		NJ
Convert integer to floating truncated	CVRIE	W/D	NJ
Convert integer to floating rounded	CVRIF	W/D	NJ
Convert integer to decimal	CVRIT		NJ
Convert floating to unsigned	CVRFU	W/D	NJ
Convert floating to signed	CVRFI	W/D	NJ
Convert decimal to unsigned	CVRTU		NJ
Convert decimal to integer	CVRTI		NJ

3.11.14 Privileged CPU Control Instructions

To control the CPU of the MASCOR 132 system four sets of registers must be controlled.

To achieve this control, instructions to set and to read the control vector, to set the program status block, to set and to read the system control registers, and to set and read the segment registers are provided. The instructions that refer to the system control registers have both single word and double word capabilities.

A WAIT instruction is provided which causes the system to enter a state where no processing takes place untill an interrupt causes a statusswitch. instructions are listed in Table 3.11.14.1.

Table 3.11.14.1 Privileged CPU Control Instructions

Title	Mnemonic	Format
Set control vector	SETCV	NL
Get control vector	GETCV	NL
Set program status block	SETPSB	NL
Wait	WAIT	NO
Set system control registers	SETSCR W/D	NK
OR into system control register	ORSCR W/D	NK
AND into system control register	ANDSCR W/D	NK
Get system control registers	GETSCR W/D	NK
Set address match register	SETAMR	NL
Get address match register	GETAMR	NL
Get real time clock	GETRTC	NL
Set segment registers	SETSGR	NL
Get segment registers	GETSGR	NL
Set segment register group	SETSGRG	NL
Get segment register group	GETSGRG	NL

3.11.15 Privileged Primary Storage System Control Instructions

To control the primary storage system of the MASCOR 132, the CPU has the capability to set up, search, and invalidate entries in the PSS directory and its corresponding high speed directory, and release information from the buffer back into main storage. An additional operation provides access to the hashing function which is used by the PSS to translate the page identification field of the system

virtual address to pointer into the PSS directory.
The instructions provided to carry out these functions are listed in Table 3.11.15.1.

Table 3.11.15.1 Privileged Primary Storage System Control Instructions

Title	Mnemonics	Format
Set PSS directory registers	SETPSS	NL
Search PSS directory	SRCHPSD	NL
Invalidate high speed PSD entry	IHSDE	NL
Invalidate high speed directory	IHSD	NO
Hash high order part of address	HASH	NL
Release page now in buffer	RELPAGE	NL
Release segment from buffer	RELSEG	NL
Release buffer	RELBUF	NO

3.11.16 Input/Output Control Operations

In order to control the flow of data that can move between the primary storage sub-system and the high speed secondary storage units, the standard input/output channels, and the peripheral processing units instructions are provided which will allow reading and writing of control words, the signalling of the presence of control words, and the masking to prevent or enable interruptions due to such signalling. These instructions are listed in Table 3.11.16.1.

Table 3.11.16.1 Input Output Control Operations

Title	Mnemonic	Format
Set External Signal Mask Register	SETESMR	NK
Get External Signal Mask Register	GETESMR	NK
Output into External Signal Register, or	OUTSOR	NK
Output into External Signal Register, and	OUTSAND	NK
Control Input Signal Line Group	CONSLG	NK
Input External Word	INEXW	NK
Output External Word	OUTEXW	NK

CHAPTER 4

HIGH SPEED SECONDARY STORAGE UNIT

INDEX

4.1 INTRODUCTION

The High Speed Secondary Storage Unit (HSSSU) provides the system with a secondary storage unit with a high access rate and a high transfer rate.

The HSSSU can be physically implemented either as a magnetic drum or as a fixed head magnetic disk.

The HSSSU has two basic interfaces:

1. the control interface
2. the primary storage subsystem interface

The control interface connects the HSSSU to general purpose processors which are responsible for initiating unit operations, and for accepting notification of the completion of such operations.

The PSS interface is used for three distinct purposes:

1. the transfer of data to and from the PSS;
2. the transfer of unit commands from the PSS to the unit; unit commands determine the actual function performed by the unit.
3. the transfer of unit status information from the unit to the PSS, normally at completion of an operation.

The typical unit operation is initiated by a start signal received through the control interface. The unit will then execute a set of one or more unit commands which will be fetched through the PSS interface. Upon completion of the last command the HSSSU will store its status back in the PSS and then stop, waiting for a new start signal.

Each HSSSU has the capability to support more than one set of control interface lines and more than one set of PSS interface lines (see Figure 4.1.1).

The operation of the HSSSU is determined by its control registers, the functions associated with the control interface, the sequencing of commands and the functions associated with each command. The following sections will describe such functions in detail.

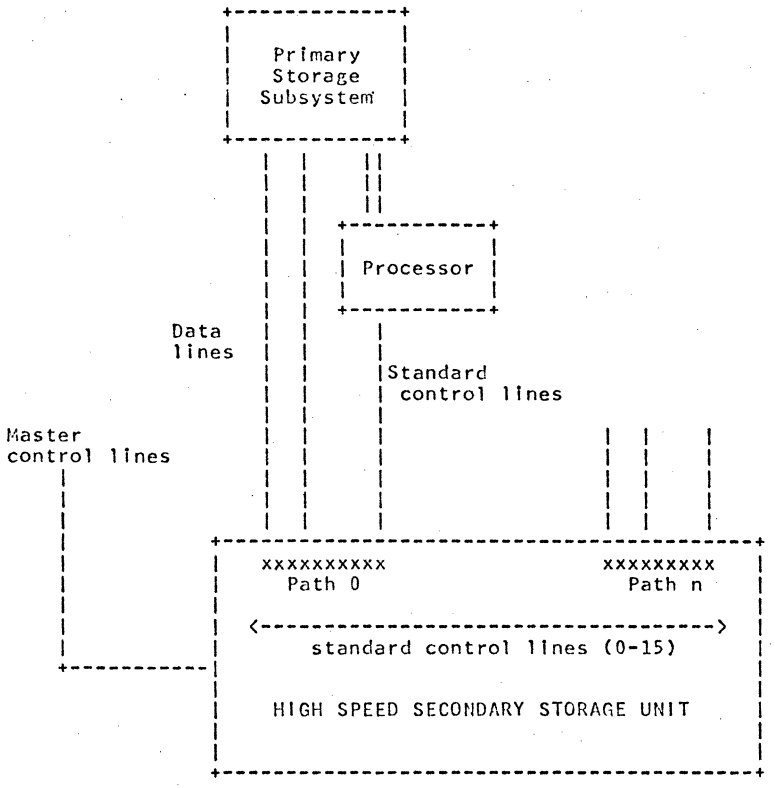


Figure 4.1.1 HSSSU Interface Structure

4.2 CONTROL REGISTERS

Associated with each HSSSU there are four unit control registers, namely the Standard Control Path Register (SCPR), the Unit Command Path Register (UCPR), the Unit Status Address Register (USAR) and the Unit Command Address Register (UCAR).

4.2.1 Standard Control Path Register

The Standard Control Path Register (SCPR) is 4 bits wide and determines which standard control interface path (see Section 4.4) is active. An HSSSU can have up to 16 standard control interface paths.

At any given time the HSSSU will respond to standard control signals only if they are coming through the active standard control interface and will only send signals through said interface.

The SCPR can only be set in conjunction with a unit reset signal (see Section 4.4).

4.2.2 Unit Command Path Register

The Unit Command Path Register (UCPR) is 4 bits wide and determines which primary storage subsystem interface path is active. An HSSSU can have up to 16 primary storage subsystem interface paths.

