Burroughs

B 7800 Information Processing Systems

REFERENCE MANUAL

PRICED ITEM

Printed in U.S.A. August 1979 5010796

Burroughs

B 7800 Information Processing Systems

REFERENCE MANUAL

PRICED ITEM

PCN 5010796-001

Printed in U.S.A. August 1979 5010796

Burroughs believes that the information described in this manual is accurate and reliable, and much care has been taken in its preparation. However, no responsibility, financial or otherwise, is accepted for any consequences arising out of the use of this material. The information contained herein is subject to change. Revisions may be issued to advise of such changes and/or additions.

Warning: This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. As temporarily permitted by regulation it has not been tested for compliance with the limits for Class A computing devices pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference, in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

Correspondence regarding this document should be addressed directly to Burroughs Corporation, Box CB7, Malvern, PA. 19355, Attn: Systems Documentation Dept., TIO East.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT.)

TABLE OF CONTENTS (CONT.)

TABLE OF CONTENTS (CONT.)

TABLE OF CONTENTS (CONT.)

TABLE OF CONTENTS (CONT.)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (CONT.)

LIST OF TABLES

INTRODUCTION

This system reference manual presents the technical details about the general architecture, the components, and the subsystems of the Burroughs B 7800 Information Processing System, which is the most advanced, the largest, and the most powerful member of the Burroughs family of 700 systems.

The chapters of this reference manual are as follows:

Chapter 1, *Description of the B 7800 System,* introduces the idea of the interaction of independently operating computing, input/output, and memory modules through an exchange and a presentation of the range of configurations of the system.

Chapter 2, *System Architecture,* discusses data representation, Polish notation and stack concepts, processor control words, and the concepts of the input/output subsystem map.

Chapter 3, *Central Processor Module,* contains a functional description of the operation of the central processor module, an explanation of hardware interrupts, and a brief description of each program operator.

Chapter 4, *Input/Output Subsystem,* contains a general description of the operation of the input/output module, functional descriptions of the subsections of the input/output module, and detailed descriptions of the control words and descriptors associated with each type of peripheral device that may be included in the system.

Chapter 5, *Memory Subsystem,* contains a general description of the memory subsystem and details about both the memory control module and the memory storage unit.

Chapter 6, *Maintenance Diagnostic Processing,* contains a general description of the maintenance diagnostic processing, a functional description of both the programmer and card tester, and a general description of the control words associated with maintenance diagnostic processing.

The term *software,* as used in this manual, applies to that category of Burroughs Program Products defined as *Systems Software.*

Other categories of Burroughs Program Products are:

Application Program Products
Program Product Conversion Aids

CHAPTER 1
DESCRIPTION OF
B 7800 SYSTEM

THE B 7800 SYSTEM

The Burroughs B 7800 Information Processing System is a large-scale, general-purpose, balanced, flexible, multiprogramming, and multiprocessing computing system suitable for such diverse applications as time sharing, scientific problem solving, and business data processing. The B 7800 is completely object code compatible with B 7700, B 6700, and B 6800 systems and affords Burroughs users the opportunity for growth without reprogramming or recompiling. Object code users' programs that can be executed successfully on the B 7700, B 6700, and B 6800 can be executed without modification on the B 7800.

The system is able to: 1) handle complex data structures and sophisticated program structures dictated both by higher-level languages now in use and by the requirements of advanced problems; 2) efficiently manage the massive on-line and archival storage requirements of large data bases; and 3) accommodate vast networks of data communications devices.

The B 7800 is a very fast, modular parallel processing system with versatility in configuration. The B 7800 can be tailored to the processing needs of a user by arranging central processor modules, input/output processors, and memory modules on an exchange (figure 1-1). If the high performance and adaptability of the B 7800 could be attributed to a single factor, it would be to the balance attained by the controlled interaction of independently operating computing, input/output, and memory modules through the exchange. Thus, the throughput of the system as a whole is maximized, and the performance of no single element of the system is maximized to the neglect or detriment of others.

The key to the efficient balanced use of the system is the Burroughs master control program (MCP), a unique executive software operating system that automatically makes optimum use of all system resources. It is this operating system that makes multiprogramming and multiprocessing both functional and practical by controlling system resources and by scheduling jobs in the multiprogramming mix. In use, the master control program allocates system resources to meet the needs of the programs introduced into the computer. It continually and automatically reassigns resources, starts jobs, and monitors their performance.

Further implications of the modularity and flexibility of the system are its expandability (a capacity to add hardware modules without reprogramming) and its increased reliability (thus increasing availability to the user). The reliability is achieved by the use of continuous processing techniques that (in addition to providing for error detection, error correction, independence, and redundancy of power supplies) exclude faulty modules from the system and permit processing to continue (without reprogramming) even with a temporarily reduced configuration.

Although it is very large and immensely complicated, the B 7800 is comprehensible to the user. Programming is done only in higher-level, problem-oriented languages (COBOL, ALGOL, FORTRAN, PL/1, and ESPOL). The control language used in entering jobs into the system is a simple, free-form, English-like language, and the messages that pass between the system and the operator are brief, clear, and easy to learn.

DISTINGUISHING FEATURES

Although the balanced use of the principal components of the system (as a whole under the control and coordination of the master control program) is the key to the high throughput of the B 7800, the high performance of the system is in large part achieved by:

1. Improving the speed of execution of instructions.
2. Reducing or masking the overhead associated with references to memory.
3. Freeing the central processor from concern with input/output operations.
4. Employing continuous processing measures that minimize system degradation.

The three goals of the continuous processing features of the B 7800 are to: 1) keep the system running 100% of the time; 2) minimize system degradation; and 3) provide the user with tools for performing one's own data recovery. These goals are achieved by the combination of hardware and software throughout the system.

The first goal, to keep running, is achieved as follows:

1. By the high reliability of system hardware.
2. By the incorporation of error detection circuits throughout the system.

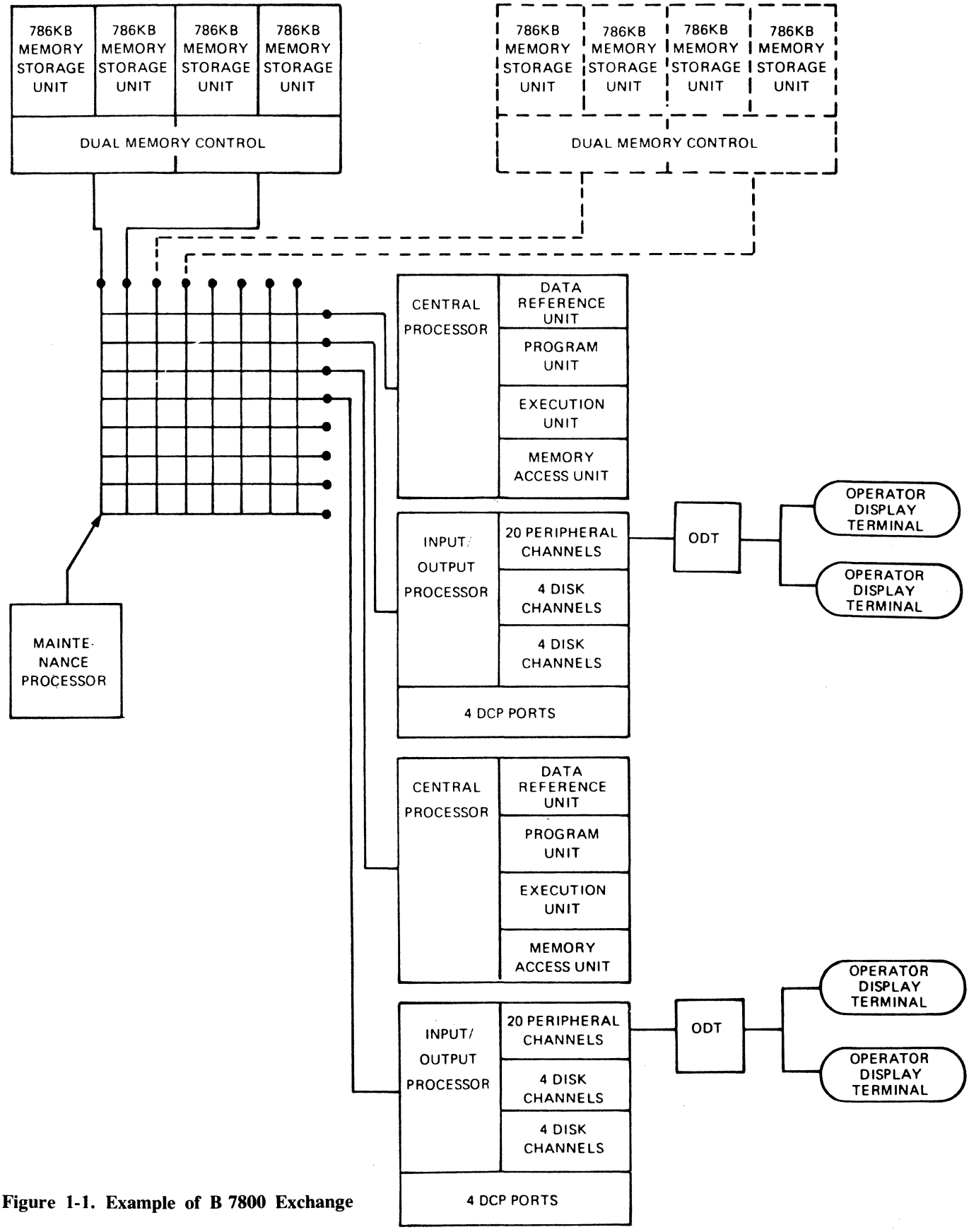


Figure 1-1. Example of B 7800 Exchange

3. By single-bit error correction of errors in memory.
4. By recording errors for software analysis.
5. By modular design. (The use of separate power supplies and redundant regulators for each module, and the use of redundant buses).
6. By the ability of the master control program to reconfigure the modules of the system to exclude (temporarily) a faulty module.
7. By automatic instruction retry. If a hardware malfunction occurs during the performance of an instruction, the master control program analyzes the error and writes the appropriate entry in the on-line maintenance log. The processor is reset to its state prior to the error and the instruction is performed again.

The detection and reporting of errors is done by hardware; analysis of errors is done by software; and the reconfiguration of the system is done by software. Because of the modularity of power supplies and the use of redundant regulated supplies for critical voltages, the impact of a malfunctioning dc supply is minimized and does not result in a catastrophic failure.

The second goal, to minimize system degradation, is achieved by providing diagnostic programs, and equipment for rapidly identifying and repairing faults and for re-establishing confidence in a repaired module before it is returned to the user's system. The diagnostic programs of the B 7800 system identify a faulty module.

By use of the maintenance diagnostic processing programs, a fault in any mainframe module is narrowed to a single clock period and to a flip-flop and associated logic circuitry. Once the maintenance diagnostic program has been used to isolate a fault to within one or more suspect circuit cards in a module, the card test facility can be used to test the card.

In addition to diagnostic programs, an interpreter program, MICI (module interrogation and command interpreter), allows the manipulation, control, interrogation, and display of B 7800 mainframe modules from a standard system SPO. The strategy of controlling modules, by use of MICI, is to exercise a suspected logic circuit for fault isolation.

The third goal, to provide the user with tools for performing their own data recovery, is achieved by the use of such features as installation allocated disk, protected disk files duplicated disk files, and fault statements in the higher-level programming languages used on the system.

Installation allocated disk allows the user to specify the physical allocation of critical disk files to facilitate the maintenance and reconstruction of these files. Protected disk files allow a user to gain access to the last portion of valid data written in a file before an unexpected system halt. The use of duplicated disk files is to avoid the problem of fatal disk file errors. The master control program maintains more than one copy of each disk file row, and, if access cannot be gained to a record, an attempt is made to gain access to a copy of the record. By use of fault statements, the user can stipulate actions to be taken by one's own programs in the event errors occur.

SYSTEM CONFIGURATION

Physically, the components of theB 7800 system fall into the following categories:

1. Central components of the B 7800 system: the central processor module; input/output processor; the memory module; the maintenance processor; and the operator's console (refer to table 1-1).
2. Standard Burroughs cabinets that contain peripheral controls and exchanges: the data communications processor; and ac power supplies.
3. Standard peripheral devices that are joined to the central system by standard Burroughs peripheral controls: adapters and exchanges; standard remote devices that are joined to the central system by line adapters; and data communications processor.

The arrangement of these components into a system and the size of the system depend on the application and workload of the user. In the following paragraphs, the range of configurations of the B 7800, the maximum configuration, and the minimum configuration is described.

Maximum Configuration

Figure 1-2 illustrates the theoretical maximum configuration of the B 7800 system.

As many as eight memory modules may be arranged on the exchange with a combined total of up to eight requestors of memory-central processor modules and input/output modules. Any single requestor of memory may address and gain access to the entire contents of high-speed main memory (1,048,576 words, or 6,291,456 eight-bit bytes). On the maintenance bus (which services the memory control modules, central processor modules, and input/output modules) a maintenance processor is placed.

Table 1-1. Central Components of the B 7800 System

B 7811	System includes: one central processor (8 MHz), one input/output processor with 24 data switching channels,* one maintenance processor, one operator console with dual displays and control.	B 7801	Additional central processor.
		B 7882	Additional input/output processor.
B 7821	System includes: two central processor (8 MHz), two input/output processors with 24 data switching channels each,* one maintenance processor, one operator console with dual displays and control.	B 7007	Dual Access Memory Control Module and four Memory Storage Units - 3,145,728 bites of storage, error correcting memory, 8-way interleaving that permits 8-word transfer to and from memory.
		B 7008	Dual Access Memory Control Module and two Memory Storage Units - 1,572,864 bites of storage, error correcting memory, 8-way interleaving that permits 8-word transfer to and from memory.
B 9955	Additional operator console and control with dual operator displays.		

* Throughout this manual the I/O Processor is referred to as the I/O Module.

At rates of up to 6.75 million bytes per second, a single input/output module is capable of transferring data simultaneously between main memory and 28 peripheral controls (including eight high-speed controls) and between main memory and as many as four data communications processors. At present, the maximum number of high-speed, medium-speed, and low-speed peripheral devices that may be attached through controls and exchanges to a single input/output module or that may be included in the input/output subsystem of the B 7800 is 255.

By suitable cross-connection through exchanges, it is possible to establish pathways between disk files, disk packs or magnetic tape units, and more than one input/output module; hence, these peripheral devices can be shared by all of the input/output modules in the system.

Among the peripheral devices available are disk file and disk pack memory modules that constitute a virtual memory that, in effect, greatly expand the storage capacity of the main memory of the system. These modules are interfaced with the input/output module.

In addition to the 255 peripheral devices that may be included in the input/output subsystem, there is a vast network of remote terminals, remote controller, and remote computers that can be accommodated by as many as 1024 remote lines. These are served by the four programmable data communications processors which can be controlled by a single input/output module. Normally, each line handles a number of remote devices; systems that have more than one input/output module can have more than one data communications network. Theoretically, the maximum number of data communications processors that could be included in a B 7800 system is 28. (Currently, the software can only handle a maximum of eight.)

Minimum Configuration

The smallest possible B 7800 system is composed of the central components listed below.

Central Components	Qty
Central processor module (CPM)	1
Input/output module (IOM)	1
Memory control module (MCM)	1
Memory storage cabinet (MSC)	1
Memory storage unit (MSU)	2
Maintenance Processor	1
Operator's console	1

In addition, the minimum configuration must contain a disk file memory subsystem large enough to hold the master control program, a card reader, a line printer, a magnetic tape unit, peripheral controls, and ac power cabinets. In practice, other peripheral devices and their controls are used with this minimum configuration.

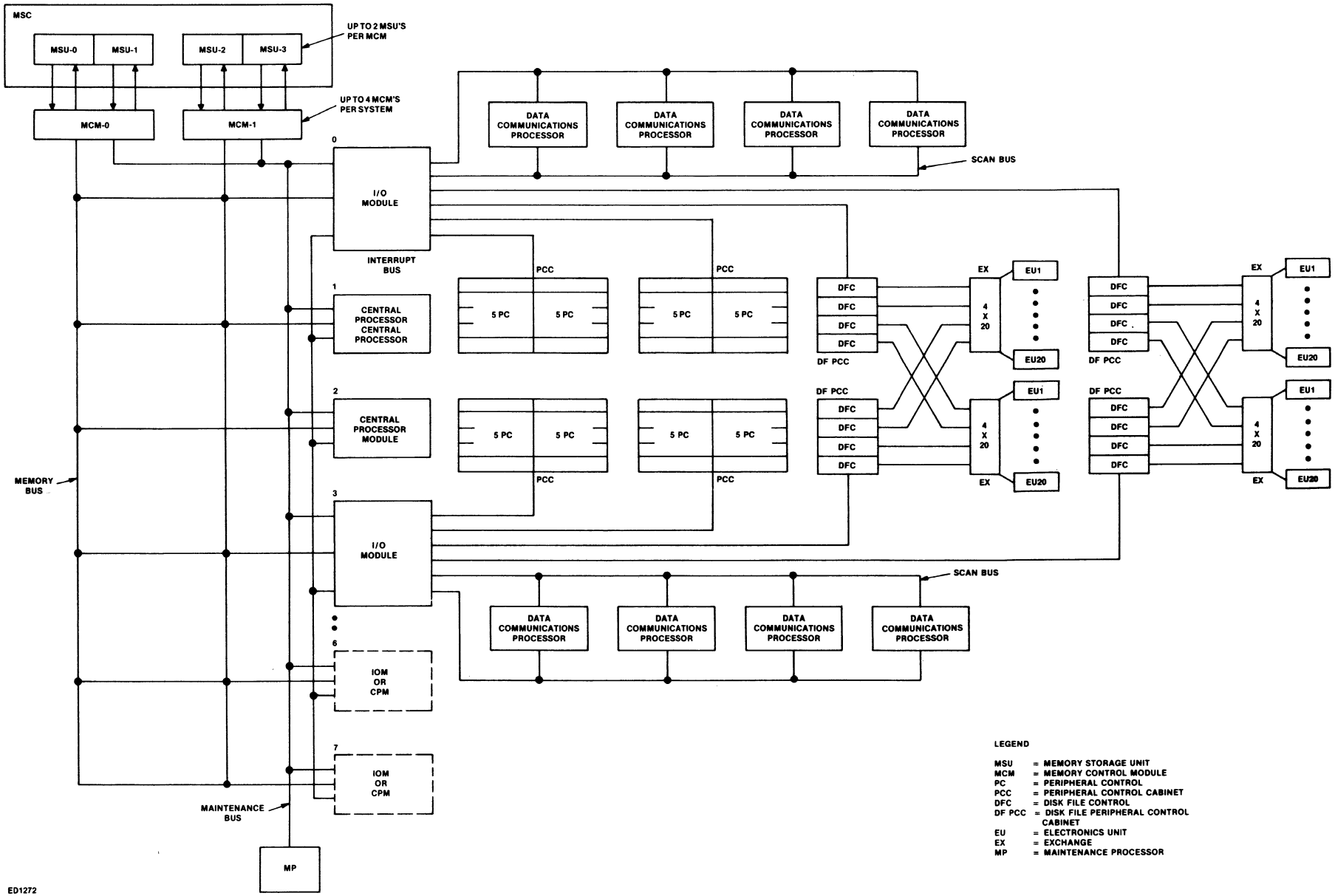


Figure 1-2. Maximum Configuration of the B 7800 System

CHAPTER 2
SYSTEM ARCHITECTURE

SECTION 1
DATA REPRESENTATION

GENERAL

The basic information structure used in the B 7800 Information Processing System is the word. Each word contains 48 information bits, three tag bits, and one parity bit. (See figure 2-1-1.) The information bits may be used to store character values, logical values, or numeric values. The tag bits are control bits which identify the type of information contained in the information field. The tag bits are inaccessible to normal state (user) programs. The parity bit is used to check for correct information transfer between the CPM or IOM and main memory.

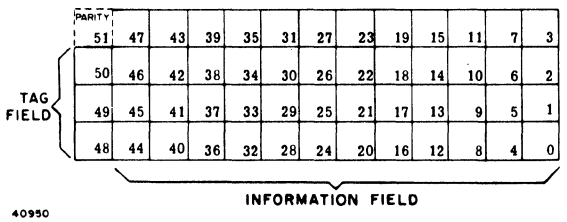


Figure 2-1-1. Word Structure

INTERNAL CHARACTER CODES AND COLLATING SEQUENCES

Extended Binary Coded Decimal Interchange Code (EBCDIC) is the primary internal character code of the B 7800. EBCDIC is an eight-bit alphanumeric code containing four zone bits and four numeric bits. Other internal codes which may be used include the American Standard Code for Information Interchange (ASCII), and the Burroughs Common Language Code (BCL). ASCII is the primary data communication code; BCL is used to interface with peripheral units. Numeric EBCDIC and BCL codes may be packed into four-bit digits by internal commands which delete the zones and compress the numeric portion of the characters. In general, characters are collated according to their internal binary value. Character codes and collating sequences are provided in the appendices.

NUMBERS AND NUMBERING SYSTEMS

The B 7800 is a digital computer; that is, values are stored internally in binary digits (bits). Data displayed in registers and printed forms may be in octal or hexadecimal format. Generally, we think in terms of, and manually perform arithmetic with, decimal numbers. Thus, an understanding of all of these numbering systems is desirable.

The decimal system is based on the ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, and upon the powers of ten. The binary system is based upon the two digits 0 and 1, and the powers of two. Two raised to the third power (2^3) is 8, the base of the octal system. Two raised to the fourth power (2^4) is 16, the base of the hexadecimal system. The set of digits for each number system is shown in figure 2-1-2.

The digits 0 through 9 and the alphabetic characters A through F comprise the 16-character requirement for the hexadecimal numbering system. The letter A is assigned a value of 10. B equals 11, etc., to F, which equals 15.

DECIMAL	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
BINARY	0 1
OCTAL	0 1 2 3 4 5 6 7
DECIMAL	0 1 2 3 4 5 6 7 8 9
HEXADECIMAL	0 1 2 3 4 5 6 7 8 9 A B C D E F

40951

Figure 2-1-2. Number Base Graphic Characters

Binary Notation

Because a binary digit may have only one of two values, it can be represented by a flip-flop or a bit. A number in internal binary representation is then a series of bits which are either on or off. When a bit is on (1), its position determines the value. Consider an example of five bits.

The least significant bit, if on (1), has a value of 2^0, or 1; the next most significant bit to

$2^0 = 1$

0 = off bit

1 = on bit

value of position = $2^4\ 2^3\ 2^2\ 2^1\ 2^0$

...0 0 0 0 1 = 0 +0 +0 +0 + 1 = decimal 1

...0 0 0 1 0 = 0 +0 +0 +2 + 0 = decimal 2

...0 0 0 1 1 = 0 +0 +0 +2 + 1 = decimal 3

.

.

.

...1 1 1 1 1 = $2^4+2^3+2^2+2^1+1 = 1 = 16 + 8 + 4 + 2 + 1$ = decimal 31

40952

Figure 2-1-3. Binary Integers

the left of the binary point has the value of 2^1, or 2; the third bit (count from right to left) has the value of 2^2, or 4; etc. In this way, any integer can be represented in binary form. Figure 2-1-3 illustrates some integers. Fractions in binary are much the same as integers. Here, though, the powers are negative powers with the first power to the right of the binary point having the value of 2^{-1}, or 1/2; the second bit has the value of 2^{-2}, or 1/4; the third bit 2^{-3}, or 1/8; the fourth bit, 2^{-4}, or 1/16; etc. It is apparent that while some fractions are represented correctly, others can only be approximated. However, the degree of error is very small when a sufficient number of bits are used.

Hexadecimal and Octal Notation

Since binary words are cumbersome to display, the more efficient methods of hexadecimal and octal notation are used. The hexadecimal representation of a binary word is obtained by dividing the bits into groups of four with each group assigned a successive power of 16. A binary-to-octal conversion is obtained by dividing the bits into groups of three and assigning successive powers of 8 to each group (figure 2-1-4).

The relationship between octal, decimal and hexadecimal is shown in figure 2-1-5 using the decimal number 1013_{10} (equivalent to 1765_8 and $3F5_{16}$ where the subscript 8, 10, of 16 indicates the base).

$1765_8 = 1 \times 8^3 + 7 \times 8^2 + 6 \times 8^1 + 5 \times 8^0 =$

$1 \times 512 + 7 \times 64 + 6 \times 8 + 5 \times 1 =$

$512 + 448 + 48 + 5 = 1013_{10}$

$1013_{10} = 1 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 3 \times 1^0 =$

$1 \times 1000 + 0 \times 100 + 1 \times 10 + 3 \times 1 =$

$1000 + 0 + 10 + 3 = 1013_{10}$

$3F5_{16} = 0 \times 16^3 + 3 \times 16^2 + F \times 16^1 + 5 \times 16^0 =$

$0 \times 4096 + 3 \times 256 + F \times 16 + 5 \times 1 =$

$0 + 768 + 240 + 5 = 1013_{10}$

40954

Figure 2-1-5. Relationship ot Octal, Decimal, and Hexadecimal Numbers

NUMBER CONVERSION

Binary to Decimal Conversion

Integral

This conversion is effected by adding together the value of each bit that is on. In this way, the binary number 11010011 would be equal to:

$1x2^7+1x2^6+0x2^5+1x2^4+0x2^3+0x2^2+1x2^1+1x2^0=$

$1x2^7+1x2^6+0 \quad +1x2^4+0^{+0}+1x2^1+1x2^0=$

$128 + 64 + 16 + 2 + 1 \quad 211_{10}$

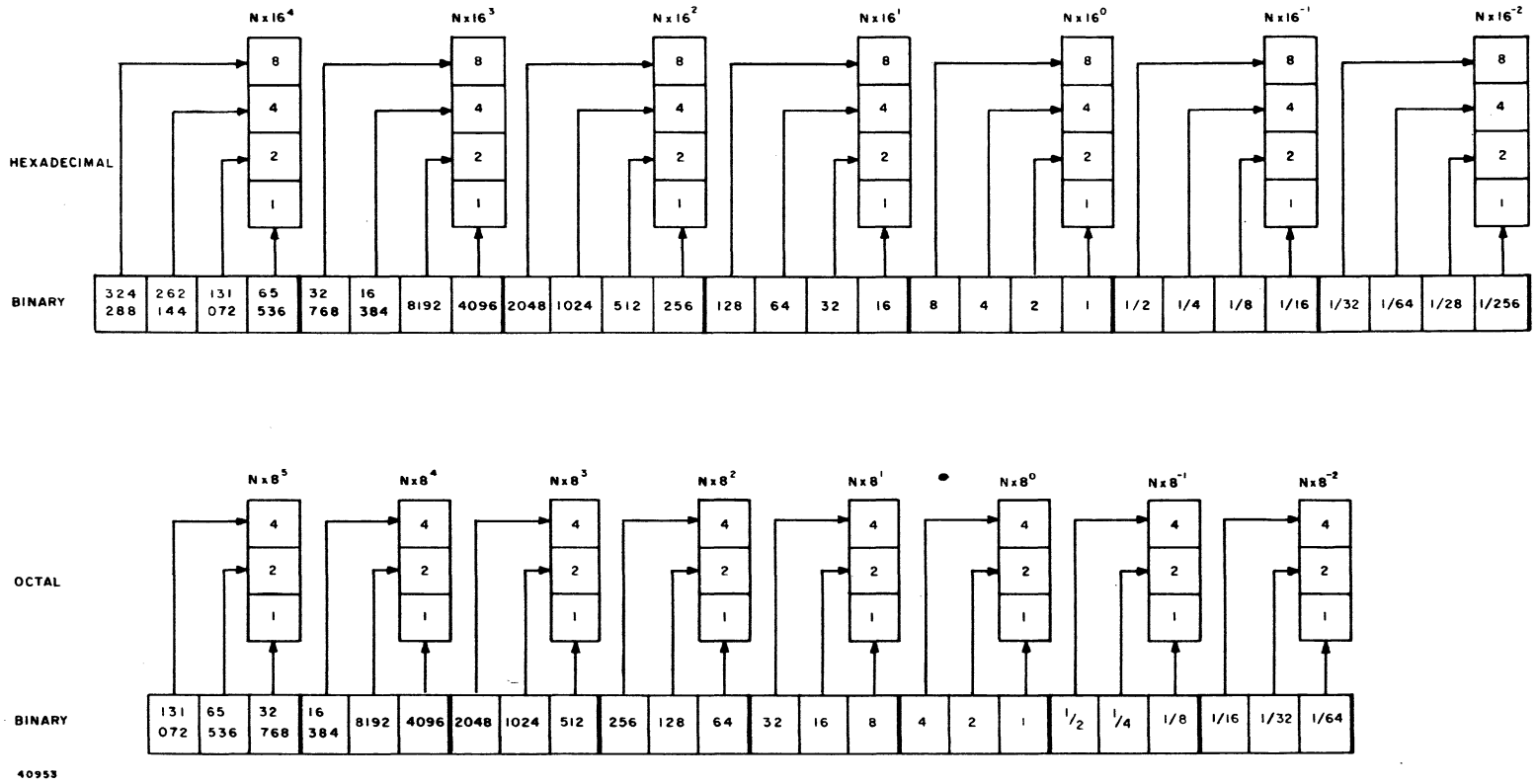


Figure 2-1-4. Binary to Hexadecimal and Octal Conversion

A second method of effecting a binary-to-decimal conversion is the "double dabble" method. In this procedure, the high-order bit is doubled (multiplied by 2) and then added to the next lower-order bit. This sum is then doubled and again added to the next lower bit. This process is continued until the entire binary number has been expended (figure II-1-6A). The correct result is obtained after the low-order bit (units) has been added.

Fractional

The above process will work for integral numbers and for the integral part of fractional numbers, but it will not work for the fractional part of fractional numbers. To convert binary fractions to decimal fractions, division is used. As was previously stated, the bits to the right of the binary point have the decreasing values of 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, etc., or, as fractions 1/2, 1/4, 1/8, 1/16, etc., respectively.

To find the decimal equivalent of a binary fraction, the lowest order significant bit is taken as the integer 1 and divided by 2. The next higher-order bit is then added into the units position of the resulting quotient, and the division is repeated. This is repeated until the binary point is reached. The result is complete when the bit to the immediate right of the binary point has been added into the units position and the result divided by 2. This process is shown in figure 2-1-6B.

Decimal To Binary Conversion

Integral

Decimal to binary conversion may be effected in several ways. If the powers of 2 are known, then the binary equivalent can be found by subtracting from the number the largest power of 2, which is smaller than the decimal number, and then recording a bit for that power of two. The largest power of 2, which is smaller than the result of the preceding subtraction, is then found, subtracted, and the corresponding binary bit recorded. In effect, this is the reverse of the first method of converting from binary to decimal.

A second method of conversion is done by successive division. The decimal number to be converted is divided by 2 and the quotient and remainder are noted. The remainder will always be either 0 or 1. Then the quotient is divided by 2, resulting in another quotient and remainder. This is repeated until the quotient is 0. The remainder, resulting from the first division, is the low order bit; the last remainder is the high order bit. This process is valid for the integral part of a number (figure 2-1-7A).

Fractional

The fractional part of a number may be converted in a method similar to the preceding method of division. The fraction is multiplied

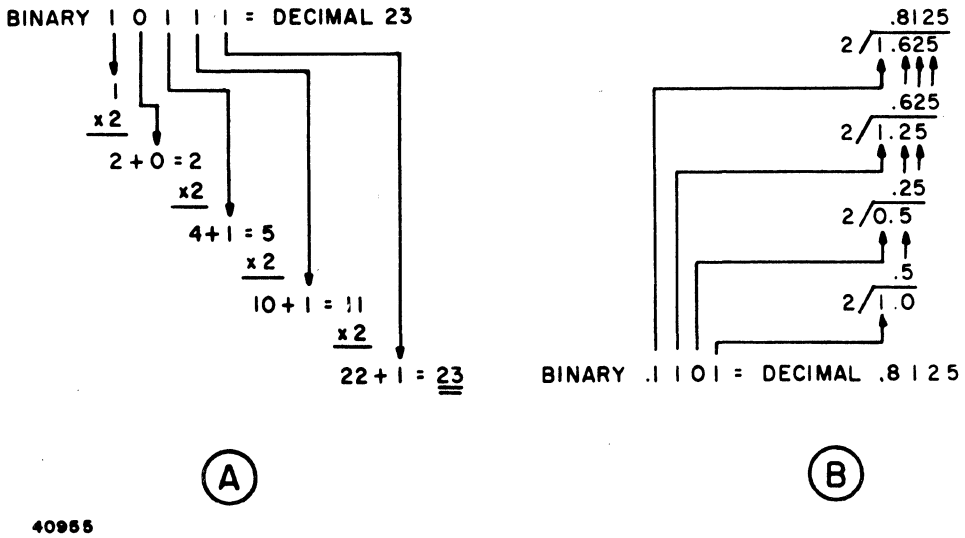


Figure 2-1-6. Binary to Decimal Conversion

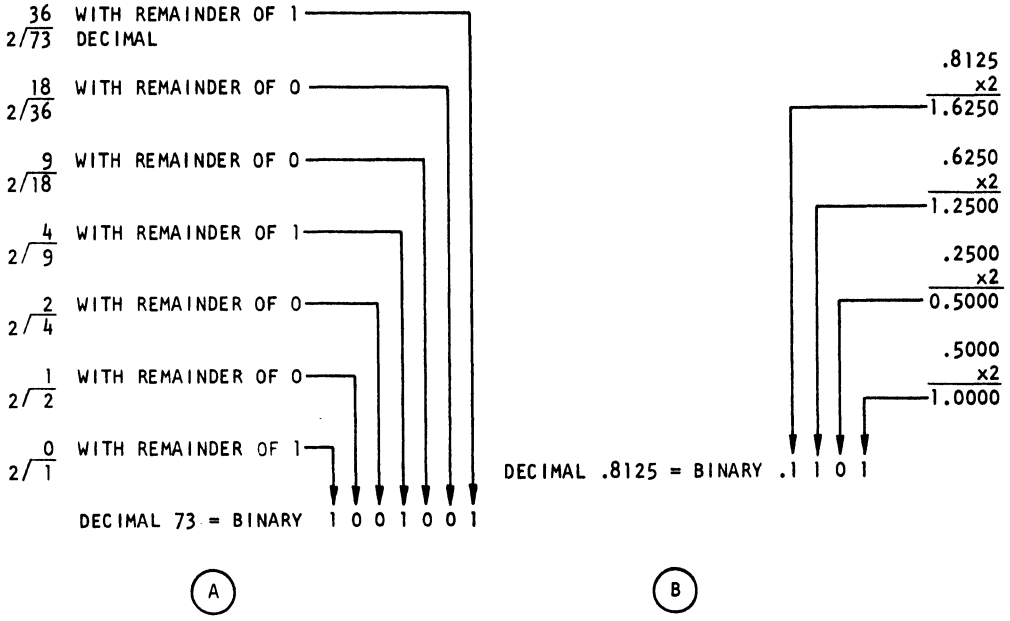


Figure 2-1-7. Decimal to Binary Conversion

by 2 and, if the result is greater than 1, the 1 is recorded in the binary string as a 1 bit. If the product remains less than 1, the binary bit is 0. The fractional part of the product is carried down and again multiplied by 2. This is repeated until the fractional part is equal to 0, or the required degree of accuracy is attained. This process is shown in figure 2-1-7B.

Decimal To Octal Conversion

Integral

To convert a decimal number to its octal form, the powers of eight may be used. Another method is to divide the number by eight. The remainder is the low-order octal digit. The quotient is then again divided by eight, and the remainder resulting is the next higher-order octal digit. This process is repeated until the quotient is zero. This method is used for the integral part of numbers (figure 2-1-8A).

Fractional

When a fractional part of the number is to be converted, multiplication is used. Here, the fraction is multiplied by eight and the integral portion formed is the first octal digit to the right of the octal point. This process is repeated until either the fraction is zero, or the desired degree of accuracy is attained. This conversion is shown in figure 2-1-8B.

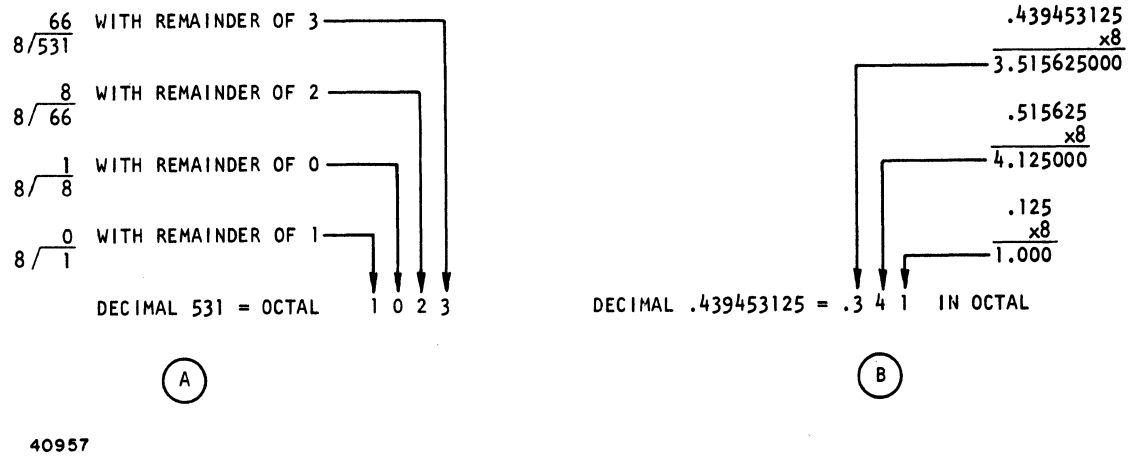


Figure 2-1-8. Decimal to Octal Conversion

8^n	n	8^{-n}
1	0	1.0
8	1	0.125
64	2	0.015625
512	3	0.001953125
4096	4	0.000244140625
32768	5	0.000030517578125
262144	6	0.00000381469726562 5
2097152	7	0.000000476837158203125
16777216	8	0.000000059604644775390625
134217728	9	0.000000007450580596923828125
1073741824	10	0.000000000931322574615478515625
8589934592	11	0.000000000116415321826934814453125
68719476736	12	0.000000000014551915228366851806640625
549755813888	13	0.000000000001818989493545856475830078125

40958

Figure 2-1-9. Powers of 8

Octal To Decimal Conversion

Octade

In octal to decimal or decimal to octal conversions, if the powers of 8 are known, then the procedure is much the same as the corresponding subtraction method of binary. The difference is the digital multiplier which will have a value of from 0 through 7 in octal. Each octal digit will be referred to as an octade. The values of the octades are shown in figure 2-1-9.

Integral

On the conversion from octal to decimal, a method very similar to "double dabble" may be used. Here, the higher-order octade is multiplied by 8 and then added to the next lower octade. This sum is then multiplied by 8 and again added to the next lower octade. This is continued until the first octade to the left of the octal point is reached. After the units octade has been added, the result should be complete (figure 2-1-10A).

Fractional

The above method is valid for the integral part of a number, but for the fractional part of a number, the following must be used. The lowest order octade is considered to be an integer. As such, it is divided by 8. The next higher octade is then added to this quotient in the

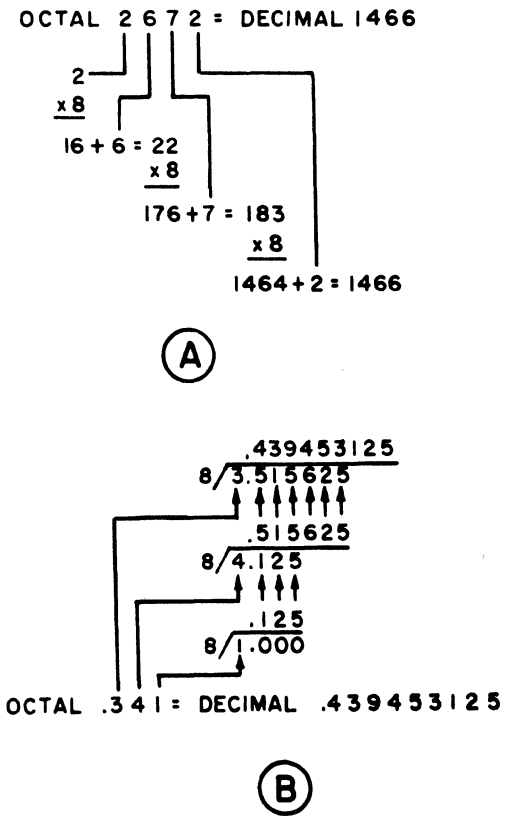


Figure 2-1-10. Octal to Decimal Conversion

units position and the sum is again divided by 8. This continues until the first octade to the right of the octal point has been added and the result divided by 8. (See figure 2-1-10B.)

Decimal To Hexadecimal Conversion

To convert an integral or a fractional decimal number to its hexadecimal form, the powers of 16 may be used. Methods similar to those used for conversion to octal representation may also be used, with the multiplication or division being by 16 rather than eight; however, such methods are very cumbersome. The simplest method is to convert the decimal number to a binary number as described earlier, and then convert the binary number to its hexadecimal representation (each four binary digits are used to form one hexadecimal digit).

Hexadecimal To Decimal Conversion

The simplest method for converting integral or fractional hexadecimal numbers to their decimal equivalent is to first convert the hexadecimal number to its binary equivalent (each hexadecimal digit is used to from four binary digits) and then convert the resulting binary number to its decimal representation as described earlier.

OPERAND FORMATS

Operands are the words of information that are worked with when processing. An operand may be used to store numeric values (a numeric operand), logical values (a logical operand), or character values (a string operand). Most operands are one word in length, and are identified by a tag field of zero. Double precision operands, which are used to store numbers in which many significant digits of accuracy are needed, are two words in length and are identified by a tag field of two. Thus, the tag field of an operand indicates the size of the operand (one or two words).

Numeric Operands

Numeric operands are used to store numeric values (numbers) in floating point format. A numeric operand may be single or double precision.

When the tag bits of a memory word (bits 50, 49, 48) are 0 (000), they denote a single-precision operand. When the tag bits are 2 (010), i.e., bit 49 set, they denote a double precision operand.

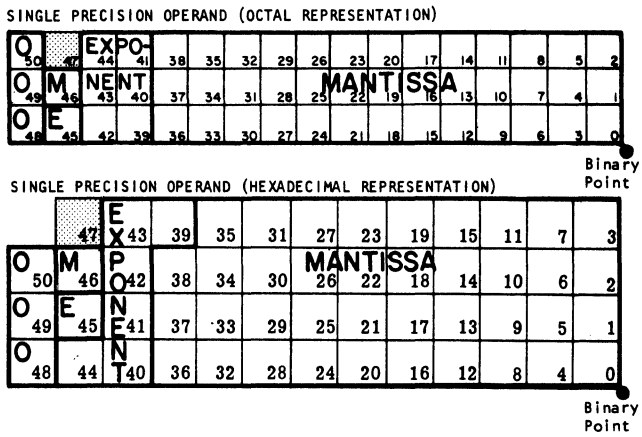
Single Precision Operands

All numeric operands are expressed in floating point form, where each numeric operand has both a mantissa and an exponent. This form may be related to power of ten notation where 13297. is the mantissa and –3, the exponent in a representation of the number 13.297 ($13297. \times 10^{-3}$). The mantissa of a single precision operand is comprised of 39 bits which make up 13 octades. The mantissa of a single precision numeric operand is considered to be an integer and is treated as such; i.e., the binary point is considered to be to the right of the least significant octade. The exponent of the number is represented by 6 bits (bits 44 through 39) which form two octades. Bit number 45 is the sign of the exponent. When 45 is off, the exponent is positive; when on, negative. Bit 46 is the sign of the mantissa, which is the overall sign of the operand.

The structure of a single precision operand is shown in figure 2-1-11. Because the exponent is an octal scale factor, the single precision operand is shown in both hexadecimal and octal representation.

Exponent Field

The exponent is a binary number which, with its sign, is an octal scale factor for the mantissa. That is, the binary point in the mantissa must be shifted left three binary places (the mantissa must be shifted right three binary places) for each increase by one in the value of the exponent. The exponent is used for automatic scaling of operands when arithmetic, comparison and integer operations are being performed. The range of the exponent is from +63 to –63 for single-precision operands.



TAG	50:3	000
	47:1	Not used
M	46:1	Sign of Mantissa. 1 = Negative, 0 = Positive.
E	45:1	Sign of exponent. 1 = Negative, 0 = Positive.
EXPONENT	44:6	Exponent.
MANTISSA	38:39	Mantissa.

Figure 2-1-11. Single Precision Operand

Mantissa Field

The mantissa is the significant part of the operand. The magnitude of the operand is obtained by multiplying the value contained in the mantissa by eight raised to the value of the exponent sign and exponent as follows:

$$V = \pm M \times 8^{\pm E}$$

where:

V = Value of number

± M = Mantissa with sign

± E = Exponent with sign

The order of number magnitude in the 39 bit mantissa, as decimal numbers and powers of base 16, 8, and 2 is shown in figure 2-1-12.

Double Precision Operands

Double precision operands are identified by a tag field of two, indicating that the operand is one of a pair of two words (figure 2-1-13).

The first word of the double precision operand is identical to the single precision operand.

The integral part of the mantissa is contained in the mantissa field of the first word. The fractional part of the mantissa is contained in the mantissa extension field of the second word.

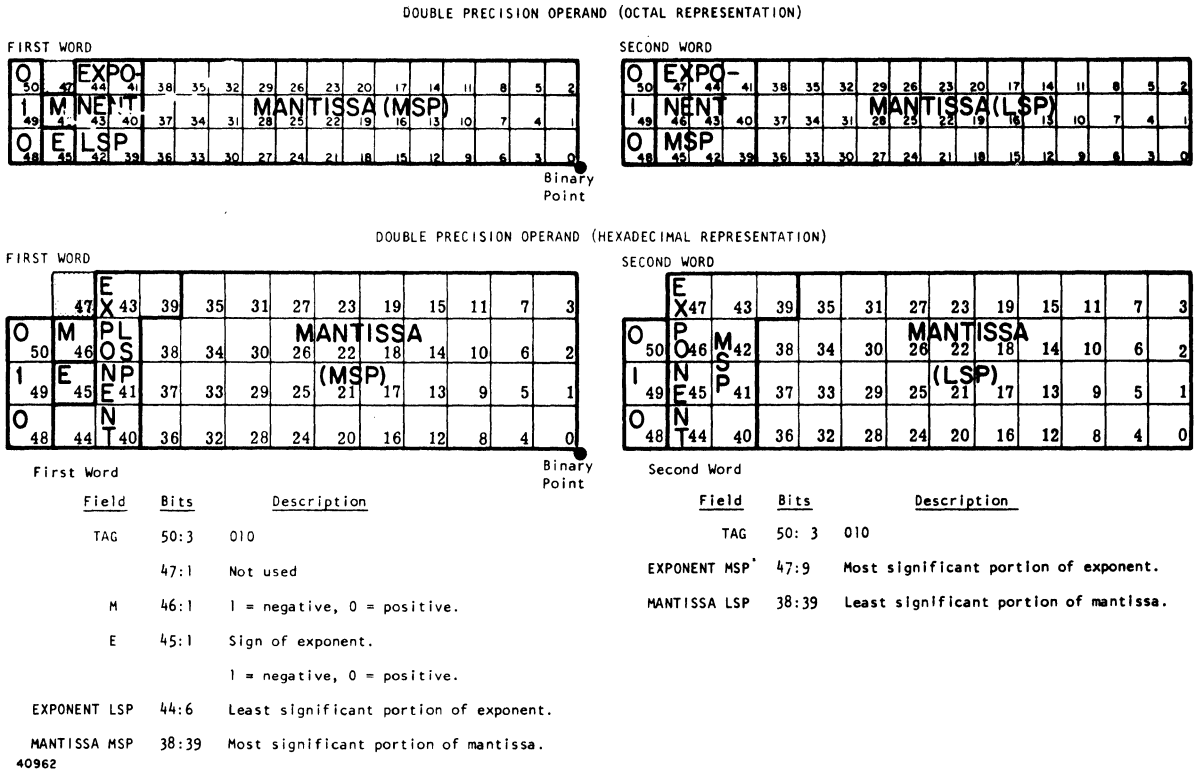
The 15-bit exponent of a double precision operand is formed by the concatenation of the

REGISTER BIT SET	DECIMAL	DECIMAL RECIPROCAL	HEX.	OCTAL	BINARY
0	1	1.0	16^0	8^0	2^0
1	2	0.5			
2	4	0.25			
3	8	0.125		8^1	2^3
4	16	0.0625	16^1		
5	32	0.03125			
6	64	0.015625		8^2	2^6
7	128	0.0078125			
8	256	0.00390625	16^2		
9	512	0.001953125		8^3	2^9
10	1024	0.0009765625			
11	2048	0.00048828125			
12	4096	0.000244140625	16^3	8^4	2^{12}
13	8192	0.0001220703125			
14	16384	0.00006103515625			
15	32768	0.000030517578125		8^5	2^{15}
16	65536	0.000015258789625	16^4		
17	131072	0.00000762939453125			
18	262144	0.000003814697265625		8^6	2^{18}
19	524288	0.0000019073486328125			
20	1048576	0.00000095367431640625	16^5		
21	2097152	0.000000476837158203125		8^7	2^{21}
22	4194304	0.0000002384185791015625			
23	8388608	0.00000011920928955078125			
24	16777216	0.000000059604644775390625	16^6	8^8	2^{24}
25	33554432	0.000000029802322387695312 5			
26	67108864	0.000000014901161193847656 25			
27	134217728	0.000000007450580596923828125		8^9	2^{27}
28	268435456	0.000000003725290298461914062 5	16^7		
29	536870912	0.000000001862645149230957031 25			
30	1073741824	0.000000000931322574615478515625		8^{10}	2^{30}
31	2147483648	0.000000000465661287307739257812 5			
32	4294967296	0.000000000232830643653869628906 25	16^8		
33	8589934592	0.000000000116415321826934814453125		8^{11}	2^{33}
34	17179869184	0.000000000058207660913467407226562 5			
35	34359738368	0.000000000029103830456733703613281 25			
36	68719476736	0.000000000014551915228366851806640625	16^9	8^{12}	2^{36}
37	137438953472	0.000000000007275957614183425903320312 5			
38	274877906944	0.000000000003637978807091712951660156 25			
*	549755813887				
39	549755813888	0.000000000001818989403545856475830078125		8^{13}	2^{39}

* FIRST 39 BITS SET. (MAXIMUM INTEGER VALUE ALLOWED).

40961

Figure 2-1-12. Order of Magnitude Chart



First Word

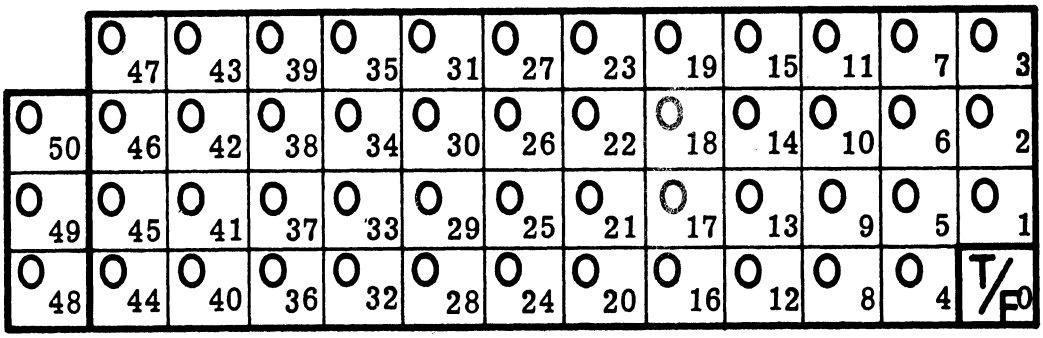
Field	Bits	Description
TAG	50:3	010
	47:1	Not used
M	46:1	1 = negative, 0 = positive.
E	45:1	Sign of exponent.
		1 = negative, 0 = positive.
EXPONENT LSP	44:6	Least significant portion of exponent.
MANTISSA MSP	38:39	Most significant portion of mantissa.

Second Word

Field	Bits	Description
TAG	50: 3	010
EXPONENT MSP	47:9	Most significant portion of exponent.
MANTISSA LSP	38:39	Least significant portion of mantissa.

40962

Figure 2-1-13. Double-Precision Operand



Field	Bits	Description
TAG	50:3	000
	47:47	All zeroes.
T/F	0:1	True/false bit.
		1 = True, 0 = False

40963

Figure 2-1-14. Logical Operand

exponent extension with the exponent. The exponent extension is more significant than the exponent.

Number Ranges and Normalization

To add and subtract two numeric operands on the B 7800, the exponents of the two operands must be equal. The B 7800 equalizes the exponents of the two operands automatically; this equalization may require that one of the operands be "normalized." Normalization occurs if the exponent difference of the two operands is greater than the number of leading zero (octal) digits in the mantissa of the operand with the larger exponent. In such cases, the larger operand is normalized, and the mantissa of the smaller operand is then shifted right until the exponents are equal.

A normalized number is a number which has the smallest exponent with which the number can be expressed without losing the most significant digit of the number. A number is normalized by shifting the mantissa to the left, (moving the binary point right) in three-bit increments until the number of leading zeroes in the mantissa is less than three. For each three-bit shift to the left (of the mantissa), the exponent is decreased by one.

Because of automatic normalization by the CPM, the range of numbers which are useable on the B 7800 includes both normalized and unnormalized numbers. In general, normalized numbers are those which the system may use for arithmetic, and unnormalized numbers are those which the system may store.

The largest and smallest numbers representable as normalized and unnormalized operands are:

The largest single precision integer	549755813887_{10}	
or	$8^{13}-1$	decimal
	0007777777777777	octal
The largest single precision number	$4.31359146673 \times 10^{68}$	
or	$(8^{13}-1) \times 8^{63}$	decimal
	0777777777777777	octal
The largest double precision integer	302231454903657293676543	
or	$8^{26}-1$	decimal
(first word)	0157777777777777	octal
(second word)	0007777777777777	
The largest double precision number	$1.9488283820502807911244 69 \times 10^{29603}$	
or	$(1-8^{-26}) \times 8^{32780}$	decimal
(first word)	0777777777777777	octal
(second word)	7777777777777777	
The smallest positive unnormalized single precision number	$1.27447352891 \times 10^{-57}$	
or	8^{-63}	decimal
	1770000000000001	octal
The smallest positive normalized single precision number	$8.7581154020 \times 10^{-47}$	
or	8^{-51}	decimal
	1771000000000000	octal
The smallest positive normalized double precision number	$1.9385458571375858335564 0 \times 10^{-29581}$	
or		decimal
(first word)	1771000000000000	octal
(second word)	7770000000000000	

The number sets are symmetrical with respect to zero. The negative number corresponding to any valid positive number may also be expressed. From the ranges above, one can see that a single precision integer must always have an exponent of zero.

Logical Operands

Logical operands (figure 2-1-14) have one of two values: true (on) or false (off). Logical values are the result of Boolean operations or relational operations. Relational operators generate a logical value as the result of an algebraic comparison of two arithmetic expressions. Bit 0 contains the logical value. Relational operators set bit 0, where conditional operators use bit 0 for the decision.

> **NOTE**
> Logical operators (LAND, LOR, LNOT, and LEQV) cause a logical operation to be performed on each bit of the two operands and the results of these operations (48 single precision values or 96 double precision values) are left in the top-of-stack operand. Logical operators may operate on logical, string, or numeric operands.

String Operands

A string operand is a single word operand (identified by a tag of zero) which is used to store characters. Character representation may be 8-bit (EBCDIC), 7-bit (USASCII), 6-bit (BCL), or 4-bit (packed BCD) characters. Generally, a string of characters is stored in one or more string operands in memory as an array or table. Such arrays or tables are addressed by means of string descriptors. The format of string operands for storage of 8-bit, 7-bit, 6-bit, and 4-bit characters is shown in figure 2-1-15.

String operands may also be used to store signed numeric characters in 8-bit, 6-bit, and 4-bit formats. Each string operand can store one signed numeric number consisting of six 8-bit characters, eight 6-bit characters, or 11 4-bit characters. Eight-bit and 6-bit characters are divided into a zone portion and a number portion. The number portion consists of the four least significant bits of each character; the remaining bits form the zone. When 8-bit or 6-bit signed numeric characters are stored in a string operand, the sign of the characters is stored in the zone bits of the least significant character. When 4-bit signed numeric characters are stored in a string operand, the sign of the characters is stored as the most significant character of the operand. Table 2-1-1 shows the bit configurations for negative and positive signs in 8-bit, 6-bit, and 4-bit formats. Figure 2-1-16 illustrates the manner in which a signed number (–4259) is stored in 8-bit, 6-bit, and 4-bit code.

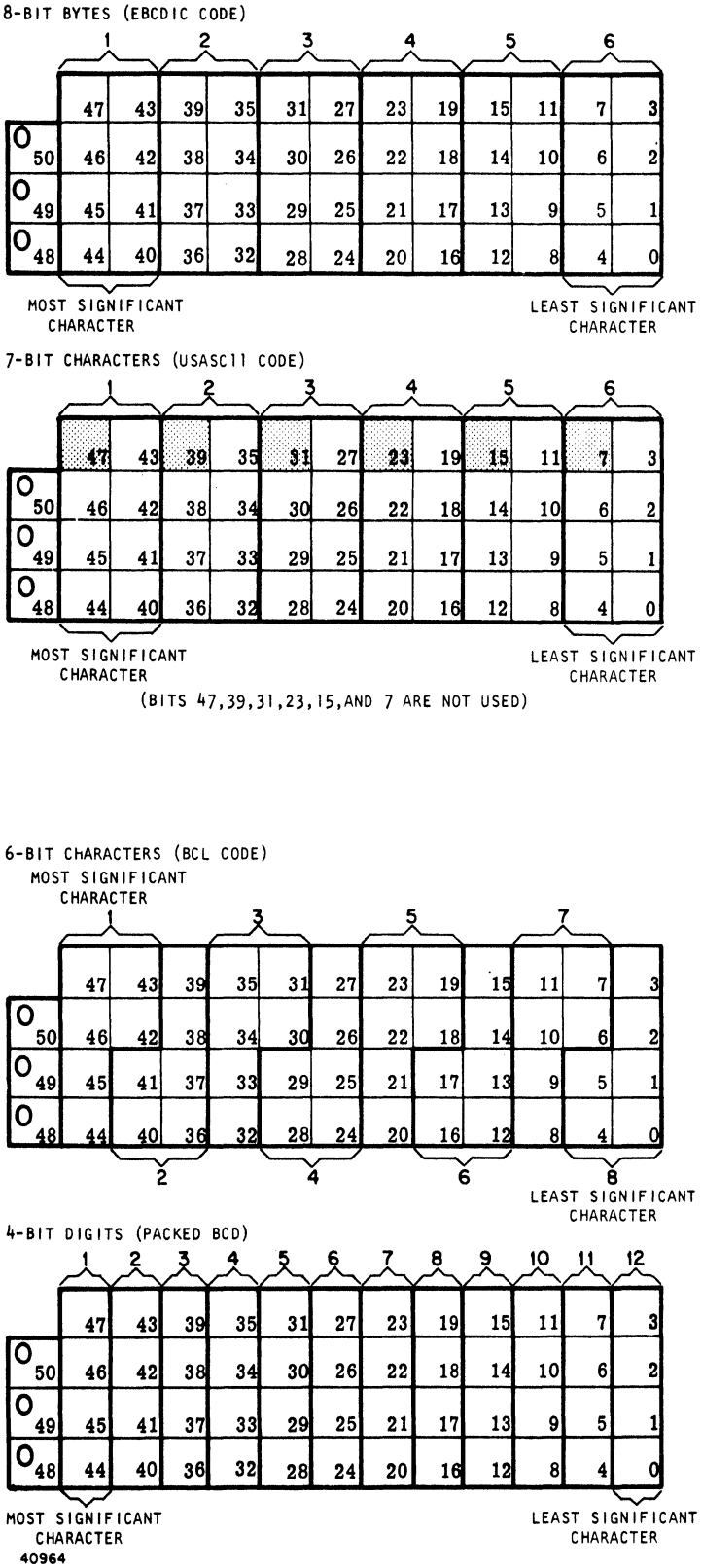


Figure 2-1-15. String Operands

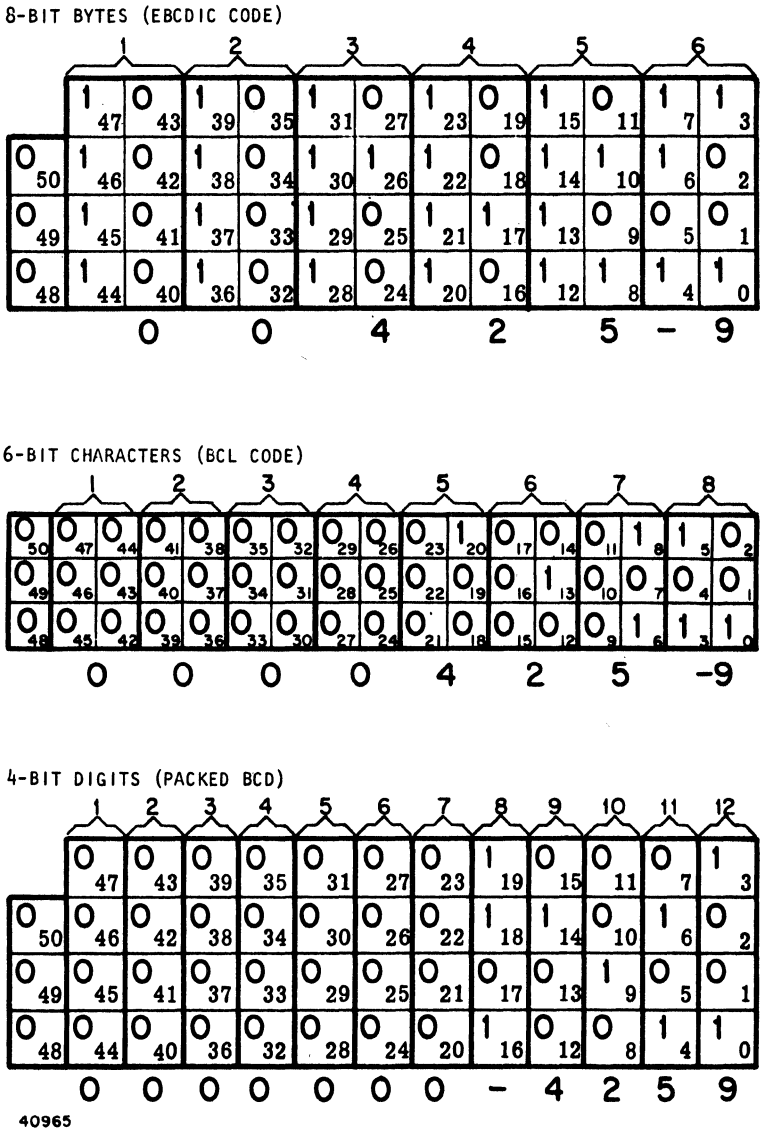


Figure 2-1-16. Use of String Operand to Store Signed Number (-4259)

Table 2-1-1. Sign Configurations of String Operands

Size	Sign Location	Negative	Positive
8-bit	Zone, least significant character	1101	Any bit configuration other than the negative bit configurations
6-bit	Zone, least significant character	10	Any bit configuration other than the negative bit configurations
4-bit	Most significant digit	1101	Any bit configuration other than the negative bit configurations

SECTION 2

POLISH NOTATION AND STACK

GENERAL

To facilitate the understanding of the B 7800 stack concept, a method of mathematical notation known as Polish notation must be understood. A problem that exists with most forms of mathematical notation is clarifying the boundaries of specific terms. This has been eliminated with the use of parentheses, brackets, and braces. However, with a complex equation, it becomes necessary to duplicate the use of the few types of delimiters that exist. It might be noted that it is common to encounter mathematical equations such as Y = 5Z + 7/2Z and Y = (5Z + 7)2Z. Two equations express different functions of Z, but one could easily be used when the other was intended. From this it can be seen that an error in notation can change the whole problem, because the parentheses have definite meaning.

Polish notation is an arithmetical or logical notational system using only operands and operators arranged in a sequence or string which eliminates the necessity of factor boundaries. The B 7800 compilers translate source statements to Polish strings, and convert these Polish strings to a series of machine instructions (program operators).

POLISH NOTATION

The essential difference between Polish notation and conventional notation is that operators are written to the right of operands instead of between them. For example, the conventional B + C would be written B C + in Polish notation. Looking at the example, A = 7 (B + C), it would be written in Polish notation as follows:

A7BC+*=

Any expression written in Polish notation is called a Polish string. In order to fully understand this concept, the rule for evaluating a Polish string should be known.

General Rules For Generation of Polish String

Figure 2-2-1 is a flow chart for generation of a Polish string. In general, the rules for generation of a Polish string may be stated as follows. If the source of expression is:

Name	Action
A Variable	Place variable in string being built and examine next symbol.
An Operator -Separator	Place in delimiter list and examine next symbol.
-Arithmetic or Boolean operator and last entered delimiter list symbol was: a. an operator of lower priority. b. a left bracket " [" or paren "(". c. a separator. d. nothing (delimiter list empty).	Place operator in the delimiter list and examine next source symbol.
-An Arithmetic or Boolean operator and last entered delimiter list symbol was: an operator of priority equal to or greater than the symbol in the source.	Remove the operator from the delimiter list and place in the string being built. Then compare the next symbol in the delimiter list against the source expression symbol.
-A right bracket "] " or parenthesis ")"	Pull out from delimiter list or until corresponding left bracket or parenthesis.

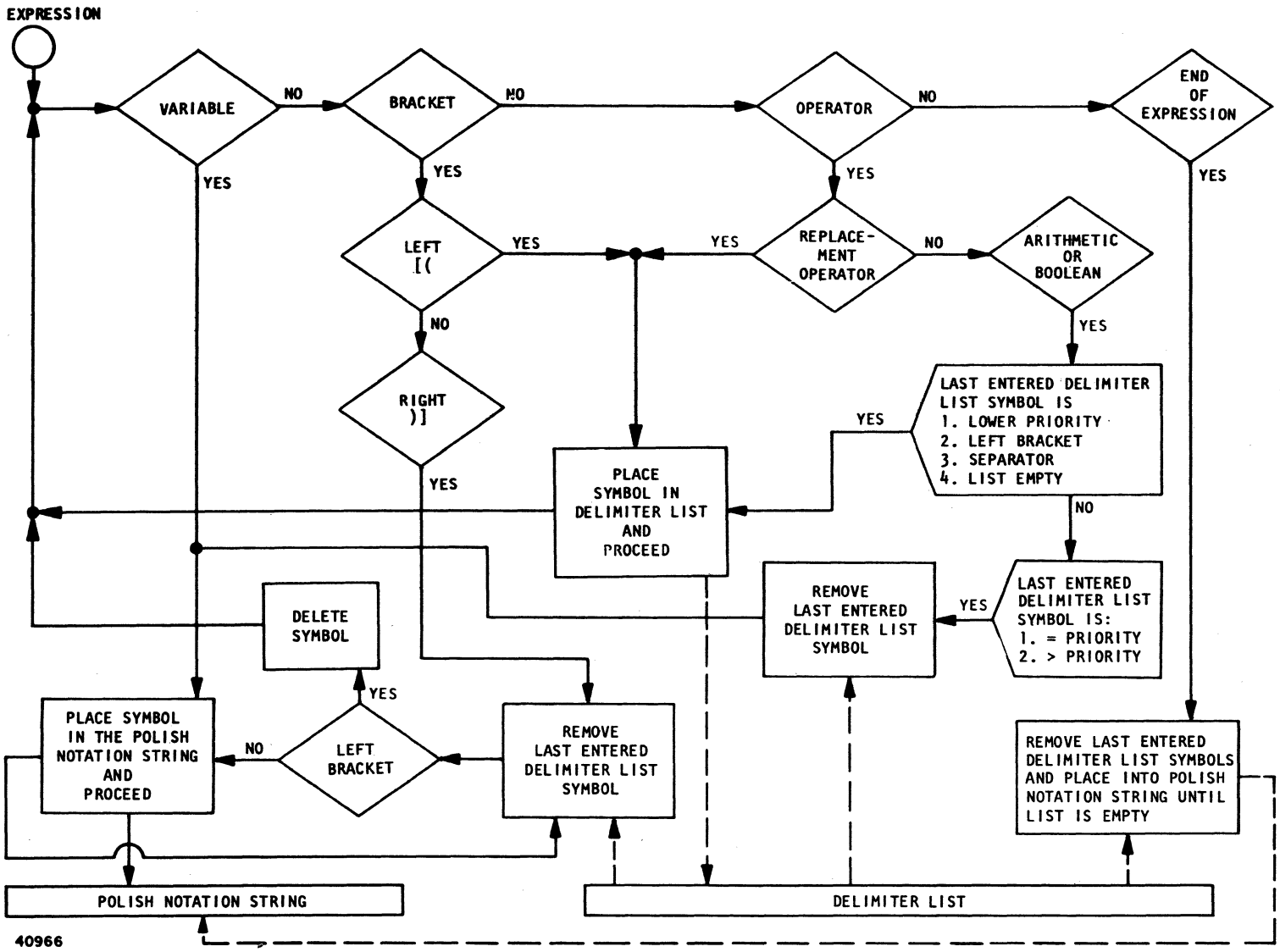


Figure 2-2-1. Polish Notation Flow Chart

Evaluating Polish String

The following procedure may be used to evaluate a Polish string.

a. Scan the string from left to right.

b. Remember the operands and the order in which they occur.

c. When an operator is encountered do the following:

1) Take the two operands which were last encountered.
2) Operate upon them according to the type of operator encountered.
3) Eliminate these two operands from further consideration.
4) Remember the result of (2) and consider it as the last operand encountered.

Following this procedure through the Polish string A7BC+* = would evaluate to A assuming the value 7 (B + C) (figure 2-2-2).

NOTE

Because replacement operators vary depending upon the language used, ← , =, and := may be used interchangeably in discussing Polish strings.

Program Code String

When a program is compiled, the source language statements are converted into a string of machine language operators. These operators are assembled into a Polish notation string and are referred to as the program code string. Each machine instruction in the string normally consists of one to three 8-bit syllables. The instructions are packed consecutively into program words. (See figure 2-2-3.) An array of program words, which can be any length, is called a program code segment. The compiler usually divides the generated code string into two or more program segments. The number of segments depend on the structure of the source program. Program segments are normally stored on disk files. When a program is executed, program segments are made present in memory as needed. Because program segments are not modified during execution, a single copy of a program segment in memory may be used for several concurrent executions of the same program; thus, the program code string is often described as "re-entrant".

Step	Symbol Being Examined	Symbol Type	Operands Being Remembered and Their Order of Occurrence (1 or 2) Before Operation	Operation Taking Place	Results Operation
a	B	Operand			
b	C	Operand	1 B		
c	+	Add Operator	2 C 1 B	B + C	(B + C)
d	7	Operand	1 (B + C)		
e	x	Multiply Operator	2 7 1 (B + C)	7 x (B + C)	7 x (B + C)
f	A	Operand	1 7(B + C)		
g	=	Replace Operator	2A 1 7(B + C)	A ← 7(B + C)	A = 7(B + C)

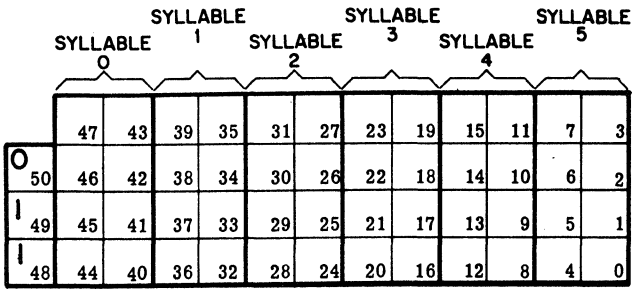
Figure 2-2-2. Evaluation of Polish String A7BC+*=

Compilation Using Polish Notation

Polish notation is used as the base for the B 7800 ALGOL compilation algorithm. An ALGOL arithmetic or Boolean expression or assignment statement may be translated to Polish notation in much the same way as the arithmetic (or algebraic) expression that already has been considered. In compiler translation, the source expression is examined one symbol at a time with a left to right scan and is combined into logical entities. As each logical entity is examined, a specific procedure is followed so that the Polish notation expression is constructed in its finalized form with one scan of the source expression.

For each program segment, there is a single segment descriptor, which defines the length and location of the program segment. The segment descriptors are stored in a special stack known as the segment dictionary.

Each job is associated with an unique job stack and with a segmented dictionary stack which may be shared by several jobs. (In addition, the MCP has its own stack and segment dictionary.) Within the job stack, a Program Control Word is provided for each point of entry into a segment of code. The PCW provides an index, not only into the segment dictionary to locate the proper segment descriptor, but also into the program segment itself to locate



Field	Bits	Description
Tag	50:3	Tag field. Value of three indicates that this word is non-modifiable (except by Overwrite operators).
	47:8	Syllable 0
	39:8	Syllable 1
	31:8	Syllable 2
	23:8	Syllable 3
	15:8	Syllable 4
	7:8	Syllable 5

Figure 2-2-3. Program Word

the proper program word and syllable. The formats of the segment descriptor and the PCW are described in detail in section 3 of this chapter.

STACK CONCEPTS

The constants and variables of a program are assigned locations within the "stack" of the program when it is compiled. The stack can be thought of as analogous to a physical stack where the last item placed on the stack is the top of the stack. When items are removed (one at a time) from the stack, the item on the top of the stack is the first item to be removed. The item at the bottom of the stack remains at the bottom of the stack until all other items have been removed from the stack. The stack not only provides an easily manageable means for keeping a dynamic history of the program as it is being processed, but also lends itself to the use of program code strings based on Polish notation.

General

A job is activated by having a processor assign to the job stack. Four top-of-stack locations are linked to the job's stack (figure 2-2-4). This linkage is established by the stack-pointer register (S), which contains the memory address of the last word placed in the stack. The four top-of-stack locations extend the stack to provide quick access for data manipulation.

Data are brought into the stack through the top-of-stack locations in such a manner that the last operand placed into the stack is the first to be extracted. The stack-pointer register (S) is incremented by 1 before a word is placed into the stack and is decremented by 1 after a word is withdrawn from the stack and placed in the top-of-stack locations. As a result, the S register continually points to the last word placed into the job's stack.

Base and Limit of Stack

A job's stack is bounded, for memory protection, by two registers: the Base-of-Stack register (BOSR) and the Limit-of-Stack register (LOSR). The contents of BOSR define the base of the stack, and the contents of LOSR define the upper limit of the stack. The job is interrupted if the S register is set to the value, contained in either LOSR or BOSR.

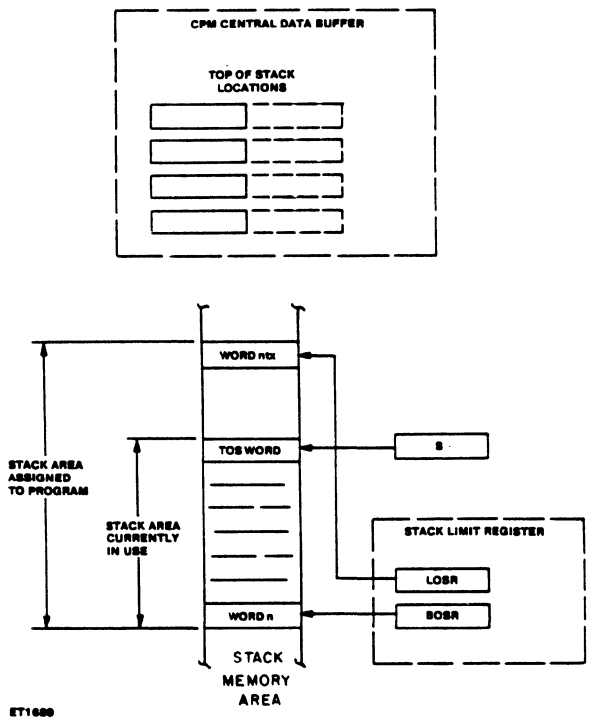


Figure 2-2-4. Top of Stack and Stack Bounds Register

Bi-directional Data Flow in the Stack

The contents of the top-of-stack locations are maintained automatically by the processor to meet the requirements of the current operator. If the current operator requires data transfer into the stack, the top-of-stack locations receive the incoming data, and the surplus contents, if any, of the top-of-stack locations, are pushed into the stack. Words are brought out of the stack into the top-of-stack locations. These words are used by operators which require the presence of data in the top-of-stack locations. These operators, however, do not explicitly move data into the stack.

Double Precision Stack Operation

Each top-of-stack location (A and B) can accommodate two memory words. For single precision operations, location A will contain one single precision operand and location B will contain the other single precision operand. However, calling a double precision operand into either top-of-stack location (A or B) will cause both halves of the double precision operand to be loaded into the A or B location. The first word is loaded into the top-of-stack location and its tag bits are checked. If the value of the tag bits indicates double precision, the second half of the operand is loaded into the second half of the top-of-stack location.

Addressing History

The B 7800 CPM provides two methods for addressing data. Direct addressing is provided by descriptors, which contain the address (core or disk) of the data. Descriptors are used to address data which are located outside of the stack area of the job. Relative addressing is provided by the Indirect Reference Word (IRW) and the Stuffed Indirect Reference Word (SIRW). The IRW and SIRW address components are both relative address components. The IRW addresses within the immediate environment of the job relative to one of 32 CPM display registers. The SIRW addresses beyond the immediate environment of the current procedure, the addressing being relative to the base of some job stack. Addressing across stacks is accomplished with an SIRW.

Direct Addressing

In general, the descriptor describes and locates data associated with a given job. String descriptors and data descriptors are used to fetch data to the stack or to store data from the stack into an array located outside the

stack area of the job. The address contained in one of these descriptors is the absolute address of an array in either system main memory or in the backup disk file, as indicated by the setting of a single bit called the presence bit. Another bit, called the double-precision bit, is used to identify the referenced data as single precision or double precision. The formats of string and data descriptors, and detailed discussions of each, are presented in section 3 of this chapter.

Relative-Addressing

Analyzing the structure of an ALGOL program results in a better understanding of the relative-addressing procedures used in the B 7800 stack. The addressing environment of an ALGOL procedure is established automatically as the program is structured by the programmer and is referred to as the lexicographical ordering of the procedural blocks. At compile time, the lexicographical ordering is used to form address couples. An address couple consists of two items:

1. The lexicographical addressing level (LL) of the variable,
2. An index value (I) used to locate the specific variable within its addressing level.

The lexicographical ordering of the program remains static as the program is executed, thereby allowing variables to be referenced via address couples as the program is executed.

The lexicographical structure of a very simple ALGOL program is illustrated in figure 2-2-5. When executed, this program would call procedure C (LL=3) from the outer block of the program (LL=2), and, in turn, procedure C would call procedure D (LL=4). The stack structure is illustrated as it would exist as procedure D was being executed. It can be seen that, as the outer block of the program was entered, and again as each procedure was entered, a Mark Stack Control Word (MSCW) was placed in the stack. The MSCW (described in detail in section 3 of this chapter) denotes the base of each lexicographical addressing level.

Display Registers

Each MSCW provides a point in the stack relative to which the variables for the associated addressing level may be referenced. The B 7800 CPM unit contains 32 display registers (D [0] through D for [31]).As shown, the base of each addressing level is addressed by

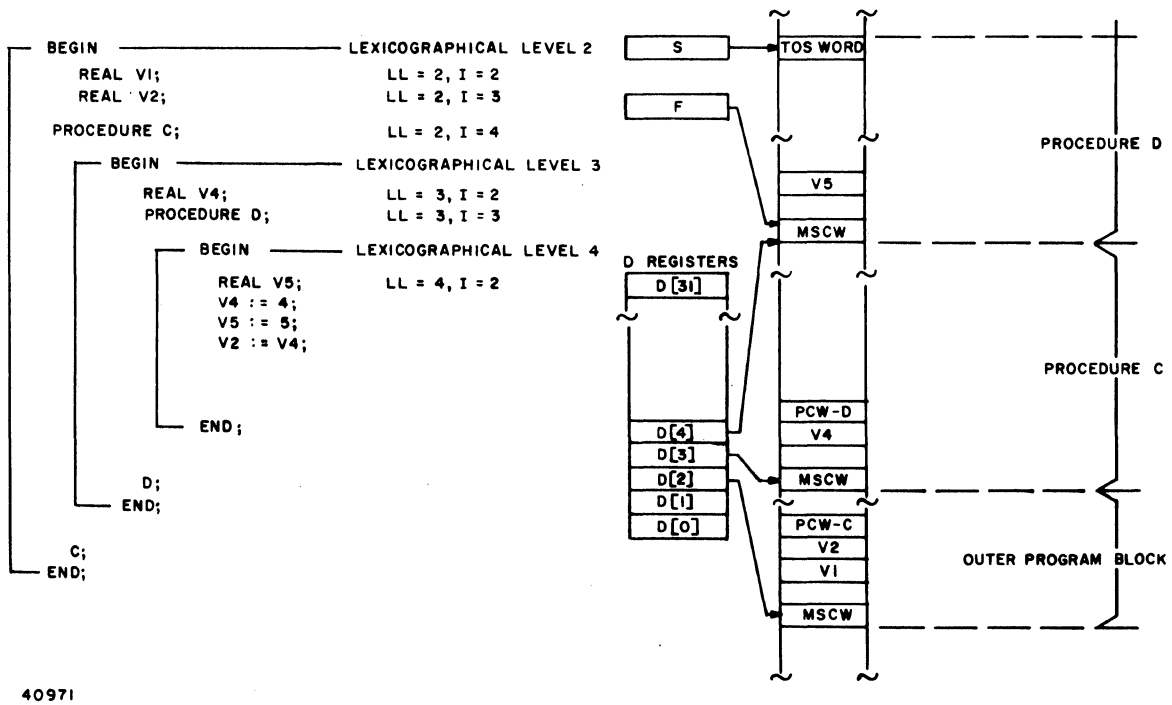


Figure 2-2-5. ALGOL Program With Lexicographical Structure and Related Stack Structure

one of these registers. The local variables of the outer block or of the procedures are addressed relative to the D registers. The D registers are updated at each procedure entry or exit.

Absolute Address Conversion

Each variable is indirectly addressed by an address couple containing a lexicographical level and an index value. The address couple is converted into an absolute memory address when the variable is referenced. The lexicographical level portion of the address couple selects the D register which contains the absolute memory address of the MSCW for the environment (lexicographical level) in which the variable is located. The index value of the address couple is added to the contents of the D register to generate the absolute memory address of the desired variable.

Addressing Environment

Thus far we have considered a very simple program in which each procedure has a different lexicographical addressing level. Generally, however, many procedures of a program may have the same lexicographical addressing level; however, no two procedures of a program may have the same addressing environment. Consider the more advanced exemplary program shown in figure 2-2-6.

This program consists of an outer block (LL=2), two procedures which have a lexicographical addressing level of three (procedures A and C), and two procedures which have a lexicographical level of four (procedures B and D). The addressing environment of the program is maintained automatically by linking the MSCWs together in accordance with the lexicographical structure of the program. This linkage is composed of the stack number (STACK NO.) and displacement (DISP) fields of the MSCW, and is inserted into the MSCW when the procedure is entered. A tree-structured addressing environment list is formed by linking the MSCW to the MSCW at the preceding lexicographical level to the procedure being entered. This tree-structured list indicates the addressing environment of the procedures.

```
BEGIN ---------- LEXICOGRAPHICAL LEVEL 2
  REAL V1;                LL = 2, I = 2
  REAL V2;                LL = 2, I = 3
  PROCEDURE A;            LL = 2, I = 4

  BEGIN ---------- LEXICOGRAPHICAL LEVEL 3
    REAL V3;              LL = 3, I = 2
    PROCEDURE B;          LL = 3, I = 3

    BEGIN ---------- LEXICOGRAPHICAL LEVEL 4
      V3 := 3;
      V1 := V3;
    END;

    B
  END;

PROCEDURE C;              LL = 2, I = 5

  BEGIN ---------- LEXICOGRAPHICAL LEVEL 3
    REAL V4;              LL = 3, I = 2
    PROCEDURE D;          LL = 3, I = 3

    BEGIN ---------- LEXICOGRAPHICAL LEVEL 4
      REAL V5;            LL = 4, I = 2
      V4 := 4;
      V5 := 5;
      A;
      V2 := V4;

    END;

    D;
  END;

  C;
END;
```

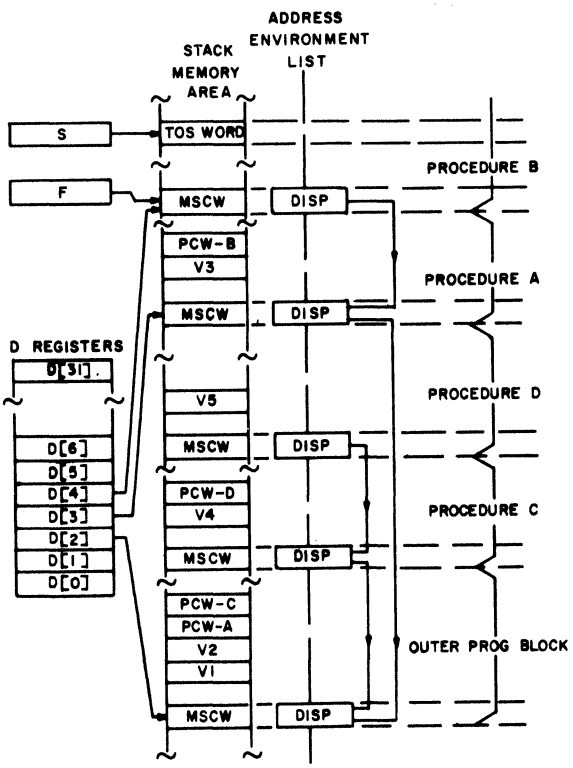


Figure 2-2-6. More Advanced ALGOL Program

Comparing the addressing tree in figure 2-2-7 with the exemplary program, one can see that when procedure B is being executed, the addressing environment includes only the variables in procedures B and A and the outer block; variables declared in procedure C and D are not addressable by procedure B. Thus, one can see that the address couples assigned to the variables in a program need not be unique. This is true because if there is no procedure which can address both of any two variables, then the two variables may have identical address couples. This addressing scheme is practical because two variables which have the same address couples will be contained within two different addressing environments.

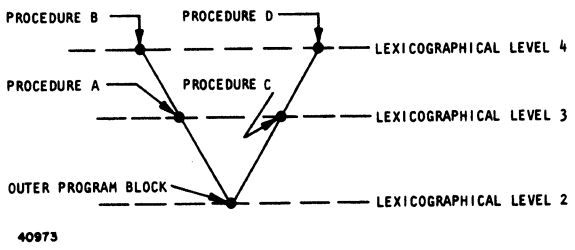


Figure 2-2-7. Addressing Environment Tree of ALGOL Program

Addressing Environment List

There is a unique set of MSCWs which the D registers must address during the execution of any particular procedure. The D registers must be changed, upon procedure entry or exit, to address the correct MSCWs. The process of changing the D registers is referred to as display update. The list of MSCWs which the D registers address is is the addressing environment list, and the areas of the stack which can be addressed relative to the settings of the D registers are the addressing environment.

Stack History

The B 7800 stack provides an easily manageable means for keeping line control information (program history) necessary for procedure entry and exit. The stack history list is a list of Mark Stack Control Words, linked together by their DF fields (figure 2-2-8).

An MSCW is inserted into the stack as a procedure is entered and is removed as that procedure is exited. Therefore, the stack history list grows and contracts with the procedural depth of the program. Mark Stack Control Words identify the portion of the stack related to each procedure. When the procedure is entered, its parameters and local variables are entered in the stack following the MSCW. When the procedure is executed its parameters and local variables are referenced by addressing relative to the MSCW.

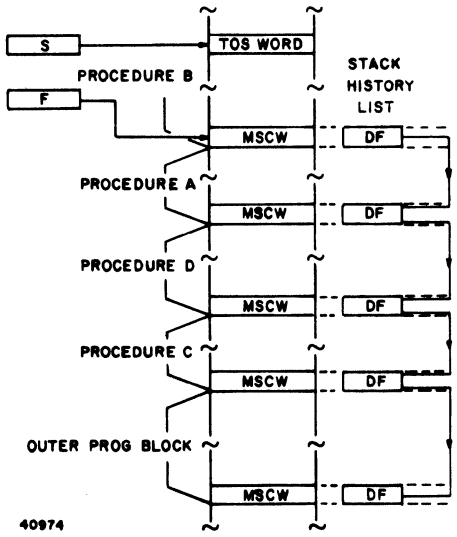


Figure 2-2-8. Stack History List

Each MSCW is linked to the prior MSCW through the contents of its DF field in order to identify the point in the stack where the prior procedure began. When a procedure is exited, its portion of the stack is discarded. This action is achieved by setting the stack-pointer register (S) to address the memory location preceding the most recent MSCW (figure 2-2-9). This topmost MSCW, addressed by another register (F), is deleted from the stack-history list by changing F to address the prior MSCW, placing this MSCW at the head of the stack history.

SIMPLE STACK OPERATION

All program information must be in the system before it can be used. Input areas are allocated for information entering the system and output areas are set aside for information exiting the system; array and table areas are also allocated to store certain types of data. Thus data is stored in several different areas: the input/output areas, data tables (arrays), and the stack. Since all word is done in the top-of-stack locations, all information or data is transferred to the top-of-stack locations and the stack itself.

At this point, an ALGOL assignment statement and the Polish notation equivalent will be related to the stack concept of operation. The example is Z:=Y + 2x(W+V), where :=

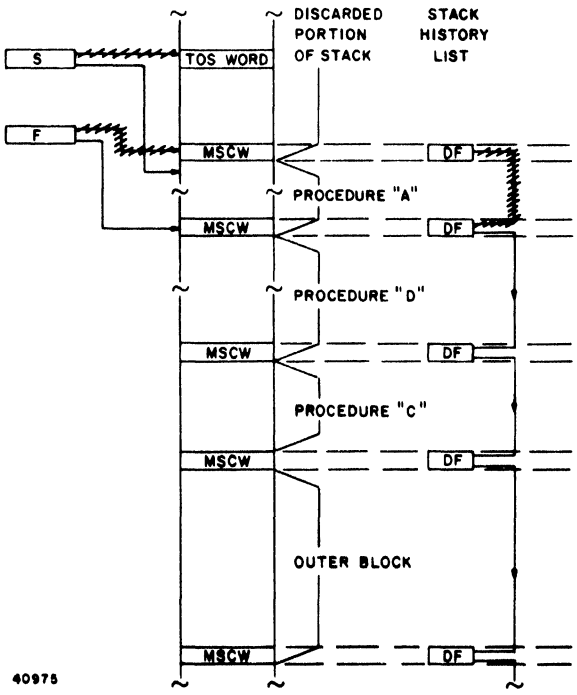


Figure 2-2-9. Stack Cut Back on Procedure Exit

means "is replaced by." In terms of a computer program, this assignment statement indicates that the value resulting from the evaluation of the arithmetic expression is to be stored in the location representing the variable Z.

When Z:=Y + 2x(W+V) is translated to Polish notation, the result is ZY2WV+ x +:=. Each element of the example expression causes a certain type of syllable to be included in the machine language program when the source problem is compiled. The following is a detailed description of each element of the example, the type of syllable compiled, and the resulting operation (see figure 2-2-10 and table 2-2-1).

In the example statement, Z is to be the recipient of a value, so the address of Z must be placed in the stack. This is accomplished by a Name Call (NAMC) syllable which places an Indirect Reference Word (IRW) in the stack. The IRW contains the address of Z in the form of an "address couple" that references the memory location reserved in the stack for the variable Z.

Since Y is to be added to a quantity, Y is brought into the top of the stack as an operand. This is accomplished with a Value Call (VALC) syllable that references Y. The value 2 is then brought to the stack, with an eight-bit literal syllable (LT8). Since W and V are to be added, the respective variables are brought to the stack with Value Call syllables. The ADD operator adds the two top operands and places the sum in the top of stack. This example assumes, for simplicity, single-precision operands not requiring use of additional top-of-stack locations which are used in double-precision operations.

The multiply operator (MULT) is the next symbol encountered in the Polish string; when executed, it places the product "2x(W+V)" in the top of the stack. The next symbol, ADD, when executed, leaves the final result "Y+2x(W+V)" in the top of the stack.

The store syllable (STOD) completes the execution of the statement Z:=Y + 2x(W+V). The store operation examines the two top-of-stack operands looking for an IRW or Data Descriptor. In this example, the IRW addresses the location where the computed value of Z is to be stored. The stack is empty at the completion of this statement.

Thus, the Polish string ZY2WV + x+:= is used to produce the following code string:

```
NAMC (Z)
VALC (Y)
LT8 2
VALC (W)
VALC (V)
ADD
MULT
ADD
STOD
```

When this code string is executed on the B 7800, the value of the expression Y+2x(W+V) is stored in the stack location reserved for the variable Z.

INTERRUPT HANDLING

In the B 7800, hardware interrupts are treated as hardware-originated procedure calls. When the hardware detects an interrupt condition, the CPM causes a MSCW to be placed in the stack, then places in the stack an IRW addressing the interrupt handling procedure, places two parameters in the stack to identify and describe the interrupt condition, and then causes the interrupt handling procedure to be entered. When the interrupt handling procedure is entered, the D registers are updated to make all legitimate variables addressable. Similarly, upon return from the interrupt handling procedure, the D registers are again updated to make all of the variables of the former procedure addressable again. A detailed description of interrupt handling is provided in chapter 3.

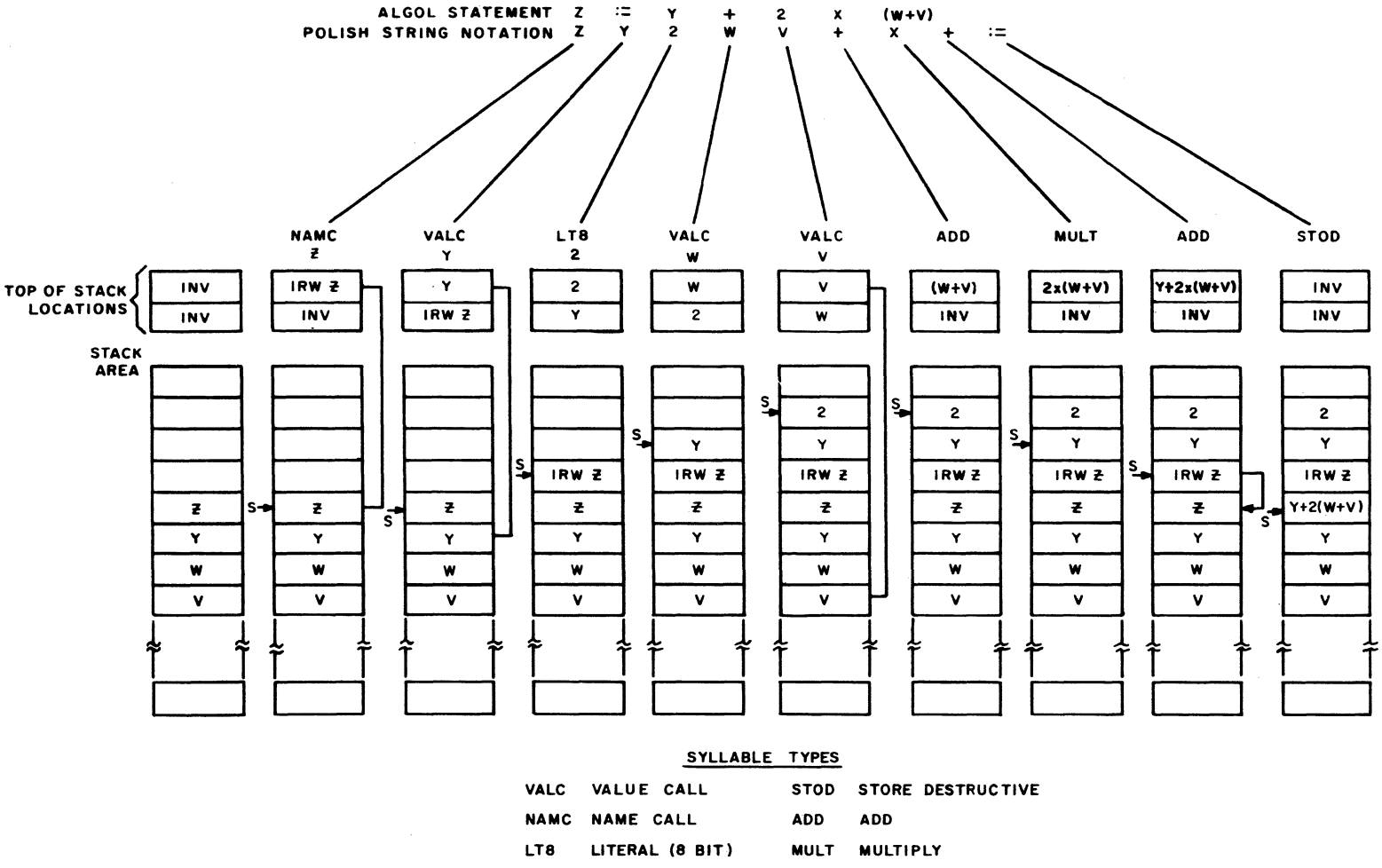


Figure 2-2-10. Stack Operation

Table 2-2-1. Description of Stack Operation

Execution Sequence	Polish Notation Element	Syllable Type Compiled	Function of Syllable During Running of the Program
0	-	-	Stack location of program variables illustrated.
1	Z	Name call for Z.	Build an indirect reference word that contains the address of Z and place it in the top of the stack.
2	Y	Value call for Y.	Place the value of Y in the top of the stack.
3	2	Literal 2.	Place a 2 in the top of the stack.
4	W	Value call for W.	Place the value of W in the top of the stack.
5	V	Value call for V.	Place the value of V in the top of the stack.
6	+	Operator add.	Add the two top words in the stack and place the result in the A location as the top of the stack operand.
7	x	Operator multiply.	Multiply the two top-of-stack operands. The product is left in the A location as the top of the stack operand.
8	+	Operator add.	Add the two top words in the stack and leave the result in the A location as the top of the stack operand.
9	:=	Operator store destructive.	Store an item into memory. The address in which to store is indicated by an indirect reference word or a data descriptor. The address can be above or below the item stored.

MULTIPLE STACKS AND RE-ENTRANT CODE

The B 7800 stack mechanism provides a facility for handling several active stacks, which are organized in a tree structure. The trunk of this tree structure is a stack containing MCP global quantities.

Level Definition

As the MCP is requested to run an execution of a program, a level-1 branch of the stack is created. This level-1 branch is a separate stack which contains only the descriptors pointing to the executable code and the read-only data segments for the program. Emerging from this level-1 branch is a level-2 branch, containing the variables and data for this job. Starting from the job's stack and tracing downward through the tree structure, one finds first the stack containing the variables and data for the job (at level 2), the segment descriptor to be executed (at level 1), and the MCP's stack at the trunk (level 0).

Re-entrance

A subsequent request to run another execution of an already-running program. Thus two jobs which are different executions of the same program have a common node, at level-1, describing the executable code. It is in this way that program code is re-entrant and shared. This results simply from the proper tree-structured organization of the various stacks within the machine. All programs within the system are re-entrant, including all user programs as well as the compilers and the MCP.

Job-Splitting

The B 7800 stack mechanism also provides the facility for a single job to split itself into two independent jobs. A common use of this facility occurs when there is a point in a job where two relatively large independent processes must be performed. This splitting can be used to make full use of a multiprocessor configuration, or to reduce elapsed time by multiprograming the independent processes.

A split of this type establishes a new limb of the tree-structured stack, with the two independent jobs sharing that part of the stack which was created before the split was requested. The process is recursively defined and can happen repeatedly at any level.

Stack Descriptor

Stack branches are located by an array of descriptors, the stack vector array (figure 2-2-11). There is a data descriptor in this array for every stack branch. This data descriptor, the stack descriptor, describes the length of the memory area assigned to a stack branch and its location in either main memory or disk.

A stack number is assigned to each stack branch. The stack number is the index value of the stack descriptor in the stack vector array.

Stack Vector Descriptor

The array size of the stack vector and its location in memory is described by the stack vector descriptor, located in a reserved position of the trunk of the stack (figure 2-2-11). All references to stack branches are made through the stack vector descriptor, indexed by the stack number.

Presence Bit Interrupt

A Presence Bit Interrupt results when an addressed stack is not present in memory. This Presence Bit Interrupt facility permits stack overlays and recalls under dynamic conditions. Idle or inactive stacks may be moved from main memory to disk as the need arises and, when a stack is subsequently referenced, a Presence Bit Interrupt is generated to cause the MCP to recall the nonpresent stack from disk.

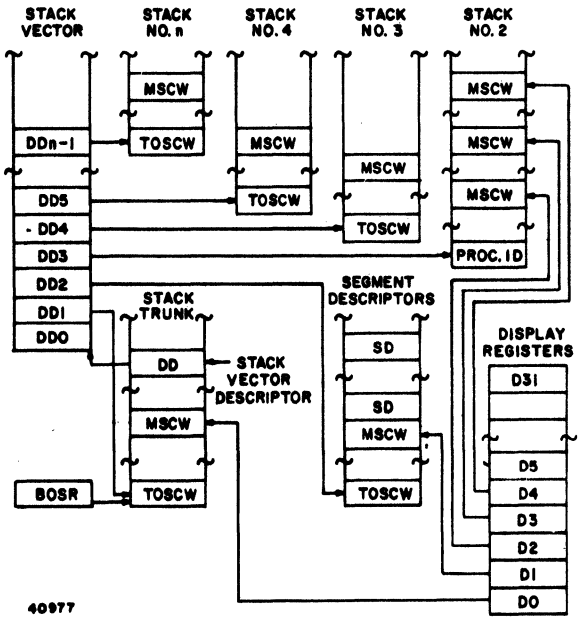


Figure 2-2-11. Multiple Linked Stacks

SECTION 3

PROCESSOR WORD FORMATS

GENERAL

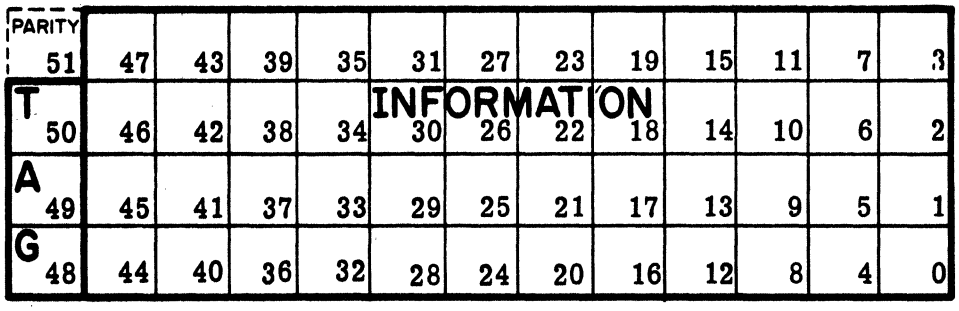
The basic information structure of the B 7800 is the word. As transferred between CPMs or IOMs and main memory, a word consists of 52 bits (see figure 2 -3-1), and is considered in three parts: a parity bit, which is used to maintain overall parity for the word being transferred; a 3-bit tag field, which indicates the type of information contained within the word, and a 48-bit information field, which contains the actual information.

The tag field not only serves to identify the type of information contained in the word but also can be thought of as an extension of the operator being executed against the word. For example, because the tag field indicates to the CPM whether the operation involves single precision or double precision operands, a single instruction (ADD) serves both types of operations. In similar fashion, if the sum obtained was a double precision number (requiring two memory words of storage), and the receiving memory word indicates that a single precision operand was resident there, the CPM will round the sum to single precision and then store it.

The tag field also prevents the user from writing over program code or read-only data areas, and prevents him from reading (as data) program code, processor control words, and uninitialized operands.

Consider the bit assignments for the tag field, as illustrated. One can see that words which have bit 48 set, such as IRW's, SIRW's, Segment Descriptors, MSCW's, RCW's, Data Descriptors, and Program Control Words, should not be alterable by the user. The CPM will not allow such words to be modified except by use of the overwrite operators. Words that are used for stack control, MSCW's, RCW's, and PCW's, have bits 49 and 48 of the tag field set. The CPM will not allow such words to be interpreted as operands.

The information field may be used to store data (logical operands, string operands, numeric operands), to store program code (program word), to address data or code outside of the stack (data descriptor, string descriptor, segment descriptor), to address within stacks (indirect reference word, stuffed indirect reference word, stuffed indirect reference word, program control word), to store information re-



Field	Bits	Description
Parity	51:1	Parity bit. Odd parity for the 52 bit word.
Tag	50:3	Value of this field indicates the usage of the information field, as described below.

Tag Value	Information Field Usage
0	Single Precision Operand, Logical Operand, String Operand, Occurs Index Word, Time of Day Function Word
1	Indirect Reference Word, Stuffed Indirect Reference Word
2	Double Precision Operand
3	Mark Stack Control Word, Return Control Word, Top of Stack Control Word, Program Word, Segment Descriptor
4	Step Index Word
5	Data Descriptor, String Descriptor
6	Uninitialized Operand
7	Program Control Word

INFORMATION	47:48	Use of this field depends on the value of the tag field.

40978

Figure 2-3-1. Basic Word Format

garding stack history (mark stack control word, return control word, top of stack control word), or to provide a parameter for use with certain operators (step index word, and occurs index word. Data words (operands) and program words were described in the previous sections of this chapter. The other various processor words are described in this section.

WORDS FOR ADDRESSING OUTSIDE OF THE STACK

There are three types of descriptors which are used for addressing data or code which is not resident in the stack. The type of descriptor is directly related to the data or code being referenced. Thus, a segment descriptor will always address a segment of program code (contained in program words), a string descriptor will always address a string operands), and a data descriptor will address an array of word operands.

The ADDRESS field in each of these descriptors is 20 bits in length; this field contains the absolute address of an array in either system main memory or in the backup disk file, as indicated by setting of the Presence bit (P). The referenced data is in main memory when the presence bit is set.

Presence Bit

A Presence Bit Interrupt occurs when the job references data by means of a descriptor in which the P-bit is equal to 0; i.e., the data is located in a disk file, rather than in main memory. The Master Control Program (MCP) recognizes the Presence Bit Interrupt and transfers data from disk file storage to main memory. After the data transfer to main memory is completed, the MCP marks the descriptor present by setting the P-bit to 1, and places the new main memory address into the address field of the descriptor. The interrupted job is then reactivated.

Index Bit

A Data Descriptor describes either an entire array of data words, or a particular element within an array of data words. If the descriptor describes the entire array, the Index bit (I-bit) in the descriptor is 0, indicating that the descriptor has not yet been indexed. The length field of the descriptor defines the length of the data array.

Invalid Index

A particular element of an array is described by indexing an array descriptor. Memory protection is ensured during indexing operations by performing a comparison between the length field of the descriptor and the index value. An Invalid Index Interrupt results if the index value exceeds the length of the memory area defined by the descriptor, or if the index is less than 0.

Valid Index

If the index value is valid, the length field of the descriptor is replaced by the index value, and the I-bit in the descriptor is set to 1 to indicate that indexing has taken place. The address and index fields are added together to generate the absolute machine address whenever an indexed Data Descriptor in which the P-bit is set is used to fetch or store data.

The Double-Precision bit (D) is used to identify the referenced data as single-or double-precision and directly affects the indexing operation. The D-bit equal to 1 signifies double-precision and causes the index value to be doubled before indexing.

Read-Only Bit

The Read-Only bit (R) specifies that the memory area described by the Data Descriptor is read-only area. If the R-bit of a descriptor is set to 1, and the area referenced by that descriptor is used for storage purposes, an interrupt results.

Copy Bit

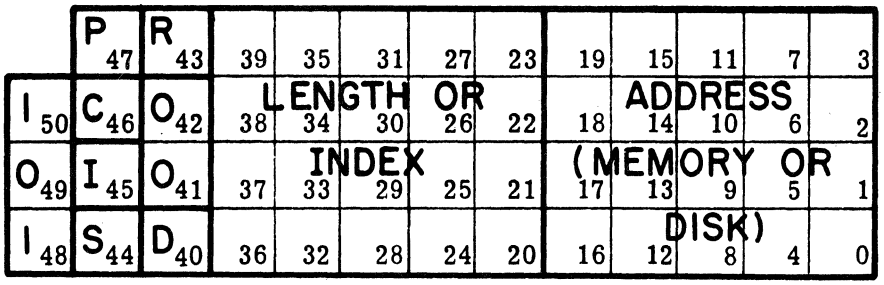
The Copy bit (C) identifies a descriptor as a copy of a master descriptor and is related to the presence-bit action. The copy bit links multiple copies of an absent descriptor (i.e., the presence bit is off) to the one master descriptor. The copy bit mechanism is invoked when a copy is made in the stack. If it is a copy of the original, absent descriptor, the processor sets the copy bit to 1 and inserts the address of the master descriptor into the address field. Thus, multiple copies of absent data descriptors all point back to the master descriptor.

Data Descriptor

Data descriptors refer to data areas, including input/output buffer areas. The data descriptor defines an area of memory starting at the base address contained in the descriptor. The size of the memory area in operands is contained in the length field of the descriptor. Data descriptors may directly reference any memory word address from 0 through 1,048,576. The structure of the data descriptor is illustrated in figure 2-3-2.

String Descriptor

String descriptors refer to strings of 4-bit digits, 6-bit or 7-bit characters, or 8-bit bytes.



Field	Bits	Description
Tag	50:3	Tag field. Value of five.
P	47:1	Presence bit. Indicates the presence or absence of data in main memory. A 0 causes a presence bit interrupt whenever the descriptor is used by a processor to obtain non-present data. A 1 indicates that the data described is in main memory.
C	46:1	Copy bit. A 0 indicates that this is the original descriptor for the particular data area. A 1 indicates that this descriptor is a copy of the original descriptor.
I	45:1	Indexed bit. A 0 indicates that an indexing operation is required before the descriptor may be used to obtain data. A 1 indicates that indexing has already taken place and the index value is stored in bit positions 39:20 (Length or Index).
S	44:1	Segmented bit. A 0 indicates that the data is not segmented. A 1 indicates that the data is divided into segments.
R	43:1	Read-only bit. A 0 indicates that the data may be referenced for reading or writing. A 1 indicates that the data can only be referenced for reading.
	42:2	Size field, must be 0 to indicate a data descriptor.
D	40:1	Double-precision bit. A 0 indicates single-precision operands, a 1 indicates double-precision operands.
Length or Index	39:20	This field contains either the length (in operands) of the memory area (if bit 45 = 0) or an index value (if bit 45 = 1). If bit 45 equals 0, the descriptor has not been indexed. This field is used for size checking during the indexing operation. If bit 45 equals 1, the descriptor has been indexed. For a double-precision operation, the index is doubled after index size checking, and the result is stored in the index field.
Address (Memory or Disk)	19:20	This field contains either a main memory or disk address. If the presence bit (bit 47) equals 1, this field contains the memory address of data. If the presence bit equals 0 and the copy bit (bit 46) equals 0, this field contains the disk address of the data. If the presence bit equals 0 and the copy bit equal 1, this field contains the memory address of the original descriptor.

Figure 2-3-2. Data Descriptor

The string descriptor defines an area of memory starting at the base address contained in the descriptor. The size of the memory area in characters is contained in the length field of the descriptor. The structure of the String Descriptor is illustrated in figure 2-3-3.

Segment Descriptors

Segment descriptors refer to areas of program code. The descriptor defines an area of memory starting at the base address contained in the descriptor. The size of the memory area in program words is contained in the length field of the descriptor. The structure of the segment descriptor is illustrated in figure 2-3-4.

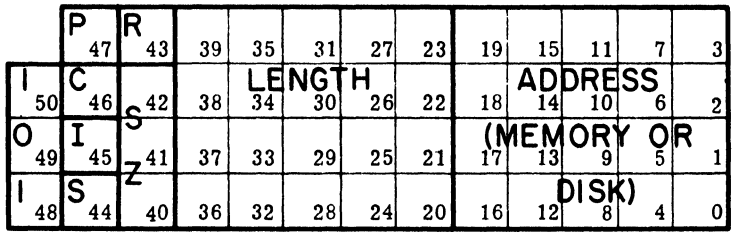
WORDS FOR ADDRESSING WITHIN STACKS

There are three types of words which are used for addressing data or descriptors which are resident within a stack. A Program Control Word is used, at the time of procedure entry, to locate a segment descriptor (and the proper word and syllable of code) for the procedure. An Indirect Reference Word is used to address within the current addressing environment of a procedure. A Stuffed Indirect Reference Word is used to address outside the current addressing environment of a procedure.

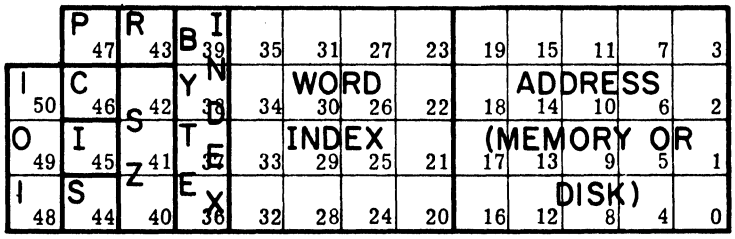
Program Control Word

The Program Control Word (PCW), and the MSCW are used during entry into a procedure. The organization of the PCW is illustrated in figure 2-3-5 and contains the following:

STRING DESCRIPTOR (NON-INDEXED)



STRING DESCRIPTOR (INDEXED)



Field	Bits	Description
Tag	50:3	Tag field. Value of five.
P	47:1	Presence bit. A 0 causes a presence bit interrupt if the descriptor is used to access data. A 1 indicates the data is present in main memory.
C	46:1	Copy bit. A 0 indicates that this is the original descriptor for the particular data area. A 1 indicates that this descriptor is a copy of the original descriptor.
I	45:1	Indexed bit. A 0 indicates indexing is required. A 1 indicates that indexing has taken place and the word and character index are in the WORD INDEX and BYTE INDEX fields.
S	44:1	Segmented bit. A 0 indicates that the data area is not segmented. A 1 indicates that the data is segmented.
R	43:1	Read only bit. A 0 indicates that the data may be referenced for reading or writing. A 1 indicates that the data can be read only.
SZ	42:3	Size field. 100 indicates character size of 8-bit bytes, 101 indicates 7-bit ASCII characters, 011 indicates 6-bit characters, and 010 indicates 4-bit digits.
Length	39:20	Bits 39:20, contain either the length of the memory area (bit 45=0) or an index value (bit 45=1). When bit 45 equals 0, this field contains the length of the area in digits, characters or bytes.
Byte Index	39:4	Byte index (Bit 45=1).
Word Index	35:16	Word Index (Bit 45=1).
Address (Memory or Disk)	19:20	This field contains either a main memory or a disk address. If the presence bit (bit 47) is 1, the field contains a memory address of the data. If both the presence bit and the copy bit (bit 46) are equal to 0, the field contains the disk address of the non-present data. If the presence bit is 0 and the copy bit is 1, the field contains the memory address of the original descriptor.

Figure 2-3-3. String Descriptor

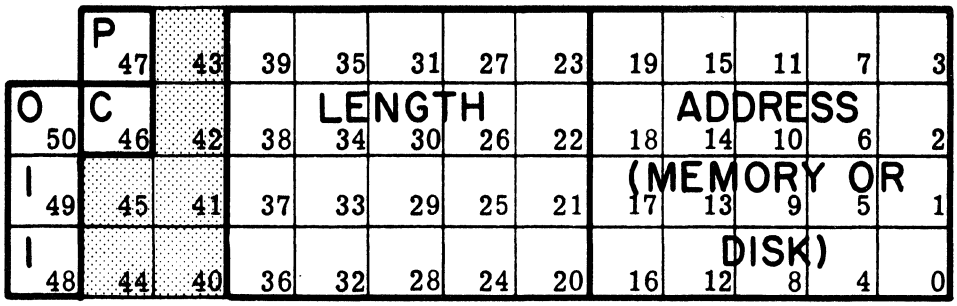
Indirect Reference Word

Referencing a variable within the current addressing environment of a procedure is accomplished through the address couple in the Indirect Reference Word (IRW). References are relative to the D register specified by the address couple. The format of the IRW is shown in figure 2-3-6.

Stuffed Indirect Reference Word

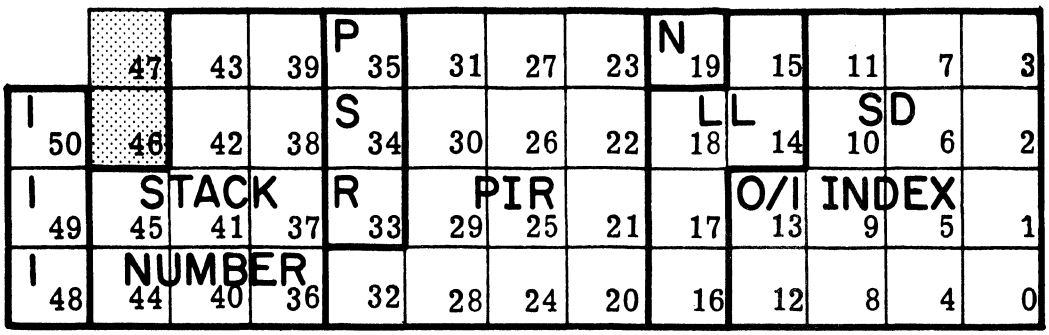
Reference to variables outside the current environment is accomplished by a Stuffed Indirect Reference Word. This addressing is relative to the base of the stack in which the variable is located.

The SIRW contains the stack number, the location (DISP) of the MSCW, and the index to



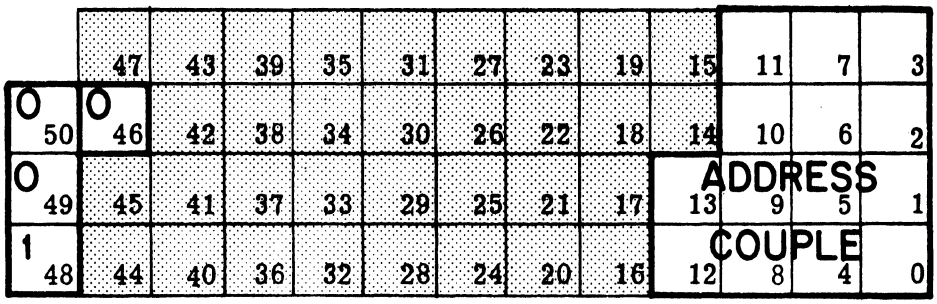
Field	Bits	Description
Tag	50:3	Tag field. Value equals three.
P	47:1	Presence bit. A 0 indicates that the segment is absent from main memory.
C	46:1	Copy bit. A 0 indicates that this is the original segment descriptor. A 1 indicates that this is a copy of the original segment descriptor.
	45:4	Not used. Unused bits may be either 0 or 1.
Length	39:20	The length of the program segment in words.
Address (Memory or Disk)	19:20	This field contains either the main memory address or the disk file address. If the presence bit (bit 47 equals 1, the field contains the main memory address of the program segment. If both the presence bit and the copy bit (bit 46) equal 0, the field contains the disk address of the non-present program segment. If the presence bit equals 0 and the copy bit equals 1, the field contains the absolute memory address of the original program segment descriptor.

Figure 2-3-4. Segment Descriptor



Field	Bits	Description
Tag	50:3	Tag field. Value equals seven.
	47:2	Not used.
Stack Number	45:10	The number of the stack which contains the PCW.
PSR	35:3	The program syllable (0-5) within the word located by PIR.
PIR	32:13	Index to the Program Base Register. Locates a word within the code segment.
N	19:1	Normal state (0) or control state (1).
LL	18:5	The level of the procedure being entered.
SD Index	13:14	The segment descriptor index. Bits 12 through 0 specify the value to be added to either D-register 0 or 1. When bit 13 equals 0, D-register 0 is selected; when bit 13 equals 1, D-register 1 is selected. The sum of the contents of the display register and the index locates a segment descriptor.

Figure 2-3-5. Program Control Word



Field	Bits	Description
Tag	50:3	Tag field. Value equals one.
	47:1	Not used
	46:1	Environment bit. Must equal zero for an IRW. (1 = SIRW).
	45:32	Not used.
Address Couple	13:14	Selects D Register (According to current program level as indicated by rLL) and provides index value (see below).

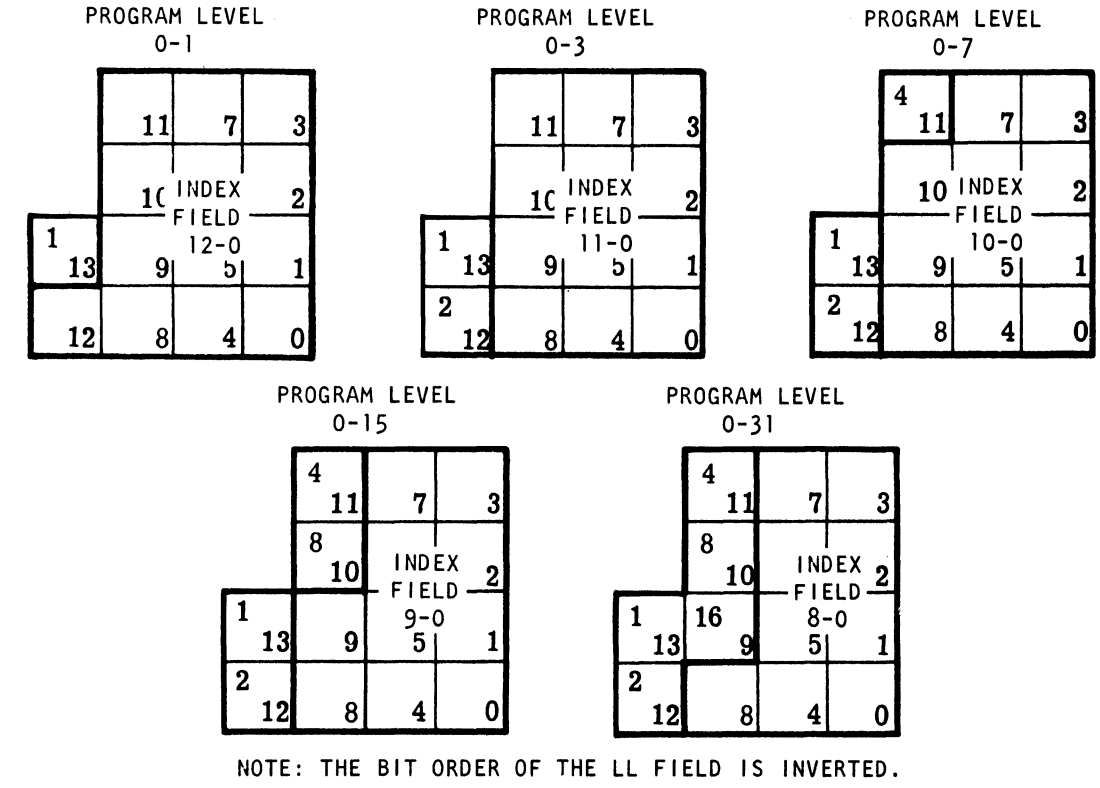


Figure 2-3-6. Indirect Reference Word

the variable relative to the MSCW. The absolute memory location of the variable is formed by adding the contents of DISP and index to the base address of the referenced stack from the stack descriptor. The contents of the SIRW (with the exception of index) are dynamic and are accumulated as the program is executed. The stack number and DISP fields are entered into the SIRW by the Stuff Environment (STFF) operator. The bit format of the SIRW is shown in figure 2-3-7.

WORDS FOR STORING STACK HISTORY

Certain words can be thought of as words used for storing stack history. These words, used for procedure entry and exit, as well as for storing the stack state for inactive stacks, include the Mark Stack Control Word, the Return Control Word, and the Top Of Stack Control Word.

Mark Stack Control Word

The Mark Stack Control Word (MSCW), together with the Return Control Word (RCW), provides a linking mechanism for the history of previous control-register settings through the stack.

The MSCW is placed in the stack by the Mark Stack operator. The MSCW is organized as illustrated in figure 2-3-8.

Return Control Word

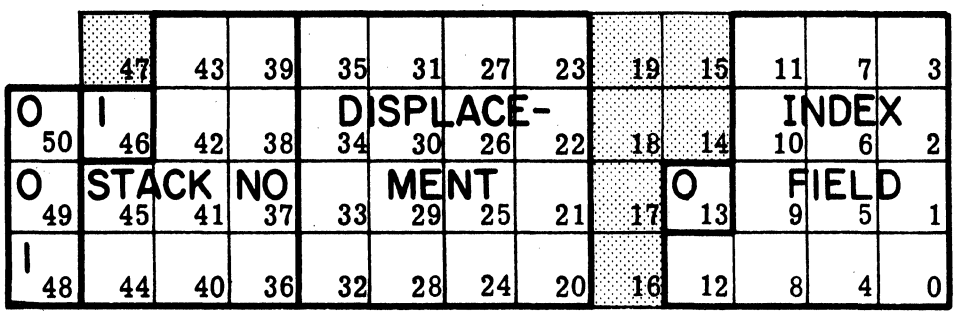
The Return Control Word (RCW) and the MSCW are used for subroutine handling. The Return Control Word stores the environment to which the subroutine will return. The organization of the RCW is illustrated in figure 2-3-9.

Top of Stack Control Word

The Top Of Stack Control Word (TOSCW) contains all information needed to restore the operating environment when a stack (or process) is activated. When a stack is active, the first word of the stack is a single precision operand containing the processor ID (a number, 0 through 7). When the stack is made inactive, the processor ID is changed to a TOSCW, containing the status of various processor flip-flops necessary to restore the stack's environment when it is again activated. The TOSCW is created by the Move Stack (MVST) operator. The TOSCW is illustrated in figure 2-3-10.

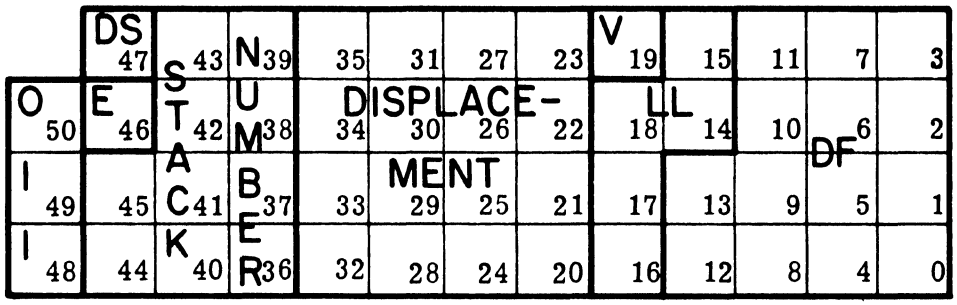
WORDS USED AS SPECIAL PARAMETERS

Certain control words are used only as a parameter to a single operator. Among these are the Step Index Word, used with the Step and Branch operator; the Occurs Index Word, used with the Occurs Index operator; and the Read Time Of Day Function Word, used with the Scan In operator.



Field	Bits	Description
Tag	50:3	Tag field. Value equals one.
	47:1	Not used.
	46:1	Environment bit. Must be a one (0=1RW).
Stack No.	45:10	The number of the stack containing the referenced word.
Displacement	35:16	This number, added to the stack base address, addresses an MSCW.
	19:6	Not used.
	14:1	Must be 0.
Index Field	12:13	This number, added to the address of the MSCW, addresses the referenced word.

Figure 2-3-7. Stuffed Indirect Reference Word



Field	Bits	Description
Tag	50:3	Tag field. Value equals three.
DS	47:1	Different-stack bit. A 0 indicates that the stack-number field refers to the current stack. A 1 indicates that the stack-number field refers to a different stack.
E	46:1	Environment bit. A 0 indicates an inactive MSCW, generated directly by the Mark Stack operator. The procedure entry has not been performed. A 1 denotes an active MSCW generated upon entry into a procedure, at which time the environment fields (stack number, displacement, value, and LL fields) are stored into the MSCW.
Stack Number	45:10	Stack-number field. Contains the number of the stack from which the PCW was obtained at procedure-entry.
Displacement	35:16	Displacement field. When added to the stack base address, locates the MSCW of the prior lexicographic level.
V	19:1	Value bit. A 0 indicates that the MSCW was generated during any operation that will be restarted from the beginning. A 1 indicates that the operator must continue after the Exit or Return which refers to this MSCW (e.g., an accidental entry by a Value Call).
LL	18:5	LL field. Denotes the lexicographical level at which the program will run when the procedure is entered.
DF	13:14	Denotes the stack history. This field is used to locate, in the stack, the preceding MSCW (i.e., the previous "F" register setting).

Figure 2-3-8. Mark Stack Control Word

Step Index Word

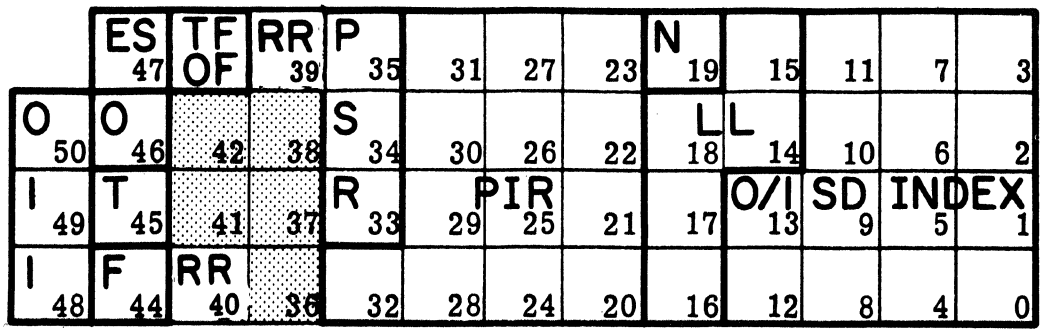
The Step Index Word (SIW) is used as a parameter to the Step and Branch operator, to increase the efficiency of this operator in iteration loops. When the Step and Branch operator is invoked, the SIW addressed by the IRW in the top of stack location is located. The increment field is added to the current value field. If the current value field is then greater than the final value field, PIR and PSR are set from the next two syllables in the program code string and the branch is made. If the current value field is not greater than the final value field, PIR and PSR are advanced three syllables, the SIW is replaced in memory, and the iteration loop continues. The format of the SIW is illustrated in figure 2-3-11.

Occurs Index Word

The Occurs Index Word (OIW) is used to index a field within an array. COBOL permits arrays to be constructed of a series of fields of a specified character size (through use of the OCCURS clause). This series of fields may not necessarily begin at a word boundary, because the array may be one of several items subordinated under a group item. The OCRX operator, together with an OIW in the A location and an index value in the B location, is used to calculate a new index value which is left in the top of the stack. The original index value is an integer which indicates the relative position of the desired field within the array. The new index value is the displacement (in characters) of the desired field from the first character of the array. The character size (specified in a descriptor) and the index value (left in the top of stack) can then be used to address the desired field. The format of the OIW is shown in figure 2-3-12.

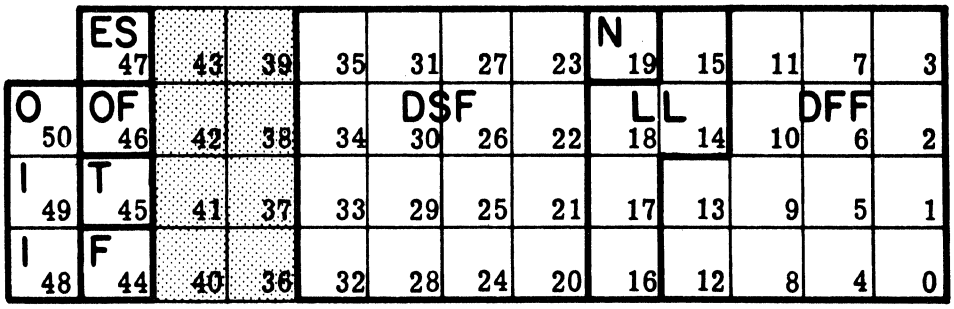
Time of Day Function Word

This word is used as a parameter to the Scan In operator, to specify that the time of day is to be interrogated by the MCP. The format of the Time of Day Function Word is shown in figure 2-3-13.



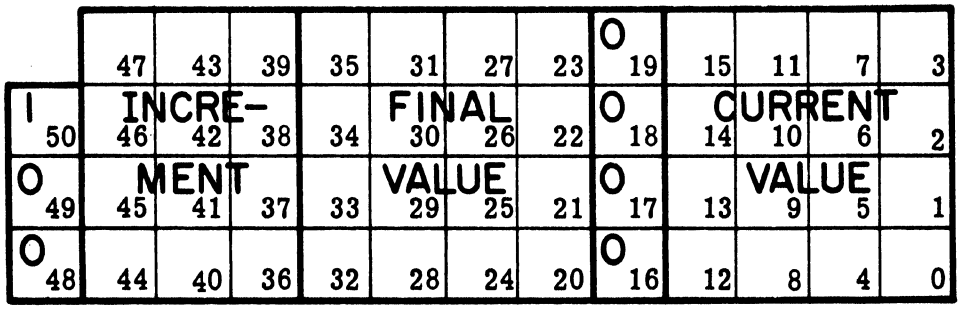
Field	Bits	Description
Tag	50:3	Tag field. Value of three.
ES	47:1	External Sign flip-flop.
O	46:1	Overflow flip-flop.
T	45:1	True/False flip-flop.
F	44:1	Float flip-flop.
TFOF	43:1	True/False flip-flop occupied flip-flop.
RR	40:1 39:1	Special hardware bits. These bits are used for controlling an early segment descriptor fetch during exit and return operations.
PSR	35:3	Program syllable of the operator to be executed after return from the subroutine.
PIR	32:13	PIR setting of the operator to be executed next in the calling routine.
N	19:1	Normal state (0) or control state (1) procedure.
LL	18:5	Level of the calling procedure when the RCW was generated (at procedure entry).
SD Index	13:14	Segment descriptor index. Bits 12 through 0 specify the value to be added to either D-register 0 or 1. When bit 13 = 0, D-register is selected; when bit 13 = 1, D register 1 is selected. The sum of the contents of the selected display register and the index locates a segment descriptor.

Figure 2-3-9. Return Control Word



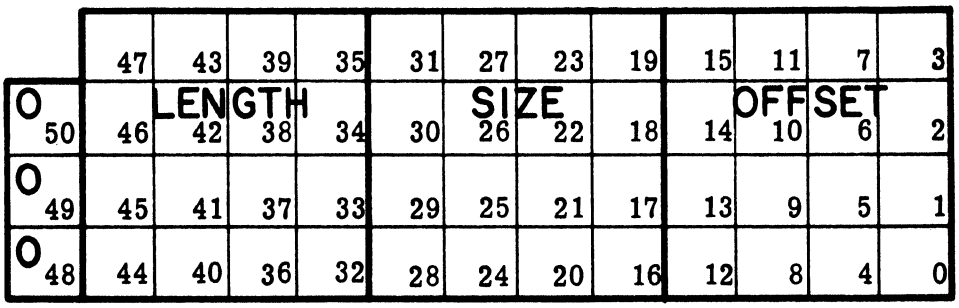
Field	Bits	Description
Tag	50:3	Tag field. Value equals three.
ES	47:1	External sign flip-flop.
OF	46:1	Overflow flip-flop.
T	45:1	True/False flip-flop.
F	44:1	Float flip-flop.
	43:8	Not used.
DSF	35:16	Delta S-register field. The value of S-register displacement above BOSR.
N	19:1	Normal-control state flip-flop. 0 = normal; 1 = control state.
LL	18:5	Lexicographic level.
DFF	13:14	Delta F-register field. The value of F-register displacement below the S-register.