At any given time all unit commands are fetched by the unit through the PSS interface selected by the UCPR. The storing of the unit status is also performed through this interface.

In addition, the fetching and storing of the key field (see Section 4.3) will also always be done through the same interface.

The UCPR can only be set in conjunction with a unit reset signal.

4.2.3 Unit Status Address Register

The Unit Status Address Register (USAR) is 40 bits wide and determines the system virtual address at which the unit status will be stored whenever necessary. The low order 5 bits of the address are assumed to be zero. The address will be called the Unit Status Address (USA).

The USAR can be set by executing a Setup unit command.

4.2.4 Unit Command Address Register

The Unit Command Address Register (UCAR) is 40 bits wide and determines the system virtual address from which the next unit command will be fetched. The low order 4 bits of the

address are assumed to be zero. The address will be called the Unit Command Address (UCA).

The UCAR is set in conjunction with a Start signal to the value of the USAR plus 32. The UCAR can be set to a new value by executing a Transfer command within a sequence of unit commands.

The UCAR is incremented by 16 on every command fetch. The incrementation is done without a carry from bit position 16 into bit position 15, i.e., the low order 24 bits of the UCA are incremented by 16 modulo 2**24.

4.3 ORGANIZATION

Each HSSSU consists of a number of modules, each module consists of a number of sectors. Each module within a unit has the same number of sectors.

The number of modules per unit and the number of sectors per module may vary from one HSSSU model to another.

At any given time, data transfers between a primary storage subsystem and a HSSSU can occur only to or from a selected sector in a selected module. Switches from one sector to another can only occur on sector boundaries.

Each sector consists of two areas:

a. an 8-word key area,

b. a (2**n)*64 word data area (where n is a function of the particular HSSSU model).

In other words, each sector contains a fixed length key and a fixed binary power of 64 word data paragraphs.

4.4 HSSSU STATES

The HSSSU is at any given time in one of five possible states, namely halted, reset, free, busy, or status pending.

The halted state is entered at the end of the power on sequence, whenever a hardware malfunction is detected or whenever a primary storage subsystem exception occurs while the unit is attempting to store the unit status (see Section 4.7). The halted state can only be left upon receiving a unit reset signal. While in the halted state the unit will perform no normal activities although it will be available for the execution of certain maintenance functions.

The reset state is entered upon the receipt of a unit reset signal at the end of the reset procedure (see Section 4.8). The reset state can only be left upon receiving a start or stop signal.

If the unit becomes ready while in the reset state, the condition will remain pending until the unit becomes free.

The free state is entered upon completion of a status storing operation not associated with the PCI flag (see Section 4.6).

The busy state is entered upon receipt of a start signal (whenever in the free or reset state) or whenever the unit becomes ready (while in the free state). The unit is in the busy state while seeking for the sector to which a command is directed and while actually performing the transfer to or from that sector. The busy state is left upon normal completion of a command whose stop flag is on or upon any abnormal occurrence which requires termination of the command.

The status pending state is entered whenever the unit attempts to store the unit status and finds that a previous status storing operation has not yet been acknowledged. The status pending state is left upon receiving the acknowledge signal for the processor notification line.

4.5 CONTROL INTERFACE

The control interface consists of two distinct sets of lines, namely the master control lines and the standard control lines.

There is only one set of master control lines for each HSSSU.

There are up to 16 sets of standard control lines. Only one set is active at any given time as determined by the standard control path register (see Section 4.2).

Control lines can be of one of three types, data, input or output.

Data lines are used to set certain unit registers in conjunction with an incoming signal on an input line.

Input lines are used to transmit signals from an external source to the unit. Associated with each input line there is an acknowledge line used by the unit to communicate to the external source acknowledgement of an input signal. An input signal is detected whenever the input line is set to 1. To acknowledge an input signal means to set to 1 the corresponding acknowledge line. Whenever the input line is set to zero by an external source, the corresponding acknowledge line is also set to zero.

Output lines are used to transmit signals from the unit to an external device. Associated with each output line there is an acknowledge line used by the external device to communicate to the unit acknowledgement of the output signal. An output signal is generated by setting the output line to 1. The external device will acknowledge the signal by setting the acknowledge line to 1. Whenever such acknowledgement occurs the output line is reset to zero by the unit.

In the definition of each control line we will indicate in parentheses if the line is a data, input or output line.

4.5.1 Master Control Lines

There are 10 master control lines, the unit reset line (input), the four control path reset lines (data), the four command path reset lines (data), and the unit check line (output).

The master control lines are permanently connected to the System Monitor Unit (See Chapter 9). Through unit reset procedures it specifies which processor is in control of the unit operations and from which PSS the unit commands are

fetched.

Whenever a unit reset signal is detected, the unit reset procedure is executed (see Section 4.8). Such a procedure resets the unit to a known, "quiescent" state. As part of the procedure the standard control path register is set to the value represented by the state of the four control path reset lines and the unit command path register is set to the value represented by the state of the four command path reset lines.

This is the only procedure by which the active control path and the active command path can be modified.

The unit check line is used to notify the System Monitor Unit of the occurrence of a hardware malfunction in the unit.

4.5.2 Standard Control Lines

Each set of standard control lines consists of three control lines, namely the start line (input), the stop line (input), and the processor notification line (output).

The start signal directs the unit to start execution of a sequence of unit commands.

The stop signal directs the unit to abort the current command and terminate the command sequence.

The processor notification signal is issued by the HSSSU in conjunction with the storing of the unit status (see Section 4.7). Section 4.8 will discuss in more detail the procedures associated with all control lines.

4.6 UNIT COMMANDS

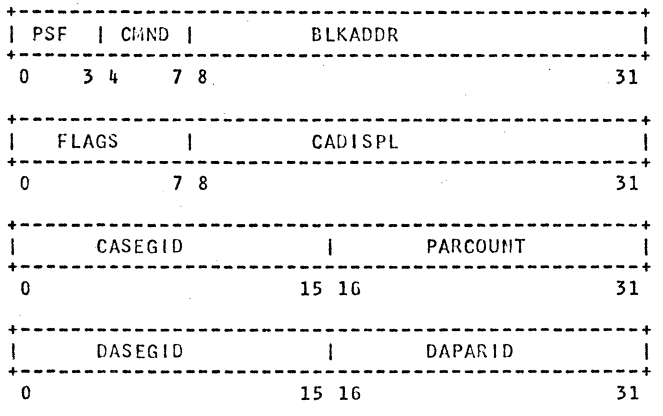
Operations on a HSSSU are initiated by the detection of the start signal. If a previously initiated operation is in progress at the time the start line is set to 1, the start signal will remain pending until the unit becomes free.

The start signal is acknowledged either as soon as the first unit command has been successfully fetched or before the status storing procedure is initiated if an error occurs.

Upon detection of the start signal the unit command address (UCA) is set to the value of the unit status address plus 32; i.e., it is set to point to the next higher 32-byte block. Normal operation is then initiated.

For each occurrence of the start signal the unit executes a sequence of unit commands consisting of commands whose stop flag (see Section 4.6.1) is off.

Each unit command is fetched by the HSSSU from the primary storage subsystem selected by the unit command path register. Commands are in the form of Unit Command Quadwords (UCQ) which have the format of Figure 4.6.1.