Figure 2-3-10. Top of Stack Control Word



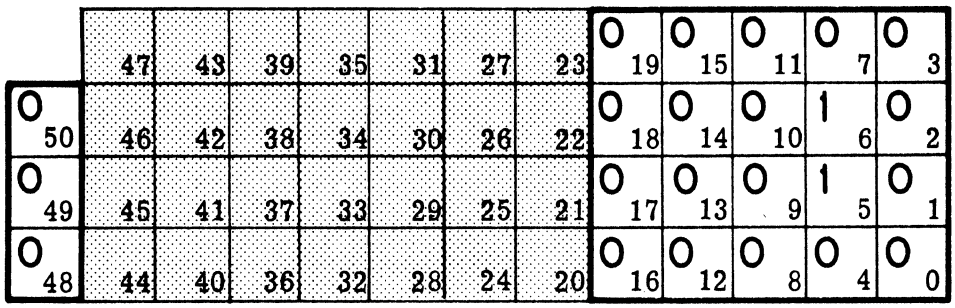
Field	Bits	Description
Tag	50:3	Tag field. Value equals four.
Increment	47:12	Increment: value to be added to current value field.
Final Value	35:16	Final value: value used to terminate the iteration loop.
	19:4	Must be 0 for S1W.
Current Value	15:16	Current value or count. The branch is made if this field is greater than the final value field.

Figure 2-3-11. Step Index Word



Field	Bits	Description
Tag	50:3	Tag field. Value equals zero.
Length	47:16	The length, in characters, of each field in the array.
Size	31:16	The size, in fields, of the array.
Offset	15:16	The number of characters preceding the first field of the array.

Figure 2-3-12. Occurs Index Word



Field	Bits	Description
Tag	50:3	Tag field. Value equal zero.
	47:28	Not used.
	19:13	Must equal zero.
	6:2	Must equal three.
	4:5	Must equal zero.

Figure 2-3-13. Time-Of-Day Function Word

SECTION 4

INPUT/OUTPUT SUBSYSTEM MAP STRUCTURE

INTRODUCTION

The B 7800 Input/Output Modules (IOM) operate in parallel with the Central Processor Modules (CPM). The purpose of the IOM is to control all data transfers between main memory and peripheral devices, or between two peripheral devices, so that the CPM is released from I/O operations at the earliest possible moment. In brief, the IOM controls not only the selection of I/O requests from lists of such requests in main memory, but also path selection to the desired devices, the initiation of requests on the appropriate device, the transfer of data as specified by the requests, and the construction of a list of completed requests in main memory. The CPM, on the other hand, builds the I/O request, places it in the appropriate list in main memory, notifies the IOM of the presence of the request (if this is the only request for the device), and then is free to continue with other processing. Routinely, the CPM checks memory for the presence of completed I/O requests and processes the completed requests.

Each IOM is, in effect, a separate computer with its own local memory, logic, arithmetic, and communication capabilities. This independent processing capability permits the IOM to perform routine input-output tasks without interrupting the CPM. Thus the IOM can control transfers of data between peripheral storage devices and main memory or other storage devices without direct supervision of the CPM. In fact, parallelism within the IOM permits it to initiate, service, and terminate data transfers for several users while the CPM is processing data for yet another user.

QUEUE-DRIVEN I/O

To allow the IOMs to properly select paths to the devices and to service I/O requests, certain structures are created by software when the system is initialized. These structures, which provide a mechanism to allow the CPMs to queue I/O requests, allow each IOM to be aware of the requests, of the devices it can service, and of the order of priority of devices served by an exchange. These structures are referred to as the I/O Subsystem Map, and hence this type of I/O is often reffered to as "map" I/O or "queue-driven" I/O. Because the use of the map allows the IOM to process many I/O operations in parallel, independent of CPM, I/O performed using the map is also known as asynchronous I/O. The IOM may also operate synchronously to process one I/O request at a time; however, such synchronous operation is used only for special applications such as system initialization and is not further discussed in this section.

The operation of asynchronous I/O is illustrated in simplified form in figure 2-4-1. When the I/O subsystem map is initialized the CPM places information about each peripheral device and the paths to it into a table in memory. During operation, the I/O subsystem map is accessed by both the CPM and IOM as I/O requests are built (by the CPM) and processed (by the IOM). In essence, the CPM builds I/O requests and places them in queues of such requests in main memory. Each request specifies the desired I/O operation and the device on which the operation is to be performed. The IOM extracts requests from these queues on a first-in first-out basis, processes each request, and places the completed requests into a queue in main memory.

Periodically, the CPM extracts the completed requests from the queue in main memory and takes the necessary action to check them.

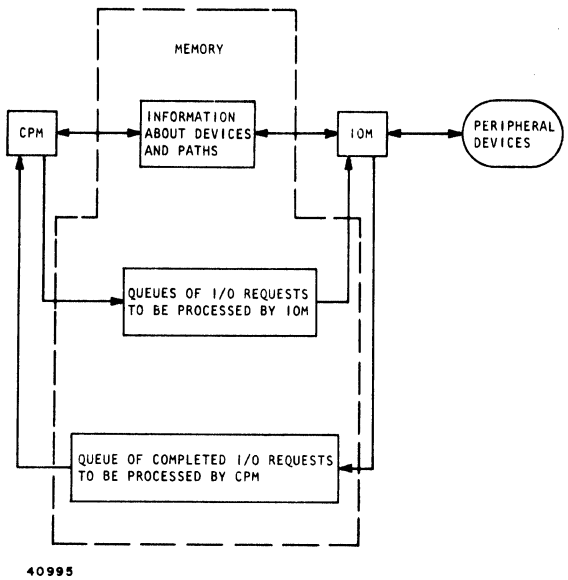


Figure 2-4-1. Asynchronous I/O Operation, Simplified Block Diagram

Once the IOM is notified (by the CPM) of the presence of an I/O request in one of the input queues, all requests in that queue will be processed by the IOM independent of CPM actions until the queue becomes empty. The CPM may place additional requests into a queue while the IOM is processing a request from the queue. Thus, once the IOM starts processing a queue, the CPM may process other programs, queue new I/O requests, and perform computations; effectively masking out the IOM transfer times.

ERROR HANDLING

From time to time conditions may arise which prevent I/O operations from being accomplished successfully. A printer may run out of paper, a card punch may be out of cards, or a device may for some reason not be ready. The design of the I/O subsystem map allows the IOMs to continue to process requests for other devices even though an error is detected on a particular device. When the error is recognized by an IOM, processing of further requests for the particular device is suspenended, the I/O request is marked as containing an error, and that marked request is linked into the queue of completed requests. The CPM is not interrupted to handle the error; however, when the CPM does process the queue of completed requests it will recognize and process the error. When the error has been processed, the CPM can again cause the IOM to process requests for the device on which the error was detected.

If such a strategy were to be applied to the handling of all input/output errors a catastrophic situation might arise. If, say, the IOM itself were the source of the error, it is conceivable that it could then process all (or many) I/O requests erroneously. However, in such cases the B 7800 IOM stops all processing of I/O requests (for all devices) and immediately interrupts the CPM. In short, I/O errors associated with a particular device cause processing of further requests for the device to be halted but allow the processing of requests for other devices by the IOM to continue. I/O errors which can be associated only with an IOM and not with a particular device cause the IOM involved to stop all processing of requests (other IOMs are not affected) and causes the system to be interrupted so that the IOM error may be processed. Provision is also made to allow the software to request that the system be interrupted when a particular I/O request is completed.

DEFERMENT OF PATH BINDING

The I/O subsystem map allows the IOM to select the transfer path for a device as the path becomes available. This dynamic path selection is logically similar to the call routing of a long distance telephone network; that is, the route of the call is selected based on the locations of the correspondents and the available paths. The user need only be concerned about the type of device to be used (card reader, magnetic tape, disk file, etc.); the MCP will associate the logical file with a physical device when the program is executed, and the IOM, when it initiates each transfer of data, will select an available transfer path to the device.

Maximum I/O throughput can be realized only if the binding of the data path between an IOM and a device is delayed until the device is ready to initiate the job. As shown in figure 2-4-2, if device 4 is to be initiated, the path required to connect CPM 1 with device 4 involves selecting between two IOM's and between two channels within each IOM. (The peripheral controls have been excluded from this figure because they do not affect the concepts

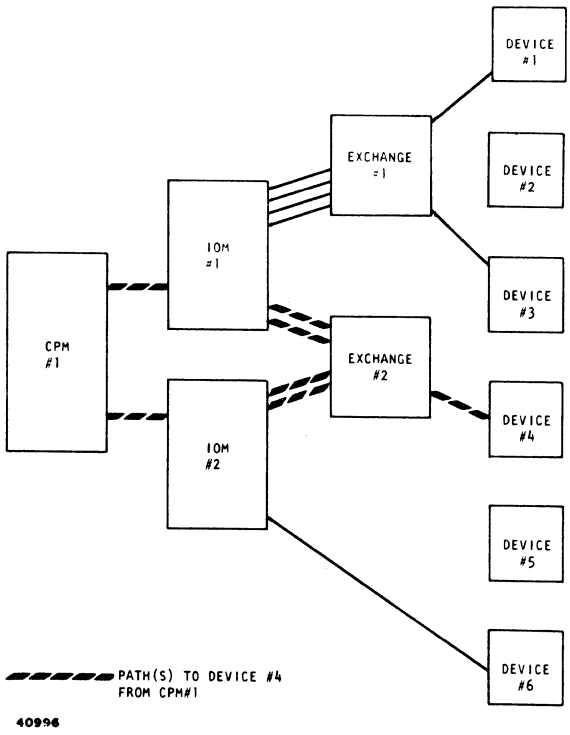


Figure 2-4-2. Data Transfer Path Selection

being described. For purposes of this discussion the peripheral controls may be thought of simply as extensions of the IOMs.) If the path to device 4 were to be preselected programmatically, a situation could develop in which the device is free but the preselected path is not. Thus, execution of the request would be unnecessarily delayed if in fact an alternate path to the device was available.

To delay binding the path programmatically generally would require that the CPM which initiated the job be involved in the operation until the request is actually initiated on the specified device. The I/O subsystem map, however, allows the IOM's to manage selection and binding of paths, allowing the CPM's to be free to do other processing. Thus, because the IOM processes I/O requests without CPM intervention, and because the IOM selects data paths at the time of execution, the total system time required to accomplish an I/O operation is limited to the amount of time required for a CPM to build an I/O request and place it in memory.

I/O SUBSYSTEM MAP

As shown in figure 2-4-3, the I/O subsystem map is made up of four major software structures in main memory. These four software structures are addressed by registers within the IOM: the Home Address words are addressed by the HA register; the Unit Table is addressed by the UT register; the Queue Head and Queue Tail words table is addressed by the QH register; and the Status Queue Header is addressed by the SQ register. The IOM uses the Queue Head word for the appropriate device to locate the I/O request. Thus, the IOM can locate any element of the map as necessary. Of course, since the map is constructed by the MCP, it too is aware of the location of each element of the map.

Commands and Requests

Before further discussion of the I/O subsystem map can take place, the difference between an I/O command and an I/O request must be made clear. An I/O command is an or-

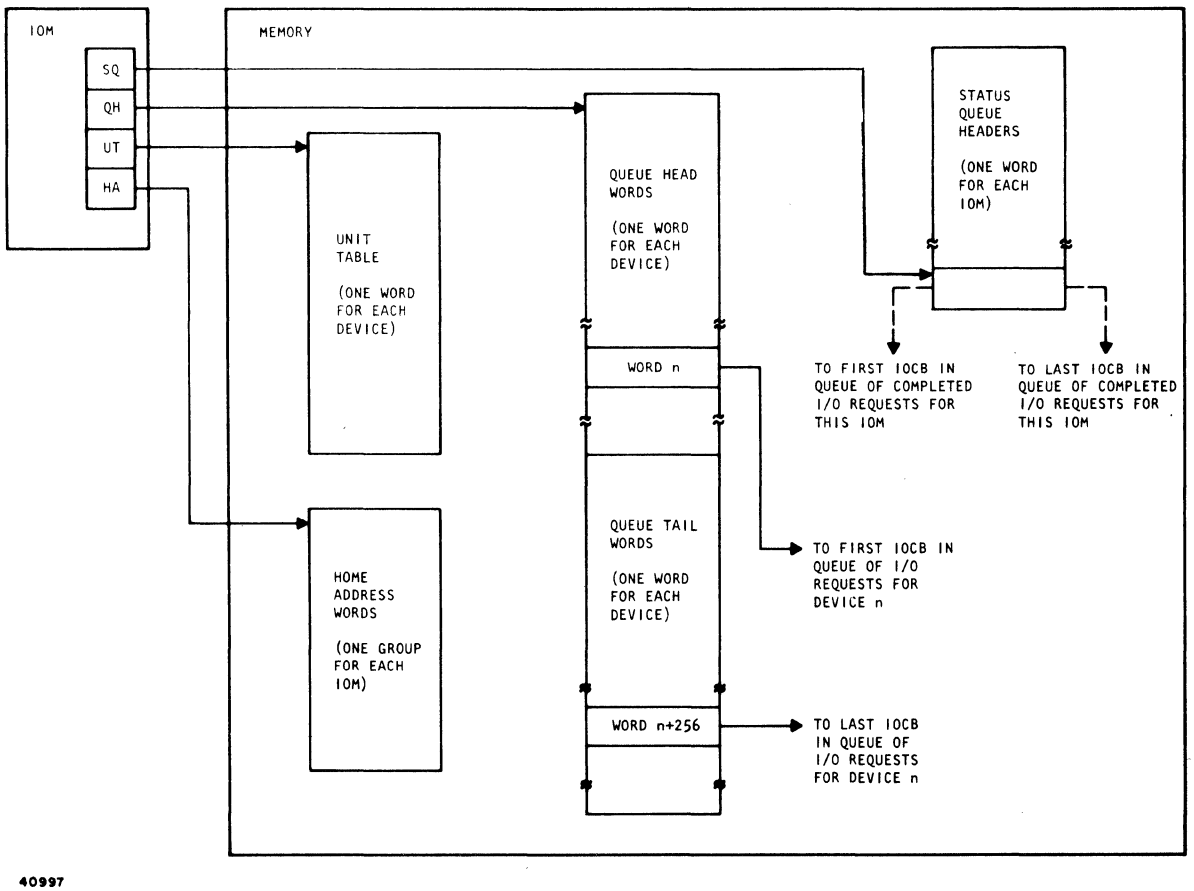


Figure 2-4-3. I/O Subsystem Map, Simplified Block Diagram

der to an IOM which can cause one operation or many operations for one device to be initiated by the IOM. Although there are special I/O commands which control but a single I/O operation, the I/O command most often associated with asynchronous I/O is the Start IO command, which causes the IOM to process I/O requests from a queue of such requests until the queue is empty. Each I/O request contains information describing a single input or output operation that is used not only by the IOM but also by the peripheral control and even the device itself. Each I/O request is made up of several words and is known as an I/O Control Block (IOCB). The IOCB is discussed in detail later in this section; I/O commands are described in chapter 4.

Map Integrity

Because the I/O subsystem may be accessed and modified by all CPM's and IOM's in the I/O subsystem, the integrity of the map is protected by special lock bits and lock words. This system of locks prevents conflicts between the IOM's and CPM's which use and modify the map. As shown in figure 2-4-4, the system consists of three types of locks; a lock bit and a lock word for each group of Home Address words, and a lock bit for each Unit Table word.

The software lock word prevents two or more CPM's from attempting to build I/O commands in the Home Address words simultaneously. This word must be unlocked before a CPM can access the Home Address words; the CPM will immediately lock this word when it gains access.

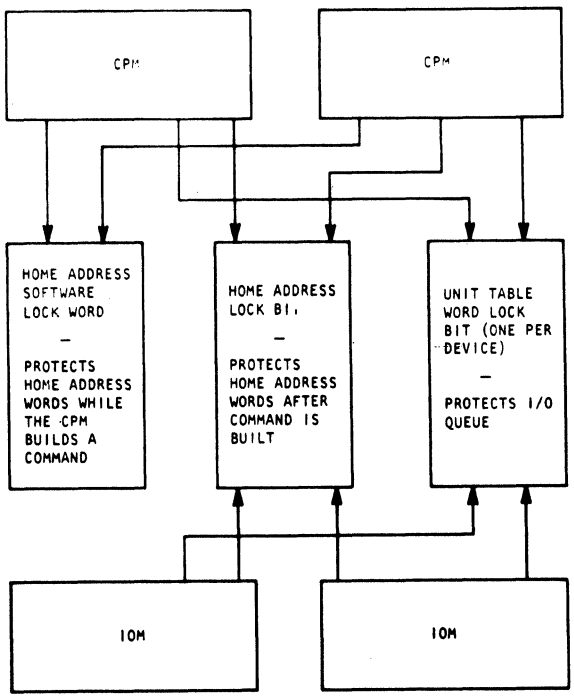


Figure 2-4-4. I/O Subsystem Map Protection

The Home Address lock bit prevents a command from being altered once it has been placed in the Home Address words for execution. The CPM locks this bit when a command is placed in the Home Address words.

In response to a channel interrupt, the IOM exchanges the contents of HA word with zero, decodes the home command, and executes the operation. When a CPM gains access to an HA block via the software lock-word, the CPM does not insert a HA command into the Home Address word until the HA lock bit is unlocked.

The lock bit in each Unit Table word protects the IO queues so that access to an I/O queue is not granted to more than one IOM or CPM at a time. The I/O queue can only be accessed when the lock bit is unlocked. Each IOM or CPM locks the bit when it is using the I/O queue and unlocks the bit when it is finished.

Home Address Words

For each IOM there exists a unique set of Home Address words in memory. The basic purpose of the Home Address words is to provide a location into which CPM's can store an I/O command until an IOM is ready to execute the command. The most generally used command is Start IO, which is used to initiate the processing of a queue of I/O requests for a device by an IOM. Other commands allow the IOM to perform special functions, such as loading into the IOM the addresses of the structures in the I/O map or performing synchronous I/O operations. Other words in the Home Address words are used as software lock words and, in certain cases, to store result descriptors for completed I/O operations.

Unit Table

For each device in the I/O subsystem there is one word in the Unit Table. This word is used both by the MCP and the IOM, and contains a lock bit which prevents conflicts of interest. This word indicates the path or paths to the unit, and provides other information needed by the IOM.

I/O Queue Head and Tail Words

For each device in the I/O subsystem there is one Queue Head word and one Queue Tail word. These words contain the address of the first IOCB and the last IOCB, respectively, in the queue of I/O requests for the unit. If there

are no IOCB's to be processed for the unit, these words will be empty.

Status Queue Headers

For each IOM there is a Status Queue Header. Fields in the Status Queue Header contain the addresses of the first and last IOCB in a queue of completed IOCB's. Thus, the Status Queue Header allows each IOM to maintain a single queue of completed I/O requests. Periodically, the MCP checks these completed requests.

Input/Output Control Block

An Input/Output Control Block (IOCB) contains the information needed by the IOM to perform one I/O operation on a device. I/O Control Blocks (see figure 2-4-5) contain information needed to link queues of IOCB's together, to describe the I/O operation to be performed, to locate the data buffer to be used for the operation, and in the case of completed IOCB's, to store the result descriptor describing the completed operation. A generalized illustration of an IOCB is shown in figure 2-4-5.

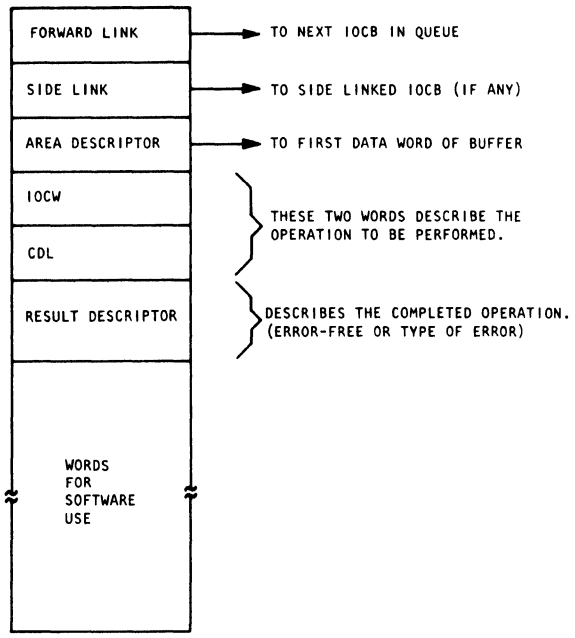


Figure 2-4-5. IOCB Format, Simplified

CHAPTER 3
CENTRAL PROCESSOR MODULE

SECTION 1
GENERAL DESCRIPTION

GENERAL

The B 7800 Central Processor Module (CPM) provides high performance through extended asynchronous operations of independent functional units within the CPM. Communications between these units are carried out by means of high-speed local storage and operation queues. These operation queues and local storage areas allow execution of operators and data exchange between operators to occur simultaneously. The operators can be deferred or restored within the pipeline flows of the CPM and can be completed out-of-sequence from the queues. The final completion of operators, with respect to code order, is ensured because different units within the CPM complete operators independent of any queue order sequence. These units are the program control unit, the data reference unit, the execution unit, and the EU store sub-unit.

Figure 3-1-1 is a simplified block diagram of the CPM showing the general interconnections and data flow between the local storage areas and the major units. The basic functions of these units of the CPM are described in the following paragraphs.

PROGRAM CONTROL UNIT

The Program Control Unit (PCU) is responsible for extracting an operator or a group of operators from the code string and initiating the processing of instructions. These instructions are placed in the appropriate queues in either the data reference unit or the execution unit.

The primary responsibility of the PCU is to allow operators to be executed when ready rather than be executed in a serial order from the code string. To accomplish this method of executing operators, the PCU is structured to be the stack machine of the CPM. This allows the PCU to change an operator or operators to a three-address operation (A, B, and R addresses) for the DRU and EU. These addresses are pointers to actual data register locations in the CDB. For example, the A and B addresses can be locations for operands to be fetched by the DRU. These same A and B locations are then read by the EU for an arithmetic operation in which the result is returned to R location in the CDB. Along with these addresses, the PCU passes a job number and other information necessary to complete the operation. The job number is used to maintain orderly processing of an operation in the pipeline levels of the DRU and EU and in deallocating CDB locations at the end of operations.

Other major responsibilities of the PCU are as follows:

1. Processes conditional branch operators (BRFL, BRTR, and STBR) independent of other units by examining the boolean when it is generated by the EU.
2. Transfers literal data directly into the CDB for use by the EU or DRU; in the case of non-concatenated NAMC, transfers the NAMC (an IRW) into the CDB for use by the DRU.
3. Provides AC (address couple) from a VALC or concatenated NAMC (for example, NAMC followed by INDX) to the DRU for obtaining an absolute address from which data is fetched and placed in the CDB for EU or DRU use.
4. Provides TOS (top-of-stack) instructions to the DRU allowing descriptor evaluation to be accomplished before an EU operator requires the referenced data.
5. Provides certain store instructions (STOD and STON) to the EU store sub-unit. This unit expedites stores as soon as possible so that time is not taken away from EU processing of other operators.
6. Processes stack operators, such as exchange and rotate stack down, by interchanging the CDB locations according to the type of stack operator being executed.
7. Contains a separate branch storage area so that any branch loop can be captured locally for fast processing.

DATA REFERENCE UNIT

The Data Reference Unit (DRU): 1) receives all data reference operations from the PCU; 2) calculates the absolute address of data; and 3) fetches the data from either the associative memory or main memory. Extensive use of pipelining allows one-

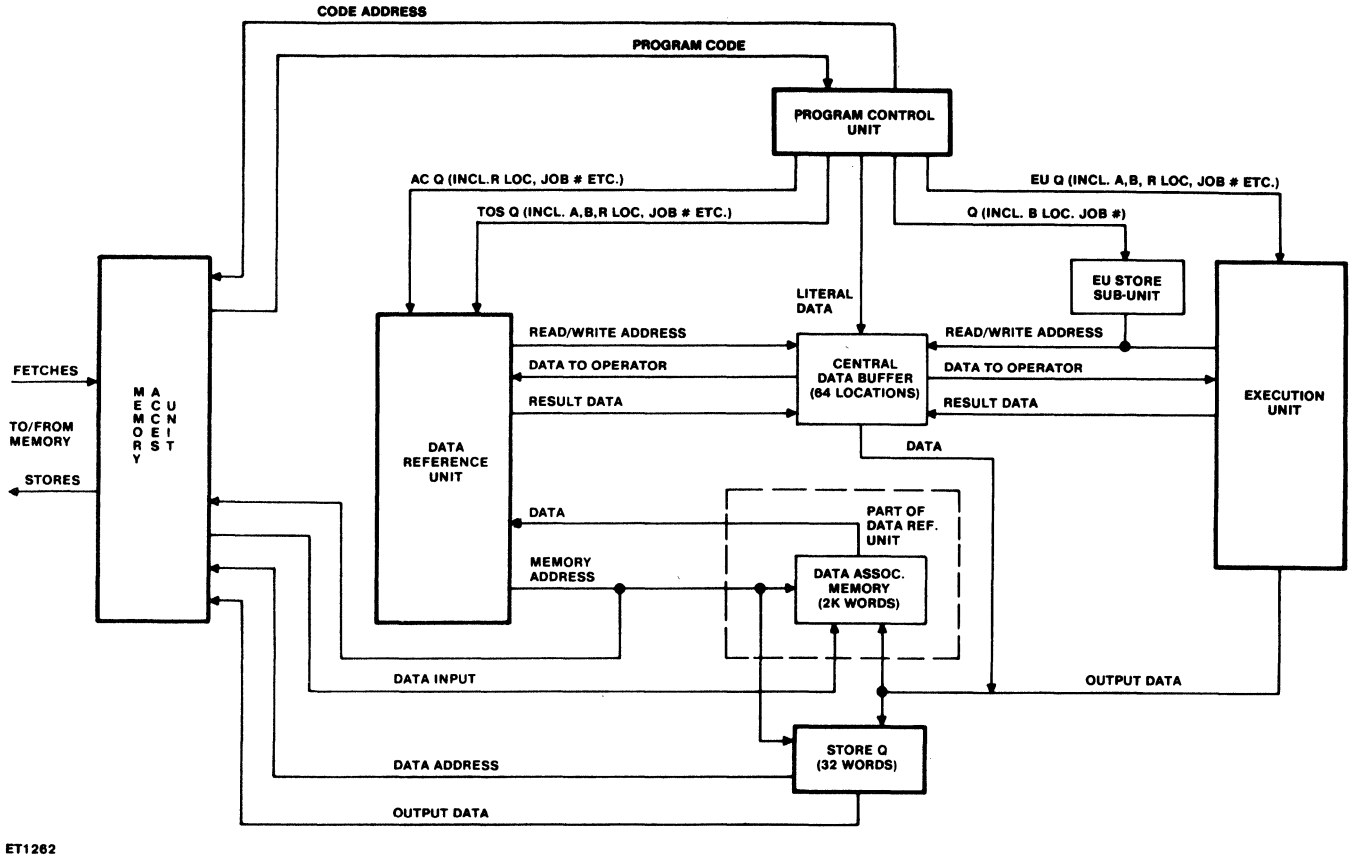


Figure 3-1-1. CPM Block Diagram

clock references to associative memory. Some other functions of the DRU are as follows:

1. Calculates start bit and field length information and places this information into the CDB for use by EU string operators.
2. Calculates and saves all write addresses for later use by the EU. During the EU portion of the write operation, the address is obtained from the DRU and is used for addressing the EU output data into both the associative memory and the store queue.
3. Compares saved write addresses against any subsequent fetch address to determine if these addresses are equal. If these addresses are not equal, the fetch is allowed to proceed if it does not address a location to be changed by a previous incomplete write. If these addresses are equal, the operation for that fetch is deferred until the EU has completed the write operation (for example, a NAMC STOR followed by a VALC with same address couple).
4. Performs all the procedure entry and exit instructions, except for assembling MSCW and RCW. (MSCW and RCW are assembled by the barrel mechanism in the EU.)

EXECUTION UNIT

The Execution Unit (EU) performs all arithmetic and logical operations on data from the CDB. Data derived by the EU may either be stored in the CDB or may be sent to associative memory and the store queue.

Extensive parallelism is provided by independent operating EU sections, which allow execution of newer operators (for example, index computation or loop control) in parallel with older operators. The EU has three processing sections: 1) arithmetic logic unit (ALU); 2) short arithmetic unit (SAU); and 3) EU central data buffer (EUCDB) logic. The basic functions of these sections are as follows:

1. SAU performs integer arithmetic operations, basically COMP, ADD, and MULT. It can operate on a maximum of 20 bits in the case of ADD and COMP operations. For MULT, an eight-bit multiply operation can be performed in three clocks to produce a 16-bit result. SAU is also used for exponent calculations and double precision. During double

precision operations, the SAU exponent calculations are controlled by macro operators from the macro code routines in the EU.

2. ALU performs most types of arithmetic and logical computations in the EU. It operates on non integer data or integer data that is greater than 20 bits.

3. EUCDB accesses the operands from the CDB for use by the ALU and SAU. It also directly controls the barrel to execute field operators and the data manipulation portion of string operators.

STORE QUEUE

The store queue is responsible for buffering data before it is sent to main memory to eliminate repeated stores to the same main memory location and to group multiword writes of adjacent words into a single main memory operation. For example, a NAMC STOR is part of a loop control in which a loop variable is updated by repeated entries into the store queue with the same memory address. Therefore, all but the newest entries (or writes) are discarded and unnecessary memory operations are not performed.

MEMORY ACCESS UNIT

The Memory Access Unit (MAU) provides the interface between the CPM and up to eight memory control modules MCMs in the main memory. Request for memory interface operations are made to MAU by the PCU, DRU, and store queue. Data fetches from memory by the MAU are forwarded to the associative memory. Program fetches are forwarded to the program buffer in the PCU. Data stores received (from store queue by the MAU) are forwarded to main memory. The MAU can perform simultaneous fetch and store operations with main memory, provided that these operations are not with same MCM.

SECTION 2
FUNCTIONAL OPERATION OF CENTRAL PROCESSOR MODULE SUBSECTIONS

GENERAL

This section contains a brief description of the operation of each of the CPM subsections described in section 1 of this chapter.

PROGRAM CONTROL UNIT

The program control unit (PCU), as shown in figure 3-1-1, consists of the instruction decode (PID) level, the instruction execute (PIE) level, along with the write level. Basically, the PID level is responsible for extracting code from the code buffer and presenting this code to the PIE level. Then, the PIE level issues the required micro operators to the write level for final distribution to the DRU and EU queues. At the PIE level, the allocation of registers in the control data buffer (CDB) and information, such as address couples, variant information, and job numbers are passed to the appropriate queues. Also, the initial stack environment for the micro operator and the top-of-stack configuration at the conclusion of that operator is controlled by the PIE level.

Because the PIE level or the write level can require resources (such as Q entries or CDB locations) which are not available, it becomes necessary to hold these levels when such conflicts occur. To accomplish this, the hold logic signals are applied as hold inputs to the clock buffers, whose outputs are routed to the control flip-flops in these levels. The PID level operates independently and does not require hold logic, but rather it has valid flip-flops to indicate that information is available for use in certain registers.

Generally, as shown in figure 3-2-1, a change of direction causes the program index registers (PIRs) to be loaded with either the branch PIR from PID or EU, the enter and exit PIR from the DRU, or the enter edit PIR from DRU. The PIR output is then added to the program base register (PBR) to form an absolute address of the next program word. This address is compared with stored addresses in an associative memory to determine whether the code resides in the program buffer. If the address compares with a stored address, the corresponding code is selected from the code buffer. Otherwise, the address is sent to the MAU for a memory fetch operation. The code is then removed from the code buffer, isolated by the barrel, and forwarded to the PID, a six-syllable register. The far left syllable in PID corresponds to current PIR in the PIN registers. As the first major register in the PCU operator pipeline, the PID is used to decode the program operator and to set up conditions for PCU registers in the PIE level.

PIR Circuits

Figure 3-2-2 shows the PIR circuits. These circuits include the PRI, PIN, PIC, PNB, PPR, and PPC registers, the 16-word branch and interrupt storages, and three PCB registers. These registers and storages contain the PIR and PSR values associated with the operators in the decode and execute levels of the PCU and in the change of processing direction. (The PIR and PSR, together with the program base register, identify the absolute address and starting syllable position of the associated program operator.)

The PRI (prime) register receives the PIR and PSR values from the following:

1. The interrupt storage.
2. The PCB register for conditional branch operators (BRFL, BRTR, and STBR).
3. The EU for dynamic branch unconditional (DBUN), and table enter edit, (TEED and TEEU) operators.
4. The PPC register for enter and exit operators.
5. The PPR for table enter edit operators so that branching back to normal code can be performed after completing edit mode.
6. The PID register for special handling of branch unconditional (BRUN) operator.

When PRI is loaded, it is declared valid and the contents in the branch storages are selected by a pointer (i.e., four LSBs of PRI) and transferred to the program read registers (PER and POR), the PID register, and the PIN register. These transfers are automatically made in anticipation that code in the branch storages is available from a previous change of processing direction. Hence, a time saving is realized, if a local check operation for code in the program buffer is not required at this time.

If code is available in the branch storages (PIN and PRI contents being equal), the address from the branch address storage is transferred to the address registers for immediate word selection from the program buffer. (This address represents the next word of code to be read from the program buffer.)

If code is not available in the branch storages (PIN and PRI not equal), the contents of PID are not

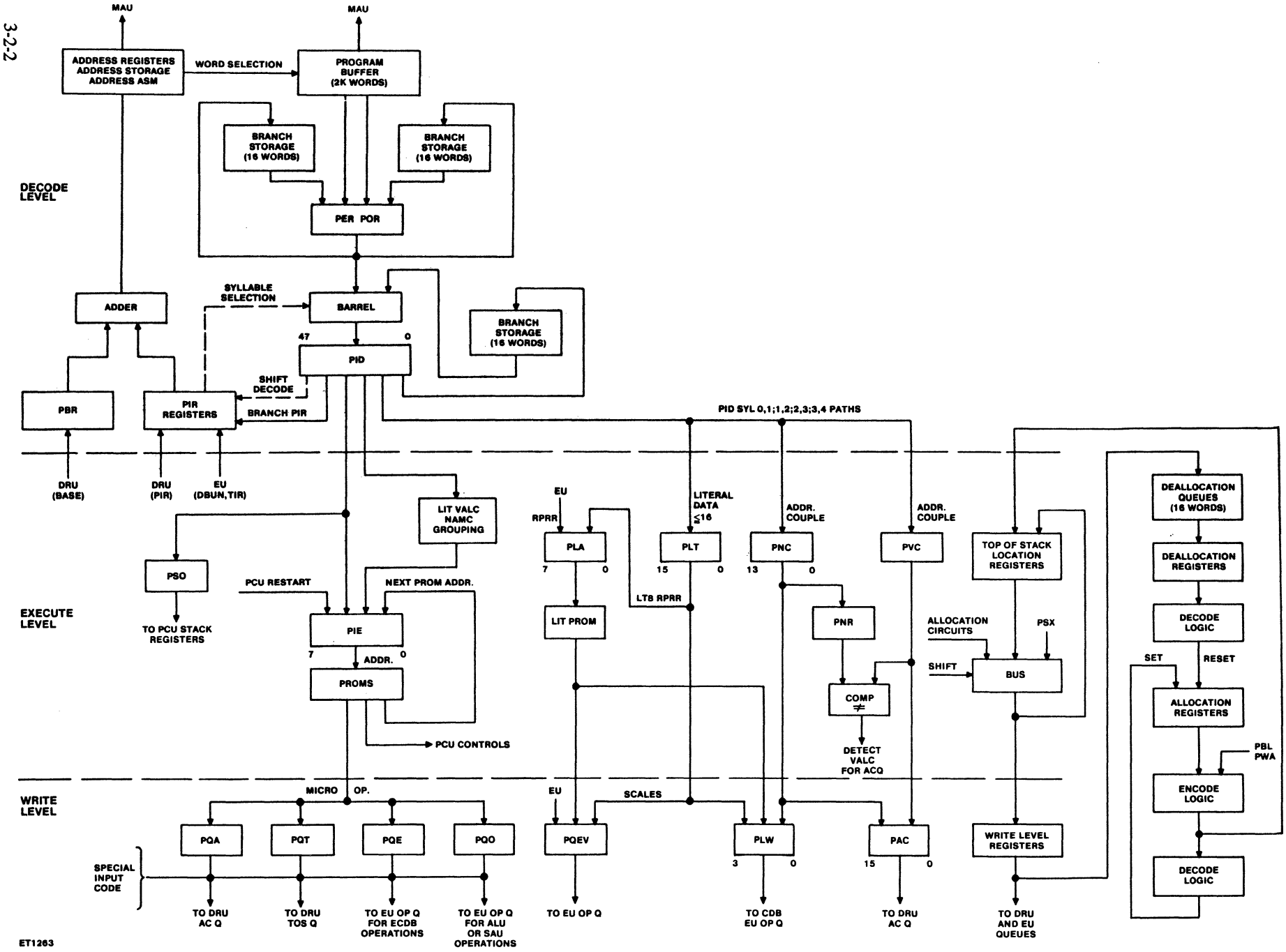


Figure 3-2-1. Program Control Unit, Block Diagram

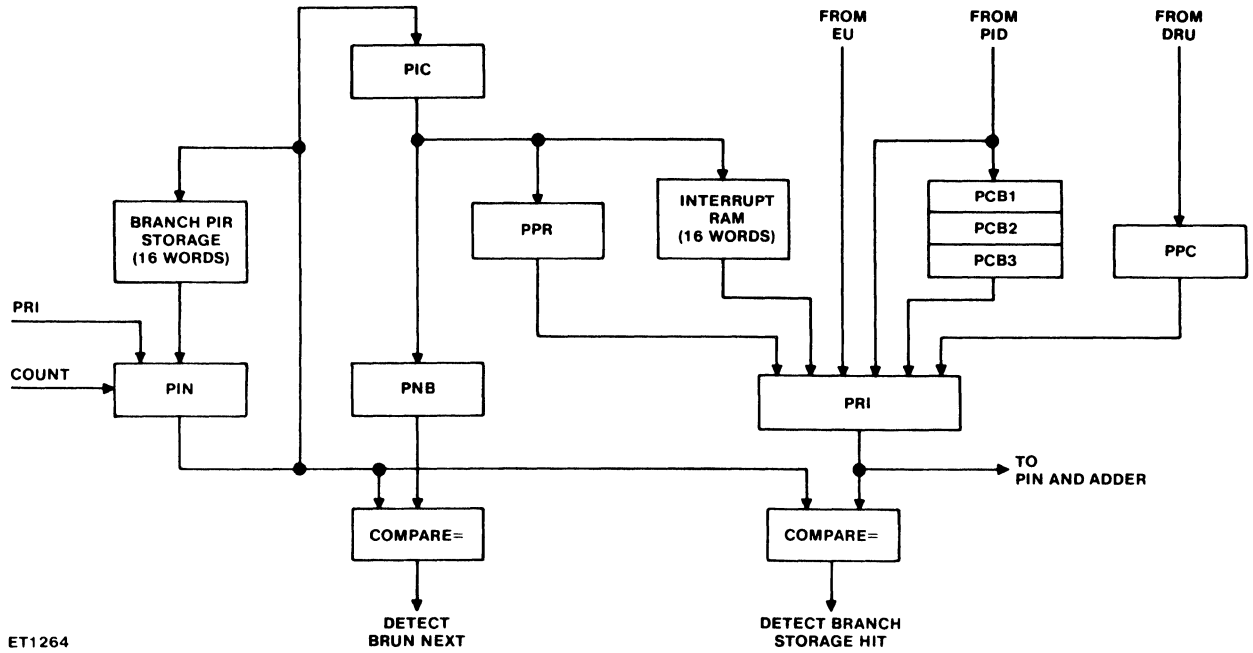


Figure 3-2-2. PIR Circuits

passed onto the PIE level. Instead, the address of the adder output (PRI + PBR) is transferred to the address registers while a local check is performed to determine if this address is pointing to code in the program buffer. When the required code is in the PID, a copy of the code in the program buffer read registers and PID register, the PIR in the PIN register, and the adder output address is written into the respective branch storages for later use.

The PIC (PIR current) register is used to maintain PIR of the first operator at the PIE level of the PCU. The most important function of PIC is to load PIR into the interrupt storage so that the PIR of that operator is available for interrupt processing.

The PIN (PIR next) register contains PIR of operator in the PID register. PIN is automatically updated by shift codes as part of the processing of each operator in the PID level. It is also the write input register for the branch storage and the syllable shift input for the barrel mechanism in the decode level.

The PNB (Next BRUN) register is used to maintain PIR of operator prior to the BRUN operator. This mechanism eliminates local check for program buffer code when BRUN loop is executed. For example, in a BRUN loop of VALC, LIT, LIT, COMP, BRTR, and BRUN, the PIR of BRTR is saved in PNB.

During the initial loop, the PIR of BRUN's target (in this example, the PIR of the VALC operator) is transferred from PID to PRI and then into the adder for normal branch operation. A copy of the code in the PID register and program buffer read registers, the PIR in the PIR register, and the address in the PAR register, is written into the respective branch storages for later use in processing the BRUN loop code.

The PIR value in the PIN register is then adjusted as each operator in the BRUN loop is preprocessed by the PCU. When adjusted, PIR compares with PIR in PNB. A BRUN next operation is set up to transfer branch storage information. The PIR value in PIN is compared with PIR value of BRTR in PNB. When PIN equals PNB, a flip-flop is set to denote a BRUN loop. BRUN loop is repeated again by comparing PIN with PRI. (PIN is read from the branch storage.) If equal, PRI branch is completed, branch storage code is used and operator processing is started.

In the example, the BRUN is exited when PCU has examined a boolean true (all 1 bits) for BRTR operator. This condition allows PCU to take a conditional branch instead of repeating BRUN loop again, as explained in "Preprocessing Conditional Branch Operator".

Preprocessing Conditional Branch Operator

The conditional branch boolean test logic, as shown in figure 3-2-3, is provided to allow conditional branches to be recognized by the PCU.

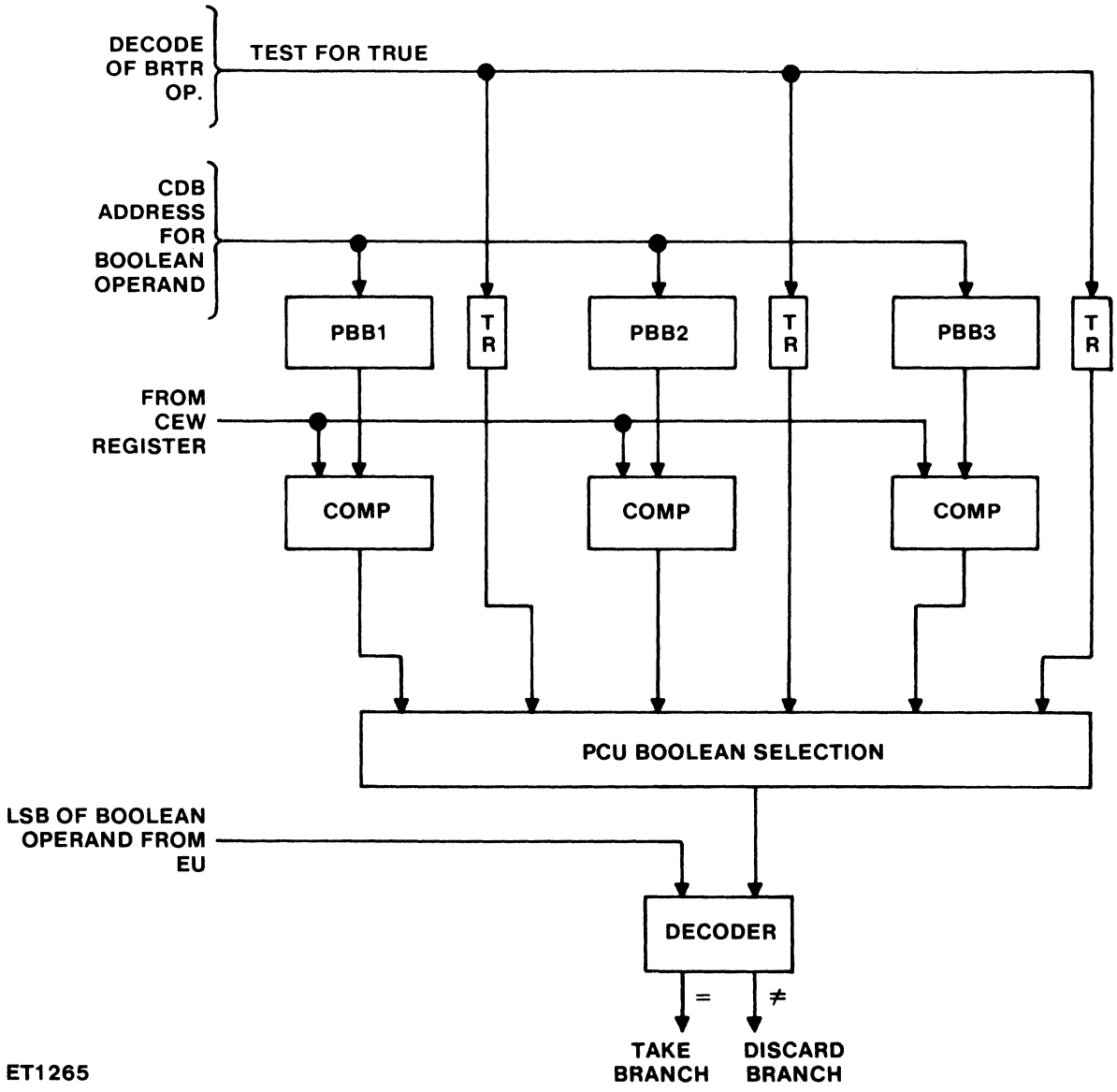


Figure 3-2-3. Conditional Branch Boolean Test Logic

When a conditional branch operator is first detected at PIE level of the PCU, the CDB address for the boolean operand is saved in a PBB register for later comparison with a value in the CDB EU write register (CEW). This register is loaded with a CDB address by the EU which processed the boolean operand.

The PBB registers are assigned to receive CDB addresses for boolean operands in the same order the PCB registers (figure 3-2-2) are assigned to receive PIR of corresponding branch operator. Controls for these assignments are derived from decodes of special valid and allocation bits in the PBB registers. The purpose for having three registers is to be able to accommodate more than one branch operation at a time.

When boolean operand is written back into the CDB by the EU, the LSB of that operand is monitored by the PCU. By comparing CEW value with PBB values, the proper TR bit value is selected for comparison with the LSB of the operand.

If TR bit value and LSB of the operand are equal, the branch is performed to obtain address of the new code string.

If these values are not equal, the branch is discarded and the PBB and PCB registers are invalidated and deallocated, respectively. (These registers can be assigned new values again.)

Address Registers

The address registers, shown in figure 3-2-4, are used for selecting program buffer locations during read and write operations. The address register contents are gated with other information from address associative memory (ASM) to develop address signals for the RAM chips of the program buffer. A description of each of the address registers is as follows:

1. PAR (program address register) contains absolute memory address of the next four to eight word group of code to be read from program buffer. The address in PAR is compared with associative

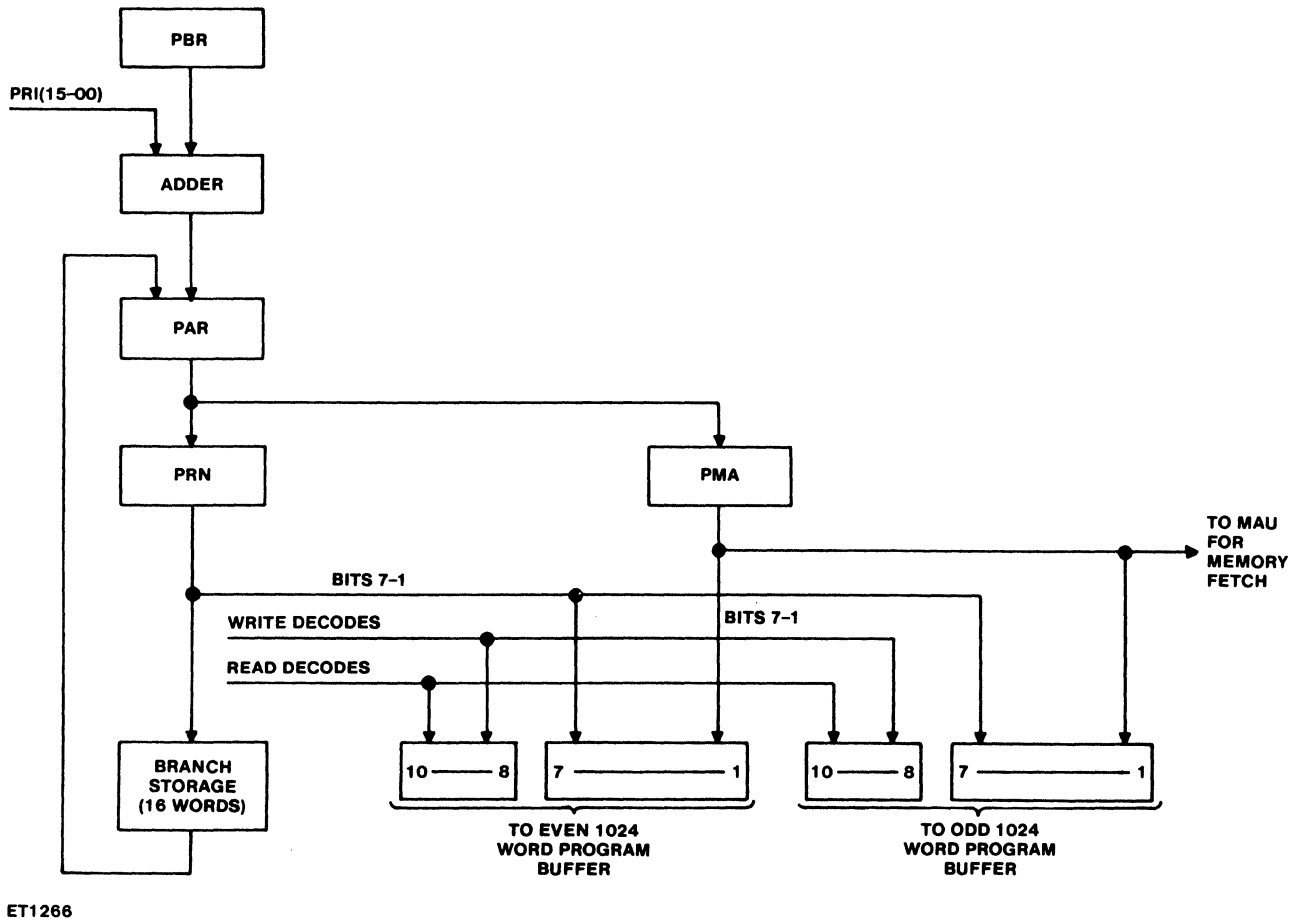


Figure 3-2-4. Address Registers

memory address to determine whether the next four words of code are local in the program buffer.

2. PRN (program read next) register contains absolute memory address of the next word of code to be read from the program buffer.

3. PMA (program memory address) register contains absolute memory address of the next word of code to be written into the program buffer.

Whenever a change of processing direction is executed, the absolute memory address for code is transferred from adder to PAR and then loaded into PRN. The adder is used to calculate absolute memory address by adding program base register (PBR) with PIR from PRI register.

For read address selection, the PRN contents (bits 7 through 1) are gated with the read decodes to develop address signals for the even and odd program buffers, each of which has 1024 word locations. The least significant bit (bit 0) of PRN defines which word from even or odd word buffer is the most significant word by setting a valid bit in the appropriate program read register. If PRN contains an even address (PRN bit 0 is low) after change of direction, both even and odd address locations are loaded simultaneously into their respective program read registers. As each word of code is loaded into the program read registers, the PRN count is increased by 1 so that successive locations can be read from the program buffer.

For write address selection, the PMA contents (bits 7 through 1) are gated with the write decodes to develop address signals for the even and odd program buffers. An address path from PAR to PMA exists to set up write address selection. The least significant bit (bit 0) of PMA defines which buffer is to receive the address signals. As each new word of code is written into the program buffer, the PMA count is increased by 1 to select the next address location in the program buffer. Controls for adjusting PMA count are derived from the MAU.

The decoding logic also provides the capability to read from one buffer and write into another buffer at the same time.

Address Associative Memory

The address associative memory (ASM) is the storage area for program code addresses. As shown in figure 3-2-5, RAM chips are used for this purpose. The ASM consists of eight groups of 4-by-4 array of 16-by-4 RAM chips. The implementation of these chips provides locations for 512 addresses, each of which represents four words of code in the program buffer.

The ASM also includes a 64-by-3-bit code priority list (PL) memory. These priority codes identify the program buffer location to be written to next.

When PRG1 or PRG2 needs to be loaded, the ASM and PL memory location to be read is selected by PAR bits 2 through 7. The eight selected ASM addresses are then compared with PAR bits 8 through 19 to determine if that address in PAR is local in the ASM.

If address is local, one of the eight comparator outputs is enabled and encoded into a three-bit code. This code, which identifies the group address (G0, G1, G2, etc.) is loaded into the program read group (PRG) register. PRG contents (bits 2 through 0) are then combined with PRN bits 7 through 1 to select proper program buffer location to be read.

If address is not local, the comparator outputs are low. A request for a new code fetch is sent to the MAU and the valid bit of program write group (PWG2) register is set. PWG2 contents are then passed onto PWG1 register, where output is combined with PMA bits 7 through 1 to select program buffer location to be written into by the MAU. A check is made to determine if the next four-word address is local in the ASM. To do this, the hardware increases PAR contents by 4 and compares new PAR contents with stored ASM addresses. If this address is not local, an eight-word code fetch is requested instead of four words from the MAU. An address path from PMA to PAR exists to restore PAR contents if address is not local.

After MAU transfers four words to the program buffer, the PL is updated by placing result of PWG plus 1 back into proper PL location. Also, the address of the four words is written into the proper ASM location for later comparison checks.

Program Buffer and Branch Storages

The program buffer provides local storage for up to 2000 words of the executing program's object code. Each word of code consists of six eight-bit syllables, and an odd parity bit. To ensure operator integrity, separate parity bits are generated on each syllable of code prior to entry into the buffer. The parity is checked as the syllables are used.

The program buffer is arranged in even and odd storage sections, each of which consists of 48 1024-by-1 RAM chips. Separate RAM chips are provided for storing parity and error information. The error information is encoded to set ER (error) and EV (error variant) bits as follows:

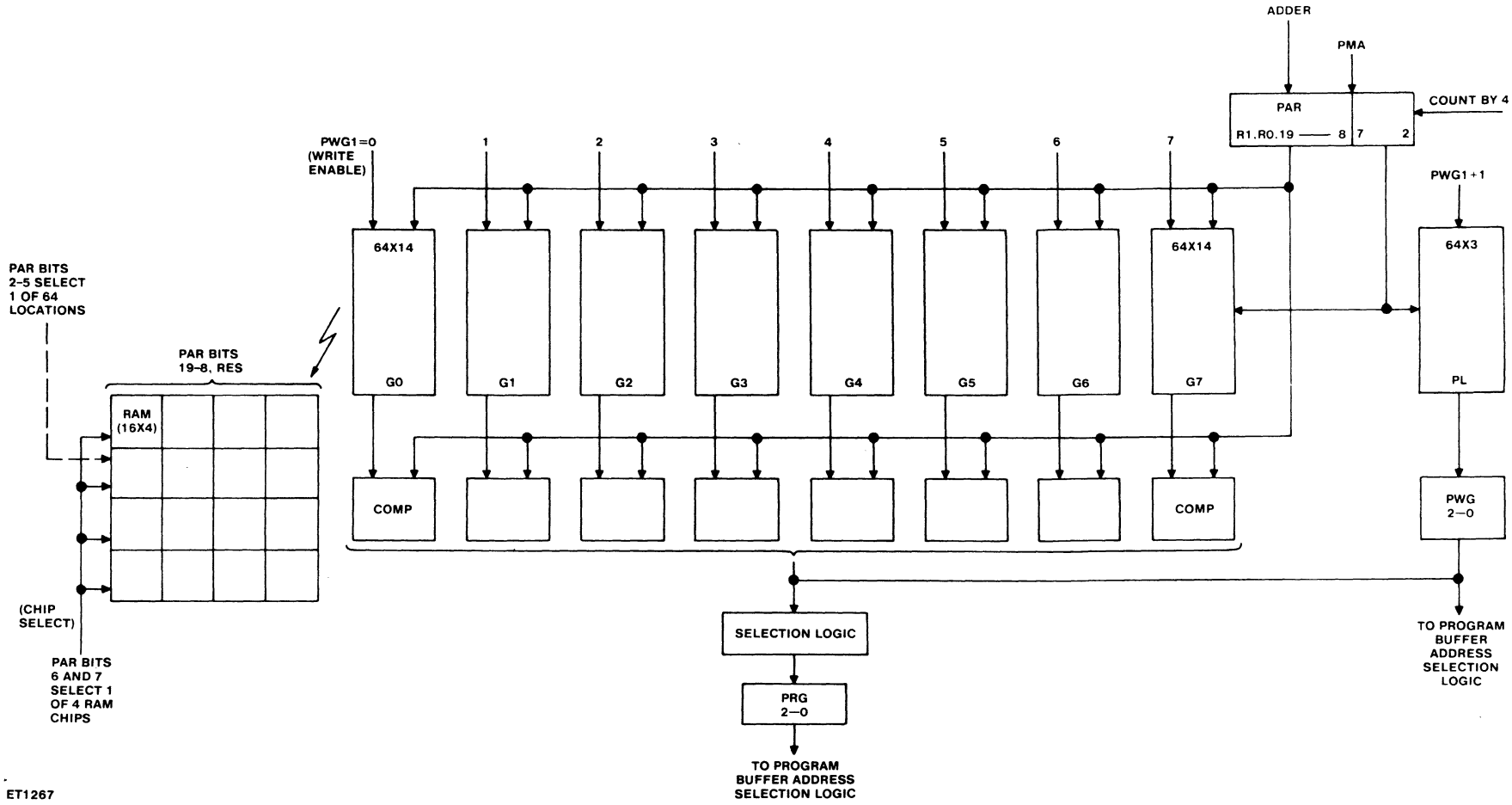


Figure 3-2-5. Address Associative Memory

ER	EV	Description
0	1	0 tag error if not in EDIT table mode
1	0	Tags other than 0 and 3
1	1	Error word from MAU

In addition to the program buffer, there are three branch storages, each of which provides local storage for up to 16 words of code, related errors, if any, and parity. These storages are used to save the code in PER, POR, and PID so that code is immediately accessible if branching back occurs. As a result, all three registers are replenished with code in one clock cycle. The manner in which these storages are accessed is controlled by the PIR circuitry.

Program Barrel

The program barrel is a shifting mechanism used for aligning and extracting the program operators from the two words of code read from the program read registers.

The program barrel has two stages, as shown in figure 3-2-6. The purpose of the first stage is to set up the next 48 bits in the PID. Both barrel stages are implemented with MFAN chips so that barrel can be used repeatedly during each clock cycle. No input selection gating is required. The output of the second barrel stage is routed directly to the PID register.

The first barrel stage shifts the outputs of POR and PER register to the left. It is capable of shifting these outputs in parallel in order to accommodate operators that overlap word boundaries. The shift amounts used by this stage are derived from PIN syllable bits. These bits are then encoded as addresses to the MFAN chips, to define the selected syllable shift for the barrel.

As shown in figure 3-2-6, the PER or POR selection decodes determine which 48 bits of the 96 bits are to be combined in the second barrel stage with PID bits. These decodes are derived from bit 0 of PRN and valid bits of PER and POR.

The second barrel stage shifts (left-justified) the next six syllables to be loaded into the PID register, as the current syllable or syllables are being sent to PIE level for execution. To load PID, both POR and PER must contain valid code. The shift amounts are derived by recognizing the type of instruction to be executed, such as one-syllable and two-syllable instructions. Like the first barrel stage, the shift signals are applied as addresses to the MFAN chips to define the selected six syllable shift for the barrel.

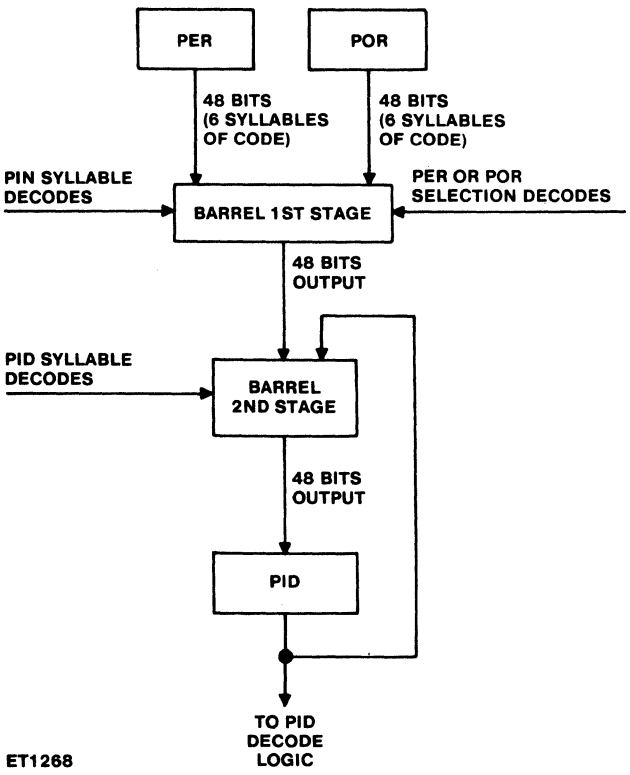


Figure 3-2-6. Program Barrel

PIE Level Registers

All operators loaded from PID level to PIE level are routed to one of four operator registers. These registers and their functions are as follows:

1. PCU stack only (PSO) register allows PCU to independently handle the operators used for adjusting items in the top of stack and to copy or delete the top of stack items within the CDB. These operators are the LIT48, MPCW, DUPL, EXCH, RSDN, RSUP, LITS, and some special pseudo lits, such as FET PSX for moving P2 parameter information within the CDB.

2. Program instruction execute (PIE) register receives all operator decodes from PID, excluding NAMC, VALC, and stack operators, and forwards them as nine-bit addresses to a network of high-speed 256-by-4-bit PROMS. The PROMS issue the required micro operator sequences and any additional information to the appropriate write registers in the PCU. Other inputs to PIE register are:

   a. LIT, VALC, and NAMC grouping logic used to facilitate preprocessing of specific groups of program operators.
   b. Next PROM address used, if more than one state of a PROM is required for passing micro operators to the write level.
   c. PCU restart used to process interrupts.

3. PCU name call (PNC) register temporarily holds the 14-bit address couple associated with the NAMC operator.
4. PCU value call (PVC) register temporarily holds the 14-bit address couple associated with the VALC operator.

The purpose of the execute level is to process as many operators in parallel as possible for PCU, DRU, and EU. At any one time, as many as three of the four operator registers can contain valid information. The combinations of the operator registers which can be loaded simultaneously are as follows:

1. PVC, PSO (only LIT).
2. PVC, PSO (only LIT), PIE.
3. PVC, PNC.
4. PVC, PNC, PIE.
5. PVC, PIE.
6. PSO (only LIT), PVC.
7. PSO (only LIT), PVC, PIE.
8. PSO (only LIT), PNC.
9. PSO (only LIT), PNC, PIE.
10. PSO, PIE.
11. PNC, PIE.

In addition to these four operator registers in the execute level, a PCU lit (PLT) register is provided to buffer certain information prior to distribution to the write level. This information includes the LT8 and LT16 data, the start and length values of bit and field operators, the scale values of scale operators, and the length and insert character parameters of edit operators.

The PID syllable's paths to the PLT, PNC, and PVC registers are arranged so that any one of the four sets of PID syllables can be loaded in the appropriate registers. For example: a LIT VALC sequence in which LIT transfers to PLT through syllable paths 1 and 2, and VALC transfers to PVC through syllable paths 3 and 4. Controls for loading these registers are derived from decode logic in PID.

To facilitate preprocessing of VALC and NAMC operators, the PCU must predict how DRU will handle address couple information.

For NAMC STOR, the address couple of the most recent STON or STOD is saved in PNR for comparison with subsequent VALC.

If a compare is made, the VALC address couple is not issued to the DRU; instead, the PCU performs the VALC. PCU obtains the CDB location of the data input to that store whose address couple was saved and places it in the PCU top of stack mechanism.

For VALC not associated with a NAMC STOR or concatenated NAMCs, the address couples are queued in the DRU for evaluation. NAMC is concatenated when the next operator in the code string is any of the following: INDX, NXLN, NXLV, STOD, STON, OVRD, OVRN, DBUN, LOAD, and PLDT.

In the case of concatenated index operators, the PCU passes additional code along with the address couple to the DRU. This code is sent with a second input to the index operator. The first input is the address couple result location. The second input allows the DRU to read the index from an assigned CDB location, when descriptor indexing is performed in the DRU.

When a non-concatenated NAMC is detected, the address couple in the NAMC is placed in the central data buffer (CDB) as an IRW for DRU use. To identify address couple as an IRW in a CDB location, a flag bit (bit 48) in that location is set.

For MKST NAMC and NAMC DBUN operator cases, the address couple is not queued in the DRU; it is immediately processed by DRU to speed-up references to PCWs.

As shown in figure 3-2-1, two special paths from the LIT PROM in the PIE level to the variant registers (PQEV) and write PLW registers exist to process processor register operations (SPRR and RPRR) for the EU and DRU. The input register for the LIT PROM is the LIT PROM address register (PLA). The purpose of PLA is to buffer read and set processor register address obtained from the EU via the CDB and read processor register address received from the PLT register. The read processor register is handled as a lit where the address for the selected processor register to be read is contained in the previous LIT operator:

By use of the LIT PROM, the processor register address is converted directly to an internal hardware address if processor register is accessed by the DRU or to a variant code if processor register is accessed by the EU.

The PQEV register receives the variant code from the LIT PROM and passes it to the EU operator queue. The variant code is then decoded in the EU to provide information for selection of a desired processor register. Registers selected by the variant codes are as follows:

Variant Codes	Register Usage
0	Program Base Register (PBR)
4	Processor Fail Register (IPF)
5	Control Mode Register (XCM)
6	Egg Timer (EGG)
8	Interrupt Identifier (ID)
9	Maintenance Processor Register (MDP)
A	Interrupt Fault Mask Register (IFM)
C	Interrupt Fault Register (IFR)
E	Interval Timer (INT)
F	Time of Day (ITD)

The PLW register receives the internal hardware address from the LIT PROM and passes it to the CDB for DRU use. Registers selected by the internal hardware address are as follows:

Internal Hardware Addresses	Register Usage
00-1F	Display Registers D[X]
20	Base of Stack (BOS)
21	Last D[1] used as SDI Base (LD1)
22	Scratch (S1L)
24	Source Index (SIR)
25	Destination Index (DIR)
26	Table Index (TIR)
28	Source Base Register (SBR)
29	Destination Base Register (DBR)
2A	Table Base Register (TBR)
2B	Scratch (S2L)
2C	Alternate D[0] Register (ADZ)
2D	Alternate Program Index Register (APIR)
41	S register (top of stack address)
42	F register (most recent MSCW address)
44	Limit of Stack (LOS)
48	Segment Descriptor Index (SDI)
50	Program Base Register (PBR)
60	Current Stack Vector Index (SNR)

Write Level Register

All micro operators issued are placed in the PCU output registers PQA, PQT, PQE and PQO. The contents of these registers are then written into the address couple queue (ACQ) and top of stack queue (TOSQ) in the DRU and the operator queue (OPQ) in the EU. The PCU output registers are loaded directly from the PROM network, as shown in figure 3-2-1, but some special input codes are also ORed with the output of these registers, when a deviation in the normal code flow is required.

Description of other input codes and their description is provided in table 3-2-1.

Table 3-2-1. Special Input Codes to DRU and EU

Code	Destination	Description
VALC	AC Q	Instructs DRU to mask LL field of address couple and to pass remaining index field as one of the inputs to an adder. The other input is the base address contained in the display register. The adder output is the absolute memory address described by the address couple.
FETSTK	AC Q	Instructs DRU to fetch a value from ASM location pointed to by S register and place it in an allocated location in the CDB. This code is issued if PIE level operator DUPL, EXCH, RSDN, or RSUP or fetch stack adjustment does not have sufficient input for execution.
STST	EU OP Q (for EUCDB)	Instructs EU to read B item from the CDB to the storage queue. Addressing of storage queue is controlled by the DRU and is accomplished by use of S + 1 register. Update of S + 1 register and storage queue pointer is performed by the DRU.

MOVE	EU OP Q (for EUCDB)	Instructs EU to move B, if B is not a reference word in a DRU CDB location. Also used to move boolean A for BRTR/BRFL. (See note below.)
MOVB	TOS Q	Instructs DRU to move B, if B is a reference word in a DRU CDB location. (See note below.)
		NOTE Because of deallocation mechanism in the PCU, if DUPL is not paired with a PIE operator, a pseudo PIE operator, called MOVE or MOVE-B is forced, which replaces B output of DUPL with a new copy of the duplicated data. Operands and EU results are moved by the EU, but DRU created references are moved by the DRU to avoid having the DRU wait for the EU.
CRLE	EU OP Q (for SAU)	Instructs EU to create a length count of all ones. This code is issued if PIE operators Transfer While True or False, Transfer While Compare, Scan While True or False, and Scan While Compare have a descriptor input instead of length count in the second word of stack.

The ACQ is bypassed if a quick fetch condition is recognized by the PCU. In such cases, PQA directly loads the address couple (RAC) register in the DRU for quick fetches. A quick fetch comes from the decoding of special operator cases. The operator cases are given in table 3-2-2.

Table 3-2-2. Operator Cases for Loading Code Into RAC Register

Operator Cases	DRU Action
MKNAMC (mark stack name)	Fetch PCW to obtain a new code segment.
RETNCVC (return continue VALC)	Fetch current RCW referenced by DLL + 1.
EXIT and RETN	Fetch current RCW referenced by DLL+1.

Issued along with each micro operator are CDB addresses, job number, and if necessary, variant and type bit information. In the case of literals, data is issued. Variant information provided by the PCU is loaded into the EU OP queue.

In special cases, literal data supplied to the CDB is also loaded into EU OP queue. For example, in the execution of VALC, LIT, ADD or LIT, VALC, or ADD, VALC results will be from the CDB and literal data will be received from the EU OP queue provided parallel inputs for the ADD operation in the EU.

As stated previously, the job number remains with the micro operator throughout the execution of the micro operator. To accomplish this, the job number is transferred to the respective unit (DRU, EU, or EU store sub unit) at the same time as the associated micro operator is transferred from the PCU write to the respective unit. This job number passes through each level of the respective unit until completion at the write level. At this time, the job complete bit for that job is set.

If an interrupt or restart occurs while the job is in process, the job number is loaded into a job interrupt register. When that job is declared the oldest job in process, the interrupt operation is executed.

Top of Stack Location Registers

The top of stack location registers contain CDB location addresses and transfer inputs by way of a bus to write level registers, as shown in figure 3-2-1. The purpose of these top of stack registers and associated controls is to maintain the top of stack locations and adjust them when operators are executed. The write level registers are used to write the proper CDB locations into the DRU and EU queues for the A, B, and D inputs of the operator being written to the queues. A write level register also contains the CDB location into which the result of the operation is to be placed.

The top-of-stack registers are controlled by the register-to-bus and bus-to-register transfer signals. The register-to-bus transfers are produced by the types of operators at the PIE level: NAMC, VALC, or stack only operator. The bus-to-register transfers are developed by the stack count and PIE delete, input, and type registers.

The major inputs to the bus are the outputs of the top of stack registers A through D. Two inputs originate in the allocation logic (described later in this section): 1) the PPA input from the PCU allocation circuits; and 2) the PRA input from the DRU allocation circuits. The PSX input is present when a com-

parison is made between PNR and PVC (explained in "PIE Level Registers"). The location of the NAMC STOR data that was stored in PSX is loaded into the top of stack registers via the bus so that VALC is not executed by the DRU. The outputs of the bus are sent to the top of stack registers at both the PIE and write levels.

PCU Allocation and Deallocation of CDB Locations

Three main allocation registers and deallocation queues are used, one each for the PCU, DRU, and EU. Allocation is accomplished by the setting of a bit in any of the allocation registers. The decode that sets an allocation flip-flop is also used to generate a six-bit hexadecimal number that can be used as a CDB address, if specified. The CDB address is maintained in one of the top of stack registers (A, B, C, or D), which were described previously. This address is used to address the top of stack data.

After locations have been allocated, they must be deallocated when the data contained in those locations are used. The deallocation process is accomplished by use of deallocation queues and their associated control registers, as shown in figure 3-2-1.

Allocation Paths

The allocation registers are used in the allocation of CDB locations for the PCU, DRU, and EU. The highest priority within the PCU and EU allocation registers is assigned to bit 11. The highest priority within the DRU allocation register is assigned to bit 15 because of the 16 locations in the DRU portion of the CDB. The encode of the next register bit to be used is decoded by a DFAN chip to set the selected register bit. (Note that if, for example, bits 8 through 10 of the register are set but bit 11 is reset, the next bit used is bit 11.) The encode is also sent to the top of stack location registers and sets a CDB address that is equal to the hexadecimal value of the bits set in the allocation register.

There are two special allocation paths within the PCU: the PBL and the PWA path. The PBL input to the encode logic is used to allocate one of four big lit locations in the CDB. The PBL register is a two-bit up counter that counts to binary three and then returns to zero. The encode output is sent to the write pointer for the PCU locations of the CDB. The PWA input to the PCU encode logic is used primarily for the reallocation of CDB locations following a PCU restart operation.

Deallocation Paths

As shown in figure 3-2-1, the deallocation paths for the PCU, DRU, and EU consist of similar logic circuits that contain deallocation queues, deallocation registers, and decoders.

The purpose of each of the three deallocation queues is to store the CDB addresses that have been allocated and to also store a valid bit, which is used to indicate that the corresponding location is to be deallocated. When a job is completed, the location that was used to supply input must be deleted from the allocation registers. By reading the queue, the PCU resets the allocation flip-flops of obsolete CDB locations.

A read pointer (JQR) and a write pointer (JQW) are applied to each queue, and the data inputs are PIE write level registers (PWA, PWB, and PWD) with their associated valid bits. In addition, the job end bit is written into the queue. This bit is used to set a flip-flop, allowing the deallocation job (JDJ) counter to count. The outputs of the queues are sent to deallocation registers.

A-deallocation (A input), B-deallocation (B input), and D-deallocation (C input or PSX) are the three registers used in the deallocation process. These registers each contain seven bits, composed of a valid bit and six bits of CDB address. These registers are loaded when the associated deallocation queue is read, provided three conditions are met: 1) the queue is not empty; 2) the oldest job is not equal to the deallocation job; and 3) a restart has not just been performed.

PCU Job Number Registers

The PCU job number registers (shown in figure 3-2-7) contain the job numbers of the operators or groups of operators issued by the PCU. Up to 16 job numbers can be active simultaneously, although the job registers can handle hexadecimal values 00 through 1F. The extra bit in these registers is used to indicate empty and full conditions. When the PCU execute level job number register (PEJ) contains hexadecimal 10 and the oldest job register (JOJ) contains hexadecimal 00, there are 16 jobs in process (full condition). When both of these registers contain hexadecimal 00, no jobs are in process (empty condition). The JCJ register is used to identify, by number, those jobs that have been completed. If, for example, job number 12 were completed, JCJ bit 12 would be set by a hexadecimal value of either 0C or 1C; however, both of these numbers would not be active simultaneously, because they are 16 numbers apart. In the job number registers, numbers are assigned from 00 through 0F hex; the fifth bit is then set, and job numbers are then assigned from 10

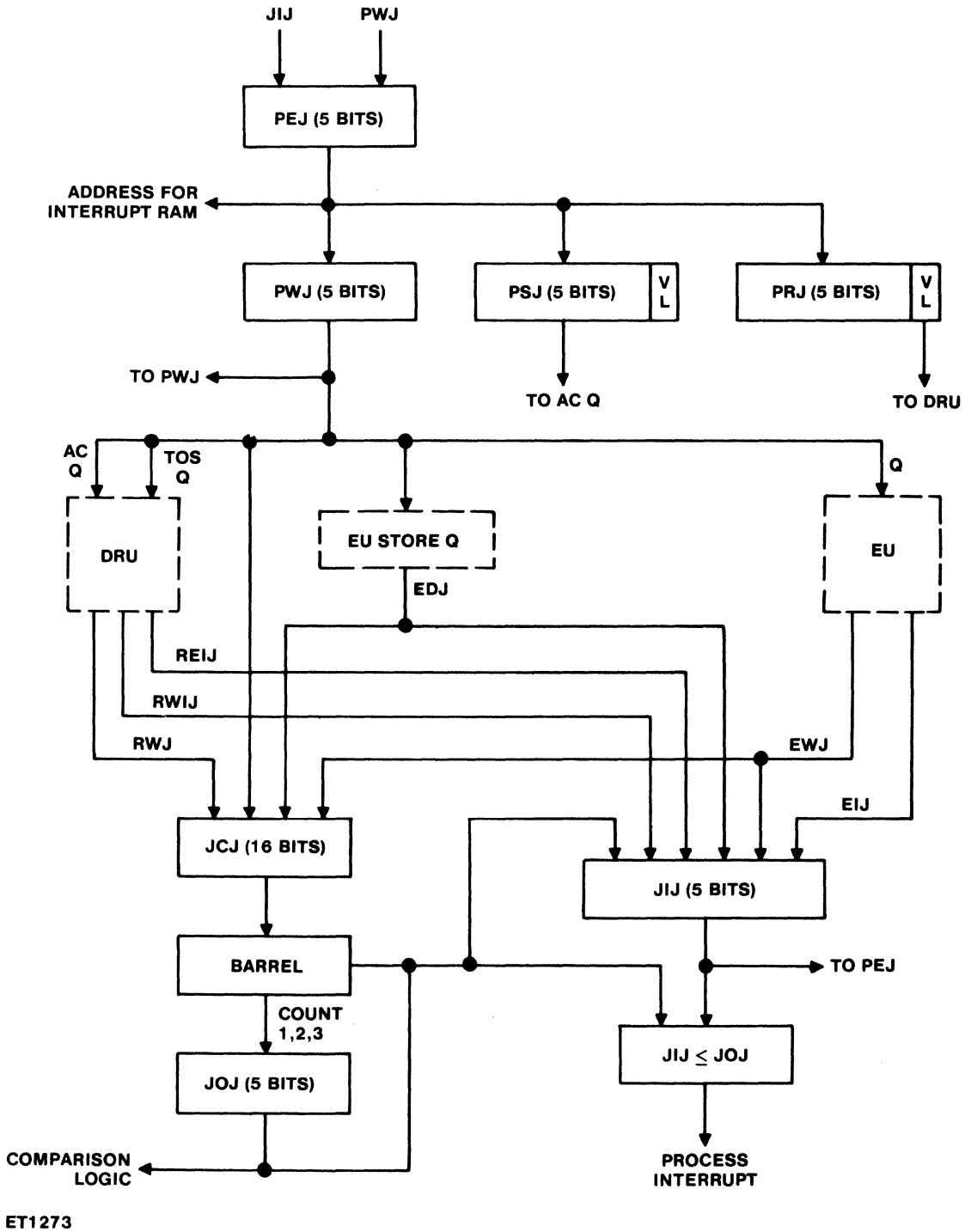


Figure 3-2-7. PCU Job Number Registers and Logic Paths

through 1F hex. Following this sequence of number assignments, the register sequence begins again with the value 00. The job number registers are as follows:

1. PCU execute level job number register (PEJ) contains the job number of the job at the execute level of the PCU. PEJ register can also receive inputs from the JIJ and PWJ registers for addressing the interrupt RAM, which contains the state of the stack for each job number. This stack information is used during a restart or for interrupt cases.

2. PCU write level job number register (PWJ) contains the job number at the write level of the PCU. This register is loaded from the PEJ register on the occurrence of each clock, unless an error occurs in the PIE level of the PCU, or it is in a holding condition.

3. PCU stack job number register (PSJ) contains the job number of the last store to stack (STST) or fetch to stack (FEST) micro operator performed by the PCU. The PSJ register is loaded from the PEJ register when the PIE level of the PCU issues an STST or FEST, at which time the valid flip-flop is set. For FEST the previous content of PSJ is written into the address couple queue (AC Q), along with the valid bit. This information is used by the DRU to ensure that any older STST or FEST operation is complete before the current FEST is executed. The valid (VL) flip-flop of the PSJ is reset when the number of the oldest job (in JOJ) is greater than the job number in the PSJ register.

4. PCU protected job number register (PRJ) contains a five-bit job number and a valid bit. When a valid bit is set, the register contains the number of the last job to perform either a fetch to stack (pop) or an overwrite operation. Under these conditions (with the valid bit set), the DRU is inhibited from performing a quick fetch of a program control word (PCW) or a return control word (RCW). The valid flip-flop of the PRJ is reset whenever the number of the oldest job (in JOJ) is greater than the job number in the PRJ register.

5. PCU job complete register (JCJ) contains 16 bits, which correspond to the job numbers of completed jobs. A job complete bit within the register is set by a decode of write level registers PWJ, RWJ, and EWJ of the PCU, DRU, and EU, respectively. The decoding logic is enabled when the job end bit and the job number reach the write level of the particular unit. The JCJ bits are reset from a decode of the JDJ (deallocate job) register, which is discussed in the description of PCU allocation of CDB locations, presented earlier in this section.

6. PCU oldest job register (JOJ) contains five bits that represent the job number of the oldest job in process. This register functions as a counter; it is incremented by the configuration of job complete bits in the JCJ register. The job complete bits are applied to a four-stage barrel and the barrel output is multiplexed to determine whether the JOJ is to be incremented by 1, 2, or 3. The increment used depends on which bit positions of the JCJ register are set. Within the JCJ register, set bits represent jobs that have been completed but not yet deallocated.

7. PCU interrupt job register (JIJ) contains five bits and an associated valid bit. The content of this register represents the number of the job that has been restarted or interrupted; the job number is loaded from one of registers REIJ, RWIJ, EDJ, EWJ, EIJ, or JOJ. When the job number is loaded, the valid bit is set but the interrupt is not processed until the value in the JIJ register is less than or equal to the value in the JOJ register. This comparison is performed to ensure that the job that was interrupted or being restarted is the oldest job; there are no older jobs still in process. If an interrupt or restart is issued by two units simultaneously, a priority scheme determines order of JIJ load. Any value in JIJ can be replaced by an older number.

Each micro operator generated by the PCU is assigned a job number, which is produced in the PEJ register. This register functions as a five-bit counter, and the value of its contents is normally displaced less than 16 from the content of the oldest job register (JOJ). If the displacement becomes 16, the PIE level is placed in a holding condition.

When a micro operator is generated at the PIE level and sent to the write level, the job number is sent from the PEJ register to the write level of the PCU. A job end bit is used to indicate to the receiving unit that when the micro operator is passed to the write level of that unit, a job complete bit is to be set for the associated job number. (For multistate PIE operators, only the last micro operator is assigned a job end bit.) From the write level of the PCU, the job number and its corresponding job end bit are written to one of four queues: 1) the EU queue; 2) EU store queue; 3) top of stack queue; or 4) the address couple queue. The job number then passes through each level of the unit to which it is written until it reaches the write level, where the respective job complete bit is set in the JCJ register. If the job was the oldest job in process, the oldest job register (JOJ) is incremented as necessary to reflect the current oldest job.

If an interrupt or restart occurs while a job is in process, the associated job number is placed into the interrupt job register (JIJ). When that job became the oldest job, the interrupt would then be processed.

DATA REFERENCE UNIT

The data reference unit (DRU) is queue-driven. The program control unit places all operations into the queues of the DRU, excluding some queue bypassed operations and operator associated information. These queues are identified as address couple queue (ACQ) and top of stack queue (TSQ). (See figure 3-2-8.) The DRU contains five distinct pipeline levels:

1. An operator level.
2. A routine level.
3. An evaluate level.
4. A compare level.
5. A write level.

The purpose of DRU input queues and pipeline levels is described in the following paragraphs.

Address Couple Queue

The address couple queue (ACQ) is the storage area for address couple operators pending evaluate level processing. The ACQ contains eight locations. The following types of information can be loaded into the queue and then read into the address couple (RAC) register:

1. JE (bit 51) indicates that the micro operator is the last in a program sequence.
2. JP (bit 50) identifies parity for job number bits in RAC.
3. JOB # (bits 49:5) allows DRU to keep track of an operation through various pipeline levels so that the operation can be recovered if an interrupt occurs. It also informs the PCU when the operation is completed.
4. RP (bit 44) identifies parity for result location bits in RAC.
5. RESULT LOC (bits 42:4) identifies CDB location address for DRU fetched data.
6. P (bit 22) identifies parity for address couple bits in RAC.
7. Type bits (bits 17:4) define special hold conditions and display buffer read operations. These type bits are defined in table 3-2-3.
8. SY (bit 13) allows address couple and read display buffer information to appear simultaneously at the DRU evaluate level.
9. MR (bit 12) indicates that the operation is preceded by a force main memory (FMMR) job; consequently, local storage is not checked or modified by the DRU. Instead, the operation is passed immediately to MAU for memory transfer.
10. OP (bit 8) identifies parity for operator bits in RAC.
11. OPERATOR (bits 7:8) identifies operator code.
12. Address couple (bits 13:14), when evaluated, provides an absolute address from which data information is fetched and placed in CDB result location for EU use.

The input of operator related information to the RAC register is through the ACQ, unless the queue is empty and the DRU is waiting for work, or a PCU quick fetch operation exists, in which case, the queue is bypassed.

Table 3-2-3. Type Bit Codes in RAC Register

RAC Code	Description
2	Hold for last push or pop.
6	Hold for a valid mark stack control word.
8	Read BOSR.
1	Read display buffer addressed by RAC bits 13:5.
3	Read display buffer addressed by XLL.
4	Read D1.
7	If IRW-LL is less than XLL read XLC; else read BOSR.
5	If RM47, read SILS(22); else read BOSR.

The contents of the write pointer (RAWP) provide the write address. To access the next location in the queue, the write pointer is increased by 1, provided that the write signal is received from the PCU and there is no PCU hold.

The queue is read by the DRU. The contents of the read pointer (RAPP) provide the read address.

Top of Stack Queue

The top of stack queue (TOS Q) is the storage area for top of stack operators pending operator level processing. Like the AC Q, the TOS Q contains eight locations. The types of information which can be loaded into the queue and then read into the operator (RO) register is similar to the AC Q. This information includes job number, A, B, and R-location addresses, type bits, memory reference bit, and operator code.

Writing into the TOS Q is performed by the PCU. Reading from the queue is accomplished by use of the TOS Q read pointer RTR in the DRU. The operator read from the queue is the instruction for the operator level. The address of TOS Q location from which the operator is read is also provided with each operator sent to the operator and routine levels. Usually the address is not used. However, when the

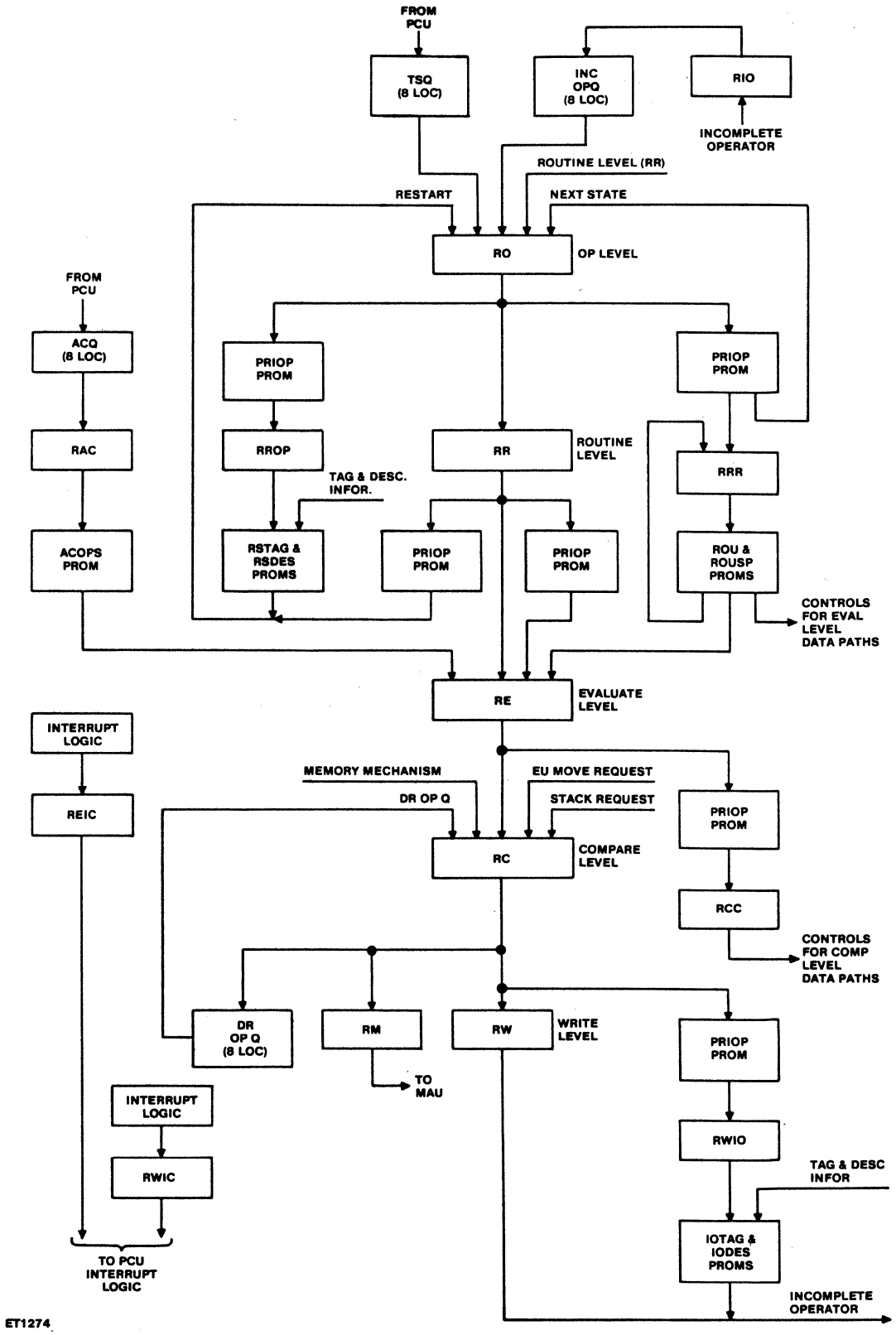


Figure 3-2-8. DRU Control Pipeline

DRU restarts an operator level operator or a routine level operator, the current RTR value is temporarily saved in TOS Q save pointer RTS. The address from the operator or routine level is loaded into RTR to provide the TOS Q location associated with the operator being re-executed. When queue is read, the contents of RTS are loaded back into RTR so that DRU may continue normal processing.

DRU Control Pipeline

As shown in figure 3-2-8, the DRU control pipeline contains: 1) five operator registers (RO, RR, RE, RC, and RW); 2) incomplete operator queue (INC OP Q); 3) deferred reference queue (DR OP Q); 4) RM mechanism control; and 5) a network of 256-by-4 PROMs.

The operator registers are the primary control registers of DRU. These registers provide the input for the DRU pipeline levels. The inputs include variant information, type bits, and an eight-bit code for addressing PROMS.

The 256-by-4 PROMs that comprise the network contained in the address couple path provide controls for passing address couple data to proper data paths in the evaluate level and to issue evaluate level operators, as eight-bit codes, to the RE register. These operators are used to inform evaluate level what to do with the address couple data.

In the pipeline, the PROM network provides operating sequences to control processing in the data and address circuits of the evaluate, compare, and write levels of the DRU. Two major PROM networks are associated with the pipeline: the primary operator PROMs (PRIOP) and routine PROMs (ROU).

The primary operator PROMs are addressed by RO, RR, RE, and RC registers, which receive inputs from various areas of the DRU. Consequently, these PROMs are identified as operator level PROMs, routine level PROMs, evaluate PROMs, and compare level PROMs. As described previously, the DRU does not operate on basic program operators, but rather it processes micro operators forwarded to it by the PCU. When a micro operator is read from the TOS Q, it is entered into the RO register. The micro operator forms an eight-bit address that is applied to the operator level PROMs and the first of a sequence of operations is generated (in some cases, the OP level PROM output sequence). If PROM output is not fed back, the OP level is available for a new operator from the TOS Q or an operator from the INC OP Q. Excluding a WAITIO operator, the OP level PROM output for a new operator always issues a routine level operator to the RRR register.

These operators form eight-bit addresses that are applied to the routine (ROU) PROMs. In turn, these PROMs produce operating sequences, such as selecting data onto the P (primary) and X buses, loading RBA and RIA registers to obtain address for DRU fetch, and setting up string and parameter information for EU processing.

Execution of a particular ROU level operator may be accomplished entirely by the ROU PROMs, but in some cases, an evaluate operator is issued to the RE register for further pipeline processing. For certain string operations, such as source and destination word fetches, the ROU PROMs make use of the routine special (ROUSP) PROMs to fill the ASM if the data is not local.

Various special paths into the RO register exist to process restart, incomplete, and routine level operator cases.

When original input for non concatenated index-type operators (INDX, NXLN, and NXLV operators not preceded by NAMC) is received, the input is decoded by use of restart tag (RSTAG) PROMs. Any unexpected input, such as 0, 2, 3, 4, 6, or 7, results in an invalid operand (INV OP) interrupt being sent to the RO register. This interrupt is eventually transferred to PCU interrupt logic via evaluate interrupt code (REIC) register. The remaining valid case is an IRW for a descriptor being fetched, where an appropriate code is fed back to the RO register as a restart operator. For example, in the INDX case, a hex code of 31 is loaded into RO register as the address for a new PRIOP PROM sequence. The sequence is used to evaluate an IRW which eventually finds a descriptor. The processing of initial inputs for other program operators is the same as in index-type operators. These program operators are as follows:

1. DBUN (dynamic branch).
2. EVAL (evaluate descriptor).
3. LOAD.
4. LODT (load transparent).
5. OVRD (overwrite destructive).
6. OVRN (overwrite non-destructive).
7. RDLK (read with lock).
8. SNGT (set to single).
9. STBR (step and branch).
10. STOR (store).
11. VALC (value call).

There is one other set of PROMs in the restart path called the restart descriptor (RSDES) PROMs. The RSDES PROMs are used in the same manner as RSTAG PROMs. If any unexpected descriptor input is received for an operator, the INV OP is sent back to the RO register for PCU interrupt processing.

Otherwise, an appropriate code is returned to RO for restart operation. The program operators that can be handled by RSDES PROMs are as follows:

1. EVAL.
2. INDX.
3. LOAD.
4. NXLN.
5. NXLV.
6. OVRD.
7. OVRN.
8. STBR.
9. STOR.
10. VALC.

The incomplete operator path supplies incomplete operators in which the operator input requires further evaluation (or chaining) until a target is found. These operators are temporarily stored in the INC OP Q by IO PROM decodes (described later in the pipeline discussion).

The RR register receives the OP level code from RO register. For some operators, code is decoded by use of a PRIOP PROM and fed back to RO for restart operations. For example, a restart code for translate (TRNS) is always developed by the PROM, but is only validated if the third input for TRNS is a source descriptor instead of an operand which is the source string. In this case, restart code sets up the routine level to handle source descriptor evaluation. During interrupt processing, RR code is provided by PRIOP PROMs and then routed as interrupt code through RE and REIC registers to PCU interrupt logic.

When the RE register contains an operator from the various pipeline inputs, the operator forms an eight-bit address that is applied to the evaluate level PROMs. The PROM output consists of a five-bit code and is loaded into the compare command (RCC) register. The information in RCC is decoded to provide commands for data and address circuitry in the compare level. RE also serves as an input to the RC register.

In addition to receiving code from RE, the RC register receives code as the result of requests for use of the compare level by operations, such as memory mechanism (RM), deferred operator queue, or for move and stack operators from the EU.

If an MAU job is started by the DRU, the job remains in RM and waits for data from the MAU. When MAU receives data from memory, the MAU interrupts the DRU, obtains control of the compare level, and sends all data to RMD for input to ASM. At the same time, the three LSBs of address from the MAU are compared with the three LSBs of address in the RM. When a compare occurs, the next word coming from the MAU is the word required by the DRU for evaluation. The job in the RM is then activated again and returned to the compare level. The difference between this job and the original job sent down the pipeline is in the compare level command. The command for the original job is to fetch data, whereas the command for the activated job is to transfer the next word in the RMD register to the RWD register for input to CDB and to activate the incomplete operator tag (IOTAG) and the incomplete operator descriptor (IODES) PROMs for possible chaining action. The functional operation of these PROMs is identical to the restart PROMs. If chaining is required, the activated job is placed in the INC OP Q as an incomplete operator.

A second MAU job can be queued in RM as long as its address does not agree with the address of a previous MAU job. If the addresses are the same, the second MAU job is placed in the deferred operator queue (DROPQ). When the job is read into compare level, the job is local in the ASM.

Requests for use of the compare level by the EU occurs when EU is required to write data from one location to another location within the CDB (called by MOVE micro operator from PCU) and to read B item from the CDB to the store queue (called by STST micro operator from PCU).

In all cases, an evaluate level hold is temporarily enabled when a compare level request is granted.

DRU Data and Address Paths

Most of the data paths into and from the evaluate level are controlled by the routine PROMs (RROU). Other inputs into this level can be controlled by the address couple operators (RACOPS).

The evaluation of an operator begins with reading in the input for the operator (figure 3-2-9). This consists of gating an appropriate source onto the primary (P) and auxiliary (X) buses and then routing them to evaluate level registers. P bus inputs are usually the contents of some CDB location. P bus inputs can also be the address adder output, various state registers, evaluate or write level data, or literal values. X bus inputs include contents of CDB locations, address adder output, computed index values, stack number, saved string lengths, write level data, and EU information for table word operators.

These P and X buses provide inputs to RED and REA (normal destination for CDE outputs), RBA and RIA (address adder base and index inputs), and REL (used for keeping track of lengths in string operators). The main function of RBA and RIA is to

hold values to be added in the address adder. Normally, an address is being calculated to be used as a fetch address. A selected field of the adder output reads out a portion of the address array into the compare level. The address array holds addresses for which the data is stored in the data array. The full adder output goes to RCA, which is compared to R0C-R3C for equality. If there is a match, the data is local and can be loaded into RWD from the data array. If not, RCA is loaded into RMA and the MAU fetches the data at this address. The MAU loads the data into RMD which is then loaded into the data array and also into RWD. RMD is also used to accept data from the EU for stores and moving-type operations. RMD is then loaded into RWD. The only input to RED is P bus. Descriptors can be moved from RED through RCD into RWD or they can be constructed in RCD by use of the RROU PROM table controls, and then moved into RWD.

Associative Memory

The associative memory (ASM) serves as a local data buffer that provides fast access to frequently used variables and descriptors to increase the speed of memory fetching. Therefore, references are first made to the ASM; if the data is not local, main memory is then accessed. The ASM consists of the following major functional elements:

1. Address Array, which is comprised of 64 blocks, each of which contains four 13-bit address groups (0 through 3). Each 13-bit address group consists of an 11-bit address and two special residue bits (S0 and S1).

2. Data Array, which contains 2,048 words, each of which consists of a parity bit, three tag bits, 48 data bits, two residue bits (R0 and R1), and an error bit. The data array is also comprised of 64 blocks, with each block containing four groups (0 through 3), and each group consisting of eight words.

3. Priority List Array, which is a 4 x 4 register file that contains 64 eight-bit locations which are configured as four blocks, each of which has 16 locations. (Of the eight bits available in each word, only five are used for the priority function.)

The data input to the address array is taken from DRU address bus RAB (19-09, S0, S1), which provides 11 bits of address and two special residue bits. The array (which is composed of RAM1 chips) is addressed by bits 08 through 03 of RAB; the address and chip select inputs can originate from a store-to-stack (push), the DRU write data register, adder output, memory address register, or deferred reference queue address. Each block in the address array consists of four 13-bit fields; each field is comprised of an 11-bit address and two special residue bits.

Seventeen bits (19 through 03) of a memory address or stack plus 1 (XSP) are applied to the address array; bits 19 through 09 are the address written into the array, and bits 08 through 03 identify the block to which the address is written.

The data array is used to store memory data from either MFD (for memory fetching operations) or CEDE (for memory stores). The 11 bits of address required to address the 2,048 words of the data array are supplied by RCA and compare level signals RCL G1 and G0. Bits 08 through 03 of RCA identify the block, bits 02 through 00 identify the word within a group, and RCL G1 and G0 identify the group.

The priority list array is used to provide a history of the groups contained in the address array. Since there is a fixed amount of storage (four locations, or groups) for addresses with the same block value (bits 8 through 3), one of the existing addresses must be overwritten if a fifth address is to be loaded. Which of the addresses is overwritten is determined by the priority list entries for that block. The array contains 64 locations, one for each block of the address array. Five of the eight bits available in each priority list word are used; two bits indicate the address that has not been referenced for the longest time, known as the oldest group. Two bits indicate the second oldest group, and the last bit indicates whether the newest (fifth) entry has a greater value than the newest existing group entry for that block in the priority list array. Therefore, when a fifth address is applied to the address array, the new address and data are written into the oldest existing group and the group ordering (from newest to oldest) is then adjusted by the priority list array to reflect the current priority.

Stack Address Registers

The stack address registers (shown in figure 3-2-10) are used during ENTR, EXIT, MVST, and MKST, and are maintained automatically by the DRU as part of the stack address adjustment process. Figure 3-2-10 shows that stack addresses are transferred from the adder output (RA0) to certain stack address registers and from one stack address register to another by use of input selection gates.

The registers and the general use of each are as follows:

1. S register contains the address of the top item in the stack. This address is passed to the DRU compare level for fetch stack operations and to the Store Q for stack-cut back operations.

2. XSP register contains S register contents plus one. This register provides a quick method of adjusting S register during POP and PUSH operations. For POP operations, both S and XSP values are decreased by 1 (that is, transferring S into XSP and decrementing S). For PUSH operations, both S and XSP values are increased by 1 (that is, transferring XSP into S and incrementing XSP).

3. XLOS (Limit of Stack) register contains the address of the highest usable location in the active stack. During Move to Stack operations, the XLOS is loaded with the base address plus length (minus 1) from the fetched descriptor for the requested stack.

4. XLSP register contains LOS register contents plus 32. This register extends the range for above S conditions because certain operations in the DRU pipeline can reference items above the upper limit of the stack when the CPM is ready to report a stack overflow condition. If above S condition occurs (RCA <—XLSP), the PCU issues store to stack (PUSH) operators to clear the top of stack.

5. F register identifies the top-most MSCW in the stack. During EXIT, the DF field of the MSCW being cut back is subtracted from D[LL] and the result is loaded into XF. The new F is used to fetch the previous MSCW. During MVST, a new value for XF (DSF of the TOSCW from the stack being entered + BOSR) is loaded into XF. This new F value is later used in calculation of MSCW. During ENTR, the XF is read onto the stack bus (XSB) to fetch IRW at F + 1 for evaluation, to fetch inactive MSCW and load store address list (SAL) with mark address (XF) so that the EU can activate the MSCW, and to write RCW into the stack at F + 1.

6. XSR (Save Register) holds S value so that the S register can be returned to the previous value if DRU is doing an EXIT and EU is reporting a conditional branch or restart condition. To accomplish the returning of S, along with F, to the previous value, the XSR and XSP are loaded into XS and XF, respectively, and the XSP increment flip-flop is set. Then on the next cycle, the XSP is incremented by 1 (S + 1).

LL Data Paths

As shown in figure 3-2-11, the major elements in the LL data paths include the current lex level (XLL) register, lex level save (XLS) register, lex level counter (XLC), LL decode logic, and PROMS.

The XLS and XLC registers are used as alternate registers during execution of dynamic branch unconditional (DBUN), enter (ENTR), and exit (EXIT) operators when contents of PCW (for DBUN and ENTR) or RCW (for exit) are distributed. Because addressing environment of CPM is not ready to be changed at this time, the LL field in the PCW or RCW cannot to loaded in XLL register. Therefore, the LL field is saved in XLS register until addressing environment is to be changed. At this time, XLL is exchanged with XLS to provide a copy of previous LL when DRU enters into new address environment. This copy of LL is saved because other sections of the CPM have not changed over to the new address environment. Then, if recovery is required, a return to old address environment can be made.

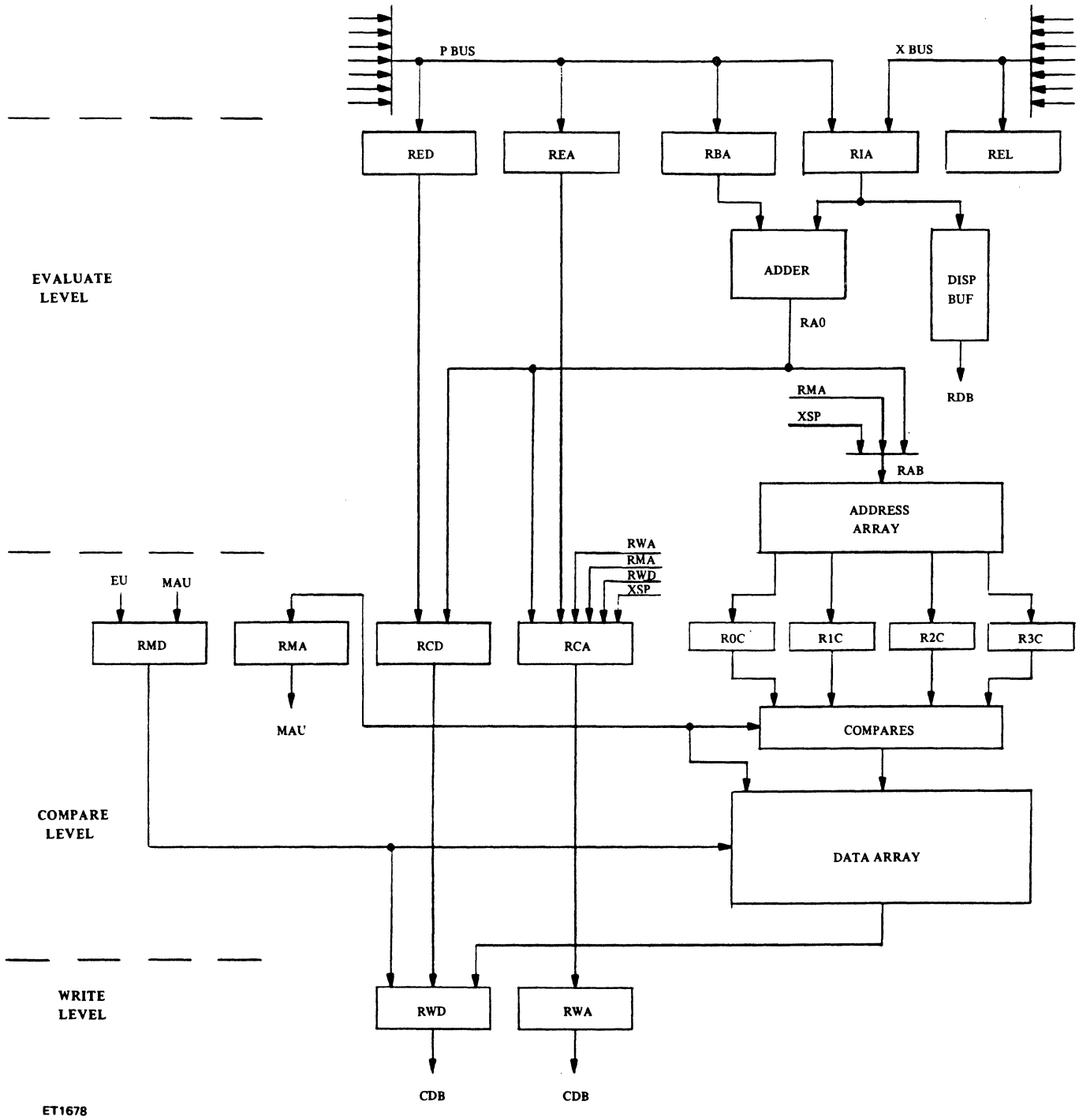


Figure 3-2-9. DRU Data and Address Paths

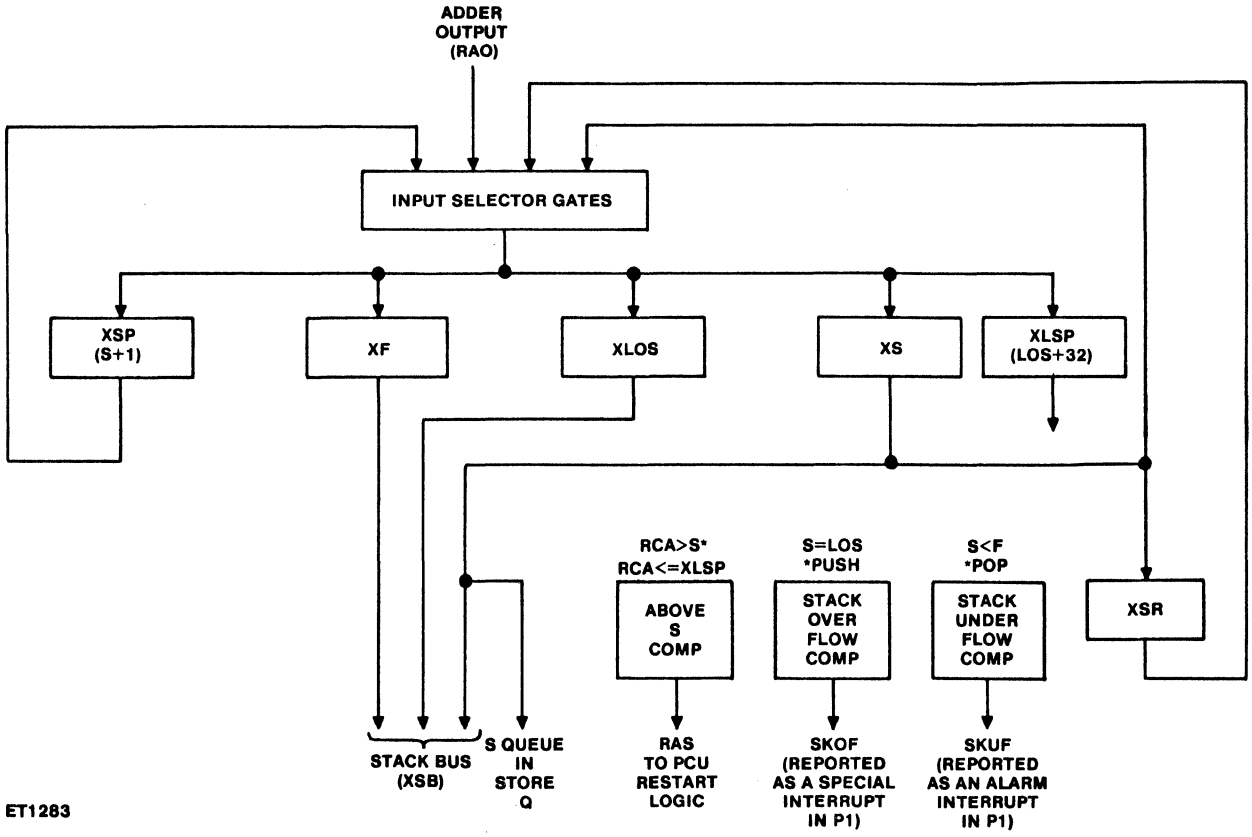


Figure 3-2-10. Stack Address Registers

The XLC register is primarily used for updating display registers. When XLL and XLS are exchanged, the XLC is initialized to LL being run at. When update display is performed, the XLC is used as a marker for counting down through the LLs. Also, at the beginning of ENTR, the LL of IRW, which found the PCW, is saved in XLC. Therefore, anytime an IRW evaluation is performed for ENTR, the LL of IRW is loaded into XLC. This is a new method of handling stack number displacement where a decision is based on whether the LL of IRW, which points to the PCW, is equal or less than current LL.

If LL of IRW is less than the current LL (XLL value), a mark can be fetched and its stack number displacement field can be extracted for a new mark being built (LL + 1).

If LL of IRW is equal to current LL, the LL of IRW is at the top-most environment of the stack (i.e, LL is in current stack). Thus, calculation of DLL - BOSR provides displacement for the mark which is in the current environment.

The LL bus forwards outputs from XLL or XLS to EU barrel for building a MSCW being entered or a RCW, respectively.

EXECUTION UNIT

The execution unit (EU) is the only unit in the CPM which operates on value data. It also has responsibility in assembling some control words.

A simplified block diagram of the EU is shown in figure 3-2-12. As shown in this figure, the EU contains three major processing sections (EUCDB, ALU, and SAU). Each is explained in section 1 of this chapter. Other major sections of the EU include the barrel, the macrocode routine, the execution write register (EWR), and the input queues. Barrel is controlled by EUCDB and ALU. These controls allow either a right or left-justified field of any length (from 0 through 48) to be selected as the barrel output. Macrocode routines are issued by the EU when complex operators require a sequence of simple operations for processing. EWR, an autonomous section of the EU, contains independent controls and grants priority to inputs from ALU, SAU, and barrel.

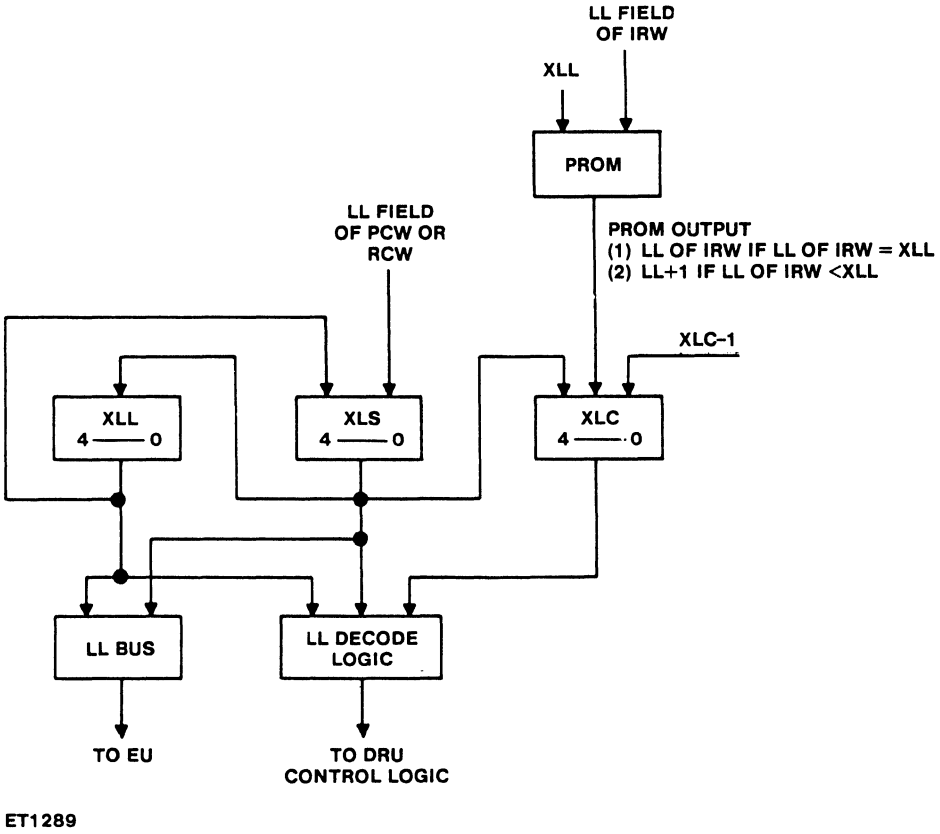


Figure 3-2-11. LL Data Paths

The EU, like the DRU, is queue-driven. All operators and operator information are placed into the operator queue by the PCU. The string information input, such as source and destination pointer values to the EU, is placed into the parameter queue when it is forwarded to the EU by the DRU. EU is responsible for performing operations on the string data which includes comparing, translating, modifying, and moving the data.

EU Operator Queue

The EU operator queue (EU OP Q) is the storage area for micro-operators and related information pending EU processing. The use of the queue allows the PCU and EU to operate independently. The information to be written into the EU OP Q is placed in the PCU operator input registers (PQO and PQE); PCU write address registers (PWA, PWB, and PWR); PCU data write register (PLW), and PCU write job number register (PWJ). The PCU loads these registers, then initiates the write cycle. EU OP Q is read by the EU and the operators read from the EU OP Q are the addresses for the EU PROM networks.

Parameter Queue

The purpose of the parameter queue is to buffer parameter information (source pointer, destination pointer, segment length, and certain controls) for use in building destination words in the EU. The DRU provides new parameter information for each destination word produced. In most string operations, if the operation is to produce five destination words and each of these five words is written into consecutive addresses in memory destination array, then five parameters must be passed from DRU to EU.

The exception is in table edit operators which require separate parameters for each micro-operator. Therefore, several parameters may be required to build one destination word.

The source, destination, and segment length information is received into the parameter queue as binary values. These binary values represent the actual starting bit and the number of bits to be moved from source word to destination word. The conversion from character representation to binary values is performed by PROMs during DRU to EU transfers.

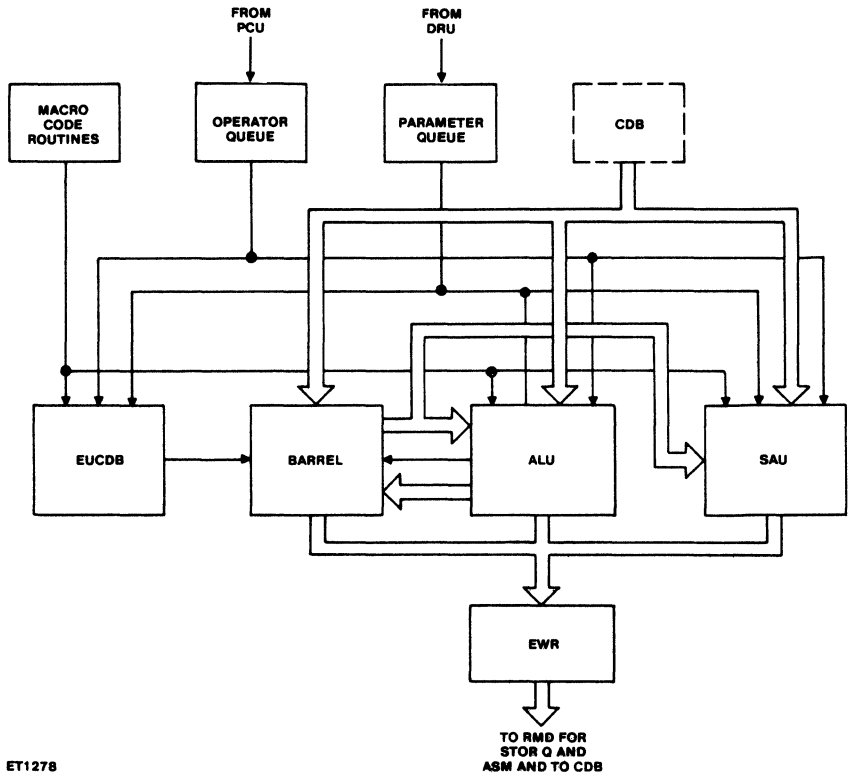


Figure 3-2-12. Execution Unit Block Diagram

The source, destination, and segment length information is used to set up the shift and allow registers in the EU barrel.

EU Control Pipeline

Like the DRU, the pipeline processing technique is implemented in the EU. There exists within the EU four distinct processing levels: a look ahead level, a read level, an ECDB level, and a terminate operator level.

As shown in figure 3-2-13, each processing level has a pointer register. Basically, the pointer registers (EL0, EQ0, EC0, and ER0) are used to remember a particular EU OP Q location from which an operator was read if a restart is required at the ECDB level. Functionally, the EU OP Q is divided into two parts: the operator queue for the ALU and SAU and the EUCDB queue.

The ETO pointer register serves to define EU OP Q full for PCU. When ETO equals OP Q write pointer PQEW, the PIE hold logic in the PCU is temporarily enabled until OP Q can accept the operator from the PCU.

In order that EU may know when a pointer register has been loaded, there is a separate valid bit associated with each pointer register. In addition to the valid bit, the EL0 contains a last load (LL) bit, which indicates that PCU has loaded EU OP Q since the last EU pipeline advance or that EL0 received a restart operation from ER0. The LL bit simply allows EU to remember that an operation exists in the EU OP Q even though EU OP Q read and write pointers are made equal by the last pipeline advance. For restart case, the ER0 valid bit allows transfer of ER0 to EL0 before the next EU OP Q read occurs.

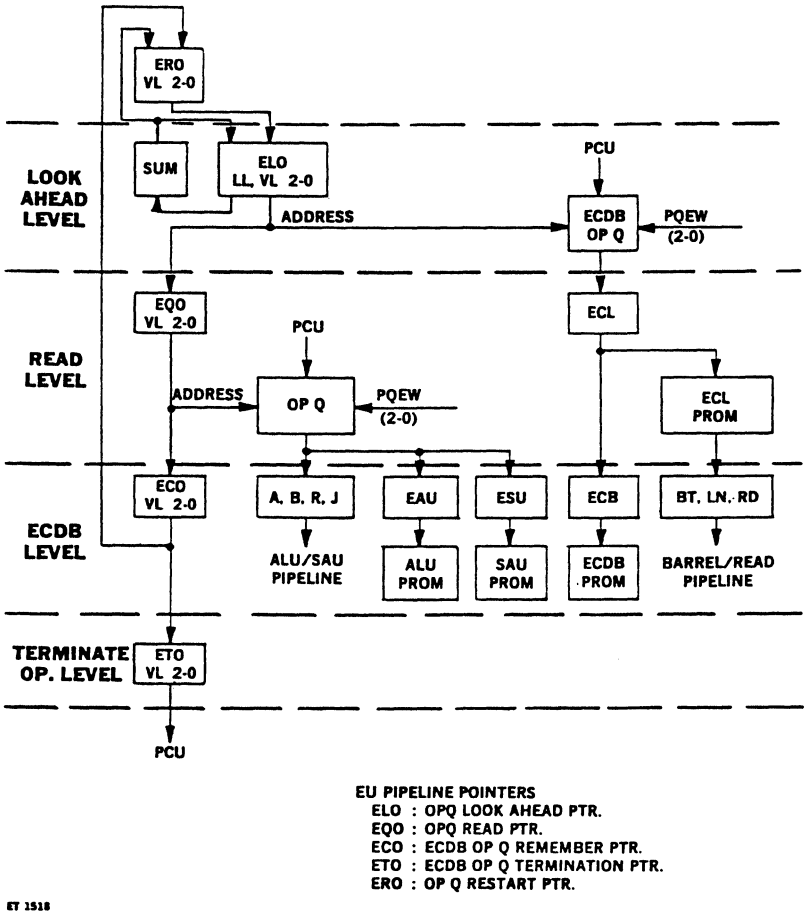


Figure 3-2-13. EU Control Pipeline

When a look-ahead read is performed by EL0 (pointers EL0 and POEW $\neq$), the EQ0 is loaded with EL0 pointer value, and, if present, an eight-bit microcode from the OP Q is read into ECL. Also, the contents of EL0 pointer are updated by use of a separate adder and the updated content is loaded back into the pointer for use on the next cycle. The eight-bit microcode in ECL is used to address the EUCDB look-ahead PROM (ECL). The EQ0 pointer reads the ALU and SAU OP Q location that is associated with the ECDB OP Q location read on the previous cycle.

On the next cycle, as the EQ0 pointer value is loaded into EC0, the ECB register is loaded from ECL and the microcode from OP Q is loaded into EAU, ESU, or both, and the first of a sequence of ALU and SAU operations is generated through ALU and SAU PROMs, provided that the unit is not waiting for ECDB to supply data. Also at this time, if the ECL PROM is addressed, the EU CDB read address from the OP Q or, for special cases, EU generated address is loaded into CER by ECL PROM. Using these addresses allows CDB data to be read onto the ALU and SAU input buses of the EU. The ECL PROM is also used to set up shift, right allow, and left allow amounts used by the EU barrel. Although ECL PROM provides shift and allow information, the barrel controls are implemented by the ECDB PROM. The controls include barrel input load; barrel shift, right allow, and left allow register loads; and barrel output destination.

It should be noted that initial functions of the ECL and ECDB PROMs are identical because the EU store subunit can obtain control of CER as the transfer of ECL to ECB occurs. If this should occur, then the ECDB PROM functions provide the necessary controls; otherwise, the ECDB PROM functions are ignored.

At this point in the pipeline operation, the ECDB can iterate, and, if it does, the EC0 pointer holds. This hold is removed when ECDB provides necessary data to the barrel, ALU, or SAU, or when an EU restart is required.

If ECDB releases hold, the pointer information is not needed anymore so EC0 pointer value is loaded into ET0 which indicates ECDB has terminated the operator.

For restart case, the OP Q pointer value in EC0 is loaded into ER0 and then back into EL0 to start an operator through the pipeline again. For example, the ECDB calls macrocode for copy action or to integerize the input to the ALU. After copy action or integerizing is completed, a Return To Pipeline is performed where the ECDB operator is started with new data. Instead of reading from A or B locations, the selected location for a read is W. (The selection of W is controlled by Remember W flip-flop ERW.) The W location contains new data for the operator. (In this example, a descriptor with copy bit ON or integerized data is being processed.)

When the ECDB operation ends, the ALU can be operating on data. For example, after ECDB sends two inputs to the ALU for a divide and then ends, the ALU performs the divide operation. The pipeline can again activate ECDB which can supply data for an SAU operation.

EU Code Paths

The EU code paths (shown in figure 3-2-14) consist of four major functional areas: 1) the operator queue; 2) the ECDB, ALU, and SAU PROMs; 3) the EU macrocode (EUMC) PROM; and 4) the pointer registers associated with these PROMs. The operator queue consists of 4-by-4 register files that can be simultaneously read and written. Functionally, the operator queue is divided into two parts: the ECDB queue, and the operator queue for ALU and SAU processing. Micro operators written to the queue originate in the PCU and are applied to the queue via registers PQE and PQO.

The ECDB operators are read from the queue by look-ahead pointer ELO (described later under "Pipeline") and the eight-bit microcode read from the queue is applied to CDB look-ahead register ECL. These operators form eight-bit addresses that are applied to the ECDB PROM network via CDB operator register ECB. The ALU and SAU portion of the operator queue is read by queue operator pointer EQO, which is also described in the discussion of the operator queue pipeline. These operators also form eight-bit addresses that are applied to the ALU and SAU PROM networks. The PCU micro operator specifically addresses the ALU, the SAU, or both.

ECDB PROM Network

The 256-by-4 PROMs that comprise the networks contained in the EU code paths produce operating sequences to control processing in the EU. Some of the functions of these operating sequences are as follows:

1. To select read address paths to CDB EU registers.
2. To transfer the CDB bus to ALU and SAU input buses.
3. For store-to-stack operations, to transfer EW to the RMD register (write into store queue and associative memory.)

The ECDB PROMs are addressed by the eight-bit register ECB. These PROMs provide three major outputs. One output is fed back to the ECB register to indicate the next state (or address) in the PROM output sequence. A second major output is used to select the ALU or SAU PROMs (via arithmetic selection circuits) if the PCU micro operator requires use of the barrel prior to being processed by the ALU or SAU. For example, if scan while compare (SCWC) micro operator is issued to EUCDB by the PCU, a delimiter is read from the CDB and is shifted by the barrel to update required character in preparation for comparison with the source character by the ALU.

The third major output of the ECDB is applied to macrocode register EMC. This output is used if a copy action operation is to be sent to the ALU, or if it is necessary to integerize the input to the ALU. This function is performed if input for a DBUN or LLLU operator or certain types of scale right and bit operators require integerizing.

ALU PROM Network

Four major functional areas are associated with the ALU PROM network: 1) arithmetic selection circuits; 2) operator save register EOS; 3) arithmetic unit operator register EAU; and 4) macrocode register EMC. The arithmetic selection circuits handle inputs from the operator queue or macrocode (through the micro operator selection circuits), the ECDB PROMs, and the EOS register. This register is used

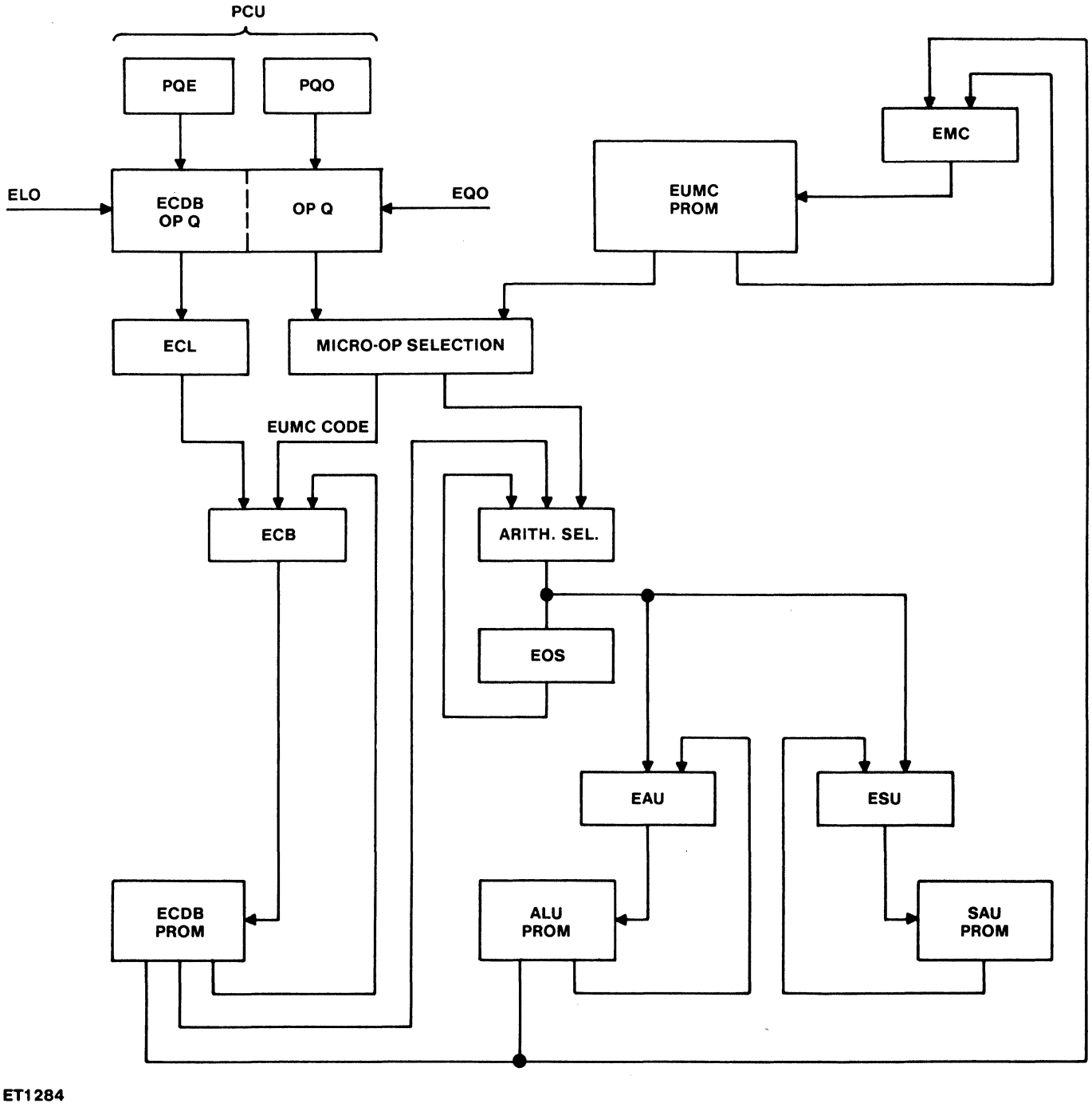


Figure 3-2-14. EU Code Paths

to temporarily save an operator destined for the ALU if the ALU is currently processing the previous operator.

When the ALU is finished processing, the operator saved in EOS is then selected into the EAU register. This register is loaded with eight-bit operators that are used to address the ALU PROM and produce the PROM sequences needed to process the data in the ALU. For double-precision operators and difficult operators, such as scale operators, the ALU makes use of the macrocode register (EMC) to obtain additional micro operators via the EUMC PROMs. Some of the functions of operating sequences provided by the ALU PROMs are as follows:

1. To direct the AI bus to one of the four ALU input registers.
2. To set the end-around-carry mode of the ALU adder.
3. To complement the AL or AM to AB registers.
4. To direct barrel data to BS (save) register, EW register, or back to ALU via the AI bus.

SAU PROM Network

The SAU PROM network is addressed by the short arithmetic unit register ESU, which receives input from the arithmetic selection circuits. For PCU micro operators, the SAU has priority over the ALU if the integer length for ADD and COMP operators is less than or equal to 20 bits and for multiply operators, if less than or equal to eight bits. Whenever these conditions are satisfied, the ALU is inhibited from requesting the EW register and from loading the barrel. If the integer length requirements are not met, the SAU aborts the job; it is then inhibited from requesting the EW register, and the job is processed by the ALU. Some of the functions of operating sequences provided by the SAU PROMS are as follows:

1. To control carries into and out of the SAU adder.
2. To load the SAU adder output to the SY register.

EUMC PROM Network

The function of the EUMC PROM network is to provide micro operators required for processing but not issued by the PCU. If a condition arises during the execution of an ALU or ECDB micro operator that requires a micro operator not issued by the PCU, the required code is then supplied by the EUMC PROM network. The code is passed through the appropriate selection circuits and is applied to the specific PROM address register involved. For data associated with new EUMC microcode sent to one of the three PROM address registers, the EUMC PROMs re-issue CDB read addresses to the CER. The EUMC PROMs also issue CDB write addresses to the CEW for resultant data to be temporarily stored in the EU data file, the EU local file, or the exponent file in the CDB.

CDB EU READ POINTER REGISTER

The CDB EU read pointer register (CER) is used to select an EU CDB location for reading. The location to be read is determined by the ECDB and EUMC operator being performed or by the store data address from the EDB register. The CER is loaded, if there is no barrel, SAU, or ALU hold (ECDB.HDL); or if an address input from the Store Data OP Queue exists and RMD is ready to accept data from the CDB.

Figure 3-2-15 shows the inputs to the CER. The signals, which transfer the inputs to the CER, are developed by decode logic. This logic receives various commands from EUCDB PROMs, ECL and EUMC PROMs, or from EUMC PROMs through special registers. These registers are loaded by macrocode but are selected by EUCDB code to determine which CDB address is read. These CDB addresses are:

1. Read A (address in EPA).
2. Read B (address in EPB).
3. Read R (address in EPR).
4. Read W2 and W3.
5. Read X0, X1, and X2.