Word	Bits	Field Name	Field Symbol
0	0-3	Path Selection Field	PSF
	4-7	Command Field	CMND
1	0-7	Flag Field	FLAGS
	0	Program Control Interrupt	PCI
	1	Stop	STOP
	2-7	Undefined	
	8-31	Control Address displacement	CADISPL
2	0-16	Control Address Segment Identifier	CASEGID
	16-31	Paragraph Count	PARCOUNT
3	0-15	Data Adress Segment Identifier	DASEGID
	16-31	Data Address Paragraph Identifier	DAPARID

Figure 4.6.1 HSSS Unit Command Quadword

The Path Selection Field (PSF) determines which primary storage subsystem data path, out of 16 possible, has to be used for the data transfer associated with the command. Such selection is totally independent from the path selected for unit command fetching, unit status storing, and key fetching and storing (which is determined by the unit command path register).

The Command Code field (CMND) identifies the type of operation to be performed.

The Block Address field (BLKADDR) identifies the module, sector and paragraph address within a sector at which the operation must initiate. The last n bits of the block address identify the paragraph within the sector (see Section 4.3 on the HSSSU organization). The high order 24-n bits identify the module and sector according to a model dependent encoding.

The Control Address Segment Identifier (CASEGID) and the Control Address Displacement (CADISPL) form a system virtual address identifying a data area for the key or a new value for unit status or command address registers.

The Data Address Segment Identifier (DASEGID) and the Data Address Paragraph Identifier (DAPARID) form the high order 32 bits of the system virtual address identifying a data area. The low order 8 bits of the address are assumed to be zero.

The Paragraph Count (PARCOUNT) determines the number of paragraphs (i.e., 64 word blocks) to be transferred.

4.6.1 Unit Command Flags

The unit command flags (bits 0-7 of word 1 of the UCQ) permit modification of the basic operations associated with each command. Each flag is generally interpreted in the same way in each command for which it is meaningful; in some commands certain flags are ignored.

Table 4.6.1.1 provides the list of the defined flags. Bit positions to which no meaning has been assigned should be set to zero in order to guarantee compatibility with possible future additions.

Table 4.6.1.1. Unit Command Flags.

Bit	Flag Name	Flag Symbol
0	Program Controlled Interrupt	PCI
1	Null Data Transfer	NDT
2	Stop	STOP
3-7	--	

The Program Controlled Interrupt flag (PCI) is used to notify the processor controlling the HSSSU that a specified point in the command sequence has been reached. Whenever the PCI flag is on in the active command, the status storing

procedure is executed before initiating execution of the command.

The effect of the Null Data Transfer flag (NDT) is a function of the direction of the data transfer associated with the command. It only affects the handling of the data proper, not the transfer of the key.

When the NDT is associated with a read command, data transfer will be suppressed if the flag is on. When it is associated with a write command, all zero data will be stored in the appropriate sectors when the flag is on.

When the stop flag (STOP) is off it indicates that the next sequential unit command must be executed. In that case, no status storing takes place upon normal completion of the operation. If the stop flag is on, status is stored and the command sequence is terminated upon completion of the command.

4.6.2 Command Codes

The HSSSU recognizes seven commands whose codes are given in Table 4.6.2.1.

Table 4.6.2.1. Command Codes.

Code	Command
0001	Setup
0011	No operation
0100	Read
0101	Write
0110	Seek
0111	Transfer

All other codes are Invalid.

4.6.3 Setup Command

In the Setup command the path selection field, the block address, the paragraph count and the data address fields are ignored. The NDT flag does not apply.

The Setup command resets the unit status address register to the value specified by the control address fields. The low order 5 bits of the displacement are ignored and assumed to be zero.

No data transfer takes place. The command is considered completed as soon as the USAR is set to the new value.

If an exception occurs during execution of a setup command, status will be stored at the old value of the unit status address.

4.6.4 No Operation Command

In the No Operation command all fields are ignored with the exception of the PCI and STOP flags which are interpreted normally.

No data transfer takes place. The command is considered completed as so as it is decoded.

4.6.5 Read Command

The Read command is used to transfer data from the HSSSU to the primary storage subsystem. The Read command will operate as follows:

1. Upon reaching the sector identified by the block address, eight words of data will be transferred from its key area to the control address referenced in the UCQ (relative to the primary storage subsystem selected by the unit command path register). The low order 5 bits of the control address will be ignored by the HSSSU but are not assumed to be zero.

2. Data transfer from the HSSSU to the PSS will start from the paragraph identified by the blockaddress in the HSSSU and move data starting at the data address into the PSS.

3. If the end of sector is reached before the block count is exhausted, the contents of the key area of the next sector are checked for being all zeroes. If they are not, the data specification flag in the unit status will be set; data transfer will in any case proceed from the first paragraph in that sector. The procedure is repeated if necessary until the paragraph count is exhausted.

4. When the paragraph count is exhausted, data transfer is terminated.

In the process of transferring data, the HSSSU will collect information which will determine at the end of each sector, if any detectable errors have occurred. A data error exception will occur if such an error is detected. Data transfer and command sequencing are immediately terminated.

4.6.6 Write Command

The Write command is used to transfer data from the primary storage subsystem to the HSSSU. The write command will operate as follows:

1. Upon reaching the sector identified by the block address, eight words of data are transferred from the control address referenced by the UCQ (relative to the primary storage subsystem selected by the unit command register) to the key area of that sector. The low order 5 bits of the control address will be ignored by the HSSSU, but are not assumed to be zero.

2. Data transfer from the PSS to the HSSSU will start from the data address in the PSS and move data starting at the paragraph identified by the block address into the HSSSU.

3. If the end of sector is reached before the paragraph count is exhausted, the contents of the key area of the next sector are set to all zeroes; data transfer will then proceed to the first paragraph in that sector. The procedure is repeated as necessary until the paragraph count is exhausted.

4. When the paragraph count is exhausted data transfer is terminated.

5. Zeroes are stored in the remaining paragraphs of the last referenced sector; upon reaching the next end of sector the command is considered completed.

In the process of transferring data, the HSSSU will construct information which will be stored with the data for purpose of error detection at read time.

4.6.7 Seek Command

In the Seek command, the control address fields, the paragraph count and the data address are ignored. The NDT flag does not apply.

The Seek command will set the unit in the seek state (BUSY) until the end of sector of the sector selected by the block address is reached. At that point the command is considered completed.

4.6.8 Transfer Command

In the Transfer command the block address, the paragraph count, and the data address fields are ignored. The NDT flag does not apply. The STOP flag is ignored.

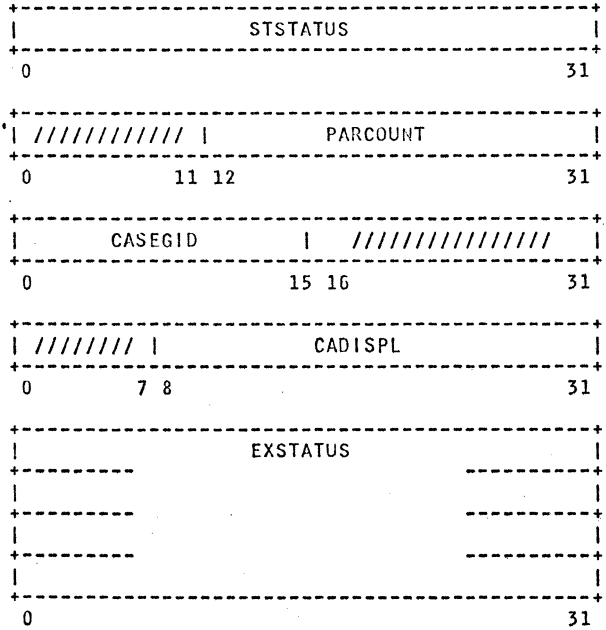
The Transfer command resets the value of the unit command address to the value of the control address. The low order 4 bits are ignored and assumed to be zero.