A and B Selection Path

The A and B selection paths are the source for loading:

1. The A and B address from the EU OP Q.
2. The W2 (Hex 2E) and W3 (Hex 2F) address.
3. The EPA and EPB registers.
4. For A selection only, the result address (EPR) into the CER.

Store Data Read Address Path

The store data read address is loaded into EDB as part of the EU store subunit processing. EDB is transferred into CER when the store counter register indicates a store exists, the older EU operators are processed, and the SAL address is declared valid by DRU. Transferring the store address to CER also prevents other transfers to CER from occurring at the same time. The store address can only be loaded into CER when the transfer of the next operator from ECL to ECB takes place. (Refer to subparagraph headed EU Control Pipeline.)

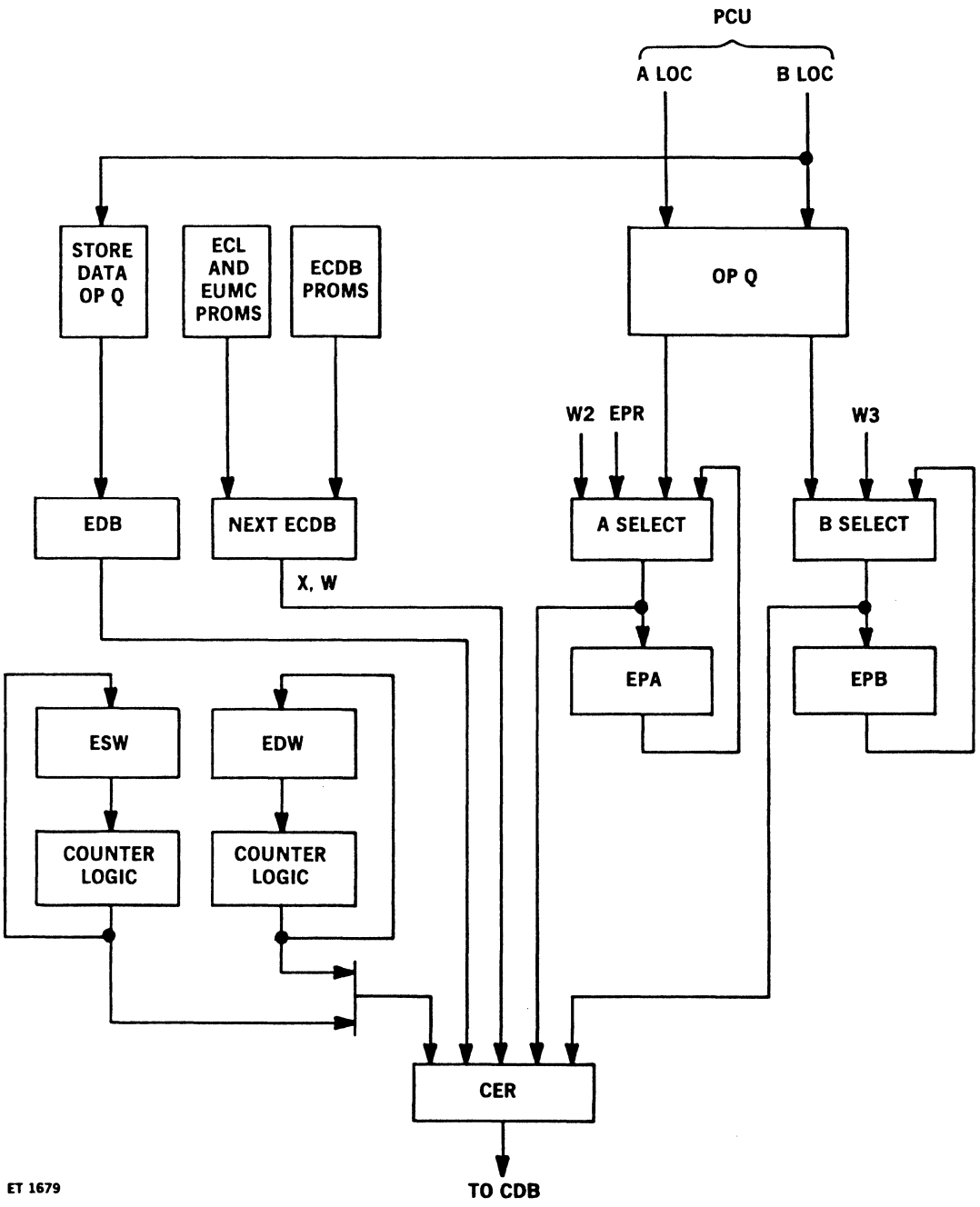


Figure 3-2-15. EU CDB Read Address Paths

Source and Destination Pointer Path

The source and destination pointer path is used for reading string data into the barrel for processing. The source word register ESW and destination word register EDW pointers are used to locate the data to be read from DRU string data file in the CDB. ESW and EDW select locations 10 through 17 and 18 through 1B, respectively, in the DRU string data file of the CDB.

The pointer that is to be loaded into CER is selected by a transfer signal that is derived from EUCDB CDB read category PROM when string operators are processed. Pointers are also updated to show the location in the CDB when transfer is ended. Parity is maintained on the contents of the pointers and is updated each time the pointer value is updated.

The pointers are reset whenever clear queue occurs or EU has aborted a string operator. This reset provides the start locations (location 10 for source words and location 18 for destination words) to allow counting the pointers to address the next string location. (The reset of these pointers provides compatibility between locations assigned by the DRU and locations read by the EU.)

EU BARREL

The data paths of the barrel consist of the barrel input (BN) register, selection logic for pack and unpack data, coarse and fine shift logic, and the barrel output logic. (See figure 3-2-16.) The coarse and fine shift logic is implemented with MFAN chips so the barrel may be used repeatedly during each clock cycle. The coarse part of the barrel shifts the input data by multiples of 1.

Because of the barrel inputs from the EU read register (ER), CDB bus, ALU output, and barrel output, the BN register has input selection gating. The desired input selection is controlled by ALU PROM commands.

The ER register loads the BN register and is the buffer register for checking parity on data transfers from the CDB bus to the EU barrel and for decoding the number of leading zero digits of mantissa data. The ER register is loaded any time a CDB transfer to BN takes place or a CDB bus and an AI bus to ALU condition exists.

The output of the barrel is selected to BN register for additional barrel operations, to EU write (EW) register for transfer to CDB, to the AI bus for transfer to ALU, to the SI bus for transfer to SAU, or the barrel save (BS) register. The BS register holds data that is to be ORed with new barrel data.

Contents of the left and right allow registers (ELA and ERA) determine the lowest and highest bit number, respectively, to be transferred at the barrel output.

A special flip-flop is used to correct barrel output parity when the barrel is in a hold condition (for ALU to BN transfers only). The purpose of this barrel output parity correction is to use the accumulated parity of the EBR and EW registers before the information becomes invalid on the next clock cycle.

SHORT ARITHMETIC UNIT (SAU)

The SAU performs SP integer arithmetic operations of 20 bits or less, eight-bit multiply operations to produce 16-bit result, and double-precision exponent calculations. The loading sequence for the SAU consists of one-cycle operand load and two-cycle operand load.

In the one-cycle load, both operands are received simultaneously by the SAU (the VALC from the CDB and either a LIT from the PCU or the SAU output from a previous operator). Excluding the multiply operator, the next cycle allows the PROM address to be accessed again. The multiply operator requires another cycle to complete the operation.

For a two-cycle operand load, the SAU receives the A operand on the first cycle and the B operand during the second cycle to complete the operation.

The basic functions of the SAU sections (figure 3-2-17) are:

1. ESX register provides input for the parity check and residue generator logic, temporary storage for the result exponent during double-precision, and temporary storage for the carry save carries during multiply operations.

2. ESY register provides one of the operands for arithmetic operations, the complemented operand for a subtract operation, and temporary storage for partial products and carry sums during multiply.

3. Adder logic is a standard two-input adder which propagates all carries on each adder pass. AFANs are used as adders. Subtraction is performed by adding the B complement to A.

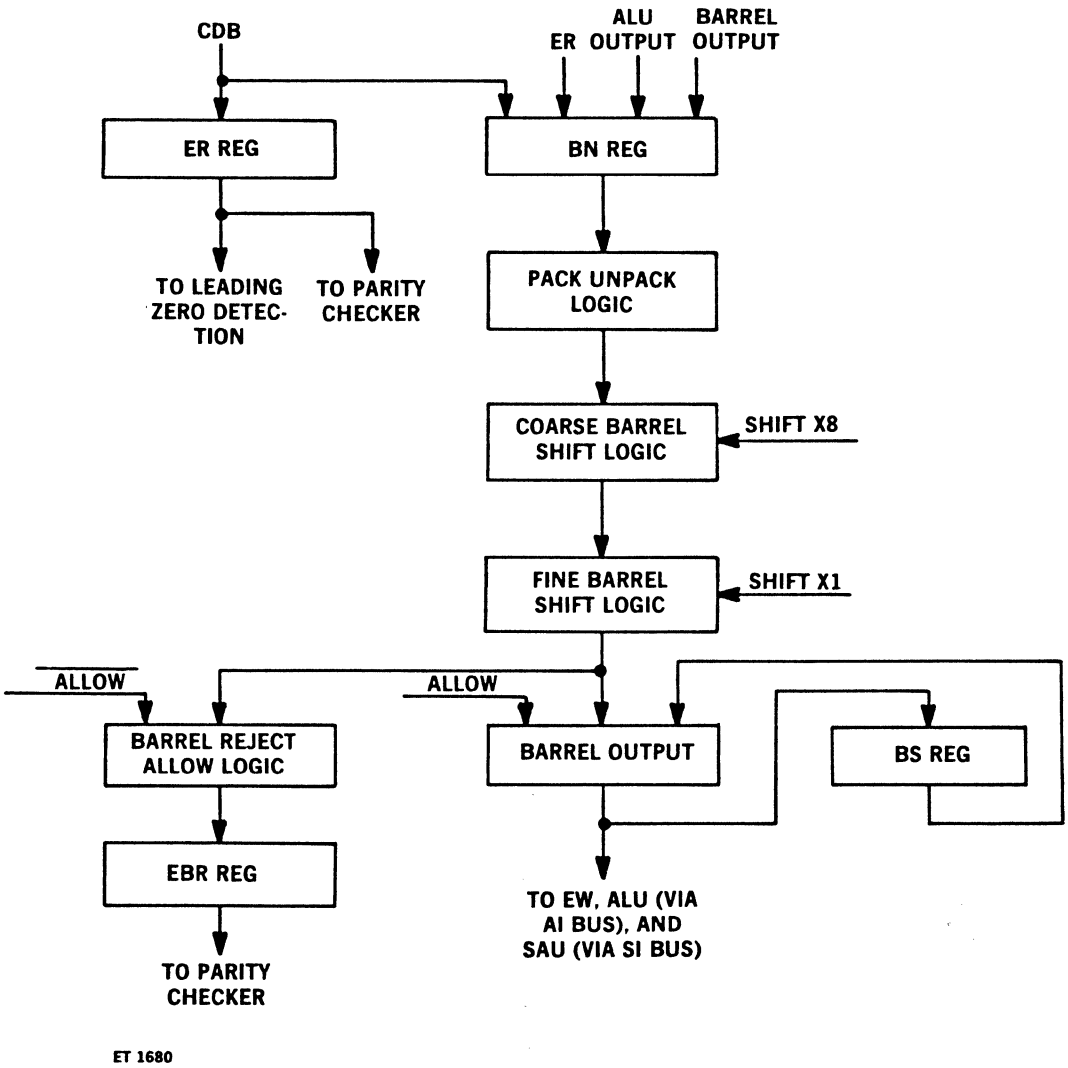


Figure 3-2-16. Barrel Data Paths

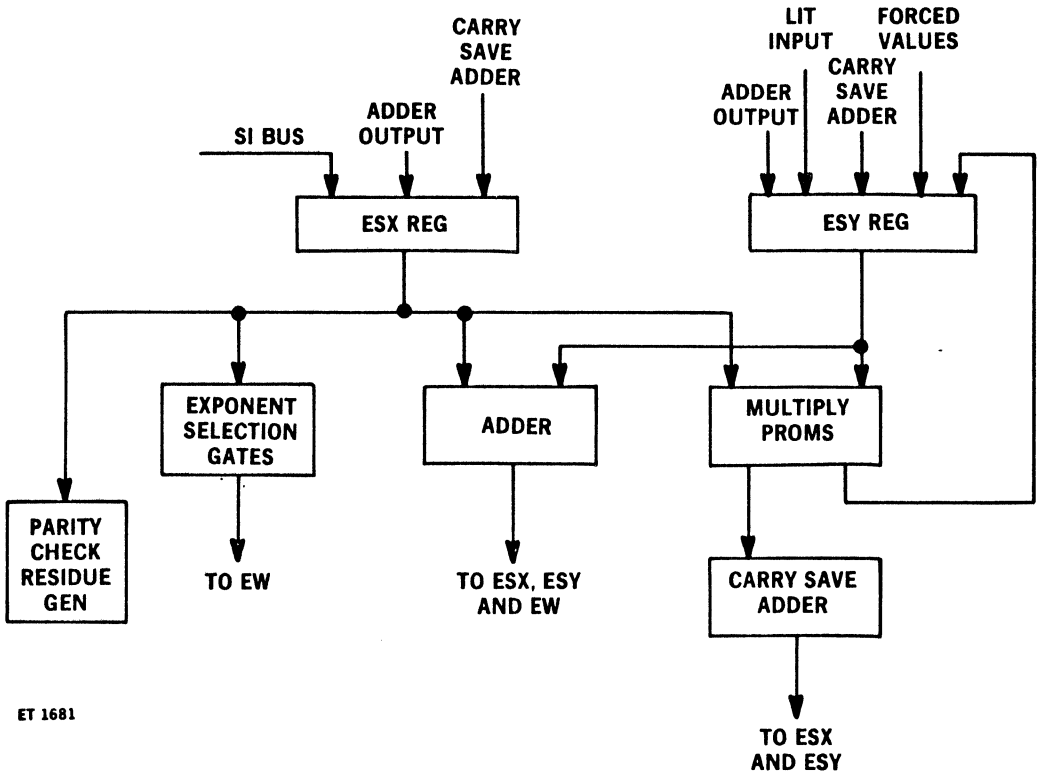


Figure 3-2-17. SAU Data Paths

4. Multiply PROMs multiply the entire eight-bit multiplicand by eight bits of multiplier. Then, on the adder pass, the accumulated partial product is added to the carry number. The ESY register enters the accumulated partial product while the ESX register provides the carry number.

5. Carry save adder generates sum and carry for each of the partial product inputs.

6. Exponent selection gates allow exponent insertion into the EW.

The SAU parity logic netword checks the parity of information passed to the ESX register from the SI bus.

The residue logic maintains residue on SAU adder output. The residue check is performed at the EW level of the EU.

ESX and ESY registers have the residue loaded at the same time the information is loaded. The residue comes from the following sources:

For ESX Register:

1. The S + 1 address is sent from the I bus through the SI bus with residue. (For mark stack operator case.)
2. Information from the SAU adder (ESA) has residue.
3. The residue in ESX is saved (residue is loaded back into ESX) and is multiplied by the ESY residue to obtain the residue of multiplication result.
4. Most inputs have no initial residue. In these cases, residue is generated with the data.

For ESY Register:

1. Information from the SAU adder has residue.

2. Value for LIT is sent from the PLW through the PCU pointer queue with residue. (PLW has a residue generator.)

3. EU I and J word and digit information are sent from the pointer select logic with residue.

4. Forced values are loaded with forced residue.

5. The residue in ESY is saved and is multiplied by ESX residue to obtain residue of multiplication result.

ARITHMETIC LOGIC UNIT (ALU)

The ALU is used to perform most types of arithmetic and logical computations in the EU. The ALU operates on non-integer or integer data that is greater than 20 bits. Error detection includes residue and parity checks. Residue is used to detect errors in the exponent and mantissa data paths and associated data registers and in the repetition counter circuitry. Parity is used to detect errors in data that is adjusted for specific ALU operations.

Figure 3-2-18 is a block diagram of the ALU data paths. The basic functions of the ALU sections are:

1. AS register is primarily used as a shift register in arithmetic and count operations and an input to the parity and residue generator logic.

2. AA register is used as an accumulator register and an input register for the main adder.

3. AB register is used to supply data in complement or true form to the main adder and the logic unit.

4. AX register is the primary input to the exponent adder.

5. BX register is the secondary input to the exponent adder.

6. Exponent adder is used for adding or subtracting exponents during single-precision floating-point arithmetic operations.

7. Exponent difference (XD) register contains the exponent value that is used to determine shift and allow inputs for the barrel operation.

8. Conversion PROMs perform the following operations:

   a. BCD to binary conversion for the input convert destructive ICVD and input convert update ICVU operators.
   b. Binary to decimal conversion for the scale right final SCRF operator with an input equal to, or greater than, 32 bits.
   c. Count the number of 1 bits in the A operand for the count binary 1's CBON operator.

9. Multiply (MUL) register is the input to the MULT PROM for loading one or two octal digits of the multiplier from the AS register. It is also the input to MULT PROM for loading the scale right PROM output for double-precision scale right operators or for a single-precision scale right operator with an input less than 32 bits.

10. Multiply (MULT) PROMs select proper multiple into the AB register. The selected multiple is entered in true form to add that multiple to the partial product or in complement form to subtract that multiple from the partial product.

11. Scale right PROM provides a binary value that is a representation of the number 10 to the minus scale factor (not greater than 12). Selection of the binary value is done by the ALU variant register, which contains scale factor provided by the PCU and the repetition counter. The RC counter is first loaded with a count (16-LX) that represents the number of octal digits in the number (A operand) to be scaled. RC count is then decreased by 1 or 2 on each multiplication cycle until RC equals 0. The multiples for each multiply cycle are selected by the MULT PROMs in accordance with the binary value supplied to the MUL from the scale right PROM.

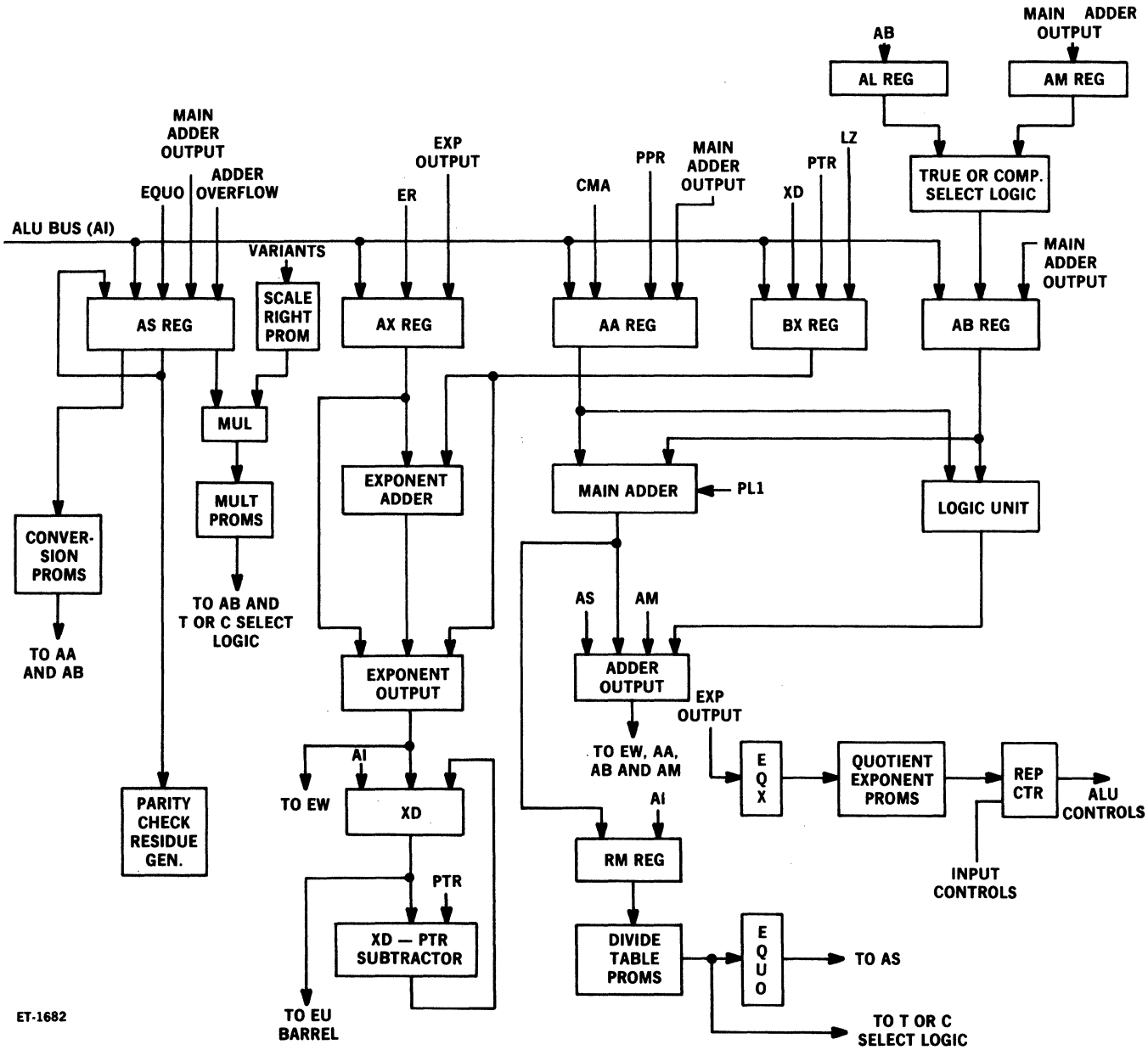


Figure 3-2-18. ALU Data Paths

12. Main adder is a standard two-input adder which propagates all carries on each adder pass. AFANs are used as adders. Subtraction is accomplished by adding B complement to A. In order to accommodate multiply and divide operations, an extension of the adder output is provided to pass the final result into an accumulating register. The extension is identified as guard bits in the ALU logic. The PL1 adds an additional 1 into the adder for certain multiply and divide operations.

13. Logic unit is a portion of the main adder AFAN chips. It provides an efficient means for executing logical operations. The generate and propagate outputs, which are developed for each bit position by the AFANs, are selectively gated to derive the logical AND, OR, and exclusive OR functions.

14. AL and AM registers are used as temporary storage locations to form new operands for certain arithmetic operations.

15. XD-PTR subtractor is used in ADD operations in which the mantissa with the smallest exponent is shifted right to equalize the exponents. The number of LZ of the mantissa with the greatest exponent is subtracted from the exponent difference to determine the right shift value for the mantissa with the smallest exponent value. This value is loaded back into XD and then applied to digit conversion. PROMs, where the proper shift and allow barrel controls are selected for right shifting the mantissa.

16. RM register contains six MS bits of the remainder for use by the divide table PROMs. These PROMs are programmed to predict the trial quotient for the next cycle and also to select proper multiples into AB register. Trial quotients from the divide table PROMs are loaded into EQUO and then routed to the two LS bits of the AS register on each cycle.

17. Quotient (EQX) register is the input to the quotient exponent PROMs. These PROMs determine the number of iterations needed to develop the integer portion of the quotient.

18. Repetition counter holds and releases ALU PROM sequences and controls certain data decodes within the ALU PROM sequences.

The ALU parity logic is composed of an AS parity generator, parity update logic, and the logic to select the type of parity checking required. The parity generator provides parity on information shifted into the AS register and parity on certain operator inputs to AA and AB registers.

Parity update occurs because right shifting of data into the adder output causes loss of bits with unknown parity. In such cases, an ALU PROM command ensures that adder output parity data is corrected to accommodate the loss bits.

The residue generator is used to generate residue for the exponent and mantissa data.

Inputs to AX from the barrel (through the ALU bus) and from the ER register are not transferred with residue, so residue must be generated. During ALU bus transfers, residue is provided by the residue generator. For ER transfers, the residue is generated by a PROM (on card EF0). Inputs to BX from either I digit or J digit pointer, or leading 0's logic, have residue already available.

The residue of the exponent adder output is automatically checked at EW level to make sure that the residue is correct. If residue is incorrect, then an EW residue error is reported as an alarm interrupt in EU interrupt error register EIE. Also, connected to the exponent adder output residue are the EQX Residue flip-flops, which, along with EQX PROM, report residue errors that only occur during IDIV operator processing. If this residue is incorrect, then an EQX register residue error is reported as an alarm interrupt in the EAE register.

All residue for AA, AB, and main adder output is controlled by their respective ALU PROMs. The residue is adjusted, along with the data with which it is associated. First, the basic adder residue is determined, then the basic residue is corrected by the PROMs to provide the proper output residue. Sign bits are never included in residue generation. However, because the residue is adjusted for sign when it is used, the signs are included in the residue check.

Inputs to AA from the CDB Memory Address bus (CMA), procedure return register (PPR), and for DP divide operations from ALU bus (through EW) have residue already available. Also, inputs to AB from the conversion PROMs have residue already available.

In many cases it is necessary to catch up on residue. This catch-up residue occurs when an operator has two inputs to process. In such cases, the residue generator output is added to, or subtracted

from, the adder output residue according to the type of operator being processed. For example, at the start of a DP ADD operator, the A input is loaded into AA and its residue is generated by the residue generator. Then, on the next cycle, the B input is loaded into AB, the residue generator output is loaded into AA residue bits, and the residue for B input is generated. This residue (or catch-up residue) is added to the main adder output residue to develop the final residue of the adder inputs. All residue for adder output is controlled by the ALU PROMs.

The residue check for AA, AB, and main adder output is performed at the EW level. If incorrect residue occurs, then an EW residue error is reported as an alarm interrupt in the EIE register.

EU RESULT ADDRESS REGISTERS

The EU result address registers (shown in figure 3-2-19) are used to maintain the CDB location into which the result or temporary result of the operator is to be placed. The CDB location of the operator result is derived from the PCU, which uses PWR to write the proper CDB location into the EU OP queue for the operator being written into the queue. The CDB location of the temporary operator result is supplied by EUMC PROMs during the execution of EU microcode. As the operator is processed through the EU pipeline, the CDB location is passed down the pipeline to the EU level that is executing the operator at that time.

The EU result address registers are identified as follows:

1. EU result CDB address register (EPR).
2. CDB level result CDB address register (ECR).
3. Barrel level result CDB address register (ERB).
4. AIU result CDB address register (EAR).
5. SAU result CDB address register (ESR).
6. CDB EU write pointer register (CEW).
7. Interrupt read register (EIR).

EU Result CDB Address Register (EPR)

As each CDB address is read from the EU OP queue, it is saved in the EPR register. The saved address is only entered into the pipeline if microcode was required to complete an EUCDB or ALU operation. A new CDB address is loaded into EPR each time the conditions for the next operator transfer into EUCDB level are satisfied. EUCDB is not in hold, and microcode is not being issued to EUCDB, SAU, or ALU.

CDB Level Result CDB Address Register (ECR)

This register is the first EU result address register in the EU pipeline. Inputs to ECR are received from the EU OP queue, EUMC PROMs, and the EPR register.

Transferring into ECR from EU OP queue is enabled whenever EMCV flip-flop is reset to indicate that EU microcode is not being generated. When EU microcode is generated, the CDB location (W or X) is explicitly selected by EUMC PROMs.

Barrel Level Result CDB Address Register (ERB)

The ERB register receives the result address from the ECR register. The ECR contents are loaded into ERB in accordance with pipeline controls and EU barrel destination level End Of Current Operator flip-flop EBDE. (Loading ERB is inhibited if flip-flop EBDE is reset.)

ALU Result CDB Address Register (EAR)

The EAR register receives the result address from the ECR register if the transfer of CDB to ALU input bus (AI) is in progress; or from the ERB register, if the transfer of barrel output to the AI bus occurs. EAR is loaded with AI bus contents when both the AI bus to the ALU flip-flop EAIA and the ALU destination level End of the Current Operator flip-flop EADE are set.

SAU Result CDB Address Register (ESR)

As the EAR register does, the ESR register receives the result address from the ECR register if the transfer of CDB to SAU input bus (SI) is in progress. Or, it is received from the ERB register if transferring barrel output to the SI bus occurs. ESR is loaded with SI bus contents when the SI bus to the SAU flip-flop is set.

CDB EU Write Pointer Register (CEW)

When writing into the CDB location (EU data file, working storage, or X storage), the area is selected by the contents of CEW. The contents of CEW are decoded to set valid bits for EU data file locations 20 through 2B. There are 12 valid bit flip-flops JEV, one for each of the EU locations. The JEV flip-flop associated with the EU data file location just filled

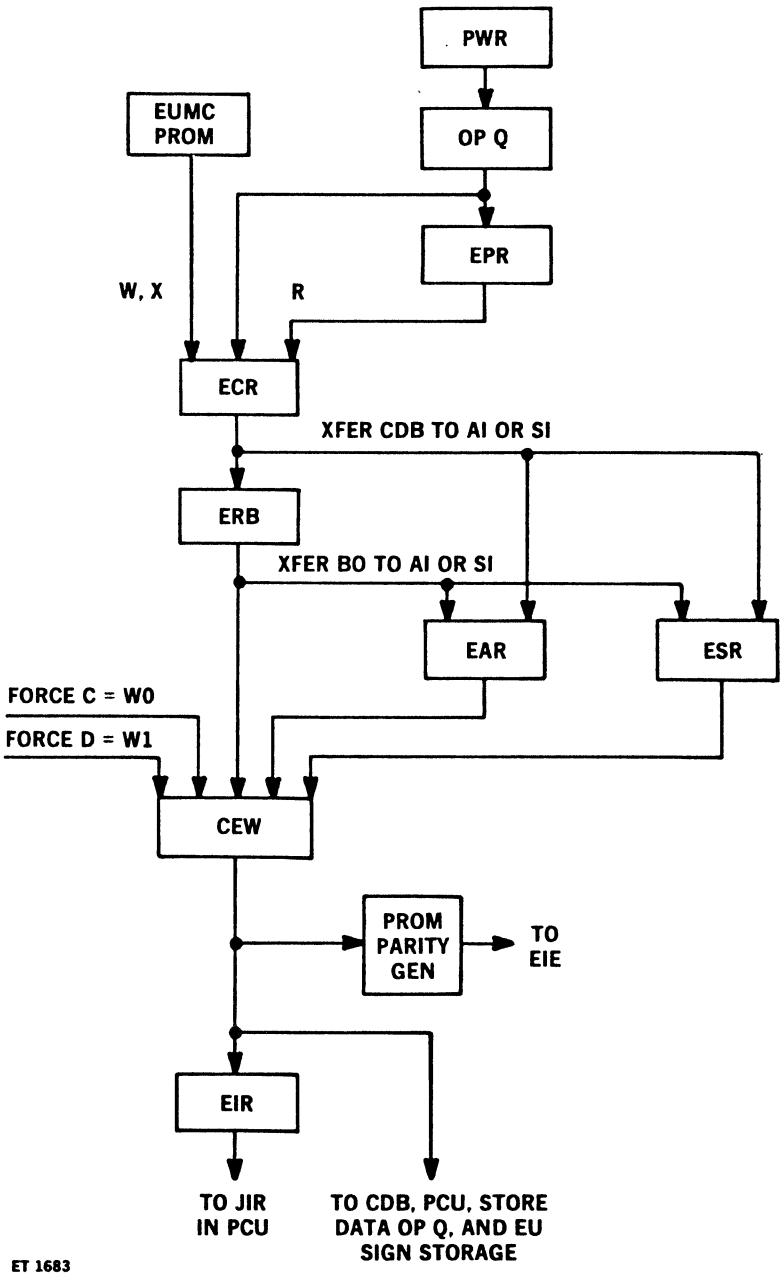


Figure 3-2-19. EU Result Address Registers

is set to record that the location has valid data. This location is now ready to be read by the EU as an input for the next EU operator or by the DRU as and input for indexing descriptors.

The order of priority for transfer into CEW is EAR, ESR, and ERB. These transfers are based upon the EW load priority for the corresponding units (ALU, SAU, and barrel).

As shown in figure 3-2-19, a Hex C or Hex D (for addressing W0, or W1 local storage locations, respectively) is forced on the input of CEW.

In the W0 address case, the decimal overflow digits that could have resulted from any adder pass are saved in W0 during scale left operations. Also, the remainder, which could result from executing the first DP divide cycle, is saved in WOLS (an EU data file RAM location) and W0 for use when the second DP divide cycle is started.

For W1 address case, the intermediate result of an operator is temporarily saved in W1 for interrupt information. However, in most cases, the data in W1 is replaced by new data before an interrupt is reported in the EU.

The PROM parity generator is used to check parity on the addresses when they are loaded (or forced) into CEW. The CEW parity is checked any time the CEW valid bit is set and write inhibit is not set.

Interrupt Read Register (EIR)

The EIR register is loaded with the result address from CEW as long as an interrupt job is not requested by the EU. The transfer of EIR to job interrupt register JIR only occurs when the associated job number of the result address is accepted into the JIJ register.

EU JOB NUMBER REGISTER

The EU job number registers (figure 3-2-20) contain the job number of the operator in process. As the operator is processed through the EU pipeline levels, the job number is transferred to the job number register at that level. The job number registers are:

1. CDB Level Job Number register (ECJ).
2. Barrel Level Job Number register (EBJ).
3. ALU Level Job Number register (EAJ).
4. SAU Level Job Number register (EWJ).
5. EW Job Number register (ESJ).
6. Interrupt Level Job Number register (EIJ).

The EU pipeline conditions for loading and transferring these registers are identical to those EU pipeline conditions described for the EU result address registers (ECR, ERB, EAR, ESR, CEW, and EIR).

The PROM parity generator is used to check parity on the job numbers when they are loaded into EWJ. If bad parity exists, signal EWJP.ERH is generated and sets the alarm bit in the EU interrupt error register EIE.

EU STORE SUBUNIT

The purpose of the EU store subunit is to execute transfers from the CDB to memory data register RMD, as require by the STOD and STON operators. The EU is therefore relieved of handling these transfers and a significant time gain is realized. The EU store subunit provides a queue that is completely separate from the PCU, with a separate EU store job number register and CDB address register. Therefore, the STOD and STON operators do not affect the timely execution of subsequent operators. This feature is especially important for the smooth functioning of loop control and index computation in the SAU.

The transfer from execution write register EW to RMD is attempted when the STOD or STON operator immediately follows the operator that calculates the data to be stored. The transfer occurs if all older operators are finished and RMD is ready when the final single-precision result is in the EW register. This advances the job number more rapidly to the next store or branch operation, and requires no EU time. If the store data must be read from the CDB (through the EU bus), the EU store subunit does not capture EU read register CER until the store is the oldest operator and the store address list (SAL) is ready. The EU is unaffected unless it simultaneously attempts to read data from the CDB.

As shown in figure 3-2-21, the EU store subunit has a queue that is completely separate from the PCU, with a separate EU store job number register EDJ and CDB address register EDB, and a store counter register ESCT. The ESCT register is used to provide commands for controlling EU store operations.

Addressing of the EU store queue for both write and read operations is accomplished by PSQ and ESQ pointers. To write into the queue, one of the PSQ signals is generated to select the location and effect the write operation. The count in the PSQ is increased by 1, whenever a STON or STOD is at the execute level of the PCU, provided that PCU PROC internal or PCU restart is not in process and no PIE hold exists. The ESQ pointer is counted so that the

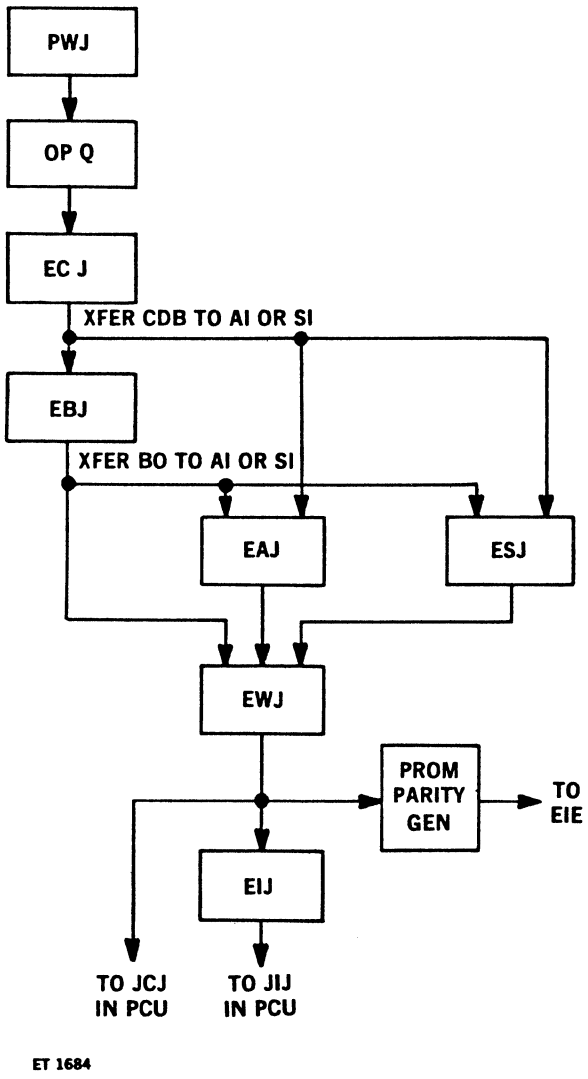


Figure 3-2-20. EU Job Number Registers

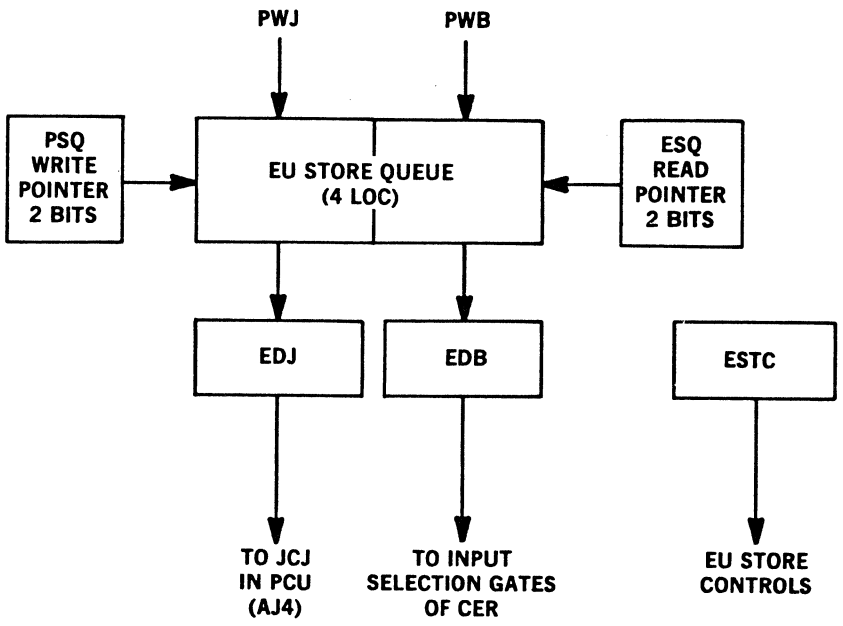


Figure 3-2-21. EU Store Subunit

job number and CDB address are available at the output of the queue when the EU store unit is idle and the queue has information (not empty) or when EU store job just read is in process and the queue has another job.

CENTRAL DATA BUFFER

The central data buffer (CDB) consists of two 64-location memories that are used for passing data between units of the CPM and for storing results of operators. The CDB holds the first four top-of-stack locations for each of up to four operators in the pipeline; although this capability is generally not fully utilized, there are usually less than four operators active at a given time. Two copies of the CDB are used. One handles data read by the DRU, and the other provides storage for data read by the EU. In this way, both the DRU and EU can read data while the CDB is being written. The write data is supplied by the DRU, the EU, and the PCU. The two copies of the CDB (for handling DRU storage and EU storage) are described in the following paragraphs.

DRU Data Storage

The storage of DRU data in the CDB is shown in figure 3-2-22. As indicated, the DRU copy of the CDB can be functionally divided into two major areas. One area consists of data files written by the DRU, EU, and PCU; the second area consists of address files for DRU data and string information. The data and address files in this copy of the CDB can be read only by the DRU, but can be written to by the DRU, EU, and PCU. The files are addressed by four groups of address lines: DRU write (CRW), DRU read (CRR), EU write (CEW), and PCU write (CPW). The data and address files are described in the following paragraphs.

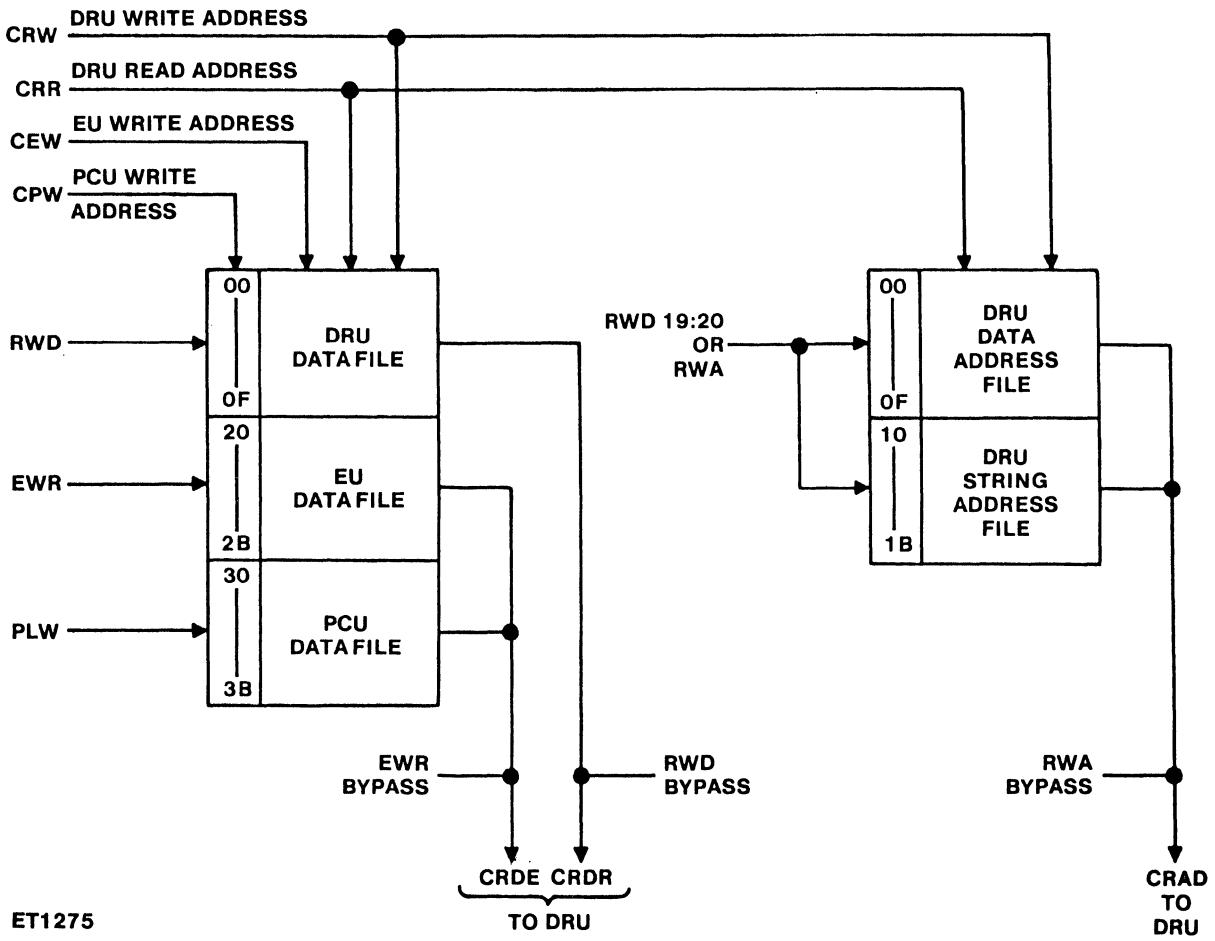


Figure 3-2-22. Central Data Buffer DRU Data Storage

Data Files

The DRU data file contains 16 locations, which are accessed by addresses 00 through 0F. This file is written to by DRU write level data register RWD; each location of the file can contain a 52-bit data word and two information bits. The portion of the 52-bit word is contained in bits 38 through 0; bits 50 through 39 on which an early read is performed (as described in later paragraphs). The remaining two bits are used for residue checking.

The information bits (L2 and L9) are used to define operand length fields. L2 signifies that the length is less than or equal to 20 bits, and L9 indicates that the length is less than or equal to 9 bits. An early read is also performed on these bits.

The EU data file contains 12 locations, accessed by addresses 20 through 2B. This file is written to by EU write level register EWR; each location of the file can contain a 20-bit operand, a parity bit, and two information bits (L2 and L9), on which an early read is performed. These bits are used to indicate that the operand length is either less than 20 bits (L2) or less than 9 bits (L9).

The PCU data file is a small lit file that contains 12 locations and is accessed by addresses 30 through 3B. The file is written to by PCU literal write register PLW; each location of the file can contain 20 bits: a 16-bit literal or address couple, a parity bit, a bit to indicate an IRW, and two information bits. An IRW is flagged by bit 48. When this bit is set, the address couple written to the PCU data file is identified as an IRW. This is done when a non-concatenated NAMC is detected by the PCU. The address couple in the NAMC is then placed in the PCU data file as an IRW. The information bits (L2 and L9) define the length of the literal as either less than 20 bits or less than 9 bits, respectively. Bits 48, L2, and L9 are subject to early read operations.

Address Files

The DRU address file contains the addresses of the words written to the DRU data file and the addresses for fetching the least significant word of double-precision operands. The addresses are written by DRU write address register RWA or the least significant 20 bits of write data register RWD into locations 00 through 0F, as addressed by the DRU write address line. Each location of the address file can contain a 20-bit address, two residue bits, a copy bit (CB), and a mom bit (MB). The copy bit is set to signify that copy action is to be taken for a present bit interrupt. References to non-present copy descriptors cause the mom descriptor to be fetched, using the base field of the copy descriptor. If the mom descriptor is present, its base field overwrites the non-present copy descriptor address saved in the DRU data address file and the mom bit is set. The base field of the mom descriptor is input via the 20 least significant bits of the RWD register.

The DRU string address file enables the DRU to construct descriptors that point to its associated data in the DRU string data file, which is located in the EU portion of the CDB. The string address file is also written to by DRU write address register RWA. The file consists of 12 locations (addresses 10 through 1B), each of which can contain a 20-bit address and two residue bits.

Bypass Functions

If any of the DRU data or address files is being written to while that address is being read, the information is written but is also supplied directly to the read bus from the associated write register. Therefore, if the address being written to by the RWD, RWA, or EWR register is equal to the address specified on DRU read address line CRR, the bypass function is invoked, and the information being written is read onto the CRDE, CRDR, or CRAD bus, as applicable.

Early Read Function

Before information is read from the data files, an early read operation is performed on certain bits and/or fields of the data word. The early read permits an evaluation of these flags to determine the action that should be taken to process the micro operators in the DRU. The early read function is performed on the L2 and L9 flags (discussed in previous paragraphs), bit 48 of the words contained in the PCU data file, and on bits 50 through 39 of DRU data words. These bits of the DRU data represent the tag field, descriptor control bits (such as present bit, copy bit, and index bit), and the descriptor size field.

EU Data Storage

The storage of EU data in the CDB is illustrated in figure 3-2-23. As indicated, this copy of the CDB can be functionally divided into three parts; each can be read by the EU only. The storages are addressed by the EU read (CER), DRU write (CRW), EU write (CEW), and PCU write (CPW) address lines; each carries six bits. The data and address files and storages of this portion of the CDB and their associated flags are discussed in the following paragraphs.

Flags

When information is written to the EU copy of the CDB, four-bit flags are generated for each location written to; these flags are examined when the locations are read. They are not stored in the CDB memories, but are placed in LFAN circuits. The flags indicate whether the information being read is an integer, a double-precision operand, or a floating point number; they also indicate integer length. They are provided by the DRU to enable the EU to properly process the data. The decodes of these flags are as follows:

Bit	3	1	0	Meaning
	0	0	0	Floating point number
	0	0	1	More than 20-bit integer
	0	1	1	Less than 9 bits
	0	1	0	Less than or equal to 20 bits, more than 8
	1	0	0	Non tag zero
	1	0	1	Double precision
	1	1	0	Set copy bit
	1	1	1	Copy action

Bit 2 = bit 46 of operand (sign bit).

DRU Data File

The DRU data file contains 16 locations; each are accessed by addresses 00 through 0F. The file is written to by DRU write data register RWD and each location of the file can contain a 52-bit data word. This file is used with the DRU data file RAMs for storage of the most significant words of double-precision operands. Words read from this file (and all other data files and storages of the EU copy of the CDB) are placed on the EU data bus, CED.

DRU String Data File

The DRU string data file contains 12 locations; each are accessed by addresses 10 through 1B. The file is also written to by DRU write data register RWD and it contains source and destination words to be used by the EU. As described previously, this storage is used with the DRU string address file located in the DRU data storage portion of the CDB.

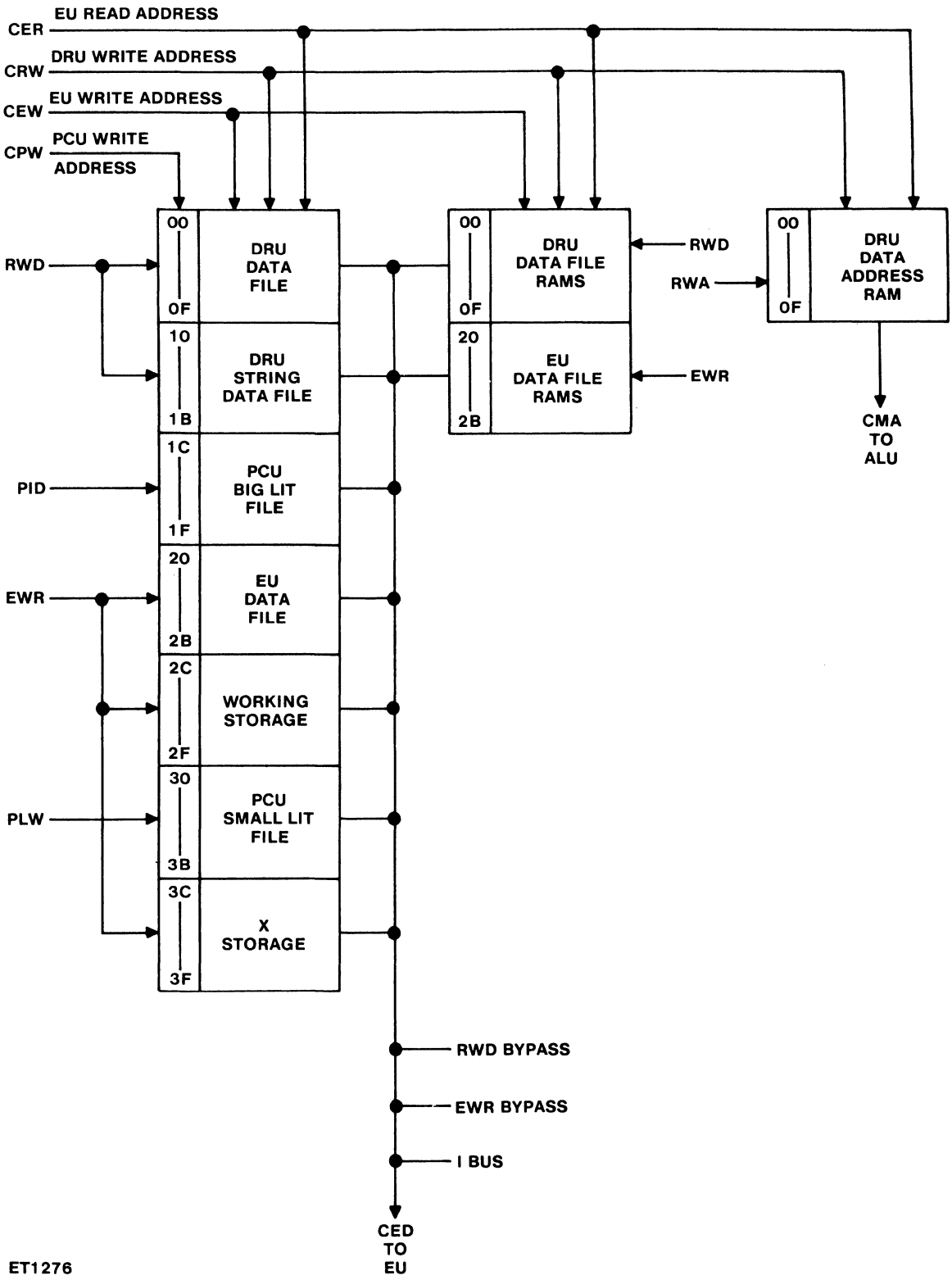


Figure 3-2-23. Central Data Buffer, EU Data Storage

PCU Big Lit File

The PCU big lit file contains four locations (1C through 1F); each are addressed by PCU write address line CPW. This file contains LIT48 and makes program control word (MPCW) operators transferred from PCU instruction decode register PID. The CDB locations are, effectively, a queue that is pointed to by the PCU for writing the operators. The locations read are determined by the locations that are allocated by the PCU; the information is then passed to the EU via the CED bus. For LIT48 operators, the EU passes the data from the CDB to a general location that is readable by the EU so that the LIT48 is available to any subsequent operator. When the EU reads out an MPCW operator, a seven tag is added and the stack number is inserted by the EU.

EU Data File

The EU data file consists of 12 locations; each are accessed by addresses 20 through 2B and contain 48-bit words. The file is written to by EU write level register EWR and contains result data of EU operators. The EU data file may contain the most significant half of double-precision operands.

Working Storage

The working storage area contains four locations (addresses 2C through 2F); each are written to by EU write level register EWR. These locations are used as a general work area for items such as source data and partial products, and provide temporary storage for interrupt parameters P1 and P2. Each location contains 48 bits.

PCU Small Lit File

The PCU small bit file consists of 12 locations (30 through 3B); each are written to by PCU literal write register PLW. Operators LIT0 through LIT16 and non-concatenated NAMC are executed entirely by the PCU, and the literal data is written to the CDB for use by the EU. The small lit file is 20 bits wide.

X Storage

The X storage area contains four locations (3C through 3F); each are written to by EU write level register EWR. These locations are used as a general work area for items such as delimiter characters for compare operators. 3F is reserved as EU copy of SNR. Each location contains 20 bits.

Data File RAMs

The data file RAMs are associated with the DRU and EU data files. They contain the same number of locations and have the same addresses as the corresponding data files, and are written to by the RWD and EWR registers. The purpose of the data file RAMs is to store the least significant words of double-precision operands.

DRU Data Address RAM

The DRU data address RAM consists of 16 locations (addresses 00 through 0F) that are written to by DRU write address register DWA. These locations contain the absolute addresses of data stored in the DRU data file. The data addresses are read out by EU read address line CER and are placed on the CDB memory address bus (CMA) for copy action by the ALU.

Bypass Functions

If any of the files and storages input by the RWD or EWR registers is being written to while the same address is being read, the information is written but is also applied directly to EU data bus CED. Therefore, if the address being written to by either the RWD or EWR register is equal to the address specified on EU read address line CER, the bypass function is performed.

I Bus

The I bus is used to enter information from hard registers (such as the processor fail register, control mode register, and time of day register) onto the CED bus. These registers are enabled onto the CED bus by decoding the contents of the EU variant register. (This register is loaded by the PCU with coded information that specifies the appropriate hard registers.)

STORE QUEUE

The major data paths within the store queue are queue are shown in figure 3-2-24. The purpose of the queue is to reduce the number of store operations to memory that must be made by the processor. Store operations are queued because of the possibility that there will be a subsequent store to the same address; if this condition arises, the earlier store is not performed. As indicated in figure 3-2-24, the store queue is a 32-deep buffer for store operations consisting of an address (RCA 19-0 from the DRU) and its associated data (RMD 51-0, also from the DRU). Addresses in the queue are compared to a new address eight at a time. If a match is found, the old address is removed from the queue. This operation is known as an invalidation check, which is described, along with other store queue operations, later in this section.

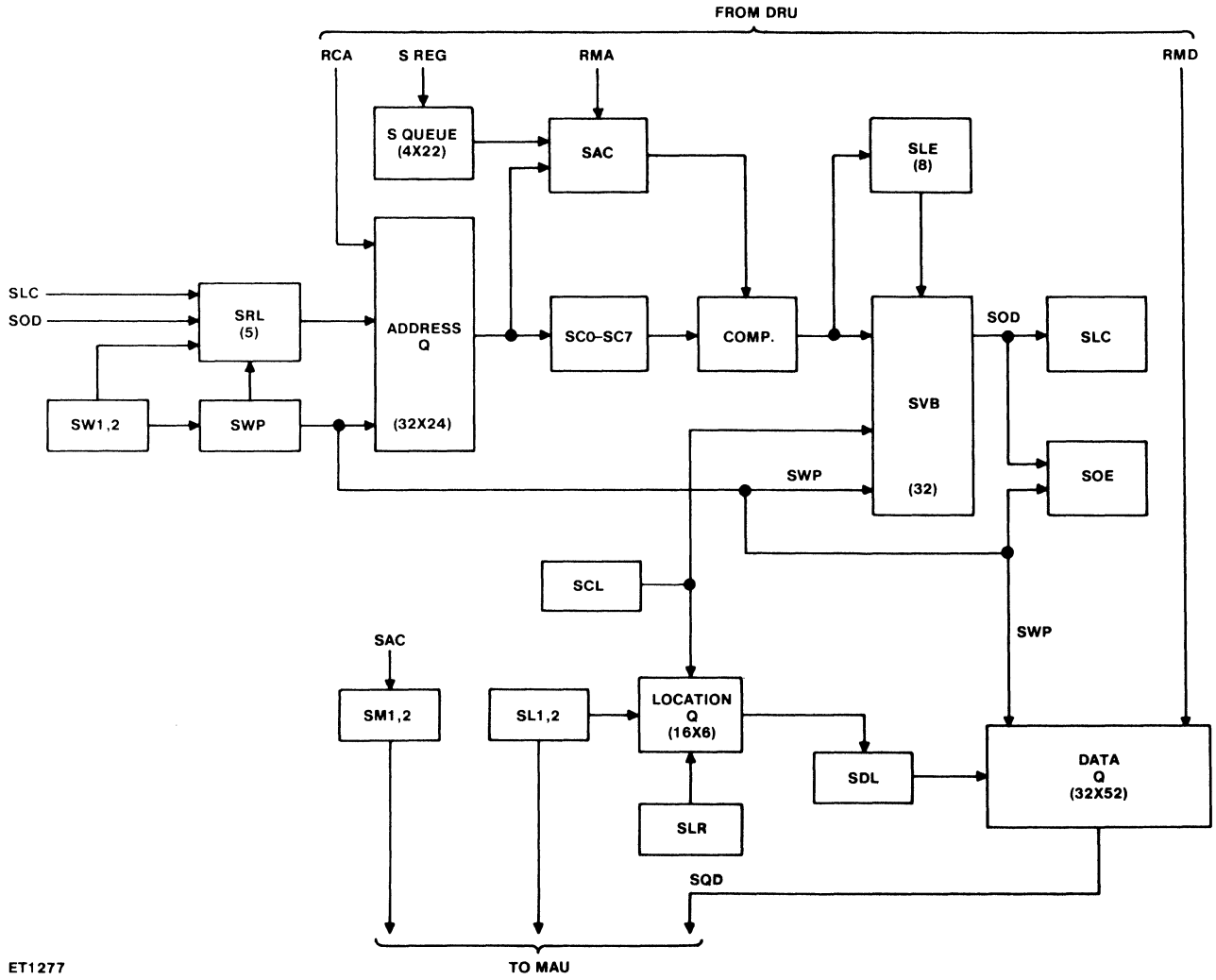


Figure 3-2-24. Store Queue, Block Diagram

When a procedure is exited, the stack (S) register is changed to a lower address. The temporary locations containing the variables required for the exited procedure can then be discarded. The stack cut back routine compares entries in the queue against the old and new S values, and deletes those that are less than or equal to the old S value and greater than the new S value.

The store queue also can create two eight-word jobs simultaneously for the MAU. Since all stores must be performed in the same order as in the original code stream, the make MAU job routine starts with the oldest entry in the queue and then examines subsequent store addresses. If the addresses are contiguous, they are grouped as a single n-word job and the store queue sends the starting address and length to the MAU.

When fetching operations are performed in the DRU, data is loaded into a particular block and group in the ASM. The store queue fill check routine determines whether a block and group just loaded had destroyed a location to be stored to. If so, the store queue begins storing until the location involved receives the new data. If this were not done, fetches for that location (which would go to main memory, since the location is no longer local) would return the old information.

A store to main memory is also initiated when the store queue is purged, which causes all store queue words, starting with the oldest entry in the queue, to be stored to memory. A store queue purge occurs whenever any of DRU operators SPRR62, SPRR63, FMMR, MVST, RDLK, SLMT, or SINH is executed. Note that if RDLK is preceded by FMMR, the store queue is not purged.

The store queue contains four individual queues, as follows:

1. Address queue. Composed of R44N chips; contains thirty-two 24-bit words organized as four eight-word blocks. Holds the store addresses input on the RCA lines from the DRU.

2. Location queue. Composed of R44N chips; contains 16 six-bit words organized as two eight-word jobs for the MAU. Holds the pointers to data to be stored.

3. S queue. Composed of R44N chips; contains four 22-bit words organized as two two-word groups. Holds old and new S values for each of two stack cut back jobs.

4. Data queue. Composed of RAM1 chips; contains thirty-two 52-bit words. Holds data associated with addresses contained in the address queue.

The queue performs four major jobs, as described in the following paragraphs.

Invalidation Check

The invalidation check is performed to possibly remove a previous store to the same address. Two pointers are used. The write pointer (SWP) points to the newest entry in the queue, and the oldest entry pointer (SOE) points to the oldest entry in the queue. When an address (RCA) and its associated data (RMD) are received by the queue, the write pointer is incremented, the address is written into the address queue, and the read pointer (SRL) value is made equal to that of the last checked pointer (SLC) plus 1. Successive blocks of eight words are read out of the address queue and into a comparison network. The newest entry is compared to the older entries. When a comparison is found between the newest entry and an entry in the block read from the address queue, the corresponding bit in the SVB register is reset to invalidate the older entry.

Stack Cut Back

The stack cut back function compares entries in the address queue to the old and new S values, and deletes those that are less than or equal to the old S value and greater than the new value. The old and new S values are stored in the S queue, which can contain the values for two stack jobs. The value in SWP (at the time when the exit had been stored in either SW1 or SW2) is used as the start location in the address queue for this job (similar to SWP in invalidation check).

Two comparisons are made for each block of eight addresses. The first comparison is against the old S value. Addresses less than or equal to the old S, set corresponding bits in SLE. The second comparison is against the new S value. If an address is greater then the new S and its corresponding SLE bit is set, that address is effectively removed from the queue by resetting its valid bit.

Make MAU Job

The store queue can create two jobs for the MAU. Since all of the stores must be accomplished in the same order as the original code stream, the make MAU job function starts with the oldest entry in the queue and examines subsequent store addresses. If these addresses are continuous, they are grouped as a single job, and the store queue sends the storage address and length to the MAU.

When a make MAU job is requested, read level location pointer SRL is loaded with the oldest entry decode (SOD), and an eight-word block of address is loaded into output registers SC0 through SC7. The specific address pointed to by the three least significant bits of SRL is then compared to the contents of address compare register SAC. At the same time, the compare level location pointer (SCL) contains the identical address, which is passed to the location queue. An equal compare increments either of length registers SL1 or SL2, which doubles as the write pointer to the location queue. In addition, one of the memory address registers (SM1 or SM2) is loaded with the first (starting) address. Then, SAC is incremented, and compared against the valid location in the address queue. Each time an equal comparison occurs, the location queue is loaded by the SCL register, and SL1 or SL2 is incremented. This process continues until either an unequal compare occurs, the queue becomes empty, or the maximum number (8) of contiguous addresses have been found. The location queue is read into data location pointer SDL by read pointer SLR, and the associated data is then taken by the MAU via the SQD output. Each time the MAU takes data, it increments the SLR to obtain new data.

Fill Check

The address array of the ASM can contain four addresses with the same block assignment. (A block is defined as 1 of 64 (0 through 63) blocks, each block contains four addresses.) When the DRU fetches data from main memory, the data is loaded into one of four eight-word groups in the data array of the ASM. (A group is defined as one of four locations in a particular block (0 through 63) of the data array.)

If four addresses with the same block assignment are loaded in the address array and a fifth address is to be loaded, the latter address must push out the

oldest address in that block. If the push-out address was loaded because of a store prefetch, the push-out address will be in the store queue. After this address is invalidated in the ASM, the only place it will be is in the store queue. Therefore, any subsequent fetches to that address are declared non-local in the ASM and a main memory fetch is required, where the old data resides, not the new data that is in the store queue. To prevent this condition from occurring, a fill check is performed in the store queue. The fill check compares the push-out block and group against entries in the address queue. When a comparison is made, the store queue is emptied up to and including the block where the equal was found by causing MAU jobs to be made. This action occurs before any load of the ASM takes place.

MEMORY ACCESS UNIT

The Memory Access Unit (MAU) consists of thirteen functional sections interfaced as shown in figure 3-2-25. A description of each of the functional sections is given below:

1. The priority resolver logic is responsible for granting the services of the MAU to the highest priority requesting unit. The order of priority for service is: a) data reference unit (DRU); b) program control unit (PCU); and c) storage queue (SQ). However, a fetch interrupt request or store re-request has priority over an initial fetch for DRU or PCU.
2. The fetch address register is used to buffer all memory address transfers from the requesting units (DRU and PCU) and to buffer the memory address update for each word fetched from memory. The contents of this register are routed to CW register for constructing the memory control word and to limit comparator for selecting the MCM. Also, the three least significant address bits of the fetch address register are routed to the DRU for identifying word location in the ASSM that is to receive the memory word.
3. The store address register is used to buffer all memory address transfers from the storage queue and to buffer the memory address update for each word stored to memory. The use of the contents of this register is the same as the fetch address register.
4. The length fetch register is used to buffer all word length transfers from the requesting units (DRU and PCU) and to buffer the word fetched from memory. The contents of this register are routed to CW register to construct the memory control word and to control logic for executing the update of the fetch address and fetch length registers and for determining job termination.
5. The length store register is used to buffer all word length transfers from the storage queue and to buffer the word length update for each word stored to memory. The use of the contents of this register is the same as the length fetch register.
6. The limit comparator and decode logic is responsible for comparing the six most significant bits of address in the fetch address or store address register with address limits supplied by each MCM in the memory system. The output of the comparator is decoded to select one of eight memory buses.
7. The bus address fetch and bus address store registers are used to buffer one to eight possible memory bus signals from the limit comparator and decode logic.
8. The control word register is used to assemble memory control word for transfer to selected MCM. It is also used to assemble an error word for transfer to requesting unit if a memory related error is detected by the MAU during a fetch or flashback operation.
9. The output register is used to buffer data transfers from requesting unit of CPM to memory.
10. The input register is used to buffer data transfers from memory to requesting unit.
11. The parity check and generate logic is required to generate odd parity for all words being transferred to memory and to check for odd parity of all words being fetched from memory. Also, this section is required to generate odd parity for each of the six syllables of the program word being transferred to PCU.
12. The residue check logic is responsible for checking and verifying the residue bits of the memory addresses transferred from the requesting units.

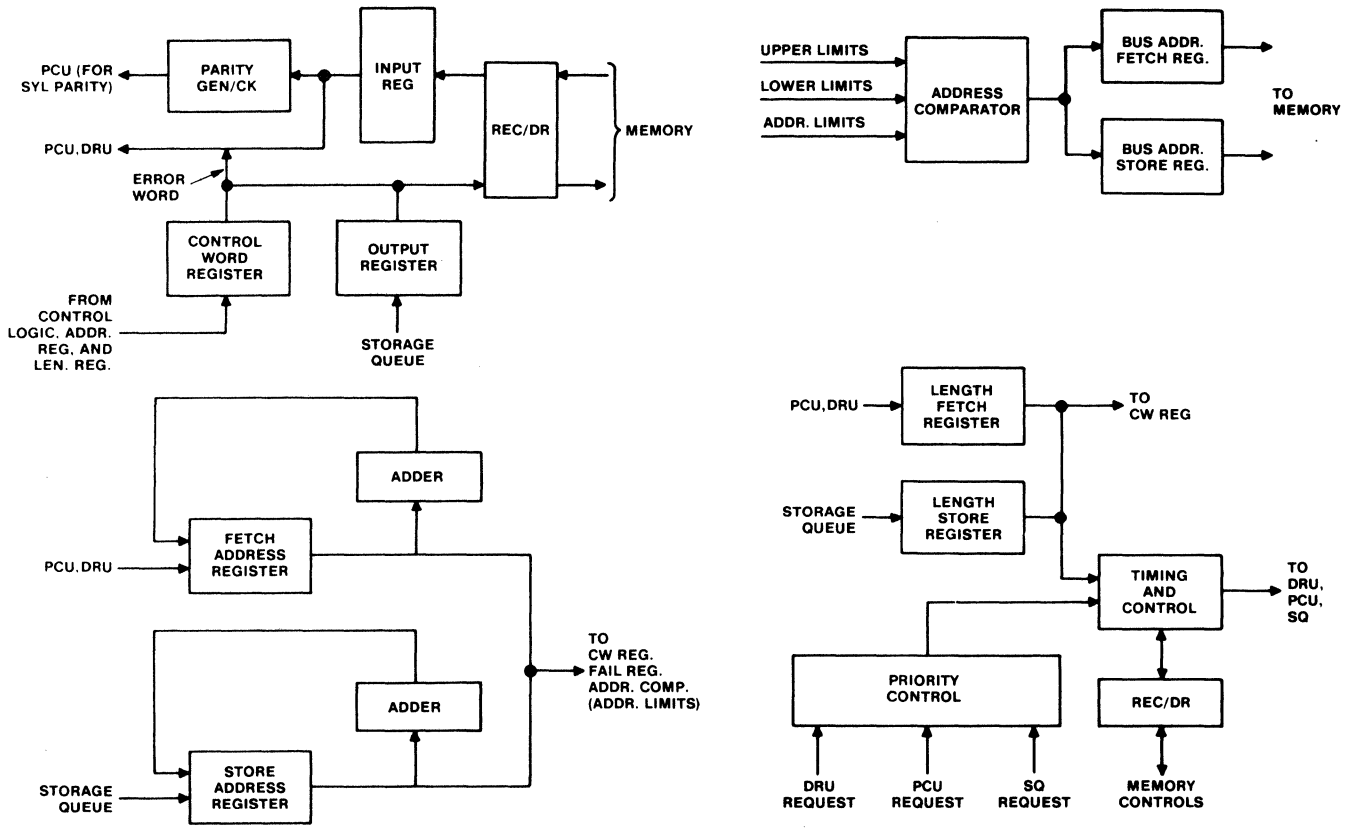
When a requesting unit of the CPM requires the services of the MAU, it is required to raise its request line to the MAU and place a 20-bit address (plus residue), length information, and control information on its interface lines to the MAU.

When the requesting unit has priority, the MAU loads the above information into its respective registers and control flip-flops and performs one of the following operations:

1. **Single data word fetch.**
2. **N-length data word fetch.**
3. **N-length overwrite.**
4. **Single-word overwrite.**

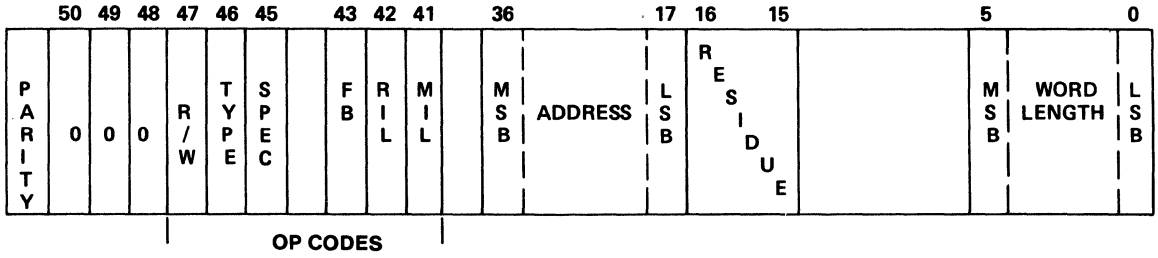
Upon determining the type of operation requested, the MAU constructs a memory control word (MCW) and transfers it to memory. (See figure 3-2-26 for the format of the MCW.) Upon transferring the MCM to memory, the MAU is required to perform one of the operations listed below:

1. If a single word store operation was specified, the MAU raises its request line to the specific MCM. In order to alternately transmit the MCW and the data word to be stored, the MAU continues to transmit the MCW, followed by the data word to be stored, until an acknowledge signal is received from the MCW.



ET1281

Figure 3-2-25. Memory Access Unit, Block Diagram



47	46	45	43	42	41	OPERATION
0	0	1	0	0	0	SINGLE DATA WORD FETCH
0	0	0	0	0	0	N-LENGTH DATA WORD FETCH
0	1	1	0	0	0	FAIL REGISTER FETCH
1	0	1	1	0	0	SINGLE-WORD OVERWRITE WITH FLASHBACK (READ WITH LOCK ONLY)
1	0	0	0	0	0	N-LENGTH OVERWRITE
1	0	1	0	0	0	SINGLE-WORD OVERWRITE
1	0	1	0	1	0	LOAD REQUESTOR INHIBITS
1	0	1	0	0	1	LOAD MEMORY LIMITS

ET1285

Figure 3-2-26. Control Word Format

2. If a multiple word store operation is specified, the MAU raises its request line to the applicable MCM, and then sends the MCW to the MCM. When the MCM acknowledges receipt of the MCW, the MAU begins the data transfer under the control of the send data signal.

3. If a fetch operation is specified, the MAU raises its required line and sends the MCW to the applicable MCM. When the MCM acknowledges receipt of the MCW, the MAU enables its memory receiver circuits. Information from the MCM can now be accepted by the MAU. However, the MCM is required to transmit a data present strobe to the MAU, causing the information present on the memory bus to be transferred to and detected by the MAU. The data present strobe is required for each word transferred from memory to the MAU.

While performing a data transfer the MAU is required to detect and/or report the following memory error conditions:

1. Invalid address (IVE) is declared if the address in the store address register compares with the address limits of more than one or no MCM. (This error condition causes termination of the memory access operation.)

2. Residue address error (RAE) is declared if the MAU receives a requesting unit address whose residue bits do not agree with its address configuration.

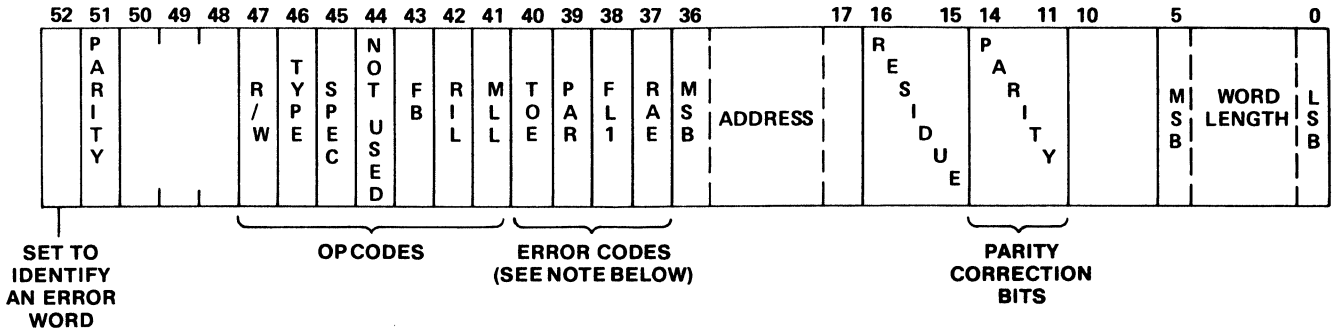
3. Parity error fetch (PEF) is declared if a fetch of data from MCM is received by the MAU with incorrect parity.

4. Parity error store (PES) is declared if a data transfer from the storage queue is received by the MAU with incorrect parity. The data with incorrect parity is transferred to the MCM.

5. Time out error fetch (TOF) and time out error store (TOS) are declared if the MAU receives no acknowledged signal (ACK) from the MCM during a writing period not to exceed two milliseconds. (These error conditions cause termination of the memory access operation.)

6. Fail 1 fetch (F1F) and fail 1 store (F1S) are declared when the selected MCM responds with a fail 1 (uncorrectable error) indication to the CPM.

If a memory related error is detected by the MAU during a fetch or flashback operation, the word, which is involved at the time the failure, is replaced by an error word. This error word, which is a copy of the control word, is contained in the control word register, then transferred with other related information to the unit (PCU and DRU) that is to receive the expected memory word. (See figure 3-2-27 for format of the error word.)



OP CODES

47	46	45	43	42	41	
0	0	1	0	0	0	SINGLE DATA WORD FETCH
0	0	0	0	0	0	N-LENGTH DATA WORD FETCH
0	1	1	0	0	0	FAIL REGISTER FETCH
1	0	1	1	0	0	SINGLE-WORD WITH FLASHBACK (READ WITH LOCK ONLY)
1	0	0	0	0	0	N-LENGTH OVERWRITE
1	0	1	0	0	0	SINGLE-WORD OVERWRITE
1	0	1	0	1	0	LOAD REQUESTOR INHIBITS
1	0	1	0	0	1	LOAD MEMORY LIMITS

ERROR CODES					PARITY CORRECTION BITS			
40	39	38	37		14	13	12	11
1	0	0	0	TIMEOUT ERROR	1	0	0	0
0	1	0	0	WRONG PARITY ON A FETCH FROM MEMORY	0	1	0	0
0	0	1	0	FAIL 1 INTERRUPT	0	0	1	0
0	0	0	1	ADDRESS RESIDUE ERROR	0	0	0	1

NOTE: THE ERROR CODES AND CORRESPONDING PARITY CORRECTION BITS ARE APPLIED DIRECTLY TO CW REGISTER.

ET1288

Figure 3-2-27. Error Word Format

SECTION 3

INTERRUPTS

INTRODUCTION

An interrupt is a means of diverting a processor temporarily from the job which it is doing if certain predetermined conditions occur, so that some higher priority job may be done immediately. Interrupts are processed by the interrupt handling mechanism of the MCP. When the interrupt has been processed, the MCP will (if conditions permit) reactivate the interrupted process.

The interrupt handling mechanism of the MCP deals with two classes of interrupts: hardware interrupts and software interrupts. Hardware interrupts are generated automatically by the B 7800 system (when interrupt conditions occur) and are processed by the MCP interrupt procedure. Software interrupts are programmatically defined, and are used both by the MCP and by object programs for communication between processes. This discussion deals only with hardware interrupts.

HARDWARE INTERRUPT SYSTEM

The B 7800 hardware interrupt system is a primary interface between the MCP and the hardware. Interrupt conditions may be detected by the Central Processor Module (CPM), the Input/Output Module (IOM), or the Memory Control Module (MCM). When detected, interrupt conditions are processed by the Fault Control Logic of the CPM. Normally, the CPM prepares the stack for procedure entry, places the necessary parameters in the stack, and causes an entry into the MCP interrupt procedure.

CPM STATES AND MODES

The CPM operates in either of two states: control state, used only by the MCP; or normal state, used both by user programs and by the MCP. Normal state is always used when executing user programs. Control state is used when executing certain portions of the MCP, including the MCP interrupt handling procedure. In the control state, External interrupts and Interval Timer interrupts are inhibited (except during an IDLE or PAUS instruction) and the CPM may execute privileged instructions which it may not execute in normal state.

In addition to the two states, the CPM can be in any one of four interrupt handling modes: Normal Mode (Control Mode 0), Control Mode 1 (CM1), Control Mode 2 (CM2), and Control Mode 3 (CM3). The CPM operates in normal mode until an interrupt condition is detected. Control modes 1 through 3 allow for recursive attempts to enter MCP interrupt handling procedures by the fault control logic of the CPM. The CPM halts if these attempts not successful. The CPM will return to Normal Mode if an interrupt condition is handled successfully in CM1, CM2, or CM3.

There is no direct connection between the states of operation and the modes of operation of the CPM. The CPM may be in control or in normal state while in any control mode. In a system which contains more than one CPM, any or all of the CPMs may operates in control state or normal state, as well as in any of the interrupt modes. The CPM states are described below; the interrupt modes are further described in the discussion of interrupt processing.

Control State

Entry into control state (from normal state) occurs when the MCP enters or returns to a control state procedure (an MCP SAVE procedure), or when the CPM executes a Disable External Interrupts operator (DEXI). (Control state procedures have bit 19, the N bit, of the PCW set.) While the CPM is operating in control state the reporting of external interrupts to the MCP interrupt handling routine is disabled. Additionally, the CPM may execute certain privileged operators while in control state which may not be executed in normal state. When the CPM is operating in control state the control state flip-flop (XPST) is set and an inhibit interrupt condition is enabled (except during an IDLE instruction).

The interrupts which are inhibited while in control state include the Channel interrupts, the IOM Error interrupts, and the Interval Timer interrupt. Although the processing of these interrupts is inhibited, the appropriate bit in the CPM Interrupt (Fault) register will be set if one of these interrupts is detected, and the interrupt will be processed when the CPM enables External interrupts either by returning to normal state or by executing an IDLE or PAUS operator.

The Egg Timer interrupt and PROC to PROC alarm bits, although External interrupts in priority, is not inhibited in control state.

The operators which are enabled only when the CPM is in control state include Set Interval Timer (SINT), Inhibit Parity (IGPR), Set Memory Inhibits (SINH), and Set Memory Limits (SMLT). Also, B 7800 operators TCOD, SMMD, and ACDB are enabled when CPM is in control state.

Normal State

Return to normal state (from control state) occurs whenever the MCP initiates or returns to a normal state procedure (non-SAVE procedure), or when the CPM executes an Enable External Interrupt operator (EEXI). (Normal state procedures have bit 19, the N bit, of the PCW reset.) When the CPM is operating in normal state, the processor state flip-flop (XPST) is reset. When a CPM returns to normal state after servicing an interrupt, it does not necessarily return to the program which was executing when the interrupt was detected. The selection of the job to be run is a function of the MCP.

FAULT CONTROL LOGIC

The fault control logic of the CPM contains four registers which are used to record and process hardware interrupts: the Fault register, Fault Mask register, Processor Fail register and Control Mode register. The Fault register is used to indicate the detection of one or more interrupt conditions (one bit for each condition). The Fault Mask register is used to inhibit (mask out) the processing of one or more interrupt conditions. (The Fault register may be read in such a way as to obtain only interrupt conditions which are not masked out; thus indicating an interrupt condition which must be processed by the MCP.) The Processor Fail register further identifies errors which are internal to the CPM and CPM-MCM interface errors. The Control Mode register is used to identify the interrupt mode (Normal, Control Mode 1, Control Mode 2, and Control Mode 3) in which the CPM is operating.

In addition to the CPM registers, a Memory Fail Register in each Memory Control Module (MCM) is used to give detailed information about memory-related failures concerning that MCM. (Discussion of the MCM fail register is given in Chapter 5 of this manual.) For IOM error interrupts, detailed information about the IOM failure is given in Chapter 4 of this manual.

Fault Register

The Fault register contains one bit each of the possible interrupt conditions. (See table 3-3-1.) The low order bits of the register are associated with interrupts which have the highest priority for processing by the CPM; the high order bits are associated with interrupts which have the lowest priority. When interrupt conditions are detected, the bits associated with those conditions are set in the Fault register.

Normally, the Fault register is set by the interrupt condition. As each interrupt condition is processed, the bits in the register are selectively reset. Programmatic control of the Fault register is accomplished by use of the Set Processor Register (SPRR) and Read Processor Register (RPRR) operators. The RPRR operator causes the contents of the register to be placed in the stack, and the register itself to be reset. The SPRR operator causes an inclusive OR setting of the register; that is, bits are set, but bits which already are set are not reset.

Fault Mask Register

The Fault Mask register allows the processing of certain interrupts to be inhibited or deferred. Alarm interrupts, Syllable interrupts, and the Egg Timer interrupt may not be masked. The Special interrupts and the other External Interrupts have a corresponding bit in the Fault and Fault Mask registers (table 3-3-1). An interrupt condition will only be recognized by the CPM if the Fault Mask register bit for that condition is set (logical one). If the Fault Mask bit is reset for an interrupt condition, that interrupt bit will still be recorded in the Fault register but will go unnoticed by the fault control logic. If the mask configuration is later changed, then interrupts (including those resident in the Fault register when the mask is changed) which are now unmasked will be recognized and processed. In this way, processing of selected interrupts can be deferred.

The Fault Mask register may only be set programmatically. The Read Processor Register operator causes a simple read of the register (without reset); the Set Processor Register operator causes a simple set of the register (each bit is set either to logical one or to logical zero).

Interrupt Identification

Each interrupt condition reported to the MCP is identified by a unique literal value, known as interrupt parameter P1. (See table 3-3-1.) This parameter is passed to the MCP interrupt procedure by the fault control logic to identify the condition which is to be processed. The P1 parameter is derived from the contents of the Fault and Fault Mask registers through a series of gates. Interrupt conditions reported in the Fault register which are not masked out by the Fault Mask register are used to make up the P1 parameter.

Normally, this parameter is read and placed into the stack by the fault control logic, although it may be read into the stack programmatically. In either case, the resultant action is as follows. The value of P1 is read into the stack and the bits which were set in P1 are reset in the Fail register. In a particular P1 parameter, all interrupts of a particular priority level

Table 3-3-1. B 7800 Interrupt Bit Assignments

Interrupt	Fault Register (Bit)	Fault Mask Register (Bit)	Interrupt Ident. (P1) (Bit)	
Alarm (First Priority)				
Loop (LOP)	0	N	0	
Memory Parity (MPA)	1	0	1	
Memory Fail 1 (FL1)	2	N	2	
Invalid Address (No Access) (NAM)	3	E	3	Plus Bit 25
Stack Underflow (SKU)	4		4	
Invalid Program Word (IPW)	5		5	
Processor Internal (PI)	6		6	
Syllable (Second Priority)				
Memory Protect (MPR)	9		0	
Invalid Operand (NVO)	10		1	
Divide By Zero (DBO)	11		2	
Exponent Overflow (XOV)	12	N	3	
Exponent Underflow (XUN)	13	0	4	Plus Bit 24 or
		N		Bit 23 (See Note)
		E		
Invalid Index (NVX)	14		5	
Integer Overflow (NTO)	15		6	
Bottom of Stack (BSK)	16		7	
Presence Bit (PB)	17		8	
Sequence Error (SEQ)	18		9	
Segmented Array (SEG)	19		10	
Programmed Operator (PGO)	20		None	
Privileged Instruction (PVO)	21		11	
Special				
Stack Overflow (Third Priority) (SKO)	24	24	1	
Interval Timer (Fifth Priority) (INT)	23	23	0	Plus Bit 22
External (Fourth Priority)				
Channel 0 (C0)	26	26	0	
Channel 1 (C1)	27	27	1	
Channel 2 (C2)	28	28	2	
Channel 3 (C3)	29	29	3	
Channel 4 (C4)	30	30	4	
Channel 5 (C5)	31	31	5	
Channel 6 (C6)	32	32	6	
Channel 7 (C7)	33	33	7	Plus Bit 21
IOM Error 0 (E0)	34	34	8	
IOM Error 1 (E1)	35	35	9	
IOM Error 2 (E2)	36	36	10	
IOM Error 3 (E3)	37	37	11	
CPM Error 4 (E4)	38	38	12	Also responded to in control state.
CPM Error 5 (E5)	39	39	13	
CPM Error 6 (E6)	40	40	14	
CPM Error 7 (E7)	41	41	15	
Egg Timer	None	None	None	

NOTE

On syllable interrupts ID Bit 24 indicates class 1 interrupt (PIR, PSR, PSDI, PBR have not been modified, ID Bit 23 indicates class 2 (PIR, PSR, PSDI, and PBR are undefined).

which are not masked out are reported, but only one priority level is reported on each read. The priority level reported will be the highest priority level for which there is at least one bit set in the Fault register which is not masked out. If the value of the P1 parameter is read programmatically (using the Read Processor Register operator), and if there are no unmasked interrupts to report, a word of all 0's is read into the stack. (The fault control logic will read P1 only when there is an unmasked interrupt to report.)

Processor Fail Register

The Processor Fail register (table 3-3-2) provides specific information about processor internal interrupts. The type of processor internal interrupts is identified by one of twelve bits in the Processor Fail register. These bits are Store Queue Error (SQ), DRU Write Level Interrupt (RWI), DRU Evaluate Interrupt (REI), EU Subunit Error (EU1, EU0), EU CDB Parity (ETO), General EU parity Error (ET1), PCU PROC. INT (PCU), PIR Residue Error (PIR), Job Number (JOB), MAU Residue Error (MAR), and MAU Store Parity Error (MSP). Usually only one of these bits is set for a given internal error. However, it is conceivable to have a MAR error occur during a memory fetch operation and then to have it replaced in the Processor Fail register by another failure in a subsequent processor operation. It can be determined that a MAR error occurred during a fetch and was the cause of the processor interrupt by observing that P2 interrupt parameter contains a MAU error word in which bit 37 is on. (See figure 3-2-28.)

The Processor Fail register contains several bits which are useful in analyzing MAU detected errors. Whenever the MAU detects an error, the MAU loads the Box number, the most significant six bits

Table 3-3-2. Processor Fail Register

Processor Fail Register (Bit)	Description
3:4	Box Number (BN)
9:4	Memory Address (MA)
11:2	MAU Requestor: MU1 MU0 0 1 = STORE QUEUE 1 0 = DRU 1 1 = PCU
12	MAU Residue Error (MAR)
13	MAU Store Parity Error (MSP)
14	MAU Single Error (MSE)
15	MAU Wrong Channel
16	Spare
17	Store Queue Error (SQ)
18 & 21:2	DRU Write Level Interrupt (RWI) and Error Type (RT1, RT0): RT1 RT0 0 0 = Parity error from compare level categories, or parity error from RWJ, RWR, or RWQ, or residue error from RCD (39:20) or RCD (35:16) or MAU reports processor internal (from control word). 0 1 = Residue error from RWA. 1 0 = Parity error from RWD. 1 1 = Parity error from compare level command or variants, or more than one copy of an address in the address array (loaded into R0C, R1C, R2C, and R3C).
19 & 21:2	DRU Evaluate Interrupt (REI) and Error Type (RT1, RT0): RT1 RT0 0 0 = Parity error from RDB, RSB, or routine table categories. 0 1 = Residue error from RED, REA, or REL. 1 0 = Parity error from RED or REL. 1 1 = Parity error from operator or routine level categories, or parity error from routine level operator, or parity error from CRRR or CRRE, or parity error from RRR (routine), or parity error from routine hold and end categories, or parity error from AC operator (RACQ), or parity error from address couple categories.
23:2	EU Subunit Error (EU1, EU0): EU1 EU0 0 1 = SAU 1 0 = ALU 1 1 = Barrel
25:2	EU Parity Error (ET1, ET0) ET1 ET0 0 1 = ESX or ER parity error 1 0 = EAS or EW parity error 1 1 = CER or ESX parity error
26	PCU PROC. INT. (PCU)
27	PIR Residue Error (PIR)
28	Job Number Parity Error (JOB)
29	Non-Recoverable State (MES)

of memory address, and the MAU related information into the Processor Fail register. When the Processor Fail register is read by using the Read Processor Register operator, the Processor Fail register is cleared.

When an alarm interrupt occurs during a store type operation in the MAU, the processor interrupts immediately. However, the MAU may detect more than one error on fetch operations before the Processor Fail register is read because the MAU errors do not necessarily result in interrupts until an attempt is made to process the memory word involved at the time of failure. In some cases, the memory word is never used because a conditional branch is taken or because it is an unused portion of an eight-word fetch.

When information regarding a MAU-detected error is loaded into an empty Processor Fail register, the MAU Single Error (MSE) bit is set. If a subsequent MAU error occurs before the Fail register is cleared, the MAU will overwrite the previous information in the Fail register and reset MSE. Thus, MSE reset in the Fail register indicates that the MAU and memory information may not correspond to the failure that caused the interrupt.

The MSE bit in the Processor Fail register indicates whether all information necessary to retry an operation has been preserved by the EU when an alarm interrupt occurs. If an alarm interrupt occurs, such as Processor Internal, and the MCP finds MES reset, instruction retry will be attempted.

Control Mode Register

The Control Mode register indicates the interrupt mode in which the CPM is operating. The use of interrupt modes provides for recursive entries into the fault control logic. The progression to higher interrupt modes is controlled automatically by the hardware. In addition, programmatic control of the Control Mode register may be accomplished by use of the Read Processor Register and Set Processor Register operators.

The Control Mode register contains two bits which display the interrupt modes of the CPM as follows:

XCM01	XCM00		
0	0	=	Normal Mode
0	1	=	Control Mode 1 (CM1)
1	0	=	Control Mode 2 (CM2)
1	1	=	Control Mode 3 (CM3)

The CPM will be halted with the last interrupt displayed in the Fault register if an interrupt is detected in CM3.

The CPM operates in Normal Mode while not attempting to process an interrupt. When an interrupt condition is detected, the CPM advances to CM1 and attempts to call the procedure pointed to by D [0] +3 (the MCP interrupt procedure) from the stack of the user program. If an interrupt is detected while in CM1, the CPM advances to CM2, changes the stack environment by moving to an alternate stack (determined by indexing the stack vector by the CPM number), and attempts to call the MCP interrupt procedure again. If an interrupt condition is detected in CM2, the CPM advances to CM3, changes the entire environment by setting D [0] to the value in the ADZ register, moves to the proper alternate stack in the new environment and attempts to enter the interrupt handler at the new D [0] +3.

If still another interrupt is detected while in CM3, it is obvious that a recursive interrupt processing situation exists, and the CPM halts. If the CPM succeeds in entering the MCP interrupt procedure, the Control Mode register is reset to Normal Mode programmatically.

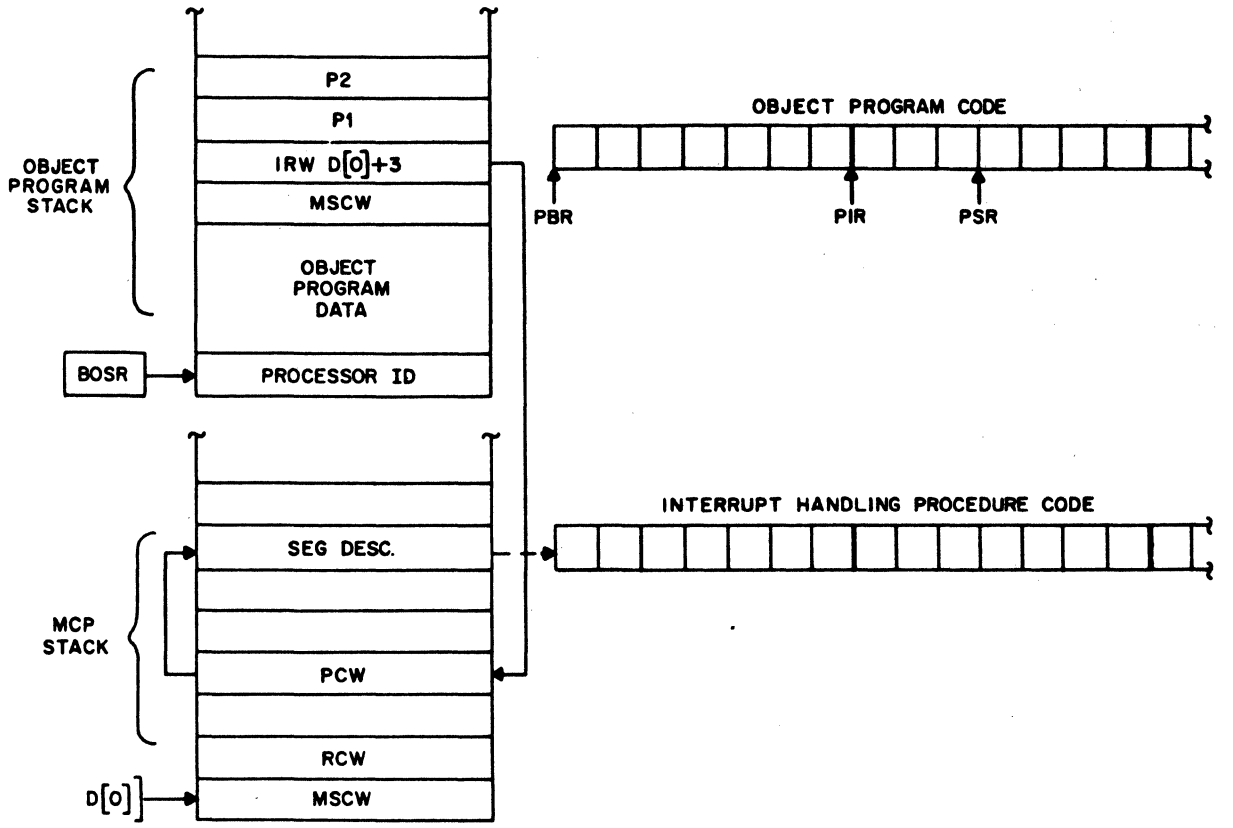
INTERRUPT PROCESSING

All interrupt conditions which are reported in the Fault Register and which are not masked out by the Fault Mask register are accumulated into a general signal to alert the fault control logic of the CPM to the fact that one or more interrupts require processing. When an interrupt requires processing the CPM will advance the Control Mode register (in most cases from Normal Mode to CM1) and will attempt to enter the MCP interrupt procedure.

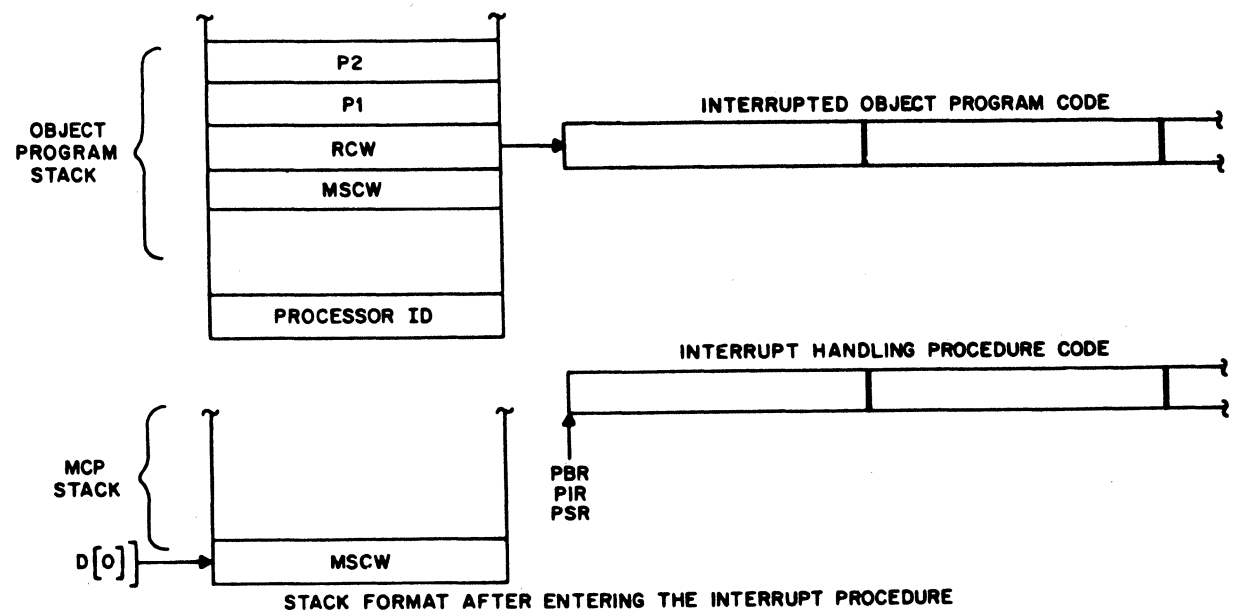
Interrupt Processing in Normal Mode

After advancing the Control Mode register from Normal Mode to CM1, the CPM will attempt to perform the following sequence of operations:

1. Read and save the P1 parameter.
2. Place a Mark Stack Control Word (MSCW) into the stack.
3. Place an Indirect Reference Word (IRW) into the stack. The IRW references a reserved location (D [0] +3) in the MCP stack. (When in Control Mode 3, the IRW references a reserved location (D [0] +3) in the Alternate D [0] stack.)
4. Place the P1 parameter into the stack.
5. Place a second parameter into the stack (the P2 parameter), giving further information about the interrupt.
6. Execute an Enter operator. The fault control logic expects to find a Program Control Word (PCW) at D [0] +3; however, an SIRW, an IRW or an IRW chain which points to a PCW are possible conditions.



STACK FORMAT PRIOR TO CALLING THE INTERRUPT PROCEDURE.



STACK FORMAT AFTER ENTERING THE INTERRUPT PROCEDURE

Figure 3-3-1. Stack Format

The two interrupt parameters (P1 and P2) that are inserted into the stack supply information describing the interrupt condition. The P1 parameter provides information concerning the type of interrupt, the interrupt priority level, and the interrupt class. The P2 parameter supplies supplementary information about the interrupt condition, such as a memory address (memory related interrupts) or a copy of the non-present descriptor (presence bit interrupts). If P2 is not used by the interrupt condition being reported, P2 will be set to zero.

When the interrupt procedure of the MCP is entered, the IRW in the stack (step 3 above) is overwritten with a Return Control Word (RCW) by the ENTER operator. As with any procedure entry, this RCW points to the point in the code string to which control is to be returned following execution of the procedure.

Figure 3-3-1 depicts the stack format just prior to and just after entering the interrupt procedure.

Interrupt Processing in CMI

When an interrupt is detected while in CM1, the CPM advances to CM2 and attempts to enter the MCP interrupt procedure from its alternate stack. The new stack is found by using the processor number as an index into the Stack Vector Array. (This array is pointed to by the Stack Vector Descriptor, located at D [0] +2.) The index into the Stack Vector Array results in a data descriptor, which points to the base of the stack for the new stack. Alternate stacks are established by the MCP at the time of system initialization.

The Bottom Of Stack Register (BOSR) is set to the base address of the new stack, which contains the Top Of Stack Control Word (TOSCW) for the new stack. A modified move-to-stack operation then causes the TOSCW for the old stack, the old BOSR setting, and the old SNR register (stack number) setting to be placed in the top of the new stack. After these parameters have been placed, the stack is marked, the IRW and the P1 and P2 parameters are placed in the stack, and the MCP interrupt procedure is entered. The stack structure just prior to entering the MCP interrupt procedure is shown in figure 3-3-2.

Interrupt Processing in CM2

At system initialization time, the MCP establishes a special CM3 operating environment at the top of memory. This environment includes an abbreviated D [0] stack with its own stack vector and an interrupt handler. The main memory address of this alternate D [0] stack is loaded into the ADZ register

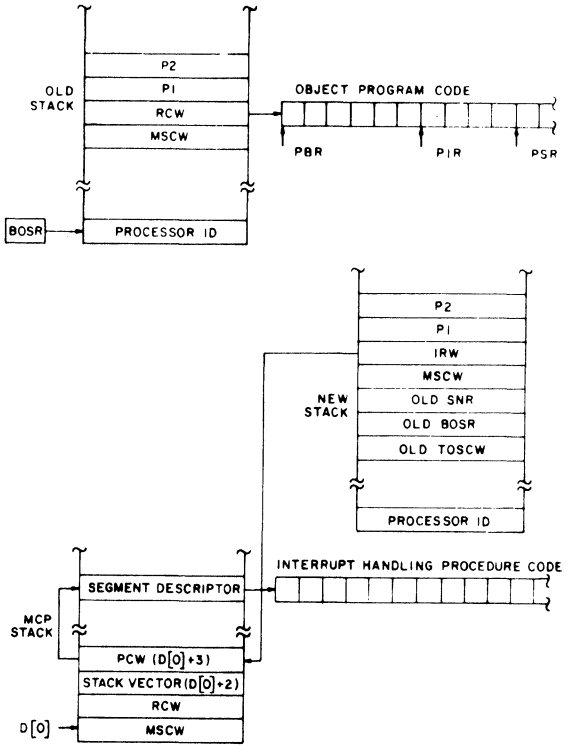


Figure 3-3-2. Stack Format Prior to Calling Interrupt Procedure While in CM1 (Move Stack Operation)

of all CPMs. When a CPM detects an interrupt while in CM2, the CPM advances to CM3, and changes to the CM3 environment by setting the D [0] register to the value in ADZ. The CPM then attempts to move to its alternate stack in the new stack vector (at ADZ+2) and enter the new interrupt handler at ADZ+3, as described in the previous paragraphs.

Interrupt Processing in CM3

If an interrupt is detected while in CM3, it is obvious that a recursive interrupt condition exists. In such cases the CPM is halted, the most recent interrupt is identified in the Fault register, and the XCM register equals 3.

Control Mode Advancement

Figure 3-3-3 illustrates the priority scheme for reporting interrupts, the conditions for advancing the Control Mode register, and the interrupt conditions which may be left in the Fault register for later servicing. In case one, the Fault register contains an Alarm Interrupt (first priority) a Stack Overflow interrupt (third priority), and may also contain Syllable interrupts (second priority), Interval Timer interrupts (fifth priority), and External interrupts (fourth

FAULT REGISTER (BEFORE REPORTING INTERRUPT)					CONTROL MODE REGISTER				REPORTED IN	FAULT REGISTER (AFTER REPORTING INTERRUPT)					
INTERVAL TIMER	EXTERNAL	STACK OVERFLOW	SYLLABLE	ALARM	NORMAL	CM 1	CM 2	CM 3	PARAMETER P1	EXTERNAL	INTERVAL TIMER	STACK OVERFLOW	SYLLABLE	ALARM	
ϕ	ϕ	1	ϕ	1	CM1	CM2	CM3	*	ALARM	ϕ	0	1	0	0	CASE ONE
ϕ	ϕ	1	1	0	CM1 CM2	CM2 CM3	CM3 *	* –	SYLLABLE STACK OVERFLOW	ϕ ϕ	ϕ 0	1 0	0 0	0 0	CASE TWO
ϕ	ϕ	0	1	0	CM1	CM2	CM3	*	SYLLABLE	ϕ	ϕ	0	0	0	CASE THREE
ϕ	ϕ	1	0	0	CM1	CM2	CM3	*	STACK OVERFLOW	ϕ	0	0	0	0	CASE FOUR
ϕ	1	0	0	0	CM1	–	–	–	EXTERNAL	ϕ	0	0	0	0	CASE FIVE
1	0	0	0	0	CM1	–	–	–	INTERVAL TIMER	0	0	0	0	0	CASE SIX

* PROCESSOR HALTS
ϕ MAY BE A ONE OR A ZERO

40354

Figure 3-3-3. Interrupt Reporting

priority). The Alarm interrupt causes the Control Mode register to be advanced (from Normal to CM1, CM1 to CM2, or CM2 to CM3), the P1 parameter reports the Alarm interrupt, and the External interrupts are still contained in the Fault register (all other interrupts are cleared from the register).

Case two shows all priorities of interrupts except Alarm interrupts present in the Fault register. The resultant action is similar to case one, in that the highest priority interrupt (Syllable) is serviced first. P1 reports the Syllable interrupt, the Control Mode register is advanced, and the Stack Overflow and External interrupts are still contained in the Fault register (in this case the Interval Timer interrupt is also left in the Fault register).

Following entry into the software interrupt procedure, the Stack Overflow interrupt is reported by another P1, the Control Mode register is advanced, the Interval Timer interrupt is cleared from the Fault register, and the External interrupts are left for later servicing. The stack structure for either case one or case two is shown in figures 3-3-4 and 3-3-5.

Case three of figure 3-3-3 shows a Syllable interrupt (second priority), an Interval Timer interrupt (fifth priority), and an External interrupt (fourth priority) all present in the Fault register. In this case, the highest priority interrupt present (Syllable) is reported in P1, the Control Mode register is advanced, and the Interval Timer and External interrupts are left for later servicing. (The External interrupt is serviced first.)

Case four shows a Stack Overflow interrupt, an Interval Timer interrupt, and an External interrupt present in the Fault register. The Stack Overflow interrupt is reported in P1, the Interval Timer interrupt is cleared from the register, and the External interrupt is left for later servicing.

Case five shows servicing of an External interrupt, leaving an Interval Timer interrupt for later servicing. Case six shows servicing of an Interval Timer interrupt. Notice that these two cases can only occur when the CPM is in Normal State. (When the CPM advances to CM1 and the MCP interrupt procedure is entered, the CPM operates in Control State and the recognition of Interval Timer and External interrupts is inhibited.)

Alarm Interrupts (First Priority)

Detection of an Alarm interrupt causes an immediate entry (or re-entry) into the fault control logic. The Control Mode register is advanced and a P1 parameter is formed which identifies all Alarm interrupts which are present in the Fault register. Syllable Dependent interrupts, Stack Overflow interrupts, and Interval Timer interrupts (if present) are cleared from the Fault register and the interval timer is disarmed. The MCP interrupt procedure is entered.

Syllable Dependent Interrupts (Second Priority)

Detection of a Syllable Dependent interrupt (if no Alarm interrupts are present) causes an immediate entry (or reentry) into the fault control logic. The Control Mode register is advanced and a P1 parameter is formed which identifies all Syllable Dependent interrupts which are present. The MCP interrupt procedure is entered.

Special Interrupts

Stack Overflow (Third Priority)

All Stack Overflow interrupts are processed by the fault control logic and cause advance of the Control Mode register. All Stack Overflow interrupts cause the P1 parameter reporting the interrupt to be formed. Interval Timer interrupts (if unmasked) are cleared from the Fault register and the Interval Timer is disarmed. The MCP interrupt procedure is entered.

Interval Timer (Fifth Priority)

Interval Timer interrupts are cleared (and the interval timer is disarmed) when an Alarm or Stack Overflow interrupt is reported. All uncleared Interval Timer interrupts cause entry into the fault control logic if the mask is set, either in normal state or if executing an IDLE, in Control state. The Control Mode register is advanced to CM1 (from Normal). (Interval Timer interrupts are inhibited when the CPM is in Control State.) The MCP interrupt procedure is entered.

External Interrupts (Fourth Priority)

Although External interrupts can occur at any time, these interrupts (with the exception of the Egg Timer interrupts) are inhibited when the CPM is in Control State. If an External interrupt occurs when the CPM is in Normal State, the Control Mode register is advanced to CM1 and a P1 parameter describing the External interrupt is formed. Then, the Interval Timer is disarmed and the MCP interrupt procedure is entered.

Memory Related Interrupts

Memory related interrupts include Memory Parity errors (MPA) which are discovered by the MAU; Memory Fail 1 (FL1) errors which are discovered by

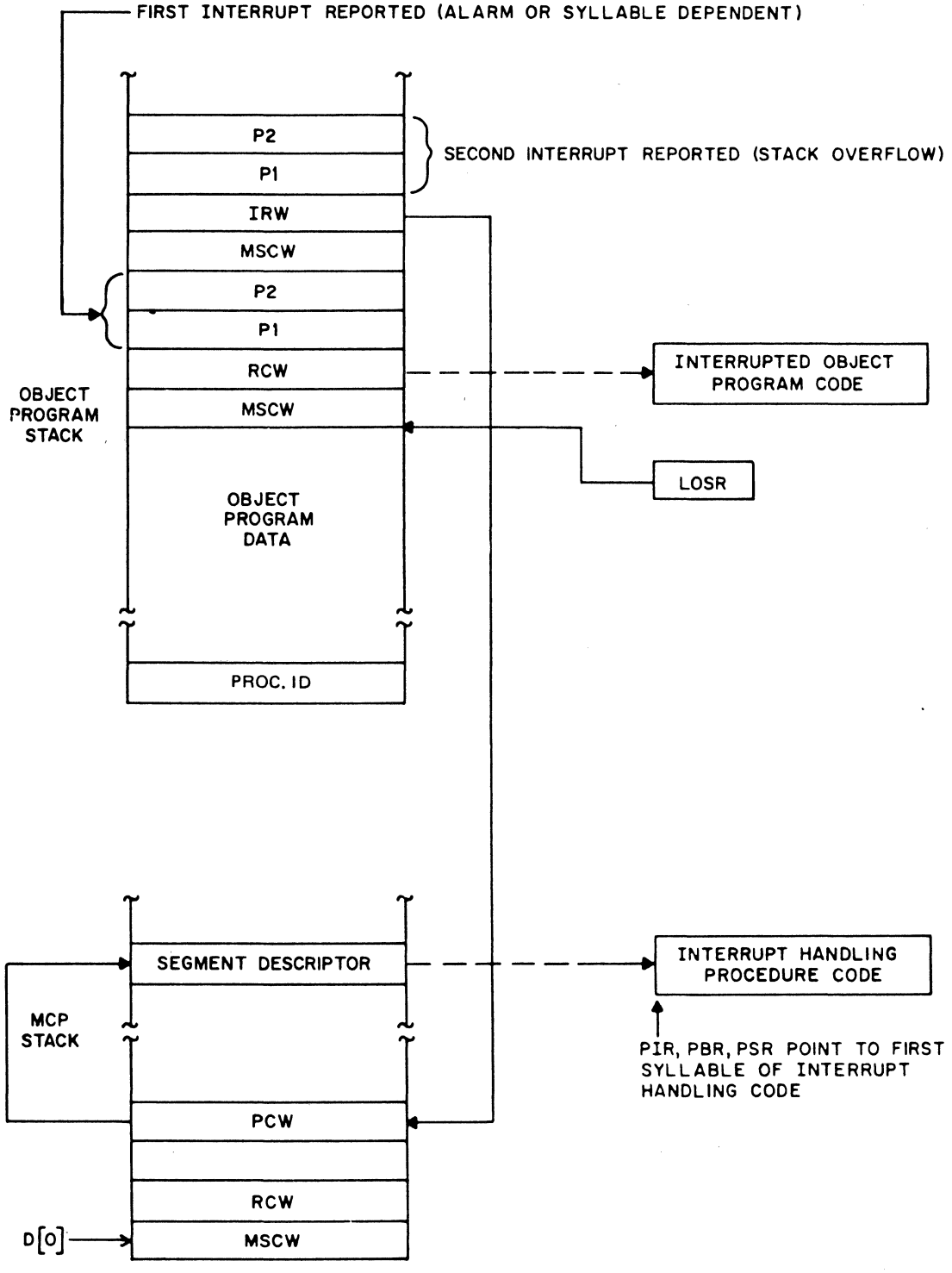


Figure 3-3-4. Stack Format Before Re-entering Interrupt Procedure to Report Stack Overflow

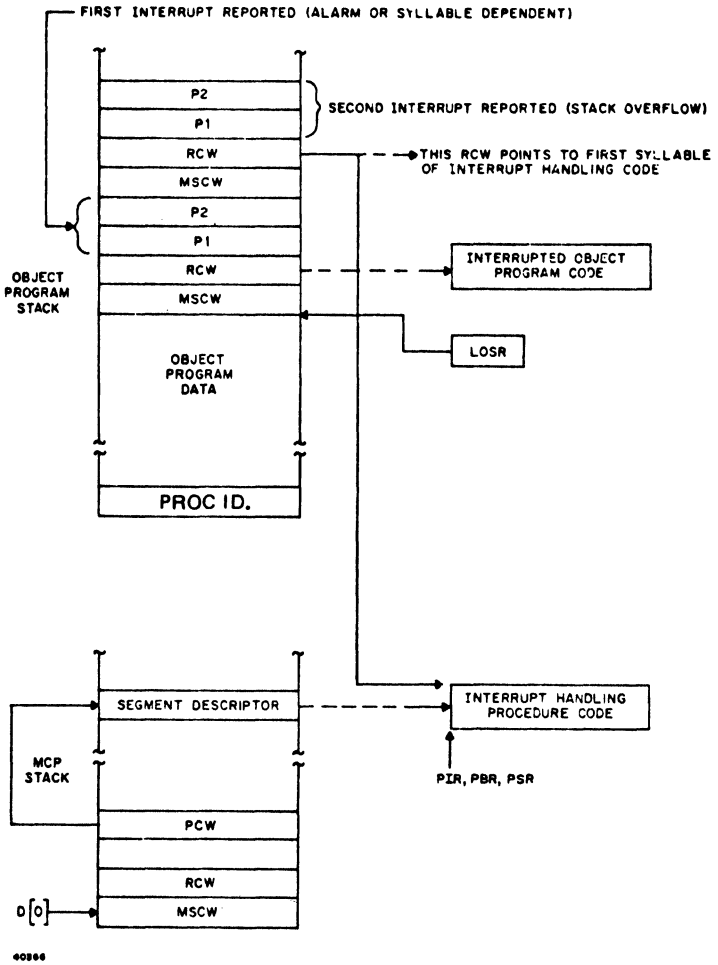


Figure 3-3-5. Stack Format After Re-entering Interrupt Procedure and Reporting Stack Overflow

the MCM and reported to the requestor; invalid address errors which are detected by the MAU in its interface with the MCMs as an NAM; and Processor Internal errors (PI) which are discovered by the MAU in its interface with the other units of the CPM. It should be noted that PI interrupt can also be set by other units in the CPM. These four types of errors are differentiated in the P1 interrupt parameter. Memory Fail 1 interrupts from all MCMs are combined into a single alarm interrupt, represented by bit 2 in the Fault register and the Interrupt ID. The identification of which MCM as involved is given in the P2 parameter or in the Processor Fail register, depending on the type of MAU operation being performed. Wrong Channel Number and Memory Time Out error indications in the Processor Fail register are combined into a single alarm interrupt, represented by bit 3 (Invalid Address) in the Fault register and the Interrupt ID. Likewise, MAU residue (address residue error) and Bad Parity to MAU error indications in the Processor Fail register are combined into a single alarm interrupt, represented by bit 6 (Processor Internal) in the Fault register and the Interrupt ID.

Explanatory information about these errors may be found either in P2 parameter or in the Processor Fail register. If P2 is not used, it will be set to zero. (See table 3-3-3.)

Interrupt Descriptions

Interrupts which can occur in the CPM are described in the following paragraphs. The interrupts are described in order of their priority. Alarm interrupts are described first, Syllable dependent interrupts second, Special interrupts third, and External interrupts last.

Table 3-3-3. CPM Handling of Memory Related Errors

Source of Request for Memory Access	Alarm Interrupts (MPA, FL1, NAM, & PI)
Data Reference Unit Fetch (includes flashback)	The error word is queued in the associative memory or reported immediately with one exception by the DRU if the error word is sent to the Central Data Buffer. In the latter case, the alarm interrupt occurs immediately. The Program Control Unit receives CDB address of error word through the interrupt mechanism. In turn PCU issues to the Execution Unit, the address of the CDB location from which error word is to be read, the address of the CDB location (W storage portion of CDB) into which error word is to be written as the P2 parameter, and a micro operator for exchanging these CDB locations. The exception is an invalid address error during a string data fetch. In this case, the error word is queued in the DRU string data file of CDB and not reported until Execution Unit attempts to process the string data involved. The error word is placed into W storage portion of CDB in the same manner as explained above.
Store Queue Write or Purge	The alarm interrupt occurs immediately. There is no P2 parameter; rather the explanatory information (including the identity of the involved MCM) is contained in the Processor Fail register.
Program Control Unit Fetch	The error word is queued in the program buffer and not reported until the Program Control Unit attempts to process the code string involved. The error word is then passed from PID to a LT48 location in the CDB and then into W storage portion of CDB as the P2 parameter.

Alarm Interrupts

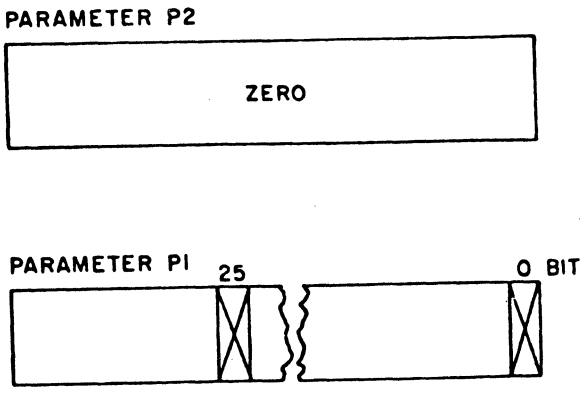
Alarm interrupts are caused by conditions which are unexpected by the CPM. They inform the system of some detrimental change in environment. In most cases, Alarm interrupts result from hardware failures. The Alarm interrupts cannot be inhibited, and always cause entry into the fault control logic. The fault control logic terminates the current operator, clears the top of stack registers, prepares the stack (MSCW, IRW, P1, P2), and causes the MCP interrupt procedure to be entered. When an Alarm interrupt is cleared from the Fault register, all Syllable Dependent interrupts present in the register are cleared. The Alarm interrupts are:

Loop
Memory Parity
Memory Fail 1
Invalid Address (no access)
Stack Underflow
Invalid Program Word
Processor Internal

Alarm interrupts generally result in termination of the process involved. Exceptions are (1) during a halt load when the MCP uses an alarm interrupt (invalid address) to determine the amount of memory available, and (2) when instruction retry by the MCP is successful after a Processor Internal interrupt.

Loop

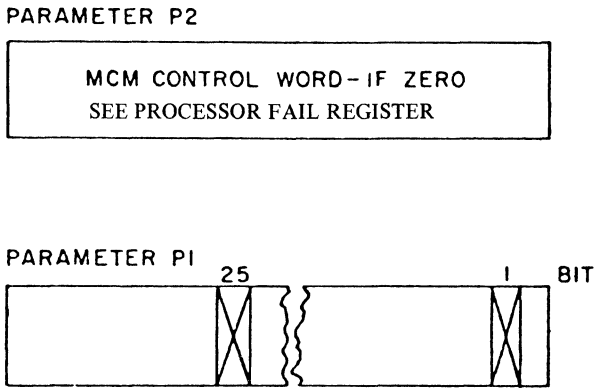
This interrupt occurs when the CPM has expended up to two seconds in the execution of one operator. This interrupt can be causes either by a hardware failure or by bad data. Should this interrupt occur, PIR may not be accurate.



Loop Interrupt Parameters

Memory Parity

This interrupt occurs if the CPM receives a memory word with a even number of 1's. Should this interrupt occur, PIR points to the word containing the operator which initiated the interrupt. Supplementary information describing the error will be contained in the Processor Fail register. (See "Memory Related Interrupts".)



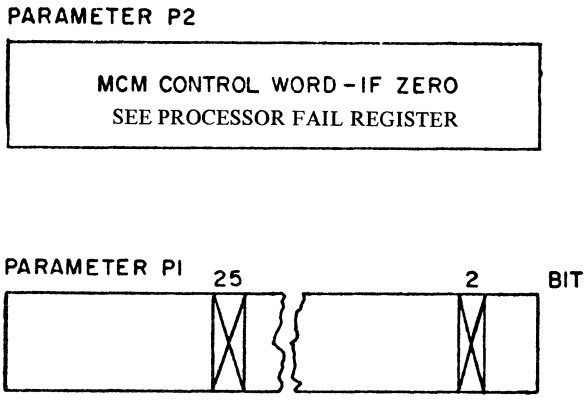
Memory Parity Interrupt Parameters

Memory Fail 1

This interrupt occurs if any of the following errors occur:

1. Data word parity error
2. Illegal operation code
3. Address is for a different Memory Module than requested
4. Data strobe error
5. Internal control error
6. Multiple bit data error

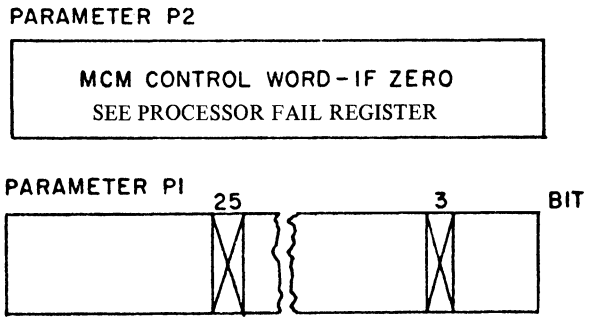
In all of the above cases, supplementary information describing the error will be contained in the MCM Fail register. (See "Memory Related Interrupts".)



Memory Fail 1 Interrupt Parameters

Invalid Address

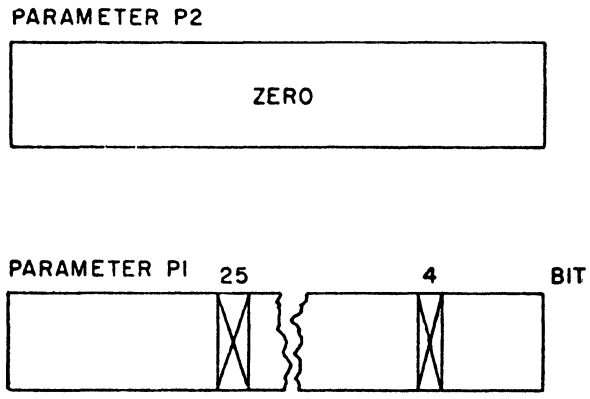
This interrupt occurs when the CPM attempts to access a memory address which is not available to the system. The Memory Module may not exist or it may be inoperative. Supplementary information is placed in the Processor Fail register. (See "Memory Related Interrupts".)



Invalid Address Interrupt Parameters

Stack Underflow

This interrupt occurs if the CPM attempts to move the top of stack (S register setting) to an address less than the address of the most recent MSCW (F register setting) during a stack adjustment. This could occur as a result either of a compiler error or a hardware control failure in executing MKST, EXIT, or MVST (all of which change F setting and could calculate an incorrect address).



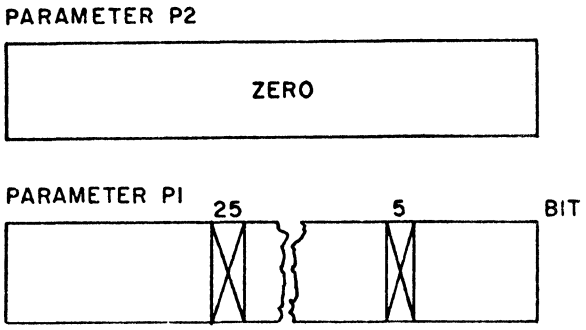
Stack Underflow Interrupt Parameters

Invalid Program Word

This interrupt occurs under any of the following conditions:

1. An attempt is made to execute a program word which does not have a tag of 3 (or tag of 0 if in Edit mode).

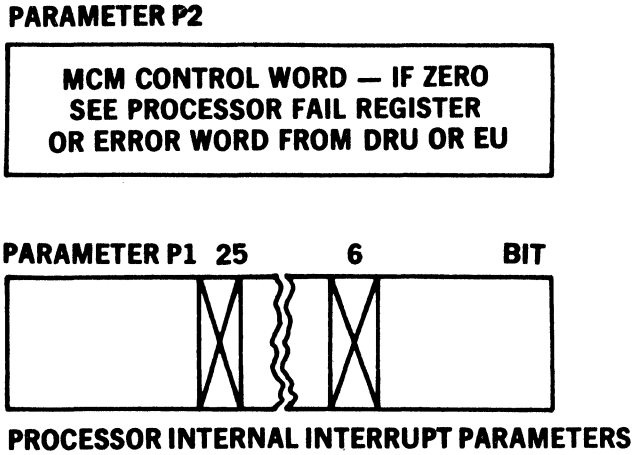
2. The Variant code (Escape to 16-bit Instruction, VARI) is detected as the second syllable of a Variant operator.
3. An attempt is made to execute an operator which is considered illegal in Edit mode or Vector mode.



Invalid Program Word Interrupt Parameters

Processor Internal

This interrupt occurs whenever an internal logic failure is detected within the CPM. The Processor Fail register provides information regarding the failure. For further information regarding memory related Processor Internal interrupts, see "Memory Related Interrupts".



PROCESSOR INTERNAL INTERRUPT PARAMETERS

ET 1688a

Syllable Dependent Interrupts

Syllable Dependent interrupts generally result from programming errors. These interrupts cannot be inhibited, and always cause entry into the fault control logic. The fault control logic terminates the current operator, prepares the stack (MSCW, IRW, P1, P2), and causes the MCP interrupt procedure to be entered. The contents of the top of stack registers may or may not be saved, depending upon the type of interrupt.

Syllable Dependent interrupts are divided into two classes. Class 1 interrupts (identified by the setting of bit 24 of parameter P1) are those interrupts in which the values of PIR, PSR, PBR, and PDR have not been modified by the operator. Class 2 interrupts (identified by the setting of bit 23 of of parameter P1) are those interrupts in which the value of PIR, PSR, PBR, and PDR have been changed. Thus, class 1 interrupts permit the operator to be re-executed; class 2 interrupts prohibit the operator from being re-executed.

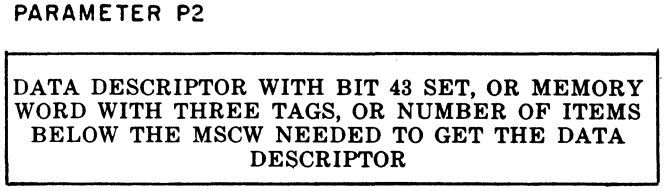
Most Syllable Dependent interrupts occur as class 1 interrupts. The only Syllable Dependent interrupts which can occur as class 2 interrupts are the Invalid Index, Bottom of Stack, and Sequence Error interrupts. The Syllable Dependent interrupts are:

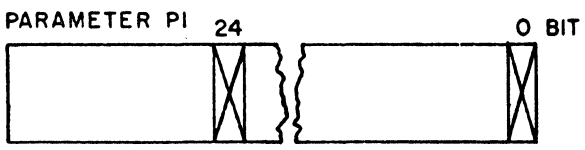
Memory Protect
Invalid Operand
Divide By Zero
Exponent Overflow
Exponent Underflow
Invalid Index
Integer Overflow
Bottom Of Stack
Presence Bit
Sequence Error
Segmented Array
Programmed Operator
Privileged Instruction

Memory Protect

This interrupt occurs when one of the following occurs:

1. A store, overwrite, read/clock, or string transfer operation is attempted using a data descriptor that has the read only bit (bit 43) set. The operation is terminated before the memory access. The data descriptor is used as the P2 parameter, except for string transfer.
2. A store operation is attempted into a word in memory that has a tag field representing a PCW, RCW, MSCW, or segment descriptor (tag = 3, 7). The memory write is discontinued when bit 48 is detected in the code word being referenced. The flashback is used as the P2 parameter.



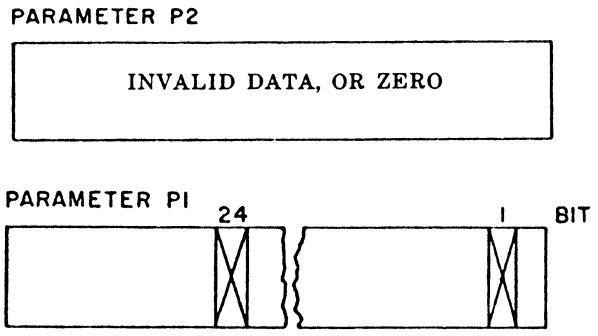


Memory Protect Interrupt Parameters

INFORMATION ON THIS PAGE DELETED

Invalid Operand

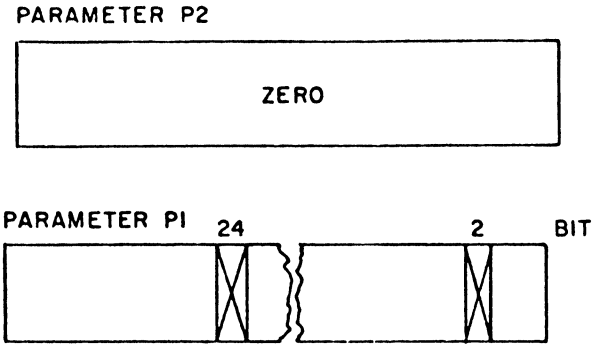
This interrupt occurs when the CPM attempts to execute a valid operator on data which is invalid for that operator or attempts to execute the invalid operator NVLD. Each operator executes checks to ensure that control words and data meet the necessary requirements of the operator. Should this interrupt occur PIR and PSR are left pointing to the current syllable.



Invalid Operand Interrupt Parameters

Divide By Zero

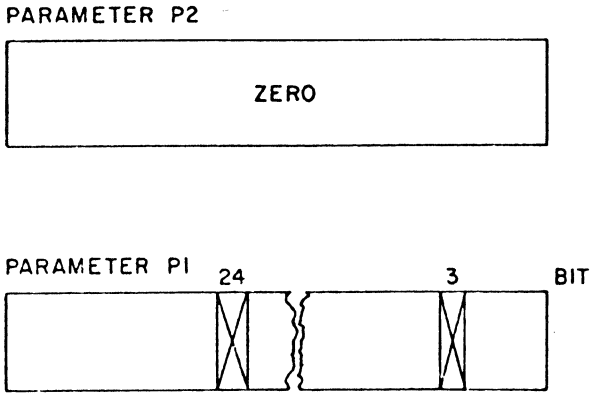
This interrupt occurs when a division operation is attempted with the divisor (contained in the top of stack) equal to 0. Should this interrupt occur, PIR and PSR point to the initiating operator, and the divisor and dividend will be left on the top of the stack (below the MSCW, RCW, P1, and P2).



Divide By Zero Interrupt Parameters

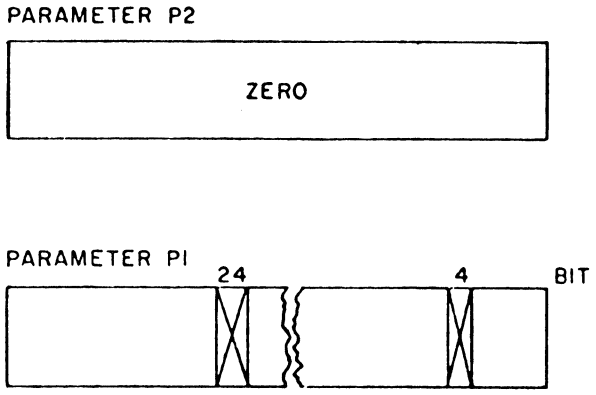
Exponent Overflow

This interrupt occurs when the capacity of a positive sign exponent field is exceeded for either single or double precision arithmetic results. Should this interrupt occur, PIR and PSR point to the initiating operator.



Exponent Underflow

This interrupt occurs when the capacity of a negative sign exponent field is exceeded for either single or double precision arithmetic results. Should this interrupt occur, PIR and PSR point to the initiating operator.



Exponent Underflow Interrupt Parameters

Invalid Index

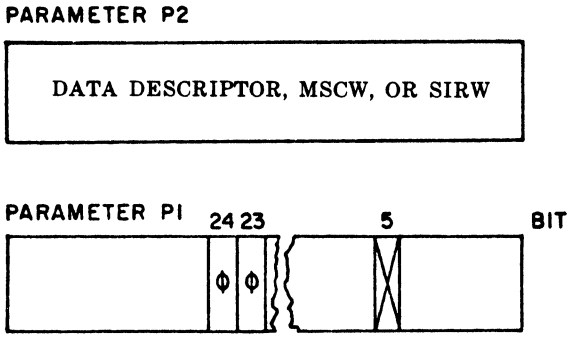
This interrupt occurs if an attempt is made to index a descriptor by an amount which is less than 0 or which is greater than or equal to the upper bound (length) in any of the following operations:

Occurs Index
Linked List Lookup
Index
Move Stack
Display Update
VALC
Stuffed IRW (pseudo operator)
Index and Load Name
Index and Load Value

If this interrupt occurs, the operation is terminated prematurely. The input operands will be left on the top of the stack (below the MSCW, RCW, P1, and P2). Except for Display Update, all operations in the list above will cause PIR and PSR to point to the initiating operator. The interrupt occurs as a class 2 interrupt (bit 23 = 1) if an attempt is made to index the Stack Vector Array descriptor (D [0] +2) during a display update operation using a stack number which is greater than or equal to the length field of the Stack Vector Array descriptor.