The Transfer command is considered completed as soon as the new value has been set in the UCAR.

If an exception occurs during the execution of a Transfer command, the command address in the unit status will point to the location following the Transfer command, not to the location pointed to by the Transfer command itself.

4.7 UNIT STATUS

Upon termination (either normal or abnormal) of a sequence of commands, upon detection of the PCI flag being set to 1 in the active command or whenever the unit becomes ready, the unit will attempt to store the unit status at the unit status address. The unit status consists of eight words, the Unit Status Double Quadword (USDQ) and its format is given in Figure 4.7.1.



Word	Bits	Field Name	Field Symbol
0	0-31	Standard Status	STSTATUS
1	12-31	Residual Paragraph Count	PARCOUNT
2	0-15	Command Address Segment Identifier	CASEGID
3	8-24	Command Address Displacement	CADISPL
4-7	0-31	Extended Status	EXSTATUS

Figure 4.7.1 Unit Status Double Quadword Area

The Extended Status field (EXSTATUS) is non-zero only when certain unusual conditions occur. The contents of the extended status field are not predetermined but are a function of the specific HSSSU model.

Upon storing the USQ the HSSSU will, however, set to zero the four-word locations following the USQ whenever such conditions do not apply.

The Standard Status field (STSTATUS) consists of a number of single bit fields identifying in detail the conditions which determined the storing of the unit status.

The Residual Paragraph Count field (PARCOUNT) is meaningful only when the status is stored in conjunction with the completion of a command sequence and indicates the number of unused paragraphs in the data area identified by the last unit command. The residual paragraph count will always be zero upon normal completion.

The Command Address Segment Identifier (CASEGID) and the Command Address Displacement (CADISPL) form the system virtual address which is 16 higher than the address of the last command whose execution has been initiated.

Whenever a condition occurs which requires the storing of the unit status, the unit checks the processor notification line to determine if the previous status has been acknowledged. If not (i.e., the processor notification line is still on) the unit enters the status pending state. As soon as such acknowledgement is received, the unit will continue with the regular procedure.

If the processor notification line is down, the status is stored at the unit status address and then the processor notification line is set to 1. If the status storing procedure was triggered by any occurrence other than the detection of the PCI flag being set to 1, the unit is set to the free state. If the status storing operation was triggered by the PCI flag, the unit will proceed to the execution of the active command.

4.7.1 Standard Status

The format of the standard status field is given in Table 4.7.1.1.

Table 4.7.1.1. Standard Status.

Bit	Field Name	Field Symbol
0	Program Controlled Interrupt	PCI
1	Status Pending Entered	SPEN
2	Unit Ready	URD
3	Unit Not Ready	UNRD
4	Invalid Command	ICMND
5	Invalid Command Address	ICA
6	Invalid Key or Data Address	IKD
7	Read Only Key or Data Address	ROKD
8	Corrected Fetch Error	CFE
9	STOP FLAG	STFL
10	Data Transfer Overrun	DTO
11	Data Specification (Non-Zero Key Detected)	DSP
12	Data Error	DERR
13	Stop Signal	STOP
14	Device Synchronization Lost	DSL

The Program Controlled Interrupt (PCI) flag is set to 1 whenever the status storing was caused by the setting of the PCI flag in the unit command.

The Status Pending Entered (SPEN) flag is set to 1 whenever the status storing operation was delayed due to the lack of acknowledgement of a previous status storing operation.

The Unit Ready (URD) flag is set to 1 whenever the unit completes the power on sequence and it is ready for operation. The status storing operation in this case is not associated with any unit command and the command address field in the unit status is undefined.

The Unit Not Ready (UNRD) flag is set to 1 in response to any command except the Setup Command whenever the unit is not in the ready status.

The Invalid Command (ICMND) flag is set to 1 whenever an incorrect command field specification or an invalid BLKADDR is detected.

The Invalid Command Address (ICA) is set to 1 wheneveran invalid system virtual address is detected by the channel while attempting to fetch a command.

The Invalid Key or Data Address (IKD) is set to 1 whenever an invalid address is detected while attempting to access the PSS for Key or Data.

The Read Only Key or Data (ROKD) is set to 1 whenever the Read Only State is associated with the PSS Key or Data Area. This exception can occur only in conjunction with the READ command.

The Corrected Fetch Error (CFE) is set to 1 whenever a correctable error has been detected and corrected in the process of fetching UCQ's, Keys or Data.

The STOP FLAG (STFL) is set to 1 whenever the status storing was due to detection of the STOP FLAG in the UCQ.

The Data Transfer Overrun (DTO) is set to 1 whenever the PSS can not keep up with the transfer of data to or from the Device.

The Data Specification (DSP) flag is set to 1 whenever a non-zero key is detected in the process of transferring data to the PSS.

The Data Error (DERR) flag is set to 1 whenever an error in the data stored in the HSSSU has been detected. In this case the Extended Status (EXSTATUS) portion of the unit status will contain additional model dependent information.

The Stop Signal (STOP) flag is set to 1 whenever the status storing was due to detection of a signal on the stop line.

The Device Synchronization Lost (DSL) is set to 1 whenever the HSSSU is discovered to have lost track of the orientation of the device. Orientation will be re-established when the device next passes its origin point.

CHAPTER 5

STANDARD INPUT/OUTPUT CHANNEL

INDEX

5.1 INTRODUCTION

The Standard Input/Output Channel (SIOC) provides the means for the transfer of data between a large variety of storage and input/output devices on one hand and primary subsystems on the other.

The SIOC device interface is compatible with the one accepted by a large number of the most popular devices.

The list of devices which can be attached to the SIOC includes drums, fixed head disk drives, movable head disk drives, mass storage devices using optically and magnetically coded strips, tape drives, line printers, card read/punch units, microfilm printers, OCR devices, etc.

The SIOC is a special purpose processor with two basic functions, namely:

1. to implement the controlled transfer of data between a primary storage subsystem and a device;

2. to control certain functions internal to the devices themselves.

The SIOC has three basic interfaces:

1. the control interface
2. the primary storage subsystem interface
3. the external (device) interface.

The control interface connects the SIOC to a general purpose processor which is responsible for initiating channel operations and for accepting notification of the completion of channel operations.

The PSS interface is used for three distinct purposes:

1. the transfer of data to and from the PSS;

2. the transfer of channel commands from the PSS to the channel; channel commands determine the actual function performed by the channel;

3. the transfer of channel status information from the channel to the PSS, normally at completion of an operation.

The external interface is used to transfer data to and from the external devices.

The typical channel operation is initiated by a start signal received through the control interface. The channel will then execute a channel program consisting of one or more channel commands which will be fetched through the PSS interface. Upon completion of the last command the channel will store its status back in the PSS and then stop, waiting for a new start signal.

Each SIOC channel has the capability to support more than one set of control interface lines and more than one set of PSS interface lines. In addition the external interface can be multiplexed among many external devices. (See Figure 5.1.1.)

Because of the necessity of controlling internal functions of the devices (e.g., tape rewind, track seek, etc.) an appreciable portion of a typical channel program execution time is spent without any transfer of data taking place.

In order to use the channel resources more effectively the SIOC is therefore organized in such a way as to maintain control simultaneously over a number of channel programs (provided they address distinct devices). Only one such channel program, however, may be transferring data at any given time.

This "block multiplexor" organization allows the SIOC to multiplex its data transfer capability among many devices on a "block" or "record" basis.

In order to support this function the SIOC is organizaed as a collection of subchannels. The number of subchannels within a SIOC is a function of the particular model. Each subchannel can support one channel program.