NOTE

Bit 23 and bit 24 may not be set simultaneously.

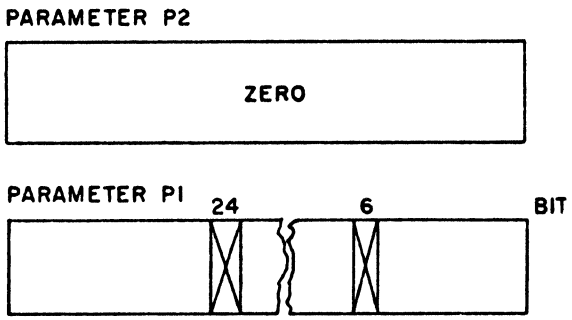


Invalid Index Interrupt Parameters

Integer Overflow

This interrupt occurs upon detection of the attempted use of an operand which exceeds the maximum integer value (2^{39} -1) by an operator which requires an integer. The following is a partial list of operators which may cause this interrupt to occur:

Integer Divide
Integerize Truncate
Integerize Rounded
Occurs Index

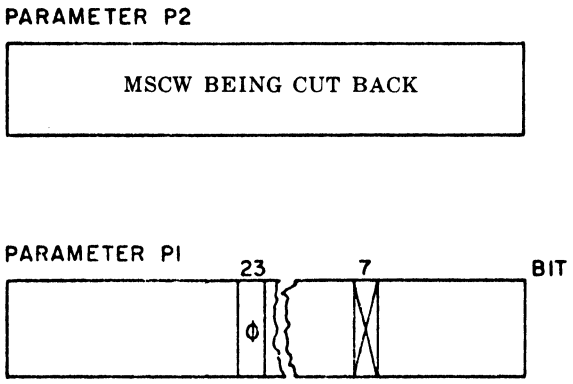


Integer Overflow Interrupt Parameters

Bottom of Stack

This interrupt occurs if a Return operator or an Exit operator causes the program stack to be cut back to its base. (The F register points to the MSCW located at the BOSR setting plus 1.) The P2 parameter will be a copy of the MSCW being cut back.

This interrupt occurs as a class 2 interrupt (bit 23 = 1).



Bottom Of Stack Interrupt Parameters

Presence Bit

This interrupt occurs when an attempt is made to access a word or group of words which are not present in main memory. All operators that access memory with descriptors may be interrupted with this interrupt. The interrupt occurs if an attempt is made to reference memory through a descriptor which has the presence bit (bit 47) reset, indicating that the descriptor points to words which are not present in main memory. There are two classes of presence bit interrupt conditions; data dependent and procedure dependent.

DATA DEPENDENT

Data dependent presence bit interrupt conditions occur when the CPM is seeking data from within its current addressing environment. In all cases except Value Call, recovery is achieved by re-executing the operator upon return from the MCP interrupt procedure. The MCP interrupt procedure makes the absent words present before return is made to the interrupted program. To permit this re-execution, the PIR and PSR settings for the current operator are saved in the RCW. Value Call always sets this RT bit for data dependent interrupts; however, Value Call never sets this RT bit for procedure dependent

interrupts. Value Call or pseudo value call will always turn on the VS bit (bit 39) and cause the V bit in the MSCW to be turned on. Figure 3-3-6 illustrates the PIR, PSR, Exit/Return, RT, VS, and RE bit relationships in the various presence bit interrupt conditions.

Accidental Entry

Procedures which have been entered accidentally during the VALC operator also require special consideration for the manipulation of PIR and PSR settings for the RCW. The VALC operator is completed after the return operator mechanism when returning from an accidentally entered procedure. A pseudo value call operator provides the ability to continue searching an IRW or data descriptor chain until an operand is located. The pseudo value call operator is activated at the end of a normal return operator if the V bit of the MSCW has been set. The V bit is set when either a VALC or pseudo value call operator enters a procedure accidentally. If a not present segment descriptor causes an interrupt during a return from an accidental entry of value call, a pseudo RT bit (Bit 45) is turned on in P1 so the presence bit procedure will finish with a return instead of an exit if the VS (Bit 39) is also on. The RT bit and pseudo RT bit are used by the software to execute the proper code. The V bit is used by the hardware to change the return into a pseudo value call so that IRW or data descriptor chain may be chased.

PIR and PSR values, pointing to the next operator syllable, are inserted into the RCW for VALC while the PIR and PSR values from the old RCW are inserted into the RCW for a value call pseudo operator.

All other operators which may incur accidental entries are restarted; therefore, the PIR and PSR settings which point to the current operator syllable are saved in the RCW. The V bit is set to zero.

Procedure Dependent

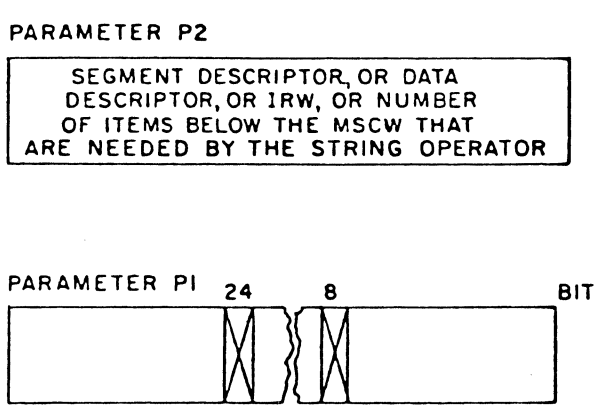
Procedure dependent interrupts occur when the CPM is attempting to enter a new addressing environment, or attempting to return to an old addressing environment. These interrupts occur during display update, and also when trying to process a non-present segment descriptor. Recovery is achieved by the Exit operator or the Return operator after the MCP interrupt procedure has made the referenced environment present. Because the CPM has not yet fetched the first operator of the new procedure when this interrupt occurs, the PIR and PSR settings from the PCW (for entry) or the RCW (for return) are stored in the RCW which is made when the MCP interrupt procedure is entered. Thus, when the referenced environment is made present, the entry or return is to the referenced environment.

Program Restart

Following a Presence Bit interrupt, a program may be restarted either by executing a Return operator or an Exit operator. The Return operator must return either an IRW or a Data Descriptor. The RT bit of the P1 parameter (bit 46) indicates to the MCP interrupt procedure whether to perform an Exit operator (bit 46 is reset) or a Return operator (bit 46 is set) when returning to the interrupted procedure.

Parameter P2

During the execution of certain string operators, if a Presence Bit interrupt occurs the P2 parameter may contain a number which indicates the number of items below the MSCW which are needed by the string operator.



Presence Bit Interrupt Parameters

Sequence Error

This interrupt occurs if an indirect reference encounters an invalid condition or reference sequence. Generally, this interrupt is caused either by a hardware error or a systems software error, and the MCP will terminate the program which generated the interrupt. The interrupt can occur as a class 2 interrupt (bit 23 = 1) only under the following conditions:

1. When a word other than a Segment Descriptor is fetched relative to the PDR during the final algorithm for the Enter, Exit, or Return operators.
2. When the F register points to a word which is not an MSCW at the beginning of execution of the Exit or Return operators.

PRESENCE BIT INTERRUPT CONDITIONS		P_2	P_1 PRESENCE BIT ID			RETURNING OPERATOR	PIR, PSR NEW RCW	SOFTWARE FUNCTION
			RT (46)	VS (39)	RE (45)			
Data Dependent	Stack Vector DD or Stack D.D. During Reference Through Stuffed IRW	DESC (4)*	0	0	0	EXIT	S_n (8)	Locate Not Present D.D. By the IRW. If necessary, make the D.D. present and return an IRW where noted
		IRW (1)	0	0	0	EXIT	S_n (8)	
		IRW (2)	1	1	0	RETURN	S_n+ 2 (8)	
		IRW (3)	1	0	0	EXIT	S_n (8)	
	Data Descriptor	DESC (2) (copy)	1	1	0	RETURN	S_n+ 2 (8)	Search Stack for copies of Not Present D.D. Make MOM and copies present, return present D.D. where noted
		DESC (7) (copy)	1	0	0	RETURN	S_n (8)	
Procedure Dependent	Stack Vector DD or Stack D.D. During Display Update	DESC (6) (copy)	0	0	0	EXIT	From RCW or PCW	Search Stack for copies of Not Present D.D. Make MOM and copies present, Return D.D. where noted
		DESC (5) (copy)	0	0	1	EXIT		
		DESC (2) (copy)	0	1	0	EXIT		
	Segment Desc	DESC (2) (copy)	0	1	0	EXIT	From RCW or PCW	Locate S.D. (MOM) via copy in P_2 AD Field Of Copy Points to MOM
		DESC (6) (copy	0	0	0	EXIT		
		DESC (5) (copy)	0	1**	1	RETURN		

1. Enter or IRWL
2. VALC
3. All Operators Except VALC, ENTR, MVST, RETN, IRWL
4. MVST
5. RETN
6. All Operators Except RETN and VALC
7. All Operators Except ENTR, VALC, or IRWL
8. S_n indicates that PIR and PSR point to current operator syllable.

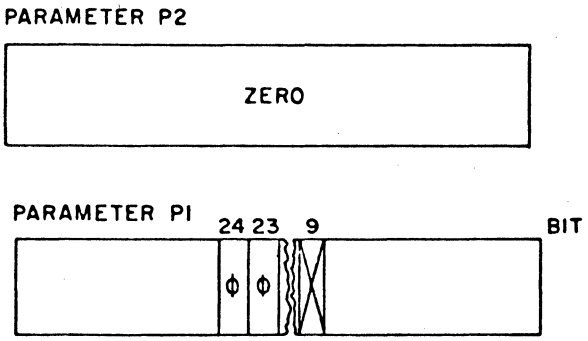
*Fetch new stack desc thru IRW only.

**IF RVLC (V-bit in the MSCW is on).

40994

Figure 3-3-6. Presence Bit Interrupt Chart

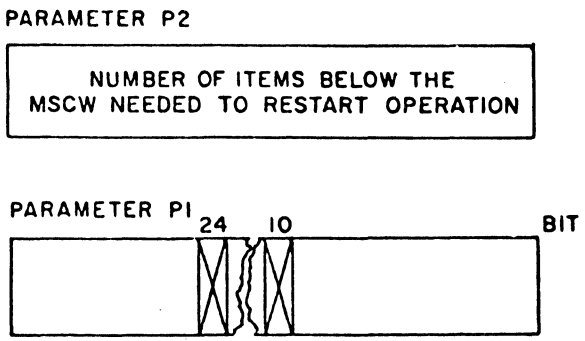
3. When tracing back through the DF links of an MSCW chain (DF locates the preceding MSCW in the stack) during an Exit, Return, or Move Stack operation and a word which is not an MSCW is fetched.
4. When a word which is not a Segment Descriptor is fetched relating to the PDR during a Dynamic Branch operator execution.



Sequence Error Interrupt Parameters

Segmented Array

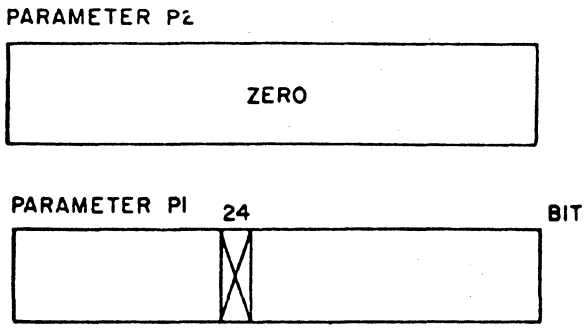
This interrupt occurs when a string operator attempts to index beyond the end of the current segment of a segmented array. Arrays in main memory may be segmented into groups of 256 words each, bounded on both ends by memory links. The memory link words are created by the MCP with the memory protect bit (bit 48) set. During string operations, each word read from memory is checked to see if bit 48 is set. If such a word is referenced, the Segmented Array interrupt will occur. The P2 parameter will indicate how many words (in the stack below the MSCW, RCW, P1, and P2) are needed to restart the operation after the new segment of the array has been made available in main memory.



Segmented Array Interrupt Parameters

Programmed Operator

This interrupt occurs if the CPM attempts to execute an operator code which is not currently assigned. The Programmed Operator interrupt acts as a communicate operator to the MCP, and allows the MCP to simulate the action of the operator programmatically, if desired. All unassigned operator codes cause this interrupt. (None of the unassigned operator codes cause Loop, Invalid Program, or Invalid Operand interrupts. Scan In Time Of Day Clock is an assigned operator: any other variation of Scan In causes the Invalid Operand interrupt.)

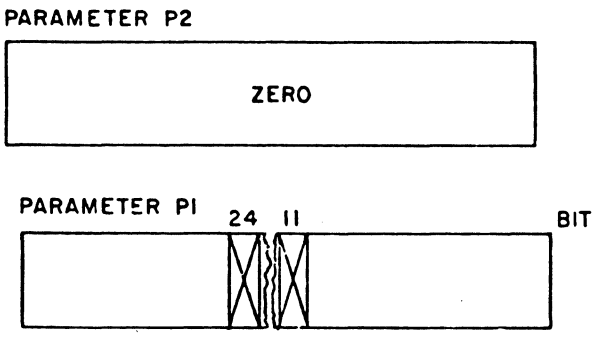


Programmed Operator Interrupt Parameters

Privileged Instruction

This interrupt occurs if an attempt is made to execute a Control State operator while the CPM is in Normal State. The Control State operators are:

Inhibit Parity (GPR)
Set Memory Inhibits (SINH)
Set Memory Limits (SMLT)



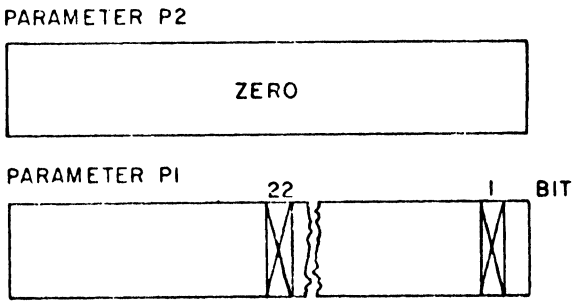
Privileged Instruction Interrupt Parameters

Special Interrupts

Special interrupts take third priority for processing. There are just two Special interrupts: Stack Overflow and Interval Timer.

Stack Overflow

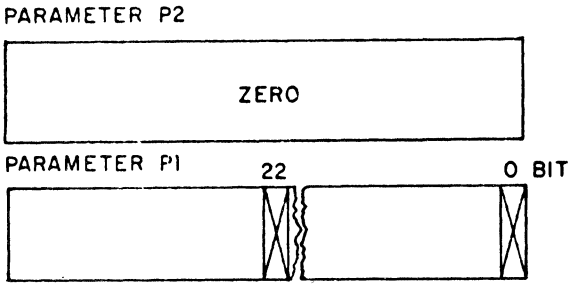
This interrupt occurs when the Stack Controller senses the use of the highest address allotted for the stack of the program (the S register and the Limit of Stack register (LOSR) point to the same address). The MCP interrupt procedure may either allocate a larger stack area, or it may terminate the program. If the current operator has not been completely executed, PIR and PSR are changed to point to the operator.



Stack Overflow Interrupt Parameters

Interval Timer

This interrupt occurs if the value in the hardware interval timer is 0 and the interval timer is armed. The timer is armed and an initial value is stored by the Set Interval Timer operator (SINT). The count in the timer is decreased every 512 microseconds until the count reaches 0 or until the timer is reset. If the timer is still armed when the count reaches 0, the interrupt occurs. The maximum interval to which the timer can be set is 1 second. This interrupt is used by the MCP to ensure that no process can control a CPM for more than 1 second without giving the MCP a chance to regain control of the CPM.



Interval Timer Interrupt Parameters

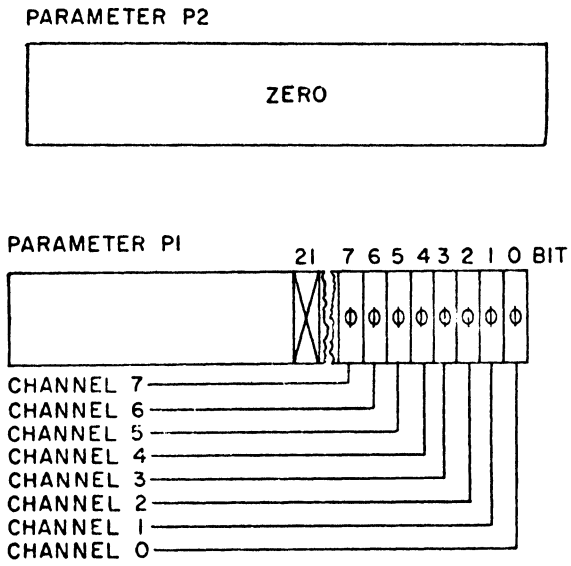
External Interrupts

External interrupts are used to inform the MCP of changes in external environment, and also to permit communications between requestor modules (CPM and IOM). Normally, these interrupts result in the momentary interruption of a program while the interrupt is handled or recorded by the MCP. Following the handling of the interrupt, the program is continued. The External interrupts are:

Channel (0 through 7)
IOM Error (0 through 7)
Egg Timer

Channel Interrupt

This interrupt may be generated by any of the eight possible requestor modules (CPM or IOM). The interrupt identification (parameter P1) indicates the source of the interrupt. This interrupt may be generated to indicate an expected event (such as IO Complete) or it may be generated by the Interrupt Channel N operator (which allows any CPM to interrupt any requestor module).

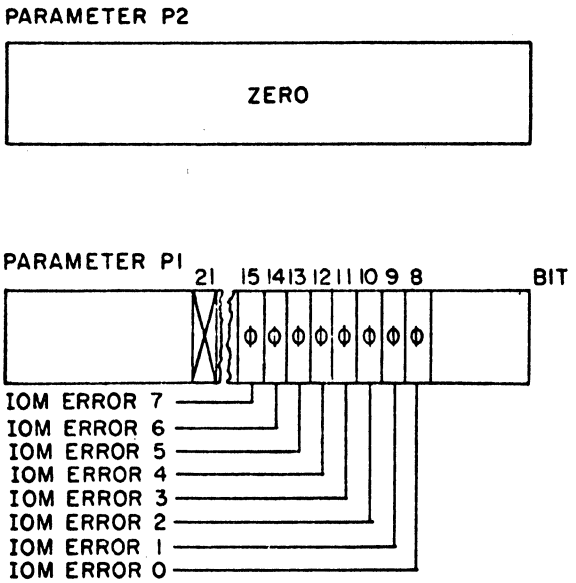


Channel Interrupt (0-7) Parameters

IOM Error Interrupt

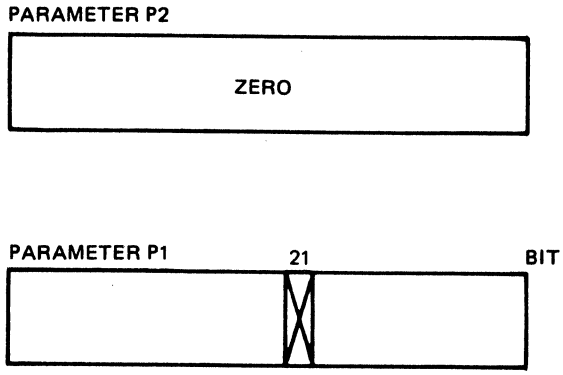
This interrupt may be generated by any of the IO modules in the system. The interrupt identification (parameter P1) indicates the channel (0 through 7) to which the IOM is connected. This interrupt is used to report errors detected by an IOM which are not device related. If possible, the IOM will link a dummy IOCB into the status queue (RESULTQ).

The dummy IOCB will contain a Result Descriptor which will further describe the error. Otherwise, the Fail Result Descriptor will be placed at Home Address (HA) + 5.



Egg Timer Interrupt

This interrupt occurs if the Egg flip-flop in the CPM is not reset every 8 to 16 seconds by the MCP. This interrupt is used by the MCP to present the CPM from looping while in the control state.



Egg Timer Interrupt Parameters

SECTION 4

OPERATORS

INTRODUCTION

Operators are machine language code generated by the compiler and stored by the master control program in memory in the area allocated to program segments. (Program segments contain no data and are not modified by the processor as the program is executed.) Program segments are sequences of instructions which are moved by the CPM as 52-bit words from memory to the program buffer. Parity is checked on all 52-bits of each program word as it is brought to the program buffer.

Each program word consists of 48 bits, 3 tag bits, and an overall parity bit. Since information will be extracted from the program buffer in syllable form there is no way to check overall parity. Therefore, as the program word is parsed into six 8-bit syllables while being loaded into the program buffer, parity is also generated on each syllable of the word and stored in the buffer with each word. The parity of each syllable can thus be checked and the integrity of the program word maintained. Tag bits are not stored in the program buffer, instead a two-bit error code is stored in the program buffer to identify tag error conditions. Figure 3-4-1 illustrates the format of the program buffer word.

An instruction may be either a Value Call, a Name Call, or an operator. (Refer to table 3-4-1.) The two high-order bits (bits 7 and 6 in figure 3-4-1) of each instruction determine the type of instruction to be executed.

Value Call (VALC) is a two-syllable instruction that requires that the 14-bit address couple in the instruction be evaluated to provide an absolute address from which data is fetched and placed in the central data buffer (CDB) for EU use.

Name Call (NAMC) is a two-syllable instruction that indicates the address couple in the instruction may be used to form an IRW, which is then placed in the CDB. However, if the NAMC operator is followed by an operator which would require evaluation of the address couple to derive an absolute address, then the NAMC is concatenated with the following operator and the address couple is evaluated immediately. NAMC is concatenated when the next operator in the program code string is any of the following: INDX, NXLN, NXLV, STOD, STON, OVRD, OVRN, DBUN, LOAD, and PLDT. If a NAMC cannot be concatenated, an IRW containing the address couple is placed in CDB for DRU use.

The 14-bit address couple in the NAMC and VALC instructions consists of a lexicographic-level field (LL) and an index field (I). As shown in figure 3-4-2, the length of each of these fields varies with the current lexic level of the active program. The LL field ranges from one to five bits in length and contains only as many bits as are required to define the current lexic level. The remaining bits are the index field. (The bits of the LL field are in inverse order so that the least-significant bit of the field is located in the most significant bit position of the address couple.)

Value Call is a two-syllable instruction that brings an operand from memory into the top of stack. A concatenation of the two Value Call syllables gives a 14-bit address couple. If the referenced memory location is an indirect reference word or a data descriptor, additional memory accesses are made until the operand is located. The operand is then placed in the top of stack in the CDB. The operand may be either single-precision or double-precision, causing either one or two words to be loaded into the stack.

Name Call builds an indirect reference word in the CDB. The six low-order bits of the first syllable for this operator are concatenated with the eight bits of the following syllable to form a 14-bit address couple.

Operators vary from 1 to 12 syllables in length. The first syllable of each operator indicates the number of additional syllables forming the operator.

Operators work on data as either full word (48 data bits plus 3 tag bits) or as strings of data characters. Word operators work with operands (single-or double-precision) in the top of the stack.

Table 3-4-1. Instruction Decode Table

Instruction Type	Identification (Bits 7 & 6)	No. of Syllables	Function
Value Call	00	2	Brings an operand into the stack
Name Call	01	2	Brings an IRW into the stack
Operator	1x	1 to 12	Performs the specified operation

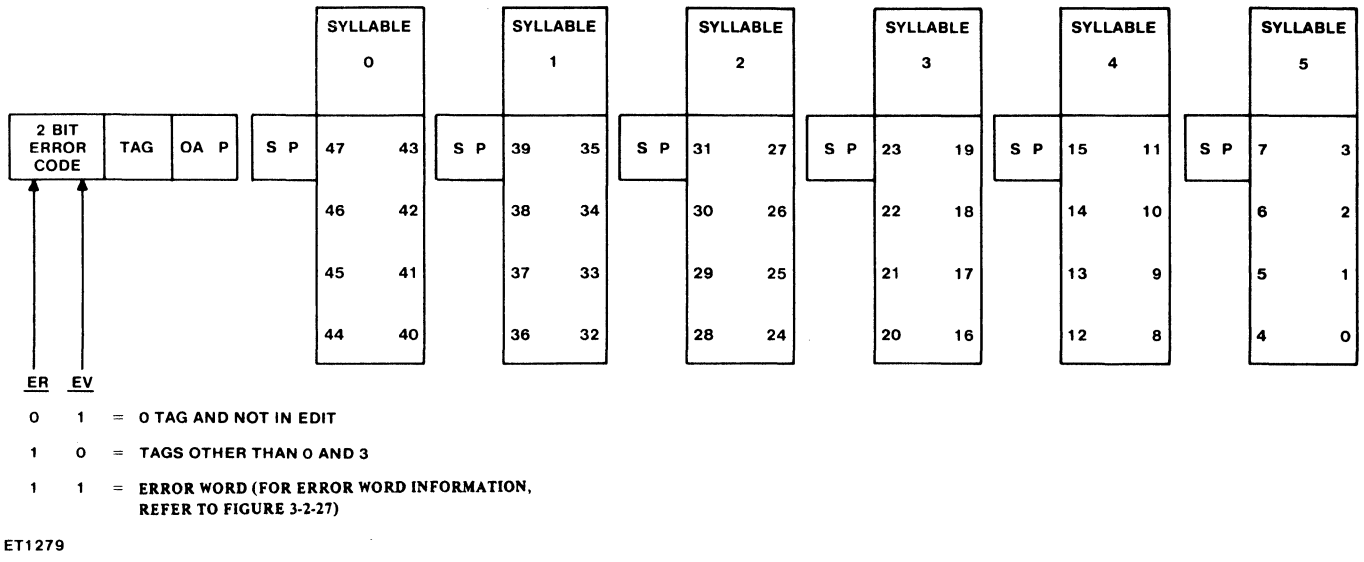


Figure 3-4-1. Program Buffer Word Format

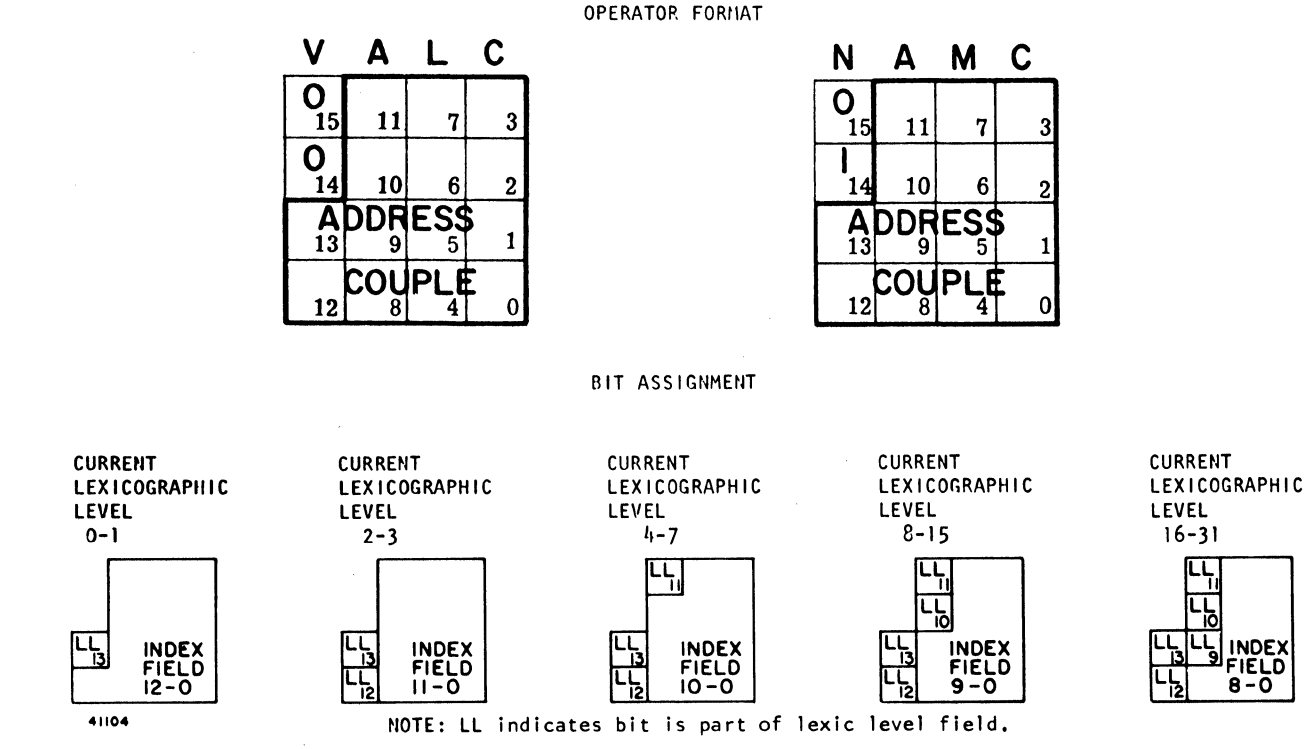


Figure 3-4-2. Address Couple Bit Assignment.

String operators are used for transferring, comparing, scanning, and translating strings of digits, characters, or bytes. In addition, a set of micro-operators (edit mode operators) provides a means of formatting data for input/output. String operators and edit mode operators use source and destination pointers located in the stack to set hardware registers.

In some of the string operators the source pointer may not be used. In this case, an operand may be in the stack; its characters are circulated as it is being used. String operators have an optional update function, producing updated source and destination pointers and counts.

If both the source and destination descriptors have size fields equal to 0, the size registers indicate eight-bit character size. When both a source and destination are required and only the size field of one is equal to 0, then the size field of the non-zero descriptor is used.

When neither size field is equal to 0, or the size fields are not equal, or the operator is not Translate or Transfer Words, the invalid-operand interrupt is set and the operator is terminated. The size field is considered equal to 0, when the source is an operand.

Logical operands may be either true (ON) or false (OFF). Logical values are the result of Boolean operations or relational operations. Relation operators generate a logical value as the result of an algebraic comparison of two arithmetic expressions. Bit 0 contains the logical value. Relational operators set bit 0, and conditional operators use bit 0 for the decision. Logical (Boolean) operators consider each bit from 47 to 0 as an individual logic value and operate on the whole operand.

GROUPING OF OPERATORS

Operators may be identified by name, mnemonic, or hexadecimal code. In this document to facilitate reference to the description of the operators, the operators are listed in the appendix in two ways: alphabetically by mnemonic, and sequentially by hexadecimal code. In each case the page number of the operator description is given.

When describing operators, considerable redundancy is eliminated by grouping operators with similar functions and only describing their differences. Also, for convenience of the user, operators used for related manipulations (such as arithmetic operators ADD, SUBT, MULT, DIVD) are described sequentially.

As shown in figure 3-4-3 all central processor program operators are grouped into one of four modes: primary (P), variant (V), edit (E). Several operators are classed as universal (U) because they can operate in any mode. (The letters in the above parentheses are used in this document as a mode-identifier prefix before the hexadecimal code associated with each operator. For example, (P) 80 indicates a primary mode operator and 80 is the hexadecimal code for the ADD operator.) In this document, the operator descriptions are grouped by mode. Preceding each group of descriptions for each mode is a list giving the order of specific operator descriptions.

The most frequently used operators are called primary mode operators. Each of the other modes is entered by first executing certain operators in primary mode. The "operator" portion of the primary mode operators begins with the first syllable and may extend for several syllables.

Primary mode operators are described in this document in the following groups: arithmetic, bit, branch, compare, enter edit mode, enter vector mode, index and load, input convert, literal call, logical, pack relational, scale, stack, store, string, string transfer, subroutine, transfer, type-transfer, miscellaneous, and universal. (In several cases a variant mode operator is conveniently described with a group of primary mode operators.)

Variant mode operators are less frequently used than primary mode operators and extend the number of hexadecimal codes available to identify the operators. Variant mode operators require two syllables. The first syllable of a variant mode operator has the hexadecimal code 95 which is the primary mode operator called Escape to 16-Bit Instruction (the mnemonic for this operator is VARI). The second syllable then gives the actual variant mode operation to be performed. The variant mode operators are described in this document in the following groups: scan, scan while, tag field, unpack, miscellaneous operators exclusive to the B 7800, and universal operators.

Edit mode operators perform edit functions (such as insert, move, and skip) on strings of data being prepared for output. The edit mode is entered from the primary mode via one of the enter edit operators (EXSD, EXSU, EXPU, TEED, or TEEU). Subsequent edit operators follow as either single micro-operators in the program string or as edit operators in a separate table which is executed as a program string. In edit mode the program buffer memory is reduced to 16 words (total available area) for processing the edit operators; the other 16 words contain the primary program syllables.

PRIMARY MODE (P) xx

1st ↓ / 2nd →	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0-3	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	0-3
4-7	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	4-7
8	ADD	SUBT	MULT	DIVD	IDIV	RDIV	NTIA	NTGR	LESS	GREQ	GRTR	LSEQ	EQUL	NEQL	CHSN	MULX	8
9	LAND	LOR	LNOT	LEQV	SAME	VARI	BSET	DBST	FLTR	DFTR	ISOL	DISO	INSR	DINS	BRST	DBRS	9
A	BRFL	BRTR	BRUN	EXIT	ST8R	NXLN	INDX	RETN	DBFL	DBTR	DBUN	ENTR	EVAL	NXLV	MKST	STFF	A
B	ZERO	ONE	LT8	LT16	PUSH	DLET	EXCH	DUPL	STOD	STON	OVRD	OVRN	PLDT	LOAD	LT48	MPCW	B
C	SCLF	DSLF	SCRT	DSRT	SCRS	DSRS	SCRF	DSRF	SCRR	DSRR	ICVD	ICVU	SNGT	SNGL	XTND	IMKS	C
D	TEED	PACD	EXSD	TWSD	TWOD	SISO	SXSN	ROFF	TEEU	PACU	EXSU	TWSU	TWOU	EXPU	RTFF	HALT	D
E	TLSD	TGED	TGTD	TLED	TEQD	TNED	TUND		TLSU	TGEU	TGTU	TLEU	TEQU	TNEU	TUNU		E
F	CLSD	CGED	CGTD	CLED	CEQD	CNED		FMMR	CLSU	CGEU	CGTU	CLEU	CEQU	CNEU	NOOP	NVLD	F

VARIANT MODE (V)xx

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
4			JOIN	SPLT	IDLE	SINT	EEXI	DEXI	IGPR		SCNI			PTPA	WHOI		4
8					PAUS	OCRX		NTGD	MIN		MAX	LOG2				INCN	8
A							RODI	SMMD*	SINH		SLMT		FMFR		ACDB*	MVST	A
B					STAG	RTAG	RSUP	RSDN	RPRR	SPRR	RDLK	CBON	LODT	LLLU	SRCH	STOP	B
D	USND	UABD	TWFD	TWTD	SWFD	SWTD	TCOD*	TRNS	USNU	UABU	TWFU	TWTU	SWFU	SWTU	RDEF	HALT	D
E																	E
F	SLSD	SGED	SGTD	SLED	SEQD	SNED			SLSU	SGEU	SGTU	SLEU	SEQU	SNEU	NOOP	NVLD	F

* Operator used for maintenance only.

EDIT MODE (E) xx

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
D	MINS	MFLT	SFSC	SRSC	RSTF	ENDF	MVNU	MCHR	INOP	INSG	SFDC	SRDC	INSU	INSC	ENDE	HALT	D
F															NOOP	NVLD	F

Figure 3-4-3. B 7800 CPM Program Operator Hexadecimal Code Assignments

Detection of an invalid operator condition terminates the operator, and an invalid operator interrupt is set in the fault register. The processor will proceed to process the interrupt whether it is in normal state or control state.

Invalid instructions are detected by the following methods:

1. Testing for unassigned operator codes. In the B 7800 all unassigned operators cause a programmed operator interrupt.
2. Testing for any value other than 011 in bit positions 50, 49, and 48 of any program word (an attempt to execute something which is not code). This results in an invalid program word interrupt except when in table mode which allows a tag 0 or a tag 3.
3. Testing for an invalid operator function; for example, an attempt to dial to a non-existent bit. This results in an invalid operand interrupt.

Bit 48 of each word in main memory is a memory protect bit. This bit is ON in all program words, indirect reference words, data descriptors, program descriptors, main memory storage links, and processor-generated control words.

PRIMARY MODE OPERATORS

Arithmetic Operators

Dyadic arithmetic operators require two operands in the top-of-stack storage. These operands are combined by the arithmetic process specified and are replaced with the resulting operand. These operands may be either single-precision, or double-precision, or intermixed types. The specified arithmetic process adapts automatically to the environment: a single-precision process is invoked if both operands are of the single-precision type and a double-precision process is invoked if either operand is of the double-precision type. Each double-precision operand occupies two words. The second word of the operand is an extension of the first word of the operand to form a 78-bit mantissa. Neither word is guaranteed to be an integer or a fraction. (For example, if an exponent is greater than +13, then every digit in both words is an integer.)

Add, subtract, multiply, and remainder divide operations with two integer single-precision operands yield an integer single-precision result if no overflow occurs in the case of add and multiply. If either or both operands are non-integer or if the result overflows, the result is non-integer.

ADD (ADD) (P)80

The add operator causes the two top-of-stack operands to be added algebraically and the sum to be left in the top of stack.

SUBTRACT (SUBT) (P)81

The Subtract operator causes the top-of-stack operand to be algebraically subtracted from the second operand in the stack and the result to be left in the top of stack.

MULTIPLY (MULT) (P)82

The Multiply operator causes the two top-of-stack operands to be algebraically multiplied and the product to be left in the top of stack. The result (or product) is right justified if there are 13 or less significant digits of result. If there are more than 13, the result is normalized and rounded.

EXTENDED MULTIPLY (MULX) (P)8F

The Extended Multiply operator causes the two top-of-stack operands to be algebraically multiplied and a double-precision product to be left in the top of stack. MULX with two single-precision inputs saves all 26 digits of the result without truncating.

DIVIDE (DIVD) (P)83

The Divide operator causes the second operand in the stack to be algebraically divided by the top-of-stack operand the quotient to be left in the top of stack.

If the mantissa of the second operand in the stack is 0, the exponent and quotient are set to 0. If the top-of-stack mantissa is 0, the divide-by-zero interrupt is set. In either case the operation is terminated.

INTEGER DIVIDE (IDIV) (P)84

The Integer Divide operator causes the second operand in the stack to be algebraically divided by the top-of-stack operand and the integer part of the quotient to be left in top of stack in integer form. If the mantissa of the second operand in the stack is 0, the exponent and quotient are set to 0. If the top-of-stack mantissa is 0, the divide-by-zero interrupt is set. In either case the operation is terminated.

REMAINDER DIVIDE (RDIV) (P)85

The Remainder Divide operator causes the second operand in the stack to be algebraically divided by the top-of-stack operand to develop an integer quotient. The reminder of this division is left in the top of stack. If both inputs to RDIV are single-precision integers, the remainder is also an integer and is left in the form of integer. If the mantissa of the second operand in the stack is 0, the exponent and quotient are set to 0. If the top-of-stack mantissa is 0, the divide-by-zero interrupt is set. In either case the operation is terminated.

INTEGERIZE, TRUNCATED (NTIA) (P)86

The Integerize (Truncated) operator converts the top-of-stack operand to an integer without rounding. The top-of-stack operand is always single-precision regardless of the inputs to NTIA. If the operand cannot be intergerized, that is, the exponent is greater than the number of leading 0's in the operand, the integer-overflow interrupt is set and the operation is terminated.

INTEGERIZE, ROUNDED (NTGR) (P)87

The Intergerize (Rounded) operator converts the top-of-stack operand to an integer with rounding. The top-of-stack operand is always single-precision regardless of the inputs to NTGR. Rounding takes place if the absolute value of the fraction is greater than or equal to 1/2. The sign of the input is also taken in consideration when 1 is added to the integer. If the sign is positive, then rounding occurs and the fraction is $\geq 1/2$. If the sign is negative, then 1 is added to the integer only if the fraction is greater than 1/2. If fraction is equal to a 1/2, then the integer part is not changed. If the operand cannot be integerized, that is, the exponent is greater than the number of leading 0's in the operand or a non-integer results from the rounding operation, the integer-overflow interrupt is set and the operation is terminated.

INTEGERIZE, ROUNDED, DOUBLE PRECISION (NTGD) (V)87

The Integerize (Rounded, Double Precision) operator converts the top-of-stack operand to a double-precision integer (exponent +13) with rounding.

Bit Operators

Bit operators set or reset bits in the top of stack or in the second item in the stack.

BIT SET (BSET) (P)96

The Bit Set operator sets a bit in the top of stack. The bit set corresponds to the value of the bit specified by the second syllable of the operator. If the program syllable defining the bit to be set has a value greater than 47, the invalid-operand interrupt is set and the operation is terminated. BSET can have an input of any kind of tag. Tag is preserved.

DYNAMIC BIT SET (DBST) (P)97

The Dynamic Bit set operator sets a bit in the second item in the stack. The bit set corresponds to the value of the bit specified by the top-of-stack operand. If the word in the top of stack is not an operand an invalid-operand interrupt is set and the operation is terminated. The word is integerized before it is used as a bit number. If after being integerized the operand is less than 0 or greater than 47, an invalid-operand interrupt is set and the operation is terminated.

BIT RESET (BRST) (P)9E

The Bit Reset operator resets a bit in the top of stack. The bit reset corresponds to the bit specified by the second syllable of the program operator. If the program syllable defining the bit to be reset has a value greater than 47, an invalid-operand interrupt is set and the operation is terminated. BRST can have an input of any kind of tag. Tag is preserved.

DYNAMIC BIT RESET (DBRS) (P)9F

The Dynamic Bit Reset operator resets a bit in the second item in the stack. The reset bit corresponds to the value of the bit specified by the top-of-stack operand. If the word in the top of stack is not an operand, an invalid-operand interrupt is set and the operation is terminated. The word is integerized before it is used as a bit number. If, after being integerized, the operand is less than 0 or greater than 47, an invalid-operand interrupt is set and the operation is terminated.

CHANGE SIGN BIT (CHSN) (P)8E

The Change Sign Bit operator complements (changes from 1 to 0 or from 0 to 1) the sign bit (bit 46) of the top-of-stack operand. CHSN must have an operand input (tag 0 or 2), otherwise an invalid-operand interrupt is set and the operation is terminated.

COUNT BINARY ONES (CBON) (V)BB

The Count Binary Ones operator counts the number of binary ones in the information part of the word in the top of stack and places this count in the top of stack. CBON must have an operand input (tag 0 or 2), otherwise an invalid-operand interrupt is set and the operation is terminated.

LEADING ONE TEST (LOG2) (V)8B

The Leading One Test operator locates the most significant information bit of the word in the top of stack. The number of that bit plus 1 is placed in the top of stack. If a one bit is not located, a 0 is placed in the top of stack. LOG2 can have an input of any kind of tag. Tag is preserved.

Branch Operators

Branch instructions function to break the normal sequence of serial instruction fetches. Branching may be either relative to the base address of the current program segment or to a location in some other program segment. Branch operators may be conditional or unconditional. Branch addresses are always checked for possible residency in the address associative memory.

BRANCH UNCONDITIONAL (BRUN) (P) A2

The Branch Unconditional operator replaces the contents of the program index register (PIR) and the program syllable register (PSR) with the next two syllables from the program string. The two syllables following the actual operator syllable provide the new PIR and PSR settings: the three high-order bits are placed in the PSR and the next 13 low-order bits are placed in the PIR.

BRANCH ON TRUE (BRTR) (P) A1

BRANCH ON FALSE (BRFL) (P)A0

In the B 7800, the conditional branch information (branch true, branch false, where branch is to, and what CDB location will be the Boolean location) is saved in the PCU to obtain a new code, if branch is required. When Boolean is being written into a CDB location by the EU, the LSB of that Boolean is monitored by the PCU. By comparing this CDB location with saved CDB location, the proper true/false bit is selected for comparison with LSB of Boolean.

If true/false bit and LSB of Boolean are equal, the branch is performed to obtain address of new code string.

If these values are not equal, the branch is discarded and the program string is continued in sequence.

DYNAMIC BRANCH UNCONDITIONAL (DBUN) (P)AA

If the top-of-stack word is either a program control word or an indirect reference to a PCW, the Dynamic Branch Unconditional operator branches to the specified syllable of the program segment. PCW can be found after chaining through normal IRWs. Note that a stuffed IRW cannot be used to find a PCW for the DBUN. If a stuffed IRW is encountered (bit 46 ON), a sequence-error interrupt occurs.

If the top-of-stack word is an operand, the program index register and program syllable register are set according to the contents of this operand as follows: The operand is made into an integer. If bit zero of the operand is 0, PSR is set to 0; otherwise, if bit zero of the operand is 1, PSR is set to 3. The next higher-order 16 bits are placed in the PIR.

DYNAMIC BRANCH TRUE (DBTR) (P)A9

If the low-order bit of the second word in the stack is a 1 and the top-of-stack word is a program control word (PCW) or an indirect reference to a

PCW, the Dynamic Branch True operator will cause a branch to the specified syllable in the program segment. Otherwise, a 1 is added to the PIR and PSR and the program continues in sequence.

If the low-order bit of the second word in the stack is a 1 and the top-of-stack word is an operand, PIR/PSR are replaced from this operand as in the DBUN operator. Otherwise, PIR and PSR are advanced and the program string continues in sequence.

DYNAMIC BRANCH FALSE (DBFL) (P)A8

If the low-order bit of the second word in the stack is a 0, and the top-of-stack word is a program control word or an indirect reference to a PCW, the Dynamic Branch False operator causes a branch to the specified syllable of the program segment. Otherwise, the PIR/PSR are continued in sequence.

If the low-order bit of the second word in the stack is a 0 and the top-of-stack word is an operand, PIR/PSR are replaced from this operand as in the DBUN operator. Otherwise, PIR and PSR are advanced and the program string is continued in sequence.

Compare Operators

The compare operators perform the specified compare of two strings of data. The true/false flip-flop is conditioned by the results of the compare.

In the B 7800, the 48-bit words are assembled and shifted for direct comparison. The boundaries of these words are determined by the destination words. The destination words end being unshifted, and the characters which are not part of the string (as defined by the original descriptor plus the length) are zeroed out.

The destructive operators are completed as soon as the results are known. Although the results are known, the Compare Update operators continue until the lengths are exhausted. If a segmented array interrupt occurs, the result is saved in the RCW.

COMPARE CHARACTERS GREATER, DESTRUCTIVE (CGTD) (P)F2

The Compare Characters Greater, Destructive operator makes a character-by-character comparison of two strings of data until it finds an unequal pair. (All comparisons are by the binary character position in the collating sequence.) If the characters in the B string (destination) are greater than the characters in the A string (source), the true/false flip-flop is set to one; otherwise, the true/false flip-flop is set to zero. If the repeat count is less than or equal to 0, the true/false flip-flop is reset.

The top of stack is an operand which specifies the number of characters to be compared. The second item in the stack is an operand or descriptor pointing at the source character string against which comparisons are to be made. The third item in the stack is a descriptor pointing to the character string to be compared. If either of the data strings has the memory protect bit ON (bit 48=1), the segmented array interrupt is set, and the operation is terminated.

COMPARE CHARACTERS GREATER, UPDATE (CGTU) (P)FA

The Compare Characters Greater, Update operator performs a compare characters greater, destruction operation except that the accesses to memory continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS GREATER OR EQUAL, DESTRUCTIVE (CGED) (P)F1

The Compare Characters Greater or Equal, Destructive operator performs a compare characters greater, destructive operation except that the true/false flip-flop is set to true if the destination is greater than or equal to the source.

COMPARE CHARACTERS GREATER OR EQUAL, UPDATE (CGEU) (P)F9

The Compare Characters Greater or Equal, Update operator performs a compare characters greater or equal, destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS EQUAL, DESTRUCTIVE (CEQD) (P)F4

The Compare Characters Equal, Destructive operator performs a compare characters greater, destructive operation except that the true/false flip-flop is set to true if the source is equal to the destination.

COMPARE CHARACTERS EQUAL, UPDATE (CEQU) (P)FC

The Compare Characters Equal, Update operator performs a compare characters equal, destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS LESS OR EQUAL, DESTRUCTIVE (CLED) (P)F3

The Compare Characters Less or Equal, Destructive operator performs a compare characters greater, destructive operation except that the true/false flip-flop is set to true if the destination is less than or equal to the source.

COMPARE CHARACTERS LESS OR EQUAL, UPDATE (CLEU) (P)FB

The Compare Characters Less or Equal, Update operator performs a compare less or equal, destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS LESS, DESTRUCTIVE (CLSD) (P)F0

The Compare Characters Less, Destructive operator performs a compare characters greater, destructive operation except that the true/false flip-flop is set to true if the destination is less than the source.

COMPARE CHARACTERS LESS, UPDATE (CLSU) (P)F8

The Compare Characters Less, Update operator performs a compare characters less, destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS NOT EQUAL, DESTRUCTIVE (CNED) (P)F5

The Compare Characters Not Equal, Destructive operator performs a compare characters greater, destructive operation except that the true/false flip-flop is set to true if the source is not equal to the destination.

COMPARE CHARACTERS NOT EQUAL, UPDATE (CNEU) (P)FD

The Compare Characters Not Equal, Update operator performs a compare characters not equal, destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

Enter Edit Mode Operators

Enter edit mode operators provide the means for transition from primary mode operation to edit mode operation. The edit mode operators in a program string are entered via the Execute Single Micro or Single Pointer. Edit mode operators may also be in a table in which case they are entered by the Table Enter Edit operator. (See also the descriptions under "Edit Mode Operators.")

TABLE ENTER EDIT, DESTRUCTIVE (TEED) (P)D0

The Table Enter Edit, Destructive operator is used to control edit micro-instructions which are contained in memory as a table rather than as part of the normal program string. This operator causes characters to be transferred from the source string to the destination string. The transfer is under control of the string of edit micro-operators which are located by the table pointer.

In normal cases, the top-of-stack word (a descriptor) is the table pointer, the second word (a single-precision operand or descriptor) in the stack is the source pointer, and the third word in the stack (a descriptor) is the destination pointer. However, if segmented bit (bit 44) is ON, then the second word in the stack, instead of being the source pointer, is the length to be used by the edit operator. The TIR in the CPM points to that edit operator.

If the first word in the stack is not a descriptor, the invalid-operand interrupt is set and the operation is terminated. If the second item in the stack is a single-precision operand, it is a source string. If the third item in the stack is not a descriptor, the invalid-operand interrupt is set and the operation is terminated. In table mode, the micro-operator words can be tagged as single-word operands (tag 0).

TABLE ENTER EDIT, UPDATE (TEEU) (P)D8

The Table Enter Edit, Update operator performs a table enter edit destructive operation. At the completion of the operation, the source pointer and destination pointer are updated.

EXECUTE SINGLE MICRO, DESTRUCTIVE (EXSD) (P)D2

The Execute Single Micro, Destructive operator transfers characters from the source string to the destination string under the control of the single micro-operator which follows this operator syllable.

The first item in the stack is a single-precision operand that defines the field length and is used as a micro-operator repeat field. The second item in the stack is the source pointer, and the third item in the stack is the destination pointer.

EXECUTE SINGLE MICRO, UPDATE (EXSU) (P)DA

The Execute Single Micro, Update operator performs an execute single micro, destructive operation. At the completion of the operation, the source pointer and destination pointer are updated.

EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE (EXPU) (P)DD

The Execute Single Micro, Single Pointer Update operator performs an execute single micro, destructive operation. At the completion of the operation, the pointer is updated.

The top-of-stack operand is used as a micro-operator repeat field. The second item in the stack is used to set both the source and destination pointers. Only the destination pointer is updated.

Index and Load Operators

The index and load operators provide the means to index the top-of-stack word and the means to load an operand or descriptor into the top of stack.

INDEX (INDX) (P)A6

The two top-of-stack items are a descriptor (or indirect reference to a descriptor) and an operand. The operand is used to index the descriptor. The Index operator places the integerized value of the second item in the stack into the 20-bit length/index field of the descriptor in the top-of-stack. The descriptor is marked indexed (bit 45 is set to 1).

If the word in the top of stack is an operand, the top-of-stack operand is exchanged with the second-item operand. If the word in the top of stack is neither a descriptor nor an indirect reference word pointing to a descriptor, the invalid-operand interrupt is set and the operation is terminated.

If the indexing value is negative or greater than or equal to the length field of the descriptor, the invalid-index interrupt is set and the operation is terminated.

If the descriptor represents an array which is segmented, the index is partitioned into two portions by dividing it by the proper divisor determined by the type of data referenced by the descriptor (D.P. word-128 S.P. word-256, four-bit digit-3072, six-bit character-2048, or eight-bit byte-1536). The quotient is used as an index to the given descriptor to fetch the array-row descriptor. The remainder is used to index the row descriptor.

Copy action can be performed in the Index operator because an IRW can be pointing to a non-present MOM descriptor.

If the double-precision bit (bit 40) in the descriptor is 1, the index value in the second item is doubled. The balance of the operation is as described in the first paragraph of this operator.

If the presence bit (bit 47) and copy bit (bit 46) are OFF, the address of the original descriptor is placed in the address field of the stack copy. If the word accessed by the indexed word in the top of stack is not a data descriptor, the invalid-operand interrupt is set and the operation is terminated. If the data descriptor accessed by the indexed word in the top of stack has the index bit (bit 45) set to 1, the invalid-operand interrupt is set and the operation is terminated.

INDEX AND LOAD NAME (NXLN) (P)A5

The Index and Load Name operator performs an Index operation. After the word in the top-of-stack is indexed, the data descriptor pointed to by this word is brought to the top-of-stack, the copy bit (bit 46) of the data descriptor is set to one.

If the presence bit (bit 47) and copy bit (bit 46) are OFF, the address of the original descriptor is placed in the address field of the stack copy. If the word accessed by the indexed word in the top-of-stack is not a data descriptor, the invalid operand interrupt is set and the operation is terminated. If the data descriptor accessed by the indexed word in the top-of-stack has the index bit (bit 45) set to one, the invalid-operand interrupt is set and the operation is terminated.

INDEX AND LOAD VALUE (NXLV) (P)AD

The Index and Load Value operator performs an Index operation. After the word in the top-of-stack is indexed, the operand pointed to by this descriptor is brought to the top-of-stack.

If the presence bit (bit 47) and copy bit (bit 46) are OFF, the address of the original descriptor is placed in the address field of the stack copy. If the word accessed by the indexed word in the top-of-stack is not a data descriptor, the invalid-operand interrupt is set and the operation is terminated. If the data descriptor accessed by the indexed word in the top-of-stack has the index bit (bit 45) set to one, the invalid-operand interrupt is set and the operation is terminated.

LOAD (LOAD) (P)BD

The Load operator places the word addressed by the indirect reference word or by the indexed data descriptor in the top of stack.

If input to Load operator is neither a data descriptor nor an IRW, the invalid-operand interrupt is set.

If the target is a descriptor, the copy bit is set.

If the descriptor is both non-present (bit 47=0) and non-copy (bit 46=0) when fetched, copy action occurs. The address from which the descriptor was fetched is placed in the address field of the copy of the descriptor left on the top of the stack.

LOAD TRANSPARENT (LODT) (V)BC

The Load Transparent operator saves the target data unchanged as the result in the top of stack. If target data is neither an IRW nor an indexed descriptor, the LS 20 bits of the target data are extracted and used as an address. Whatever is referenced by this address is then fetched.

Input Convert Operators

The input convert operators convert the various character sets (digits BLC, EBCDIC, or ASCII) to operands for arithmetic operations.

INPUT CONVERT, DESTRUCTIVE (ICVD) (P)CA

The Input Convert, Destructive operator converts four-bit digit, or six-bit BCL, or eight-bit EBCDIC (or ASCII) to an operand for internal arithmetic operations.

The first items in the stack is an operand that is integerized to form the repeat field. The second item in the stack is a descriptor used as a source pointer.

The specified number of characters are transferred from the source string to the top of stack. Only the numeric portion of the character is transferred. The transferred string is converted to a double-precision operand if the length is ≥ 12. If a double-precision operand is produced, the true/false flip-flop is set to false; otherwise, it is set to true. The sign bit of the operand is set negative if the zone of the last character transferred is 10_2 (for six-bit characters) or 1101_2 (for eight-bit characters). The tag field is set to indicate a single-or double-precision operand.

INPUT, CONVERT, UPDATE (ICVU) (P)CB

The Input Convert, Update operator performs an Input Convert, Destructive operation.

If the source is an operand, the source is rotated so that the next character to be used is left justified. At the completion of the operation, the source pointer is updated.

Literal Call Operators

The literal call operators place defined-value operands in the top of stack.

LIT CALL ZERO (ZERO) (P)B0

The Lit Call Zero operator places in the top of stack a single-precision operand with a value of 0.

LIT CALL ONE (ONE) (P)B1

The Lit Call One operator places in the top of stack a single-precision operand with a value of 1.

LIT CALL 8 BITS (LT8) (P)B2

The Lit Call 8 Bits operator places in the top of stack a single-precision operand equal in value to the second syllable of this operator.

LIT CALL 16 BITS (LT16) (P)B3

The Lit Call 16 Bits operator places in the top of stack a single-precision operand equal in value to the second and third syllables of this operator.

LIT CALL 48 BITS (LT48) (P)BE

The Lit Call 48 Bits operator places in the top of stack a single-precision operand equal in value to the next program word.

NOTE

Since the literal is synchronized by word, this operator can be 7 to 12 syllables long. Any unused syllables are filled in with the invalid operator code.

MAKE PROGRAM CONTROL WORD (MPCW) (P)BF

The Make Program Control Word operator performs a Lit Call 48 Bits operation except that the tag field is set to 111 to indicate a program control word and the stack number field of the PCW is inserted from the stack number register.

Logical Operators

Logical operators operate on the two top-of-stack operands bit for bit from bit 47 through bit 0 to obtain logical values (48 logical values for single-precision operands and 96 for double-precision operands) which are left as the top-of-stack operand. If only one of the operands associated with LAND, LOR, LNOT, or LEQV is a double-precision operand, then the other operand will be extended with 0's. Logical operators may be used to operate on logical, string, or numeric operands.

LOGICAL AND (LAND) (P)90

The Logical AND operator logically ANDs each bit (except tag bits) of the two top-of-stack operands leaving the result in the top of stack. Each bit of the top-of-stack operand is set to 1 where a 1 appears in the corresponding bit positions of the two top-of-stack operands; the other information bits in the top-of-stack operand are set to 0. The tag of the second operand is undisturbed except for a double-precision operand in the top of stack, in which case the second operand is made double precision and the tag field is changed accordingly. AND is defined as follows:

Operand A	Operand B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

NOTE
The tag field is set equal to the second item in the stack.

LOGICAL OR (LOR) (P)91

The Logical OR operator logically ORs each bit (except tag bits) of the two top-of-stack operands leaving the result in the top of stack. OR is defined as follows:

Operand A	Operand B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

NOTE
The tag field is set equal to the second item in the stack.

LOGICAL NEGATE (LNOT) (P)92

The Logical Negate operator complements each bit position (except tag bits) of the top-of-stack operand.

LOGICAL EQUIVALENCE (LEQV) (P)93

The Logical Equivalence operator compares the corresponding bits of the two items in the top of stack (except the tag bits). The two items are replaced both by a single item with a tag field equal to the tag field of the second item in the stack and by a 1 in each bit position where the corresponding bits of the two top-of-stack items were equal.

Pack Operators

PACK, DESTRUCTIVE (PACD) (P)D1

The Pack, Destructive operator packs data (as addressed by the source pointer) right justified into the top of stack in four-bit (digit) format.

The top-of-stack operand defines the length/repeat field (in digits) to be packed. The source pointer is the second item in the stack. The specified number of digits are transferred from the source to the top of stack (dropping the zones when required). If the digit length transferred is less than 13, the tag field in the top of stack is set to a single-precision operand; otherwise, the tag field is set to a double-precision operand.

If the length is not less than 25, an invalid-operand interrupt is set and the operation is terminated. If the source data has the memory protect bit (bit 48) set to 1, the segmented-array interrupt is set and the operation is terminated.

if the sign of the source data is negative, the true/false flip-flop is set to 1; otherwise, the flip-flop is reset. Sign conventions are as follows:

Data Bit Format	Sign Location	Neg. Sign Zone Bit Config.
4-bit	Most significant digit	1101
6-bit	Least significant character	10
8-bit	Least significant byte	1101 (EBCDIC)
8-bit	Least significant byte	1111 (ASCII)

PACK, UPDATE (PACU) (P)D9

The Pack, Update operator performs a Pack, Destructive operation. At the completion of the operation, the source pointer is updated.

Relational Operators

The relational operators perform algebraic comparisons on the two top-of-stack operands. The operands are removed from the stack and the result of the comparison is a logical operand which is placed in the top of stack. The result is a single-precision operand with the least significant bit set to 1 if the relation is true or a single-precision operand with all information bits set to 0 if the relation is false.

GREATER THAN (GRTR) (P)8A

If the second operand in the stack is greater than the top of stack operand, the Greater Than operator replaces the two operands with a single-precision operand which has the least significant bit set to 1.

If the second operand in the stack is not greater than the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to 0.

GREATER THAN OR EQUAL (GREQ) (P)89

If the second operand in the stack is greater than or equal to the top-of-stack operand, the Greater Than or Equal operator replaces the two operands with a single-precision operand which has the least-significant bit set to 1.

If the second operand in the stack is not greater than or equal to the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to 0.

EQUAL (EQUL) (P)8C

If the second operand in the stack is algebraically equal to the top-of-stack operand, the Equal operator replaces the two operands with a single-precision operand which has the least significant bit set to 1. If the second operand in the stack is not equal to the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to 0.

LESS THAN OR EQUAL (LSEQ) (P)8B

If the second operand in the stack is less than or equal to the top-of-stack operand, the Less Than or Equal operator replaces the two operands with a single-precision operand which has the least significant bit set the 1. If the second operand in the stack is not less than or equal to the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to 0.

LESS THAN (LESS) (P)88

If the second operand in the stack is less than the top-of-stack operand, the Less Than operator replaces the two operands with a single-precision operand which has the least significant bit set to 1. If the second operand in the stack is not less than the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to 0.

NOT EQUAL (NEQL) (P)8D

If the second operand in the stack is not equal to the top-of-stack operand, the Not Equal operator replaces the two operands with a single-precision operand with the least significant bit set to 1. If the second operand in the stack is equal to the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to 0.

LOGICAL EQUAL (SAME) (P)94

The Logical Equal operator compares all bits (including tag bits) of the two items (operands, controls words, descriptors) in the top of stack. If all bits are equal, a single-precision operand (with the least significant bit set to 1 and all other information bits set to 0) is stored in the top of stack; otherwise, a single-precision operand with all information bits set to 0 is stored in the top of stack.

Scale Operators

The scale-left operators provide a means of aligning the decimal points prior to performing arithmetic operations. The scale-right operators provide a means of converting binary arithmetic to decimal arithmetic. Most important operator in this group is Scale-Right Final. In most cases, the other scale operators are not used, except Scale Right Save.

SCALE LEFT (SCLF) (P)C0

The Scale Left operator shifts the operand in the top of stack for decimal point alignment. The operand in the top of stack is first converted to an integer and then multiplied by 10 raised to the power specified by the scale factor. The scale factor is obtained from the second syllable (the program syllable following the operator syllable).

If scaling of a single-precision operand would result in overflow, the single-precision operand is converted to a double-precision operand integer. For the scale operators, a double-precision integer is defined as a double-precision operand with an exponent equal to 13 (octal). If scaling of the operand results in an exponent greater than 13 (double-precision operand), the overflow flip-flop is set to 1.

DYNAMIC SCALE LEFT (DSLF) (P)C1

The Dynamic Scale Left operator performs a Scale Left operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second operand in the stack. The operand in the top-of-stack is converted to an integer before scaling takes place.

SCALE RIGHT SAVE (SCRS) (P)C4

The Scale Right Save operator shifts the top-of-stack operand to the right for conversion from a binary to a decimal numbering system. The operand in the top of stack is converted to an integer and divided by 10 raised to the power specified by the scale factor. The scale factor is obtained from the second syllable. If the scale factor is greater than 12, the invalid-operand interrupt is set and the operation is terminated.

The binary quotient resulting from the division is left in the top of stack. The second operand in the stack is the remainder which is converted to decimal (four-bit digits) and left justified.

DYNAMIC SCALE RIGHT SAVE (DSRS) (P)C5

The Dynamic Scale Right Save operator performs a Scale Right Save operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second item in the stack. The top-of-stack operand is converted to an integer before scaling takes place.

SCALE RIGHT TRUNCATE (SCRT) (P)C2

The Scale Right Truncate operator performs a Scale Right Save operation except that the remainder resulting from the division is deleted from the stack.

DYNAMIC SCALE RIGHT TRUNCATE (DSRT) (P)C3

The Dynamic Scale Right Truncate operator performs a Scale Right Truncate operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second operand in the stack.

SCALE RIGHT ROUNDED (SCRR) (P)C8

The Scale Right Rounded operator performs a Scale Right Save operation except that the remainder resulting from the division is deleted from the stack. If the most significant digit of the remainder is greater than or equal to 5 the quotient from the division is rounded by adding 1 to it.

DYNAMIC SCALE RIGHT ROUNDED (DSRR) (P)C9

The Dynamic Scale Right rounded operator performs a Scale Right Rounded operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second operand in the stack.

SCALE RIGHT FINAL (SCRF) (P)C6

The Scale Right Final operator performs a Scale Right Save operation except that the quotient is deleted from the stack and the sign of the quotient is copied into the external sign flip-flop. If the quotient was not equal to 0 at the conclusion of the operation, the overflow flip-flop is set.

DYNAMIC SCALE RIGHT FINAL (DSRF) (P)C7

The Dynamic Scale Right Final operator performs a Scale Right Final operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second item in the stack.

Stack Operators

The stack operators are used to adjust the relative positions of the top items in the stack and to copy or delete the top-of-stack item.

EXCHANGE (EXCH) (P)B6

The Exchange operator causes the two top-of-stack items to be exchanged.

ROTATE STACK DOWN (RSDN) (V)B7

The Rotate Stack Down operator rotates the three top-of-stack words as follows:

Before Rotation	After Rotation
Word 1	Word 2
Word 2	Word 3
Word 3	Word 1

ROTATE STACK UP (RSUP) (V)B6

The Rotate Stack Up operator rotates the three top-of-stack words as follows:

Before Rotation	After Rotation
Word 1	Word 3
Word 2	Word 1
Word 3	Word 2

DUPLICATE TOP-OF-STACK (DUPL) (P)B7

The Duplicate Top-of-Stack operator duplicates the item in the top of stack.

DELETE TOP-OF-STACK (DLET) (P)B5

The Delete Top-of-Stack operator deletes the top-of-stack item.

PUSH DOWN STACK REGISTERS (PUSH) (P)B4

The Push Down Stack Registers operator pushes down the top-of-stack items and stack buffer contents into memory.

STORE DESTRUCTIVE (STOD) (P)B8

The Store Destructive operator stores the second item in the stack into memory. The address into which the item is to be stored is indicated by an indirect reference word or indexed data descriptor in the top of stack. If the top-of-stack item is an operand, the two top-of-stack items are exchanged so that the address item is in the top of stack and the item to be stored is in the second position. After the item is stored, both the item and its address are deleted from the stack.