The operation of the SIOC is determined by its control registers, the functions associated with the control interface, the sequencing of commands and the functions associated with each command. The following sections will describe such functions in detail.

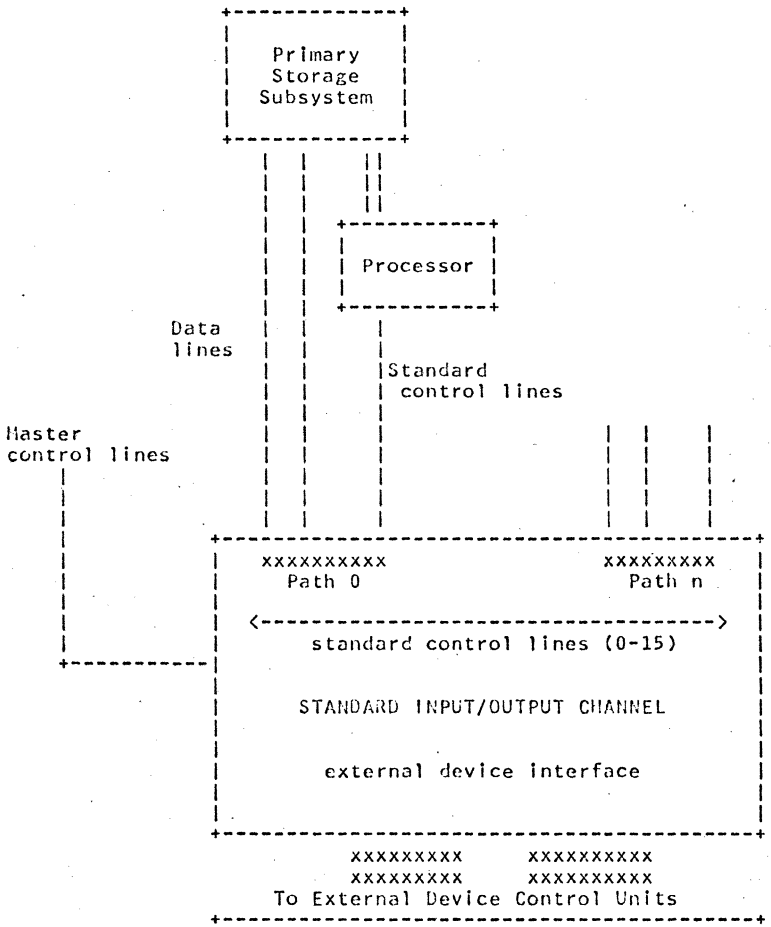


Figure 5.1.1 SIOC Interface Structure

5.2 CONTROL REGISTERS

Associated with each SIOC there are two channel control registers, namely the Standard Control Path Register (SCPR) and the Channel Command Path Register (CCPR) which affect the channel as a whole.

In addition associated with each subchannel there are three subchannel control registers, namely the Subchannel Unit Address Register (SUAR), the Subchannel Status Address Register (SSAR) and the Subchannel Command Address Register (SCAR); they only affect the operation of the specific subchannel.

5.2.1 Standard Control Path Register

The Standard Control Path Register (SCPR) is 4 bits wide and determines which, out of possibly 16, standard control interface path is active (see Section 5.4). active.

At any given time the channel will respond to standard control signals only if they are coming through the active standard control interface and will only send signals through said interface. The SCPR can only be set in conjunction with a channel reset signal (see Section 5.4).

5.2.2 Channel Command Path Register

The Channel Command Path Register (CCPR) is 4 bits wide and determines which, out of possibly 16, primary storage subsystem interface path is active.

At any given time all channel commands are fetched by the channel through the primary storage subsystem interface selected by the CCPR. In addition, the storing of the channel status is also performed through said interface.

The CCPR can only be set in conjunction with a channel reset signal.

5.2.3 Subchannel Unit Address Register

The Subchannel Unit Address Register (SUAR) is 16 bits wide and will contain the address of the unit to which the given subchannel is presently connected (if any).

The SUAR can be set by executing a Select command within a channel program.

5.2.4 Subchannel Status Address Register

The Subchannel Status Address Register (SSAR) is 40 bits wide and determines the system virtual address at which the channel status will be saved whenever necessary. The low order four bits of the address are assumed to be zero. Such address will be called the Subchannel Status Address (SSA).

The SSAR can be set by executing a Setup command within a channel program.

5.2.5 Subchannel Command Address Register

The Subchannel Command Address Register (SCAR) is 40 bits wide and determines the system virtual address from which the next channel command for that subchannel must be fetched. The low order four bits of the address are assumed to be zero. Such address will be called the Subchannel Command Address (SCA).

The SCAR is initially set to the value of the SSAR plus 16. The SCAR can be set to a new value by executing either a Select or Transfer in Channel command within a channel program.

The SCAR is incremented by 16 on every command fetch. The incrementation is done without a carry from bit position 16 into bit position 15, i.e., the low order 24 bits of the SCA are incremented modulo 2**24.

5.3 STATES AND CONTROL FLAGS

The channel as a whole and each subchannel can be in a number of basic states. In addition the setting of certain control flags will also determine the operation of the channel.

The channel is in active state whenever the execution of a channel command has been initiated and the channel end condition (see Section 5.6) for that command has not yet been detected. The command in progress will be labeled the active command. Such command must belong to a channel program associated with one subchannel. That subchannel is called the active subchannel. Associated with the active command there is the active channel status. (See Section 5.6.)

The channel will enter the status pending state whenever, upon attempting to store the channel status, it finds that a previous status storing operation has not yet been acknowledged (see Section 5.4).

The channel will enter the halted state whenever a hardware malfunction is detected or a primary storage subsystem exception occurs while attempting to store the channel status.

If the channel is not in any of the previous states, the channel is said to be available.

A subchannel is free whenever there is no channel program in progress associated with that subchannel. Otherwise the subchannel is busy.

Associated with each subchannel there is a skip state flag which is set upon completion of a command according to the setting of the skip control flag in the command and the value of the status modifier bit (see Sections 5.5 and 5.6). The skip state flag is used in controlling channel command sequencing. The skip state flag is reset to zero at the end of a command chain (see Section 5.5).

5.4 CONTROL INTERFACE

The control interface consists of two distinct sets of lines, namely th master control lines and the standard control lines.

There is only one set of master control lines for each SIOC. They control the channel as a whole and they are not associated with any specific control path or subchannel.

There are up to 16 sets of standard control lines. Only one set is active at any given time as determined by the standard control path register (see Section 5.2).

Control lines can be of one of three types: data, input or output.

Data lines are used to set certain channel registers, in conjunction with an incoming signal on an input line.

Input lines are used to transmit signals from an external source to the channel. Associated with each input line there is an acknowledge line used by the channel to communicate to the external source acknowledgment of an input signal. An input signal is detected whenever the input line is set to 1. To acknowledge an input signal means to set to 1 the corresponding acknowledge line. Whenever the input line is set to zero by an external source, its corresponding acknowledge line is also set to zero.

Output lines are used to transmit signals from the channel to an external device. Associated with each output line there is an acknowledge line used by the external device to communicate to the channel acknowledgement of the output signal. An output signal is generated by setting the output line to 1. The external device will acknowledge the signal by setting the acknowledge line to 1. Whenever such acknowledgement occurs, the output line is reset to zero by the channel.

In the definition of each control line we will indicate in parentheses if the line is data, input or output line.

5.4.1 Master Control Lines

There are 10 master control lines: the channel reset line (input), the four control path reset lines (data), the four command path reset lines (data) and the channel check line (output).

The master control lines are permanently connected to the System Monitor Unit (SMU) (see Chapter 9). Through channel reset procedures it specifies which processor is controlling the channel operations and from which PSS the channel commands are to be fetched.

Whenever a channel reset signal is detected, the channel reset procedure is executed (see Section 5.6). Such procedure resets the channel and all the devices attached to it to a known, "quiescent" state. As part of the procedure the standard control path register is set to the value represented by the state of the four control path reset lines and the channel command path register is set to the value represented by the state of the four command path reset lines.

This is the only procedure by which the active control path and the active command path can be modified.

The channel check line is used to notify the System Monitor Unit of the occurrence of a hardware malfunction in the channel.

5.4.2 Standard Control Lines

Each set of standard control lines consists of three control lines per subchannel. In other words if there are 12 subchannels associated with a given channel each set of standard control lines will consist of 36 control lines.

The three lines associated with each subchannel are the subchannel start line (input), the subchannel stop line (input) and the processor notification line (output).

The subchannel start signal directs the subchannel to start execution of a channel program.

The subchannel stop signal directs the subchannel to abort the current command and terminate the channel program.

The processor notification signal is issued by the channel in conjunction with the storing of the channel status (see Section 5.6).

5.5 CHANNEL COMMANDS AND CHANNEL PROGRAMS

Operations on a subchannel are initiated by the detection of the start signal for that subchannel. If a previously initiated operation on that subchannel is in progress at the time the subchannel start line is set to 1, the start signal will remain pending until the subchannel becomes free.

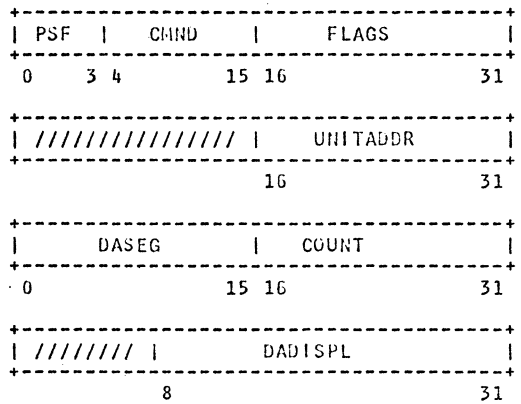
The subchannel start signal is acknowledged as soon as the channel has fetched the first channel command quadword (see below).

Upon detection of the start signal the subchannel command address register (SCAR) is set to the value of the subchannel status address register plus 16, i.e., it is set to point to the next higher 16_byte block (quadword) and normal operation is initiated.

For each occurrence of the start signal the subchannel executes a full channel program. A channel program consists of one or more command chains, each chain consisting of one or more channel commands.

Commands belonging to the same chain are connected by either data chaining or command chaining as determined by the flags in each command (see Section 5.5.1). Each command chain must start with a Select or Setup command; furthermore, if the command is a Setup command it must be the only command in the chain.

Each channel command is fetched by the SIOC from the primary storage subsystem selected by the channel command path register. Commands are in the form of Channel Command Quadwords (CCQ) which have the format of Figure 5.5.1.



Word	Bits	Field Name	Field Symbol
0	0-3	Path Selection Field	PSF
	4-15	Command Code	CMND
	16-31	Flags	FLAGS
1	16-31	Unit Address	UNITADDR
2	0-15	Data Address Segment Identifier	DASEG
	16-31	Count	COUNT
3	8-31	Data Address Displacement	DADISPL

Figure 5.5.1. Channel Command Quadword.

The Path Selection Field (PSF) determines which primary storage subsystem data path, out of 16 possible, has to be used for the data transfer associated with the command. Such selection is totally independent from the path selected for channel command fetching and channel status storing.

The Command Code field (CMND) identifies the type of operation to be performed. Often not all of the bits in the command field are used by the channel itself, but they are sent by the channel to the selected unit for further interpretation.

The Unit Address field (UNITADDR) identifies the unit to which the channel program is directed in a Select command. It is ignored in all other commands.

The Flags field (FLAGS) contains individual control bits (flags) which determine various modes of operation (see Section 5.5.1).

The Byte Count (COUNT) determines the number of bytes to be transferred.

The Data Address Segment Identifier (DASEG) and the Data Address Displacement (DADISPL) form a system virtual address identifying either a data area address, or a new value for the subchannel command and status address registers.

5.5.1 Channel Command Flags

The channel command flags (bits 16-31 of word 0 of the CCQ) permit modifications of the basic operation associated with each command. Each flag is generally interpreted in the same way in each command for which is meaningful; in some commands certain flags are ignored.

Table 5.5.1.1 provides the list of the defined flags. Bit positions to which no meaning has been assigned should be set to zero in order to guarantee compatibility with possible future additions.

Table 5.5.1.1. Channel Command Flags.

Bits	Flag Name	Flag Symbol
0-3		
4	Skip Mode	SKM
5	Skip Control	SKC
6	Stop	STOP
7		
8	Chain Data	CD
9	Chain Command	CC
10	Suppress Length Indication	SLI
11	Null Data Transfer	NDT
12	Program Controlled Interrupt	PCI
13-15		

The Skip Mode flag (SKM) in conjunction with the status modifier bit determines the setting of the subchannel skip flag at the completion of the channel command. The skip flag will always be set to the exclusive or of the SKM flag and the status modifier bit. It should be noted that for most commands the status modifier bit will always be zero.

The skip mode flag setting in a given command does not affect the execution of that command but may determine if the next command will be executed or skipped.

The Skip Control flag (SKC) determines the effect of the subchannel skip flag on the execution of the command. If both the subchannel skip flag and the skip control flag are set to 1, the command will be skipped. Skipping of a command will not reset the subchannel skip flag. It is therefore possible to skip any number of commands (provided they are either data or command chained).

The Stop flag (STOP) indicates that the command is the last command in a channel program. The Stop flag will be ignored if either the CD or CC flags are on.

The Chain Data flag (CD) permits the transfer of data between a single physical record and more than one data area in the primary storage subsystem. If the CD flag is on and the byte count is exhausted before the unit signals channel end then the data area from the next CCQ will be used.

If the channel end condition is detected before or just when the count is exhausted and the SLI flag (see below) and the CD flags are on, the current command will be assumed to be completed and the subchannel command address register will be advanced until it points to the CCQ following the first command whose CD flag is off. If any of the intervening CCQ had the SLI flag off then the channel program will be terminated as if the channel end condition had been detected just at the beginning of that command.

In other words, a string of channel commands which all have the CD flags on and the immediately following command with the CD flag off are viewed, for command sequencing purposes, as a single command (with the explained role of the SLI flags).

The Chain Command flag (CC) indicates that the next CCQ should be executed immediately following detection of a device end condition for the current command. If the channel end condition is detected separately from (necessarily before) the device end condition, the channel will become available unless there is some activity pending on some other subchannel.

The CC flag being off in a command which is actually executed (i.e., not skipped and not a part of a data chain) signals the end of a command chain.

If the STOP flag is on the subchannel, upon detecting a channel end condition (with or without device end), will release the unit, proceed to store the status at the subchannel status address and will then become free.

If the STOP flag is off the subchannel will proceed as if the CC flag were on with the difference that the unit will be released upon detecting the device and condition, the skip state flag will be reset to zero and the next command will be checked for being either a Select or a Setup command.

The Suppress Length Indication flag (SLI) determines if the occurrence of a mismatch between the data record length, as specified in the channel program, and the physical record length should stop execution of the channel program (SLI=0) or not (SLI=1).

The effect of the Null Data Transfer flag (NDT) is a function of the direction of the data transfer associated with the command. When associated with commands that transfer data to the Primary Storage Subsystem ("read" commands), data transfer will be suppressed if the flag is on. When associated with commands that transfer data from the PSS ("write" commands), data consisting of all zeroes will be supplied by the channel to the external unit whenever the flag is on.

The Program Controlled Interrupt flag (PCI) is used to notify the processor controlling the channel that a specified point in the channel program has been reached. Whenever the PCI flag is on in the active command, a status storing procedure is executed before initiating execution of the channel command. The PCI flag is also interrogated in a CCQ which is data chained to a previous CCQ.

Table 5.5.1.2 lists the actions taken by the SIOC upon occurrence of a channel end condition or exhaustion of the byte count as a function of the setting of the CD, CC, SLI and STOP flags.

The symbol X in correspondence to the flag setting in Table 5.5.1.2 denotes that the setting of the particular flag is irrelevant in that case.

Table 5.5.1.2 Action in Channel upon Exhaustion of Count or Receipt of Channel End.

FLAGS				CONDITIONS		
			Count Channel STOP	Exhausted --	Exhausted End	Not Exh'd End
CD	CC	SLI	STOP			
0	0	0	0	Stop, IL	New Chain	End, IL
0	0	0	1	Stop, IL	End	End, IL
0	0	1	0	New Chain	New Chain	New Chain
0	0	1	1	Stop	End	End
0	1	0	X	Stop, IL	Chain Cmnd	End, IL
0	1	1	X	Chain Cmnd	Chain Cmnd	Chain Cmnd
1	X	0	X	Chain Data	End, IL	End, IL
1	X	1	X	Chain Data	Next CCQ	Next CCQ

The actions listed in Table 5.5.1.2 are defined as follows:

End	Operation is terminated. Status is stored.
Stop	Device is signaled to terminate data transfer; channel waits for channel end and then the End procedure is executed.
IL	Incorrect length indication is provided in the channel status.
Chain Command	Upon receipt of device end the channel proceeds to the execution of the next command.
Chain Data	The channel proceeds to use the next CCQ to continue the same operation.
NextCCQ	The next CCQ is made current and execution is continued. Channel end condition remains outstanding.

New Chain — Upon receipt of device end the device is released, the skip state flag is set to zero and the next CCQ is made current as the first CCQ of a new chain.

5.5.2 Command Codes

The SIOC recognizes eight commands whose codes are given in Table 5.5.2.1. The symbol X indicates that the bit position is ignored. The symbol M identifies a modifier bit which is normally not examined by the SIOC bit is passed to the unit.

Table 5.5.2.1. Command Codes.

Code	Command
0000MMMM0100	Sense
0000XXX01000	Transfer In Channel
0000XX011000	Setup
0000XX111000	Select
0000MMMM1100	Read Backward
0000MMMMMM01	Write
0000MMMMMM10	Read
0000MMMMMM11	Control

All other codes are invalid.

5.5.3 Sense Command

5.5.4 Transfer_in_Channel Command

In the Transfer_in_Channel command the path selection field, the unit address field and the count field are ignored. The CD and CC flags in a Transfer_in_Channel command are ignored, but the effect of the CC and CD flags on the previous command is carried on to the next command. The SLI and NDT flags do not apply. The Stop flag is ignored.

The Transfer_in_Channel command resets the subchannel command address register to the value specified by the data address in the command. The low order 4 bits of the displacement are ignored and assumed to be zero. For the purpose of determining the effect of the SKM flag, the status modifier with the Transfer_in_Channel command is

always zero.

If this command occurs in the midst of a data chain or command chain sequence it does not affect the logic of command sequencing.

The Transfer_in_Channel command will be used to construct loops within a chained program or to link two sections of a channel program located at non-contiguous addresses.

If an exceptional condition arises during execution of a Transfer_in_Channel command, the command address in the channel status will point to the location following the TIC command, not to the location pointed to by the TIC command itself.

5.5.5 Setup Command

In the Setup command, the path selection field, the unit address field and the count field are ignored. The CD and CC flags are ignored and assumed to be zero. The SLI and NDT do not apply.

The Setup command resets the subchannel status address register to the value specified by the address in the command. The low order 4 bits of the displacement are ignored and assumed to be zero.

The Setup command can only occur as the only command in a chain.

If an exceptional condition occurs during execution of a Setup command, the channel status will be stored at the old value of the subchannel status address.

5.5.6 Select Command

In the Select command the path selection field and the count field are ignored. The CD flag is ignored and assumed to be zero; the CC flag is ignored and assumed to be 1 (which implies that the STOP flag is ignored). The SLI and NDT flags do not apply.

The Select command sets the subchannel unit address register (SUAR) to the value of the unit address field. The data address field is used as in the transfer in channel command. For the purpose of determining the effect of the SKM flag, the status modifier associated with a Select command is always zero.

The Select command can only occur as the first command of a chain.

5.5.7 Read Backward Command

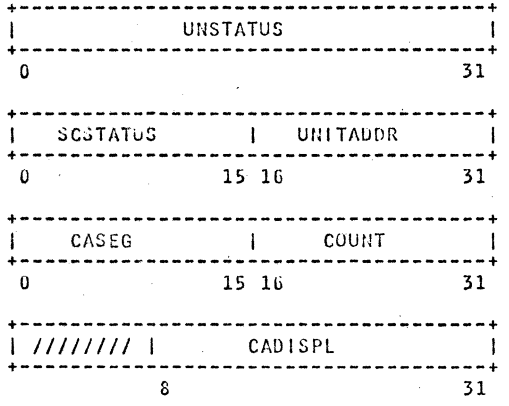
5.5.8 Write Command

5.5.9 Read Command

5.5.10 Control Command

5.6 CHANNEL STATUS

Upon termination of a channel program (either normal or abnormal), upon detection of the PCI flag on in the active command or in case of a device attention interrupt (see Section 5.6.1) the channel will attempt to store the channel status at the subchannel status address for the appropriate subchannel. The channel status consists of four words, the Channel Status Quadword (CSQ), and its format is given in Figure 5.6.1.



Word	Bits	Field Name	Field Symbol
0	0-31	Unit Status	UNSTATUS
1	0-15	Subchannel Status	SCSTATUS
	16-31	Unit Address	UNITADDR
2	0-15	Command Address Segment Identifier	CASEG
	16-31	Residual Count	COUNT

3 8-31 Command Address Displacement CADISPL

Figure 5.6.1. Channel Status Quadword.

The Unit Address field (UNITADDR) identifies the active unit on the subchannel at the time the channel status has been stored.

The Residual Count field (COUNT) is meaningful only when the status is stored at the termination of the channel program and indicates the number of unused bytes in the data field of the last executed channel command.

The Command Address Segment Identifier (CASEG) and the Command Address Displacement (CADISPL) form a system virtual address which is 16 higher than the address of the last command whose execution has been initiated.

The Unit Status (UNSTATUS) and Subchannel Status (SCSTATUS) fields consist of a number of single bit fields identifying in detail the conditions which determined the storing of the channel status.

Whenever a condition requiring the storing of the channel status is detected, the subchannel checks the processor notification line to determine if the previous status has been acknowledged. If not (i.e., the processor notification line is still on) the channel enters the status pending state. As soon as the acknowledgement for the previous status storing operation is received, the channel will continue with the regular procedure.

If the processor notification line is down, the status is stored at the subchannel status address and then the processor notification line is set to 1. If the status storing procedure was associated with the termination of the channel program, the subchannel is set to the free state. If the status storing was triggered by the PCI flag, the subchannel continues execution of the current command.

If the termination was signalled by a device check the channel will not proceed with any other command until receiving a new start signal for that particular subchannel.

5.6.1 Unit Status

The format of the unit status field is given in Table 5.6.1.1.

Table 5.6.1.1. Unit Status.

Bit	Field Name	Field Symbol
0	Attention	ATT
1	Status Modifier	STMOD
2	Control Unit End	CUEND
3	Busy	BUSY
4	Channel End	CHEND
5	Device Check	DVEND
6	Unit Check	UNCHK
7	Unit Exception	UNEXC
8	Not Operational	NOTOP
9-31	Unassigned	

The Attention flag is set to 1 whenever a device detects an asynchronous condition which is of relevance to the device handling program. The condition can only be detected whenever no operation is in progress in the I/O device and the control unit. All such occurrences are not therefore naturally associated with a specific subchannel. The SIOC will, by convention, automatically associate the attention condition with subchannel 0 (zero).

The Status Modifier flag may be set to 1 in conjunction with certain unit operations, usually to indicate if a certain condition was or was not satisfied.

The Control Unit End flag is set to 1 whenever an unusual condition is detected by a control unit after channel end or whenever a control unit has been interrogated while in the busy state.

The Busy flag is set to 1 whenever an I/O device or control unit cannot execute the command because it is executing a previously initiated operation or because it contains a pending interrupt condition.

The Channel End flag is set to 1 whenever the transfer of data or control information is terminated. The occurrence of this condition makes the channel available for another operation. Detailed handling of the channel end condition as a function of the setting of the CCQ flags has been given in Table 5.5.1.2.

The Device End flag is set to 1 upon completion of an I/O operation at the device or, on some devices, by changing the device from the non-ready to the ready state. The device end condition is always generated simultaneously with or later than the corresponding channel end condition.

The Unit Check flag is set to 1 whenever a unit detects an unusual condition different from the one associated with unit exception (see below). The sense data will contain additonal information identifying the type of condition.

The Unit Exception flag is set to 1 whenever a specific (command and device dependent) unusual condition has been detected.

The Not Operational flag is set to 1 whenever the specified unit address does not correspond to any attached device of the device is disconnected at the time.

5.6.2 Subchannel Status

The format of the subchannel status field is given in Table 5.6.2.1.

Table 5.6.2.1. Subchannel Status.

Bit	Field Name	Field Symbol
0	Program Controlled Interruption	PCI
1	Incorrect Length	IL
2	Program Check	PCHK
3	Interface Control Check	ICC
4	Chaining Check	CHC
5	Invalid System Virtual Address	ISVA
6	Read Only Data	ROD
7	Corrected Error	SEC
9	Uncorrectable Data Error	UDE
10	Corrected CCQ Error	CCE
11	Uncorrectable CCQ Error	UCE

The Program Controlled Interrupt flag is set to 1 whenever the channel status is stored upon detection of the PCI flag on in a CCQ.

The Incorrect Length flag is set to 1 whenever the channel end condition and the exhaustion of the byte count in a CCQ do not occur simultaneously, provided that the SLI flag in the CCQ is not 1.

The Program Check flag is set to 1 whenever an incorrect field specification occurs in a CCQ.

The Interface Control Check flag is set to 1 upon detection of an invalid signal on the I/O interface.

The Chaining Check flag is set to 1 whenever a channel overrun occurs during data chaining.

The Invalid System Virtual Address flag is set to 1 whenever an invalid system virtual address is detected by the channel upon attempting to fetch a command or access a data byte.

The Read Only Data flag is set to 1 whenever the Read Only data condition is detected by the channel in attempting to store data in the primary storage subsystem.

The Interface Control Check flag is set to 1 upon detection of an invalid signal on the I/O interface.

The Chaining Check flag is set to 1 whenever a channel overrun occurs during data chaining.

The Corrected Error flag is set to 1 whenever a PSS data error was detected and corrected. Occurrence of such errors does not affect the execution of the channel program.

Throughout the MASCOR Reference Manual consistent notational conventions have been used.

Familiarization of these notations will aid in a correct interpretation of the various detailed functional descriptions of the MASCOR 132 System.

Symbol	Definition
d	Displacement : A 24-bit non-negative integer used generally as the second operand in address-addition.
i,j,k	Operand register pointers : The 6-bit non-negative integers contained in various instructions which define the operands.
sd	Short displacement : A 15-bit non-negative integer, extended with zeroes on the left to form a displacement.
wd	Word displacement : A 22-bit non-negative integer, extended by two zero bits on the right to form a displacement.
Wx	Contents of General Purpose Register x. Wx has a length of 32 bits.
Dx	Contents of an even-odd General Purpose Register pair; x must be even. Dx has a length of 64 bits.
Qx	Contents of a set of four General Purpose Registers; x must be even. Qx has a length of 128 bits.
Bx	Contents of a byte position in the General Purpose Registers; x may be 0 to 255. Bx has a length of 8 bits.
Wx/Dx	Wx or Dx, depending on context.
B(x)	Value of a byte (8 bits) whose EVA is x.
H(x)	Value of a halfword (16 bits) whose EVA is x; x must be even.
W(x)	Value of a word (32 bits) whose EVA is x ; x must be a multiple of four.
D(x)	Value of a doubleword (64 bits) whose EVA is x; x must be a multiple of eight.

Symbol	Definition
W(x)/D(x)	W(x) or D(x), depending on context.
W/D	Wx, W(x), Dx, or D(x), depending on context.
K	Litteral value of the k-field : A 6-bit non-negative integer, extended with zeroes on the left as required.
IA	Instruction address : Contents of the Instruction Address Register (IAR).
ITC	Iteration Count : Contents of the Iteration Count Register (ITCR).
B'b1b2...bn'	A field of length n bits containing the bitvalues bi indicated.
X'x1x2...xn'	A field of length 4*n bits containing the hexadecimal digit values xi indicated.
D'd1d2...dn'	A field of length 4*n bits containing the binary coded decimal digit values di indicated.
C'c1c2...cn'	A field of length 8*n bits containing 8 bit representations of the alphanumeric characters ci indicated.
0	A field of appropriate length containing only zero (0...0) bits.
...	Ellipsis : A shorthand notation for an extension of the preceding symbols to fill a field to the appropriate length.
X<m>	The m-th bit of field X. The leading bit is designated as bit 0.
X<m-n>	A field of length (n-m+1) bits beginning at bit m of field X.

Symbol	Definition
:=	The assignment operator : The operation of copying the rightside identically into the leftside.
+ - * /	The arithmetic operators : addition, subtraction, multiplication, division as appropriate for the datatype in context.
**	The exponentiation operator : integer to the power positive-integer.
> = <	The basic comparision operators : Greater than, equal to, less than as appropriate for the datatype in context.
>= ¬= <=	The composite comparision operators : greater than or equal to, not equal to, less than or equal to.
¬ | &	The bit by bit logical operators : not, or, and.
MOD(x,y)	x modulo y : The remainder after integer division of x by y. The result will have the same length attribute as y.
ABS(x)	The absolute value of x : ABS produces x if x is positive and -x if x is negative. The result will have the same length attribute as x.
x||y	The concatenation operator : Concatenation produces a bit vector consisting of the bits of x followed by the bits of y. The result has a length equal to the sum of the lengths of x and y.
x@y	Address addition of x and y : Y is taken to be a displacement to be added to the base address x, or x<0-7> || MOD(x<8-31> + y<8-31> , 2**24)