If the word addressed by the indirect reference word is another indirect reference word or indexed data descriptor, or if the word addressed by the data descriptor is another indexed data descriptor, the store operation will not occur at that location, but will be retried using the address indicated by that word. This chaining of address items will continue until a "target" location is reached; however, once a data descriptor has been encountered, an indirect reference word or PCW is not allowed, and once a stuffed indirect reference word has been encountered, a normal IRW is not allowed. Either of these conditions will cause an invalid-operand interrupt.

If the word addressed by the indirect reference word is a program control word, accidental procedure entry occurs. The spontaneously generated RCW causes STOD to be re-executed upon return from the procedure.

If a data descriptor used as an address item has the read-only bit (bit 43) ON, or if the addressed word has the memory protect bit (bit 48) ON and is not a data descriptor, IRW, or PCW, the memory-protect interrupt is set and the operation is terminated.

If the presence bit in the data descriptor is 0, the presence-bit interrupt is set. After the data has been made present, the operation is restarted.

If the flashback had a tag of 2 (for an IRW address) and data for storage is single precision, then the XTND micro operator is called to convert the single-precision data to double-precision data.

If double-precision bit (bit 40) is OFF (for a descriptor address), the data for storage should be single precision. However, if the data is double precision, the SNGL micro operator is called to convert the double-precision data to single-precision data.

In either case, a PCU restart is required. The PCU restart PROM table is used to issue a PIE level XTND or SNGL micro operator, along with an exchange, before store is reissued.

STORE NON-DESTRUCTIVE (STON) (P)B9

The Store Non-Destructive operator performs a Store Destructive operation, except that only the address item is deleted from the stack. The item which was stored is left in the top of stack.

OVERWRITE DESTRUCTIVE (OVRD) (P)BA

The Overwrite Destructive operator performs a Store Destructive operation, except that the addressed location will be overwritten regardless of its contents. Chaining of address items, memory protection checks, or accidental procedure entry do not occur.

OVERWRITE NON-DESTRUCTIVE (OVRN) (P)BB

The Overwrite Non-Destructive operator performs a Store Non-Destructive operation, except that the addressed location will be overwritten regardless of its contents. Chaining of address items, memory protection checks, or accidental procedure entry do not occur.

READ WITH LOCK (RDLK) (V)BA

The Read With Lock operator is a variant of the Overwrite Non-Destructive operator. The word in the top of stack and the specified word in memory are interchanged after all local data is purged.

String Operators

The string operators are used for transferring, comparing, scanning, and translating strings of data. In addition, a set of micro-operators provides a means of formatting data for input/output.

The string operators use a repeat value and source and destination pointers which are located in the stack. For most string operators, the repeat value range is from 0 to 220-1. If the repeat value is ≤ 0, the string operator checks for valid inputs and terminates. If the string operator is an update type operator, the normal updated descriptors are produced.

The source for the string operator can either be a pointer into an array or a single or double precision operand. If the source is an operand, the source character size is determined by either the string operator or the destination character size. The first source character to be used by the string operator is the left-most character in the most significant word of the operand.

As the string operator acts upon each character in the operand, the operand is rotated left by one character so that the next character to be used is always the left-most character in the rotated source operand. For update type string operators, the operand is placed back into the stack in its rotated form. The source and destination pointers can be:

1. An unindexed data descriptor.
2. An indexed data descriptor.
3. An unindexed string descriptor.
4. An indexed string descriptor.

When one descriptor (source or destination) is a data descriptor and the other is a string descriptor, the data descriptor is converted to a string descriptor of the same type.

If both descriptors are data descriptors or there is only one descriptor and it is a data descriptor, then the conversion is made to 8-bit character string descriptors. Note that the index field inserted into converted string descriptors is the same as that found in the original descriptors.

If string descriptors, except for the translate and transfer word operators, contain different character sizes, the invalid-operand interrupt is caused.

If string operators contain an update variant, the indexed string descriptors pointing to the next character in the array to be used are left in the stack.

STRING ISOLATE (SISO) (P)D5

The String Isolate operator transfers from the source string to the top-of-stack the number of bytes specified by the repeat field. This string is right-justified and filled with leading zeros.

At the start of the operation, the top-of-stack operand specifies the length of the byte string and the second item in the stack is an operand or a descriptor used as the source pointer. If the number of bytes exceeds one word (6 bytes or 48 bits), the tag of the result is set to double precision. if the number of bits is greater than 96, an invalid operand interrupt is set and the operation is terminated. If the source data has the memory-protect bit (bit 48) set to one, the segmented-array interrupt is set and the operation is terminated.

String Transfer Operators

String transfer operators give the system the ability to transfer characters or words from one location in memory to another location in memory.

TRANSFER WORDS, DESTRUCTIVE (TWSD) (P)D3

The Transfer Words, Destructive operator transfers the number of words specified by the top-of-stack operand from the source string to the destination string. The first operand is integerized and is used as the count or repeat field. The second item in the stack (a string descriptor or operand) is the source pointer; i.e., it points at the source string. The third item in the stack (a string descriptor) is the destination pointer which is used to provide the address of the destination string. The number of words specified by the repeat field are transferred from the source to the destination. If an odd-tagged word is encountered in the source during distribution, a segmented array interrupt is generated.

TRANSFER WORDS, UPDATE (TWSU) (P)DB

The Transfer Words, Update operator performs a Transfer Words, Destructive operation. At the completion of the operation the source and destination pointers are updated to point to the next memory location which would have been processed if the length had not been exhausted. If either pointer was a data descriptor, then an indexed data descriptor is updated.

TRANSFER WORDS, OVERWRITE DESTRUCTIVE (TWOD) (P)D4

The Transfer Words, Overwrite Destructive operator performs a Transfer Words, Destructive operation bypassing the memory-protection checks.

TRANSFER WORDS, OVERWRITE UPDATE (TWOU) (P)DC

The Transfer Words, Overwrite Update operator performs a Transfer Words, Update operation bypassing the memory-protection checks.

TRANSFER WHILE GREATER, DESTRUCTIVE (TGTD) (P)E2

The Transfer While Greater, Destructive operator transfers the number of characters specified by the second operand (bits 19:20) in the stack or while the source character is greater than a delimiter. The top-of-stack operand is the delimiter. The third item in the stack is the source pointer, and the fourth item is the destination pointer.

If the second item in the stack is a descriptor, it is used as a source pointer. This means that no repeat field was given and the default field length is 1,048,575.

If either the source or destination word has the memory protect bit ON (bit 48 = 1), the segmented-array interrupt is set and the operation is terminated.

All comparisons are binary (EBCDIC collating sequence). The source character is compared with the delimiter. If the comparison is true, the true/false flip-flop is set to one; if the comparison fails, the true/false flip-flop is set to zero.

TRANSFER WHILE GREATER, UPDATE (TGTU) (P)EA

The Transfer While Greater, Update operator performs a Transfer While Greater, Destructive operation. At the completion of the operation, the source and destination pointers are updated to point at the next character in the source and destination strings, respectively. At the completion of the operation, a count of the number of characters not transferred is placed on the top-of-stack. If all the characters specified by the length field are transferred, the true/false flip-flop is set to true; otherwise, the true/false flip-flop is set to false.

If the operation is terminated because the relationship is not met, the source pointer points to the character which stopped the transfer.

TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE (TGED) (P)E1

The Transfer While Greater or Equal operator performs a Transfer While Greater, Destructive operation while the source character is greater than or equal to the delimiter.

TRANSFER WHILE GREATER OR EQUAL, UPDATE (TGEU) (P)E9

The Transfer While Greater or Equal, Update operator performs a Transfer While Greater Than or Equal operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE EQUAL, DESTRUCTIVE (TEQD) (P)E4

The Transfer While Equal, Destructive operator performs a Transfer While Greater or Equal, Destructive operation while the source character is equal to the delimiter.

TRANSFER WHILE EQUAL, UPDATE (TEQU) (P)EC

The Transfer While Equal, Update operator performs a Transfer While Equal, Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE (TLED) (P)E3

The Transfer While less or Equal, Destructive operator performs a Transfer While Greater or Equal, Destructive operation while the source character is less than or equal to the delimiter.

TRANSFER WHILE LESS OR EQUAL, UPDATE (TLEU) (P)EB

The Transfer While Less or Equal, Update operator performs a Transfer While Less or Equal, Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE LESS, DESTRUCTIVE (TLSD) (P)E0

The Transfer While Less, Destructive operator performs a Transfer While Less or Equal, Destructive operation while the source character is less than the delimiter.

TRANSFER WHILE LESS, UPDATE (TLSU) (P)E8

The Transfer While Less, Update operator performs a Transfer While Less, Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE NOT EQUAL, DESTRUCTIVE (TNED) (P)E5

The Transfer While Not Equal, Destructive operator performs a Transfer While Greater or Equal, Destructive operation while the source character is not equal to the delimiter.

TRANSFER WHILE NOT EQUAL, UPDATE (TNEU) (P)ED

The Transfer While Not Equal, Update operator performs a Transfer While Not Equal Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE TRUE, DESTRUCTIVE (TWTD) (V)D3

The Transfer While True, Destructive operator transfers characters from the source string to the destination string for the number of characters specified by the length operand while the stated relationship is met. If the relationship is not met the transfer is terminated at that point. The relationship is determined by using the source character to index a bit in the table. If the bit indexed is a one, the relationship is true. An all zero's character indexes to the most significant bit of the table.

The operator uses the top four words in the stack as follows: The top-of-stack word is a table pointer to specific addresses in the table; the second word in the stack provides the length of the string to be transferred or, it is is a descriptor, it is used as a source pointer since no repeat field was given and the default field length is set at 1,048,575; the third word in the stack is an operand or a descriptor which gives the address of the source string or is a single-precision operand which is the source string; the fourth word in the stack is a descriptor pointing at the destination string.

The table is indexed as follows to obtain the decision bit: The source character is expanded to eight bits, if necessary, by appending two or four leading-zero bits. The three high-order bits of the source character select a word from the table, indexing the table pointer. The remaining five bits of the expanded source character select (by their value) a bit from this word.

At the completion of the operation, a count of the number of characters not transferred is placed on the top of stack.

If all the characters specified by the length field are transferred, the true/false flip-flop is set to true; otherwise, the true/false flip-flop is set to false.

The table format is as follows:

Source Size	Table Length	Bits Used In Table Word
4	1 word	(31:16)
6	2 words	(31:32)
8	8 words	(31:32)

TRANSFER WHILE TRUE, UPDATE (TWTU) (V)DB

The Transfer While True, Update operator performs a Transfer While True, Destructive operation. At the completion of the operation, the source destination pointers and the count are updated. If all the characters specified by the length field are transferred, the true/false flip-flop is set to one (true); otherwise it is set to zero.

TRANSFER WHILE FALSE, DESTRUCTIVE (TWFD) (V)D2

The Transfer While False, Destructive operator performs the Transfer While True Destructive operation except that the relationship is true if the bit found by indexing into the table is a zero.

TRANSFER WHILE FALSE, UPDATE (TWFU) (V)DA

The Transfer While False, Update operator performs a Transfer While False, Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER UNCONDITIONAL, DESTRUCTIVE (TUND) (P)E6

The Transfer Unconditional, Destructive operator transfers from the source to the destination the number of characters specified by the top-of-stack operand. If the top-of-stack item is a descriptor, it is used as a source pointer. Since no repeat field was given, the field length is set by default at 1,048,575. The second item in the stack is the destination pointer. If all characters specified by the length field are transferred, the true/false flip-flop is set to one (true) by this operand; otherwise, the flip-flop is set to zero (false).

TRANSFER UNCONDITIONAL, UPDATE (TUNU) (P)EE

The Transfer Unconditional, Update operator performs a Transfer Unconditional, Destructive operation. At the completion of the operation, the source and destination pointers are updated.

Subroutine Operators

Subroutine operators are those operators which can move the program operation across machine architecture such as from stack to stack, or from subroutine to subroutine, and so on.

Any subroutine operator which can "chain" indirect reference words (IRW's) or stuffed indirect reference words (SIRW's) can obtain accidental procedure entry if a program control word (PCW) is pointed to by the IRW or SIRW last in the chain.

MARK STACK (MKST) (P)AE

The Mark Stack operator builds an inactive MSCW on top of the stack which is to be subsequently used by an Enter operator. The F register is set to the location of the MSCW.

The Mark Stack operator is normally used when an entry to a procedure is anticipated. The normal sequence of events to enter a procedure is (1) mark the stack; (2) insert an indirect reference to a program control word; (3) insert parameters, if any are to be passed to the procedure; and then (4) execute an Enter operator, which will in turn, cause an entry into the program segment located by the program control word.

INSERT MARK STACK (IMKS) (P)CF

The Insert Mark Stack operator inserts a mark stack control word in the current stack below the two top-of-stack items.

NAME CALL (NAMC) (P) 40 THRU (P)7F

Name Call builds an indirect reference word in the top of stack. The six low-order bits of the first syllable and the eight bits of the second syllable form a 14-bit address couple. This address couple is placed in the top-of-stack location of the CDB with the tag field set to 001.

In the B 7800, if the Name Call is followed by INDX, NXLN, NXLV, STOD, STON, OVRD, OVRN, DBUN, LOAD, or LODT operator, the IRW is not placed in the CDB. Instead the address couple is sent to the DRU for evaluation and, if appropriate, a memory request is initiated by the DRU. Except for write operators, the operator following NAMC is processed in the DRU. In the case of concatenated index operator, a second input is sent as a CDB address location to the DRU. The operand in the CDB location is pointing to a descriptor, the invalid-operand interrupt used by the DRU to index the fetched descriptor. Since the address computation is completed by the time the index operator is processed by the DRU, a considerable time saving is realized for concatenated index operations.

VALUE CALL (VALC) (P)00 THRU (P)3F

Value Call is a two-syllable instruction that brings an operand from memory into the top-of-stack. A concatenation of the two Value Call syllables gives a 14-bit address couple. If the referenced memory location contains an indirect reference word or a data descriptor, additional memory accesses are made until the "target" operand is located. The operand is then placed in the top-of-stack. The operand may be either single-precision or double-precision, causing either one or two words to be loaded into the top-of-stack.

If the word accessed is an indexed data descriptor, the word addressed by the data descriptor is brought to the top-of-stack. If the word accessed is a non-indexed word data descriptor, the descriptor is indexed using the second word in the stack as the index value, and the word addressed by the indexed data descriptor is brought to the top-of-stack. If the double-precision bit (bit 40) in the data descriptor is set, the second half of the double-precision operand is placed in the second half of the top-of-stack location.

If the presence bit in the data descriptor is zero, the presence-bit interrupt is set. After the data has been made present, the operation is restarted.

If a data descriptor does not address an operand, step index word, or a word descriptor, an invalid-operand interrupt is set and the operation is terminated.

If the word accessed by the Value Call is an indirect reference word (IRW), the word addressed by the IRW is brought to the top-of-stack.

If the word accessed is a program control word (PCW), an accidental entry into the subroutine addressed by the PCW is initiated. A mark stack control word and return control word are placed in the stack and an entry is made into the subprogram. Upon completion of the subprogram, a return operation will re-enter the Value Call operator flow.

If the target operand is a step index word (tag = 4) instead of an operand, the current-value field (bits 15:16) of the SIW will be placed in the top-of-stack with the tag set to zero.

The "chaining" of memory accesses continues until a target operand is reached; however, once a data descriptor has been encountered, an indirect reference word or PCW is not allowed, and once a stuffed indirect reference word has been encountered, a normal IRW is not allowed. Either of these conditions will cause an invalid-operand interrupt.

EVALUATE (EVAL) (P)AC

If the word in the top of stack is an IRW at the start of the evaluate operator, the IRW is evaluated and chained until a target operand or descriptor is obtained.

If the IRW chaining encounters an operand or step-index word, the last IRW is left in the top of stack as the result of the evaluate operator.

If the IRW chaining encounters a descriptor, the last IRW is saved unless the descriptor is an index word descriptor. If IRW chaining encounters an index word descriptor, the index word descriptor is brought to the top of stack.

If a non-index descriptor or string descriptor is an input to the evaluate operator, it is left in the top of stack as the result of the evaluate operator.

ENTER (ENTR) (P)AB

The Enter operator causes an entry into a procedure from a calling procedure. (The sequence of events to enter a procedure is: (1) mark the stack; (2) insert an indirect reference to a program control word; (3) insert parameter(s), if any are to be passed to the procedure; and, (4) execute an Enter operator.)

The Enter operator completes the MSCW and stores it at F, builds an RCW and stores it at F+1, and initializes processor state for procedure being entered.

EXIT (EXIT) (P)A3

The EXIT operator causes a called procedure to return to a calling procedure and is used when the called procedure is not required to return a result. The Exit operator returns all control registers to the position they were in prior to the calling procedure and cuts back the stack.

RETURN (RETN) (P)A7

The Return operator causes a called procedure to return to a calling procedure (as in EXIT) but is used when the called procedure is required to return a result. An operand or name in the top-of-stack is returned to the calling procedure. If a name is returned and the V bit (bit 19) in the MSCW is ON, the name is evaluated to yield an operand as in VALC (since the V-bit indicates that the RETN is to VALC which caused accidental entry).

Transfer Operators

The transfer operators transfer any field of bits from one word in the stack to any field of another word in the stack.

NOTE

For all transfer operators the values specified in the stack must be non-negative.

FIELD TRANSFER (FLTR) (P)98

The Field Transfer operator uses the three syllables following it to establish the pointers used in the field transfer. Stack adjustment takes place so that the two top-of-stack locations are full. The contents of the field in the top-of-stack, starting at the bit position addressed by the third syllable of FLTR, is transferred into a field of corresponding length in the second location in the stack. The field in the second location in the stack starts at the bit position indicated by the second syllable of FLTR and proceeds toward the low-order-bit positions. When the number of bits specified by the fourth syllable of FLTR has been transferred the top-of-stack word and the operation is complete.

If the second or third syllables of the operator are found to be greater than 47 or the fourth syllable is greater than 48, the invalid operand interrupt is set and the operation is terminated.

DYNAMIC FIELD TRANSFER (DFTR) (P)99

The Dynamic Field Transfer operator causes a Field Transfer operation using the top-of-stack operand to specify the field length, using the second operand in the stack to specify the starting-bit position of the field from which the transfer will be made, and using the third operand in the stack to indicate the starting bit of the field to which the transfer will be made.

As each of these operands is used to establish a pointer for the transfer, it is first integerized and checked for being greater than 47 or 48, as above, then is deleted from the stack. The fourth and fifth stack operands become the two top-of-stack operands, and the transfer takes place as in the FLTR operator.

FIELD ISOLATE (ISOL) (P)9A

The Field Isolate operator isolates a field in the top-of-stack word. The second syllable of the operator specifies the starting bit. The third syllable specifies the length of the field in bits. The isolated field is right-justified with all other information bits set to zero. The tag bits are set to zero.

DYNAMIC FIELD ISOLATE (DISO) (P)9B

The Dynamic Field Isolate operator performs a Field Isolate operation using the top-of-stack operand to specify the length of the field to be isolated and using the second operand in the stack to specify the starting bit. These operands are then deleted from the stack and the Field Isolate operation is performed on the next operand.

FIELD INSERT (INSR) (P)9C

The Field Insert operator inserts a field from the top-of-stack into the second word. Stack adjustment assures that the top two positions are occupied. The right-justified field in the top-of-stack is inserted into the second word starting at the position specified by the second syllable of the Field Insert operator. The third syllable specifies the length of the field to be inserted. The top-of-stack word is deleted after the field is inserted in the second word.

DYNAMIC FIELD INSERT (DINS) (P)9D

The Dynamic Field Insert operator performs a Field Insert operation, transferring a field from the top operand in the stack into the fourth operand in the stack. The second operand in the stack specifies the length of the field to be inserted, and the third operand in the stack specifies the starting bit of the field.

Type-Transfer Operators

Type-transfer operators are used to manipulate operand relative to single-precision or double-precision operands.

SET TO SINGLE-PRECISION, TRUNCATED (SNGT) (P)CC

The Set to Single-Precision, Truncated operator sets the top-of-stack operand to a single-precision operand without rounding. It functions as an arithmetic operator with operand inputs. The result of this operator is expressed in normalized floating-point form.

If descriptor is received as an input, the SNGT operator resets the double-precision bit (bit 40) in the descriptor; thereby making the data referenced by the descriptor single-precision.

If descriptor is not indexed, the SNGT operator doubles the length field (bit 39:20) of the descriptor (length field represents the number of items in the array).

If descriptor is already indexed, the index field (bits 39:12) of the descriptor is not modified.

SET TO SINGLE-PRECISION ROUNDED (SNGL) (P)CD

The Set to Single-Precision, Rounded operator sets the top-of-stack operand to a single-precision operand with rounding. It functions as an arithmetic operator with operand inputs. The result of this operator is expressed in normalized floating-point form.

If at the start of this operator, a descriptor is received as an input; the invalid-operand interrupt pointing to a descriptor, the invalid-operand interrupt is set and the operation is terminated.

SET TO DOUBLE-PRECISION (XTND) (P)CE

The Set Double-Precision operator sets the top-of-stack operand to a double-precision operand.

If single-precision operand is received as an input, the operand is extended with a word of 0's (LS word will contain all 0's) and the tags are set to 2.

If descriptor is received as an input, the XTND operator sets the double-precision bit (bit 40) in the descriptor; thereby, making the data referenced by the descriptor double-precision.

If descriptor is not indexed, the XTND operator divides the length field (bit 32:20) of the descriptor by 2 (length field represents the number of items in the array).

SET DOUBLE TO TWO SINGLES (SPLT) (V)43

The Set Double to Two Singles operator splits a double-precision operand into two single-precision results. (A becomes LS part and B becomes MS part of the result.) The SPLT operator can receive two single-precision operands as inputs. In this case, the LS word is pointing to a descriptor, the invalid-operand interrupt considered to be all 0's.

SET TWO SINGLES TO A DOUBLE (JOIN) (V) 42

The Set Two Singles to a Double operator joins two single-precision operands to form one double-precision result. (A becomes LS part and B becomes MS part of the result.) The JOIN operator can receive a double-precision operand as an input. In this case, the LS word is ignored and only the MS part of the result is used.

Miscellaneous Primary Mode Operators

Miscellaneous primary mode operators are those operators which cannot be readily described or grouped with other operators.

ESCAPE TO 16-BIT INSTRUCTION (VARI) (P) 95

The Escape to 16-Bit Instruction operator provides transition from the primary mode operators to the variant mode operators. The first syllable (VARI) indicates that the actual operator is in the second syllable. (Interrupts are not allowed between the VARI syllable and the following syllable.)

READ AND CLEAR OVERFLOW FLIP-FLOP (ROFF) (P)D7

The Read and Clear Overflow Flip-Flop operator places a single-precision operand in the top-of-stack with the least significant pointing to a descriptor, the invalid-operand interrupt bit set equal to the overflow flip-flop. The overflow flip-flop is reset.

READ TRUE FALSE FLIP-FLOP (TRFF) (P)DE

The Read True False Flip-Flop operator places a single-precision operand in the top of stack with the least significant bit set equal to the true/false flip-flop.

SET EXTERNAL SIGN (SXSN) (P)D6

The Set External Sign operator places the operand sign bit of the top-of-stack word into the external sign flip-flop.

STUFF ENVIRONMENT (STFF)(P)AF

The Stuff Environment operator places the current stack number and displacement into the stack number field and displacement field of the top-of-stack IRW. The index field of the SIRW is obtained directly from the IRW. Bit 46 is set to indicate that it is now a stuffed indirect reference word.

The stack number identifies the stack in which the LL field of the IRW points to a display. This display may not be in the stack being executed by the processor. Whatever stack is pointed to by the display is, the stack number that is placed in the SIRW.

If normal IRW's LL field is less than current LL, pointing to a descriptor, the invalid-operand interrupt the IRW's LL + 1 is used to reference a display. This display is then used to fetch the MSCW. The display pointed to by the original IRW points to the stack number in the MSCW. Therefore, the stack number and displacement fields from the MSCW are used in building the SIRW.

If LL of the normal IRW equals the current LL, the current stack number is placed into the SIRW and the display to which the normal IRWs LL points minus the current bottom of stack is calculated to obtained displacement field for the SIRW.

Universal Operators

The operators HALT, NVLD, and NOOP are universal except that they cannot follow operators EXSU, EXSD, EXPU, and EXPD; in these cases a Loop Timeout will occur.

CONDITIONAL HALT (HALT) (U) DF

The Conditional Halt operator halts the processor if the conditional halt switch is in the ON position; if the conditional halt switch is OFF, the operator is treated as a NOOP.

INVALID OPERATOR (NVLD) (U)FF

The Invalid Operator sets the invalid-operator interrupt, pointing to a descriptor, the invalid-operand interrupt

NO OPERATION (NOOP) (U)FE

No operation occurs when the No Operation operator is encountered except that the PSR and PIR are advanced to point at the next operator.

VARIANT MODE OPERATORS

Variant mode operators is the name used to describe those primary mode operators which are less frequently used. There is no functional significance to the category "variant mode." Variant mode operation extends the number of operation codes. Variant mode operators require two syllables: the first syllable is the Escape to 16 Bit Instruction (VARI) operator. The syllable following VARI is the actual operation and the syllable pointer is positioned beyond the two syllables.

Any unusual variant mode codes are detected and cause a programmed operator interrupt.

Variant mode operations are both word-and string-oriented operators.

String Operators

TRANSLATE (TRNS) (V)D7

The Translate operator transfers from the source to the destination the number of characters specified by the second item in the stack while performing the following translation.

The translation uses a table containing the translated characters. The word in the top-of-stack is a descriptor that addresses the translation table. The second operand in the stack specifies the length of the string. The third word in the stack is a descriptor addressing the source string (or an operand which is

the source string). The fourth word in the stack is a descriptor addressing the destination string. Source and destination are updated at the end of the operation.

Translation occurs as follows: Each source character is used as index into the table to locate a character. An all zeroes character locates the most significant character in the table. The located character is transferred to the destination string.

The least significant 32 bits of each table word provide four 8-bit characters. Table sizes are as follows:

1. 4-bit source digits use a 4-word table.
2. 6-bit source characters use a 16-word table.
3. 8-bit source bytes use a 64-word table.

Scan Operators

SCAN IN (SCNI) (V)4A

The Scan-In operator uses the 20 low-order bits of the top-of-stack word as the address of the Time of Day (TOD) register and reads the TOD contents to the top-of-stack. Other variants, which are valid in the B 6000 Systems, produce an invalid operand interrupt in the B 7800. The B 7800 MCP causes the correct B 7800 code to be executed to handle the interrupt.

Scan While Operators

SCAN WHILE GREATER, DESTRUCTIVE (SGTD) (V)F2

The Scan While Greater, Destructive operator scans the number of characters specified by the second operand in the stack or while the source character is greater than a delimiter. The top-of-stack operand is the delimiter. The third item in the stack is the source pointer. If the second item in the stack is a descriptor, it is used as a source pointer and the length of the character string is set to 1,048,575. All comparisons are binary. When the source is an operand, it must be a single-precision operand.

At the completion of this operator if all the characters have been scanned, the true/false flip-flop is set to one. (SGTD operator leaves no result on the stack.) If the scan was stopped by the delimiter test before the end of the string the true/false flip-flop is set to zero.

SCAN WHILE GREATER, UPDATE (SGTU) (V)FA

The Scan While Greater, Update operator performs a Scan While Greater, Destructive operation. At the completion of the operation, the source pointer and count are updated. The SGTU operator leaves the updated length on top of the stack and the updated source second from top of the stack. If all the characters specified by the length field are scanned, the true/false flip-flop is set to true; otherwise, the true/false flip-flop is set to false. The source pointer locates the character which stopped the scan.

SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE (SGED) (V)FL

The Scan While Greater or Equal, Destructive operator performs a Scan While Greater, Destructive operation while the source character is greater than or equal to the delimiter. This operator leaves no result on the stack.

SCAN WHILE GREATER OR EQUAL, UPDATE (SGEU) (V)F9

The Scan While Greater or Equal, Update operator performs a Scan While Greater Than or Equal, Destructive operation. At the completion of the operation, the source pointer and count are updated. This operator leaves the updated length on top of the stack and the updated source second from top of the stack.

SCAN WHILE EQUAL, DESTRUCTIVE (SEQD) (V)F4

The Scan While Equal, Destructive operator performs a Scan While Greater, Destructive operation while the source character is equal to the delimiter. This operator leaves no result on the stack.

SCAN WHILE EQUAL, UPDATE (SEQU) (V)FC

The Scan While Equal, Update operator performs a Scan While Equal, Destructive operation. At the completion of the operation, the source pointer and count are updated. This operator leaves the updated length on top of the stack and the updated source second from top of the stack.

SCAN WHILE LESS OR EQUAL, DESTRUCTIVE (SLED) (V)F3

The Scan While Less or Equal, Destructive operator performs a Scan While Greater, Destructive operation while the source character is less than or equal to the delimiter. This operator leaves no result on the stack.

SCAN WHILE LESS OR EQUAL, UPDATE (SLEU) (V)FB

The Scan While Less or Equal, Update operator performs a Scan While Less or Equal, Destructive operation. At the completion of the operation, the

source pointer and count are updated. This operator leaves the updated length on top of the stack and the updated source second from top of the stack.

SCAN WHILE LESS, DESTRUCTIVE (SLSD) (V)F0

The Scan While Less, Destructive operation performs a Scan While Greater, Destructive operation while the source character is less than the delimiter. This operator leaves no result on the stack.

SCAN WHILE LESS, UPDATE (SLSU) (V)F8

The Scan While Less, Update operator performs a Scan While Less, Destructive operation. At the completion of the operation, the source pointer and count are updated. This operator leaves the updated length on the top of the stack and the updated source second from top of the stack.

SCAN WHILE NOT EQUAL, DESTRUCTIVE (SNED) (V)F5

The Scan While Not Equal, Destructive operator performs a Scan While Greater, Destructive operation while the source character is not equal to the delimiter. This operator leaves no result on the stack.

SCAN WHILE NOT EQUAL, UPDATE (SNEU) (V)FD

The Scan While Not Equal, Update operator performs a Scan While not Equal, Destructive operation. At the completion, the source pointer and count are updated. This operator leaves the updated length on the top of the stack and the updated source second from top of the stack.

SCAN WHILE TRUE, DESTRUCTIVE (SWTD) (V)D5

The Scan While True, Destructive operator uses each source character as an index into a table to locate a bit in the table. In order to index the table the source character is expanded to eight bits (if necessary) by appending two or four leading-zero bits. The three high-order bits of these eight select a word from the table, indexing the table pointer. The remaining five bits of the expanded source character select a bit from this word by their value. If the bit located is a one, the relationship is true and the scan continues. An all zero's character indexes to the most significant bit of the table.

The top-of-stack word is a table pointer. The second item in the stack specifies the number of characters to be scanned or, if it is a descriptor, it is used as a source pointer and the length of the character string is set at 1,048,575. The third item in the stack is the source pointer. If all the characters specified by the length field are scanned, the true/false flip-flop is set to true; otherwise, the true/false flip-flop is set to false. This operator leaves no results on the stack. The table format is as follows:

Source Size	Table Length	Bits/Word
4	1 word	(31:16)
6	2 words	(31:32)
8	8 words	(31:32)

SCAN WHILE TRUE, UPDATE (SWTU) (V)DD

The Scan While true, Update operator performs a Scan While True, Destructive operation. At the completion of the operation, the source pointer and count are updated. This operator leaves the updated length on top of the stack and the updated source second from top of the stack.

SCAN WHILE FALSE, DESTRUCTIVE (SWFD) (V)D4

The Scan While False, Destructive operator performs a Scan While True, Destructive operation except that the relationship is true if the bit found by indexing into the table is a zero. This operator leaves no results on the stack.

SCAN WHILE FALSE, UPDATE (SWFU) (V)DC

The Scan While False, Update operator performs a Scan While False, Destructive operation. At the completion of the operation, the source pointer and count are updated. This operator leaves the updated length on top of the stack and the updated source second from top of the stack.

Tab Field Operators

SET TAG FIELD (STAG) (V)B4

The Set Tag Field operator sets the tag field (bits 50:3) of the second word in the stack to the contents of bits 2:3 of the top-of-stack word.

READ TAG FIELD (RTAG) (V)B5

The Read Tag Field operator replaces the top-of-stack word with a single-precision operand with bits 2:3 equal to the tag field of the original top-of-stack word.

Set State Operators

SET INTERVAL TIMER (SINT) (V)45 (CONTROL STATE OPERATOR)

The Set Interval Timer operator integerizes the top-of-stack operand. If the operand cannot be integerized, an integer-overflow interrupt is set and the operation is terminated. The value of the 11 low-order bits of the top-of-stack operand is used to set the interval timer associated with the processor which is executing this operator. Once set, the interval timer will start to decrement once each 512 microseconds. The associated processor is interrupted when the value has been counted to zero if the timer is still armed.

The interval timer is disarmed whenever the associated processor is interrupted by an external interrupt.

READ PROCESSOR IDENTIFICATION (WHOI) (V)4E

The Read Processor Identification operator places a single-precision operand with a value equal to the processor's number on the top-of-stack.

ENABLE EXTERNAL INTERRUPTS (EEXI) (V)46

The Enable External Interrupts operator prohibits this processor to respond to external interrupts.

DISABLE EXTERNAL INTERRUPTS (DEXI) (V)47

The Disable External Interrupts operator prohibits this processor from responding to external interrupts.

IDLE UNTIL INTERRUPT (IDLE) (V)44

The Idle Until Interrupt operator suspends program execution by this processor. External interrupts are allowed, and the processor will enter its interrupt-handling routine upon receipt of an interrupt.

READ PROCESSOR REGISTER (RPRR) (V)B8

The Read Processor Register operator reads into the top-of-stack the contents of one of the eight base registers, or one of the eight index registers, or one of the 32 D registers. Register address assignments are shown in table 3-4-2.

An invalid-operator interrupt is set and the operation is terminated if the top-of-stack word is not a descriptor or an indirect reference word at the start of the Evaluate operator.

SET PROCESSOR REGISTER (SPRR) (V)B9

The Set Processor Register operator sets the processor register addressed by the second word in the stack to the value contained in the top-of-stack word.

Unpack Operators

UNPACK ABSOLUTE, DESTRUCTIVE (UABD) (V)D1

The Unpack Absolute, Destructive operator unpacks a string of left-justified digits from the second operand in the stack. the top-of-stack operand defines the string length (in 4-bit digits) of the second operand in the stack. The specified number of digits are transferred from the second operand to the destination. The third item in the stack is a string descriptor destination pointer. Zone fill in the destination is as follows:

1. If the destination bit format is 8-bit ASCII, the digits are transferred to the destination string with the leading-zone bits set to 0011.
2. If the destination bit format is 6-bit BCL, the digits are transferred to the destination with the two leading-zone bits set to zero.
3. If the destination bit format is 8-bit EBCDIC, the digits are transferred to the destination string with the four leading-zone bits set to all ones (F).

UNPACK ABSOLUTE, UPDATE (UABU) (V)D9

The Unpack Absolute, Update operator performs an Unpack Absolute, Destructive operation. At the completion of the operation the destination pointer is updated.

UNPACK SIGNED, DESTRUCTIVE (USND) (V)D0

The Unpack Signed, Destructive operator performs an Unpack Absolute, Destruction operation except that the external sign is considered.

If the external sign flip-flop is ON (indicating negative data) then a zone of 10 is inserted in the last 6-bit character or a zone of 1101 is inserted in the last 8-bit byte. For 8-bit ASCII formatted data the negative sign is indicated in the least-significant byte by a zone of 1111. If the data format of the destination is 4 bits, the first digit position of the destination string is set to 1101 if the external sign flip-flop is ON; if the external sign flip-flop is OFF the first digit of the destination string is set to 1100.

Table 3-4-2. Register Address Assignments

Add. (dec.)	Add. (hex)	Int. Addr. hex	Reg. Name	Register Usage
0-31	0-1F		D[x]*	Display Registers
32	20		PIC	Program Index
33	21	24	SIR*	Source Index
34	22	25	DIR*	Destination Index
35	23	26	TIR*	Table Index
36	24		XLOS	Limit of Stack
37	25	20	BOSR*	Base of Stack
38	26		XF	Most Recent MSCW Address
39	27	22	S1LS*	Scratch (Spare Local Storage)
40	28		ID (CID**)	Interrupt Identifier
41	29		SCAN	MDP Control Register
42	2A		IFM (CMR**)	Fault Mask Register
43	2B			Spare
44	2C		IFR (CFR**)	Interrupt Fault Register
45	2D			Spare
46	2E		INT	Interval Timer
47	2F		ITD	Time of Day
48	30		PBR	Program Base Register
49	31	28	SBR*	Source Base Register
50	32	29	DBR*	Destination Base Register
51	33	2A	TBR*	Table Base Register
52	34		XS	Top of Stack Address
53	35		XSN	Current Stack Vector Index
54	36		PSDI	Current Segment Descriptor Index
55	37	2B	S2LS*	Scratch (Spare Local Storage)
56	38	2C	ADZ*	Alternate [D0] Register
57	39	2D	APIR*	Alternate Program Index
58	3A		ALL1	(Return zero on Read)
59	3B	21	LD1*	Last D [1] used as SD1 base.
60	3C		IPF	Processor Fail Register
61	3D		XCM	Processor Mode Register
62	3E		PGAM	Purge Store Q and associative Memories
63	3F		PGKA	Purge Store Q and associative Memories

* Soft storage in DRU of CPM (no panel indicators)

** Mnemonic used in MCP.

*** RPRR 3E resets egg timer.

UNPACK SIGNED, UPDATE (USNU) (V)D8

The Unpack Signed, Update operator performs an Unpack Signed, Destructive operation. At the completion of the operation, the destination pointer is updated.

Searching Operators

LINKED LIST LOOKUP (LLLU) (V)BD

The Linked List Lookup operator searches a linked list of words. Each word consists of a value in bits 47:28 and a link in bits 19:20. The link is an index from the base of the array to the next element in the list.

This operator expects the third stack entry (bits 27:28) to contain an argument, the second stack entry to contain a non-indexed data descriptor, and the top-of-stack to contain an index value pointing into a linked-list of words. The argument is not required to be an integer, but only the right-most 28 bits are significant after the argument has been integerized as required. The base address, size field, and argument are saved throughout the operator.

Bits 47:28 of the word addressed by the base plus the index value are compared to the argument value. If this field is less than the argument value, the process is repeated using the link as the new index. If the field is greater than or equal to the argument value, the operation is complete. At completion, the top-of-stack contains the index of the word that contains the link that points to the satisfying argument.

If the value of the link portion of the linked-list word is equal to zero, the top-of-stack is set to minus one (–1) and the operation is completed.

If the index value in the linked-list word is greater than the length value from the descriptor, an invalid-index interrupt is set and the operation is terminated.

When the first word in the stack at the start of this operator is not an operand an invalid-operand interrupt is set and the operation is terminated.

If the data descriptor has been indexed, the invalid-operand interrupt is set and the operation is terminated.

If the value in the first word is greater than or equal to the argument value, the index of the first word itself is left on top of the stack.

MASKED SEARCH FOR EQUAL (SRCH) (V)BE

The Masked Search for Equal operator searches a data word list for a word identical to the third word in the stack. At the beginning of this operator, the top word in the stack contains a data descriptor, the second word in the stack contains a 51-bit mask, and the third word in the stack contains a 51-bit argument value. If the descriptor is not present, the presence-bit interrupt is set and the operator is exited. Otherwise, if the descriptor is unindexed, the indexed bit (bit 45) is turned ON and the index field value is set to length –1.

The descriptor points to a word which is then fetched into the processor. This word is ANDed with the mask and a test is made to determine whether the result is identical to the argument.

When an equal compare is made, the index of the equal word is left on top of the stack.

When a not-equal compare is made, the index value is decreased by one and the operation is repeated (except when the index value is zero). When the index value is zero, a –1 is left on top of the stack and the operator is exited.

Subroutine Operator

MOVE TO STACK (MVST) (V)AF

The move to Stack operator causes the processor's environment (or addressing space) to terminate and to be moved from the current stack to the program stack specified by the operand in the top of stack.

The operator builds a top-of-stack control word and places it at the base of the current stack, thereby inactivating the stack.

The top of stack item is integerized and checked for invalid index against the stack vector descriptor at D[0]+2.

The stack descriptor for the requested stack is then fetched and made present. The address field is placed into the base-of-stack register. LOSR is loaded with the address field plus length –1. The top-of-stack control word is then fetched and the stack is marked "active" by storing the processor ID at the base of the stack. The TSCW is distributed and the D registers are updated.

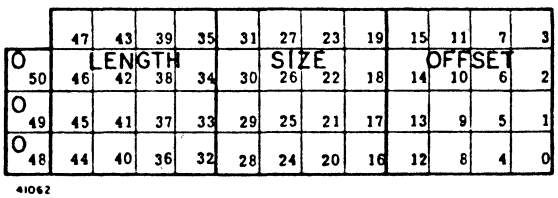
If during the integerization the top of stack item is too large, the integer-overflow interrupt is set and the operation is terminated.

If the index value is less than zero or greater than the length field of the data descriptor for the stack vector array, an invalid index interrupt is set and the operation is terminated.

Special Interpretation Operator

OCCURS INDEX (OCRX) (V)85

The Occurs Index operator is used to index a field in an array. This operator requires an Occurs Index Word (OIW) in the top-of-stack and an index value (operand) in the second stack position. The format of the IOW follows:



The operator creates a new index value from the OIW and the operand in the following manner:

The operand is integerized. If the resulting index is greater than the maximum integer value

(549,755,813,887), the integer overflow interrupt is set and the operation is terminated. If the index has a value of zero or if the index is less than zero or greater than the SIZE field of the OIW, the invalid index interrupt is set and the operation is terminated.

The LENGTH field of the OIW is multiplied by the index value 15:16 minus 1, and that value is added to the OFFSET field of the OIW, resulting in the new index value. The two original top-of-stack items are deleted and the new index value is left in the top-of-stack.

In the OIW the "length" field gives the number of characters in a field; the "size" field gives the number of fields in the array; the "offset" field indicates the beginning of the first character position in the first field of the first word.

Operators Exclusive to the B 7800

SET MEMORY INHIBITS (SINH) (V)A8 (CONTROL STATE OP)

The Set Memory Inhibits operator transfers the inhibit settings in the second stack register to the memory module specified in the top stack register. The two top-of-stack items are deleted. (All tags are legal.) The memory module number is given in the top-of-stack (bits 3:4). The inhibit field setting is given in the second item in the stack (bits 7:8).

SET MEMORY LIMITS (SLMT) (V)AA (CONTROL STATE OP)

The Set Memory Limits operator transfers the limits and availability settings in the second stack register to the memory module specified in the top-of-stack register. The two top-of-stack items are deleted. (All tags are legal.) The limits specify the range of addresses (in 16K increments) behind the module and the availability setting specifies which stack(s) (of a possible four) are to be used. (All tags are legal.) The top-of-stack gives the memory module number (bits 3:4). The second item in the stack gives module availability (bits 3:4) and memory addressing limits: upper limit (bits 15:6) and lower limit (bits 9:6).

FETCH MEMORY FAIL REGISTER (FMFR) (V)AC (CONTROL STATE OP)

The Fetch Memory Fail Register operator fetches the contents of the fail register from the memory module specified in the top-of-stack (bits 3:4). The contents of the fail register are placed in the top-of-stack.

IGNORE PARITY (IGPR) (V)48 (CONTROL STATE OP)

The Ignore Parity operator is used for confidence checking and requires the processor to be in the control state. In control mode 0, words entering the CPM are checked for correct parity but the IGPR operator sets the IGP flip-flop which inhibits transmission of parity error messages for those words with incorrect parity. Likewise, IGPR inhibits correct parity generation before storage for those words detected in the CPM with incorrect parity.

Parity error interrupts and new parity generation will be inhibited with the CPM in control mode 0 by IGPR until any one of the following occurs:

1. Some other interrupt causes the CPM to move to control mode 1.
2. Another IGPR is decoded while the CPM is in a control mode greater than zero.
3. Or the CPM returns to normal state.

Any one of the previous conditions cause the MIGP flip-flop to be reset and the CPM to resume parity error interrupts and generation of new parity.

PAUSE UNTIL INTERRUPT (PAUS) (V)84

The Pause Until Interrupt operator suspends program execution until an external interrupt or an interval timer interrupt occurs. If the processor is operating in control state, the operation continues in sequence; to clear the interrupt the INT. I.D. must be read. If the processor is operating in normal state, the interrupt is handled as in IDLE.

INTERRUPT CHANNEL N (INCN) (V)8F

The Interrupt Channel N operator sends signals to the channel or channels specified by the top-of-stack. The top-of-stack item is deleted. Bit 0 interrupts channel 0; bit 1 interrupts channel 1, and so on.

STOP (STOP) (V)BF

The STOP operator causes an unconditional halt of the central processor. The STOP operator is primarily used for diagnostic purposes. The processor may be restarted by pressing and releasing the START button on the processor control panel.

Edit Mode Operators

Edit Mode operators perform editing functions on strings of data. Edit functions are normally involved in preparing information for output. These operators include Insert, Move, and Skip, in the form of micro-operators in either the program string or in a

separate table. In the program string, they are single micro-operators and are entered by use of the Execute Single Micro or Single Pointer operators. (See the "Enter Edit Mode Operator" descriptions.) If the micro-operators are in a table, the table becomes the program string that is to be executed. This table is entered by means of the Table Enter Edit operators, and is exited through the End Edit micro-operator.

If the source or destination data has the memory protect bit (bit 48) equal to one, the segmented-array interrupt is set and the current micro-operator is terminated.

Insert Operators

INSERT UNCONDITIONAL (INSU) (E)DC

The Insert Unconditional micro-operator places an insert character into the destination string for the number of times specified by the repeat value. When this operator is entered by a Table Enter Edit operator, the repeat is in the syllable following the micro-operator syllable, and the insert character is in the next syllable (the third syllable).

When this operator is entered through an Execute Single Micro Instruction operator, the repeat field is in the top-of-stack operand and the insert character is the second syllable. The operator length is then two syllables.

INSERT CONDITIONAL (INSC) (E)DD

The Insert Conditional operator inserts the character defined by the third syllable into the destination string if the float toggle is OFF. If the float toggle is ON, the character defined by the fourth syllable is inserted into the destination string. The insertion is repeated the number of times specified by the second syllable when this operator is entered by the Table Enter Edit operation.

When this operator is entered through an Execute Single Micro instruction operator, the repeat field is the top-of-stack operand. The operator length is then three syllables.

INSERT DISPLAY SIGN (INSG) (E)D9

The Insert Display Sign operator inserts the character defined by the second syllable into the destination string if the external sign flip-flop is set; otherwise, the character defined by the third syllable is inserted.

INSERT OVERPUNCH (INOP) (E)D8

The Insert Overpunch micro-operator places a sign overpunch in the destination string character. If the external sign flip-flop is reset, the operator skips one destination string character. If the external sign flip-flop is set, the zone bits of the destination character are set to 10 for 6-bit data and to 1101 for 8-bit EBCDIC data; the destination pointer is then advanced one character. The zone bits for 8-bit ASCII data are set to 1111.

Move Operators

MOVE CHARACTERS (MCHR) (E)D7

The Move Characters operator transfers the number of characters specified by the second syllable from the source string to the destination string, if this operator is entered by a Table Enter Edit operator.

When this operator is entered through an Execute Single Micro Destructive instruction, the number of characters transferred is specified by the top-of-stack operand. The operator length is then one syllable.

MOVE NUMERIC (MVNU) (E)D6

The Move Numeric operator transfers from the source string to the destination string the number of characters specified by the second syllable, if entered by a Table Enter Edit operator. The zones are not transferred but are set to 00 for 6-bit data, to 1111 for 8-bit EBCDIC data, and to 0011 or 8-bit ASCII data.

When this operator is entered through an Execute Single Micro instruction, the number of characters transferred is specified by the top-of-stack operand. The operator length is then one syllable.

MOVE WITH INSERT (MINS) (E)D0

The Move With Insert micro-operator performs leading zero suppression from the source to the destination.

If the float flip-flop is set, a Move Numeric operation is performed. If the float flip-flop is reset and the source character numeric is zero, the character defined by the third syllable is transferred to the destination string. If the float flip-flop is reset and the source character numeric is not zero, then the float flip-flop is set and a Move Numeric is performed.

The number of characters transferred from the source string to the destination string is defined by

the repeat value. In Table Edit mode the second syllable is the repeat value and the third syllable is the character to be inserted under control of the float flip-flop. In Execute Single Micro mode the repeat field value is in the word in the top-of-stack and the insert character is in the syllable following the micro-operator syllable.

MOVE WITH FLOAT (MFLT) (E)D1

If the float flip-flop is reset and the source character numeric is zero, then the character defined by the third syllable is transferred to the destination string.

If the float flip-flop is reset and the source character numeric is not zero, then the float flip-flop is set. If the external sign flip-flop is set, the character defined by the fourth syllable (the second insert character) is transferred to the destination string; otherwise, the character defined by the fifth syllable (the third character) is transferred. Then a Move Numeric operator is performed.

In Table Edit mode, the previous operation is repeated for the number of characters specified by the second syllable; the third, fourth, and fifth syllables are the insert characters.

When this operand is entered through an Execute Single Micro instruction, the repeat field is the top-of-stack operand. The operand length is then four syllables, three of which contain insert characters.

Skip Operators

SKIP FORWARD SOURCE CHARACTERS (SFSC) (E)D2

The Skip Forward Source Characters operator causes a skip forward for the number of source characters. This is done by incrementing the source pointer. The skip amount is specified by the syllable following the micro-operator's syllable, if the entry to this operator is by the execution of the Table Enter Edit Operator. When this operator is entered through an Execute Single Micro, Destructive instruction, the number of characters skipped is specified by the top-of-stack operand. The operator length is then one syllable.

SKIP REVERSE SOURCE CHARACTERS (SRSC) (E)D3

The Skip Reverse Source Characters operator decrements the source pointer by the number of source characters specified by the second syllable, or top-of-stack operand if single micro.

SKIP FORWARD DESTINATION CHARACTERS (SFDC) (E)DA

The Skip Forward Destination Characters operator causes a skip forward for the number of destination characters specified by the second syllable, or top-of-stack operand if single micro.

SKIP REVERSE DESTINATION CHARACTERS (SRDC) (E)DB

The Skip Reverse Destination Characters operator causes a skip in reverse for the number of destination characters specified by the second syllable, or top-of-stack operand is single micro.

RESET FLOAT (RSTF) (E)D4

The Reset Float micro-operator sets the float flip-flop to zero.

END FLOAT (ENDF) (E)D5

The End Float operator transfers to the destination string the character defined by the second syllable if the float flip-flop is reset and the external sign flip-flop is set.

If the float flip-flop is reset and the external sign flip-flop is reset, the character defined by the third syllable is transferred to the destination string.

If the float flip-flop is set, the End Float operator is treated as a NO-OP and the float flip-flop is reset.

END EDIT (ENDE) (E)DE

The End Edit operator terminates the execution of this string of edit micro-operators in Table Enter Edit mode. The micro program string must end with the End Edit operator.

CHAPTER 4
INPUT/OUTPUT SUBSYSTEM
SECTION 1
GENERAL DESCRIPTION OF INPUT/OUTPUT MODULE

INTRODUCTION

The B 7800 Input/Output Module is designed to serve as a buffer and control unit for all B 7800-system input and output data transfers. The IOM services requestors from a queue of requests constructed by the Central Processing Module (CPM) and stored in the Memory Storage Unit (MSU).

The IOM is informed, via an interrupt from the CPM, of the presence of a service request in the MSU. Once informed, the IOM controls the desired input/output operation in its entirety; thus, the CPM time required to initiate an I/O operation is only that needed to construct a request, queue it in the MSU, and interrupt the IOM.

BASIC IOM CONFIGURATION

As illustrated in figure 4-1-1, the IOM consists of seven major subsections. Each subsection is totally independent of the other subsections, and operates asynchronously with them.

Control Word Flow

All control word flow (between main memory and up to 255 system peripherals) is by means of: 1) an IOM subsection, the Memory Interface Unit (MIU); 2) an IOM control subsection, the Translator (XLATOR); and 3) one of four IOM subsections, each of which is uniquely buffered to match the class of data transfer assigned to it. The XLATOR subsection routes control of a given job request to one of these subsections, depending on data class (batch, high speed, data communications, or real-time interactive).

Data Flow

All data flow between main memory and the peripherals is by means of the appropriate data-transfer subsection and/or the MIU and DSB; the XLATOR is not involved and is free for control of additional job requests. When a data transfer is complete, however, the XLATOR is given control over job termi-

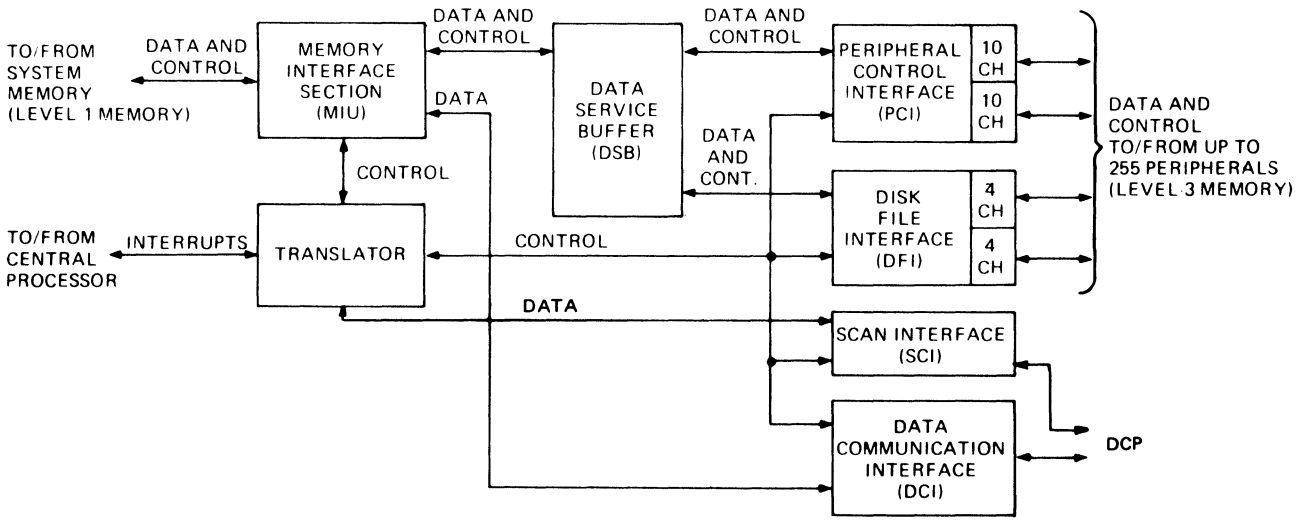


Figure 4-1-1. IOM Basic Block Diagram

nation, and control flow to main memory is accomplished by the appropriate data-transfer subsection, the XLATOR, and the MIU.

FUNCTIONAL SYSTEM INTERFACE

The functional interface between the IOM and the system is divided into the mainframe interface and the peripheral interface. These two interface areas are described in the following subparagraphs.

Mainframe Interface Configuration

As shown in figure 4-1-1, control words and data words are transferred between the IOM and the system memory (MCM/MSU). These interface signals are transferred via the MIU subsection of the IOM. Interrupt signals are transferred between the IOM translator subsection and the CPM.

IOM/MCM Interface

As illustrated in figure 4-1-2, the MIU contains eight interface areas. Each interface area is dedicated to a distinct Memory Control Module (MCM), and is connected to it by a unique memory bus. The bussed IOM/MCM interface is referred to as a memory/user pair.

A similar capability exists within the CPM which also contains eight MCM interface areas. Each CPM interface area is dedicated to a distinct MCM and is connected to it by a unique memory bus. The bussed CPM/MCM interface is also referred to as a memory/user pair.

The interface capability of an MCM is eight memory busses, each of which is connected to one and only one IOM or CPM. Therefore, the maximum combined number of CPM's and IOM's which may be bussed to an MCM is limited to eight.

The maximum number of MCMs which may be contained in a B 7800 system is four.

The typical memory-bus-configuration (figure 4-1-2) indicates the use of two IOM's, two CPM's, and two MCM's. The maximum number of MSUs with which an MCM can communicate (two) is also illustrated. Each of these MCM's can access 262K words of memory (two MSU's of 131K words each). Each IOM or CPM, when connected, can access 524K words of memory.

IOM/CPM Interface

The interface between the IOM's and CPM's of a B 7800 system consists of an interrupt bus only. The CPM informs the XLATOR section (of an IOM) about job requests via the bus, and the XLATOR informs the CPM of non-channel-related IOM errors via the bus. In addition, the XLATOR uses the bus to inform the CPM of I/O job completions (when requested by software, a SPO, or a DCP) and status changes. The interrupt bus is common to all IOM's and CPM's in a system.

IOM/Peripheral Interface Configuration

Figure 4-1-3 illustrates typical peripheral devices which may be assigned to each data-transfer class; also illustrated are the data-transfer subsection names which are henceforth referred to. The following is a brief description of the interface capability of each subsection and its physical relationship to typical peripheral equipment. The descriptions presented in figure 4-1-4 illustrate the interface capability provided when two maximum-configuration input/output modules and appropriate exchanges are used. A maximum of 28 peripheral controllers (excluding DCP's) may be connected to a single IOM.

Peripheral Control Interface (PCI)

The PCI of a single IOM consists of either one or two interface sections, depending on user requirements. Each section has 10-channel interface capability, for a total maximum capacity of 20 channels per IOM.

The controls serviced by a PCI are housed in one or two peripheral control cabinets (PCC's). Each PCI/PCC cabinet services one IOM and has a 10-channel interface capability. Up to five of these channels can be assigned to large controllers, but the remaining channels must be assigned to small controllers.

Any combination of small controls may be housed in the PCI/PCC cabinet. The large controls (SLC and MTC) may be connected to the peripheral units directly, or, in the case of the MTC only, via exchanges. Any unused channels in the PCC cabinet are left empty.

The PCI enables the IOM to interface with one to 20 peripheral controls and coordinates data transfers between the peripheral controls and the DSB as di-

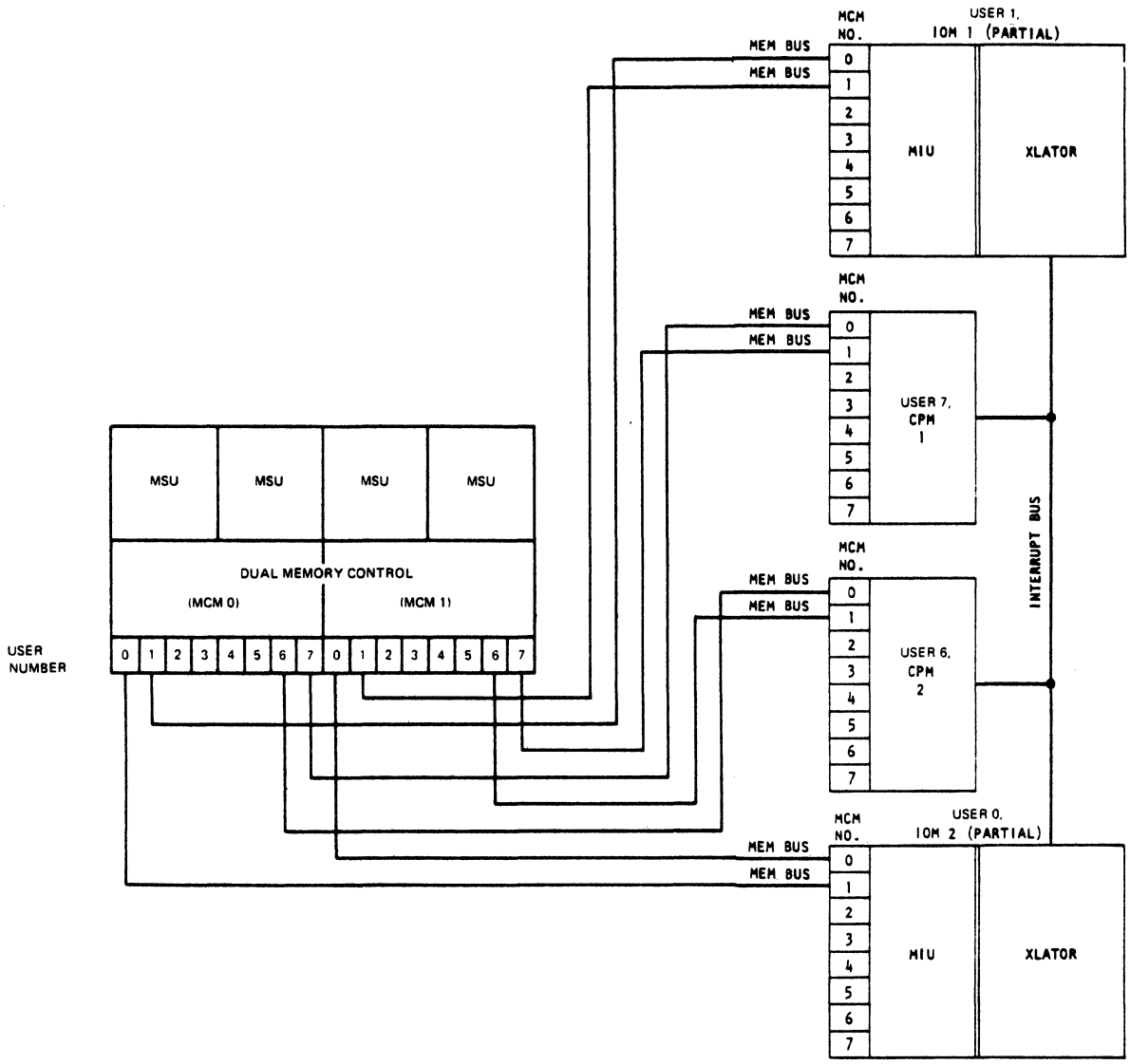


Figure 4-1-2. Typical IOM/Main Memory and IOM/CPM Interface Configurations

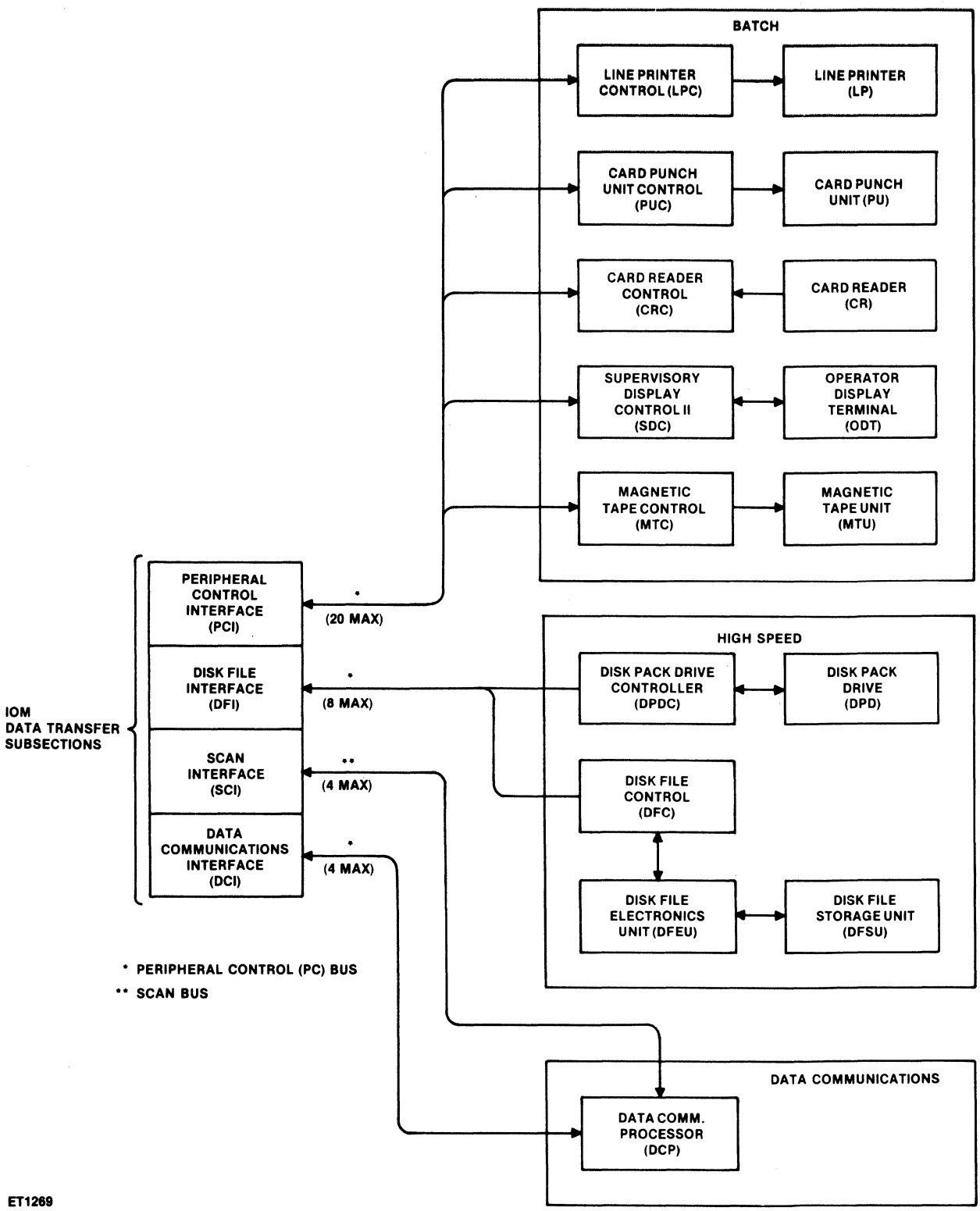


Figure 4-1-3. Typical Data-Transfer Classifications and Related IOM Subsections

rected by the translator subsection of the IOM. Upon command from the translator, the PCI initiates requests with its associated peripheral devices. The PCI controls the data transfers with the peripheral devices and notifies the translator of termination status. During data transfer operations, the PCI communicates with the DSB to obtain memory access. The order of priority for memory access is on a first-in and first-out basis.

The PCI multiplexes all 20 channels by generating overlapping one-micro-second data-service cycles and by use of windows in a self-contained local memory. In the typical configuration (figure 4-1-4) the use of two IOMs and appropriate exchanges (2x16) allows access by either IOM of 32 magnetic tape units. Both IOMs are illustrated as having access to an additional non-exchange magnetic tape unit, as well as having access to disk pack drives (via DPDC) and to ODT units (via supervisory display controls).

Disk File Interface (DFI)

The DFI of a single IOM consists of either one or two interface sections, depending on user requirements. Each section has an interface capability of four channels, for a total disk-file-channel capability of eight channels per IOM.

Each DFI four-channel area can service a single DFI/PCC cabinet. This cabinet can contain only four channels, which are dedicated to disk packs. The channels may be connected to the peripherals either directly or by exchanges. As illustrated in figure 4-1-4, the use of two maximum DFI configurations IOMs (eight channels per IOM, four each disk file and disk pack) and appropriate exchanges (2x10 for disk file, 2x8 and 2x16 for disk pack) allows access by either IOM of 20 disk file electronics units (100 disk file storage units) and 72 disk packs.

Upon command from the translator, the DFI initiates requests with its associated disk pack controls. Upon request completion, the DFI notifies the translator of the termination status and awaits re-initiation. During data transfer operations, the DFI communicates with the DSB to obtain memory access. Like the PCI, the order of priority for memory access is on a first-in and first-out basis.

Scan Interface (SCI)

The SCI subsection consists of a DCP scan interface, which provides scan-out control only, and may communicate with up to four DCP's via a scan bus. (See figure 4-1-4.) The SCI is not used for DCP scan-in functions, which are initiated by the DCP. For these functions, the DCP communicates with memory directly via an interface in the DCI and the MIU. The DCP scan bus is not shared by a second IOM.

Data Communications Processor Interface (DCI)

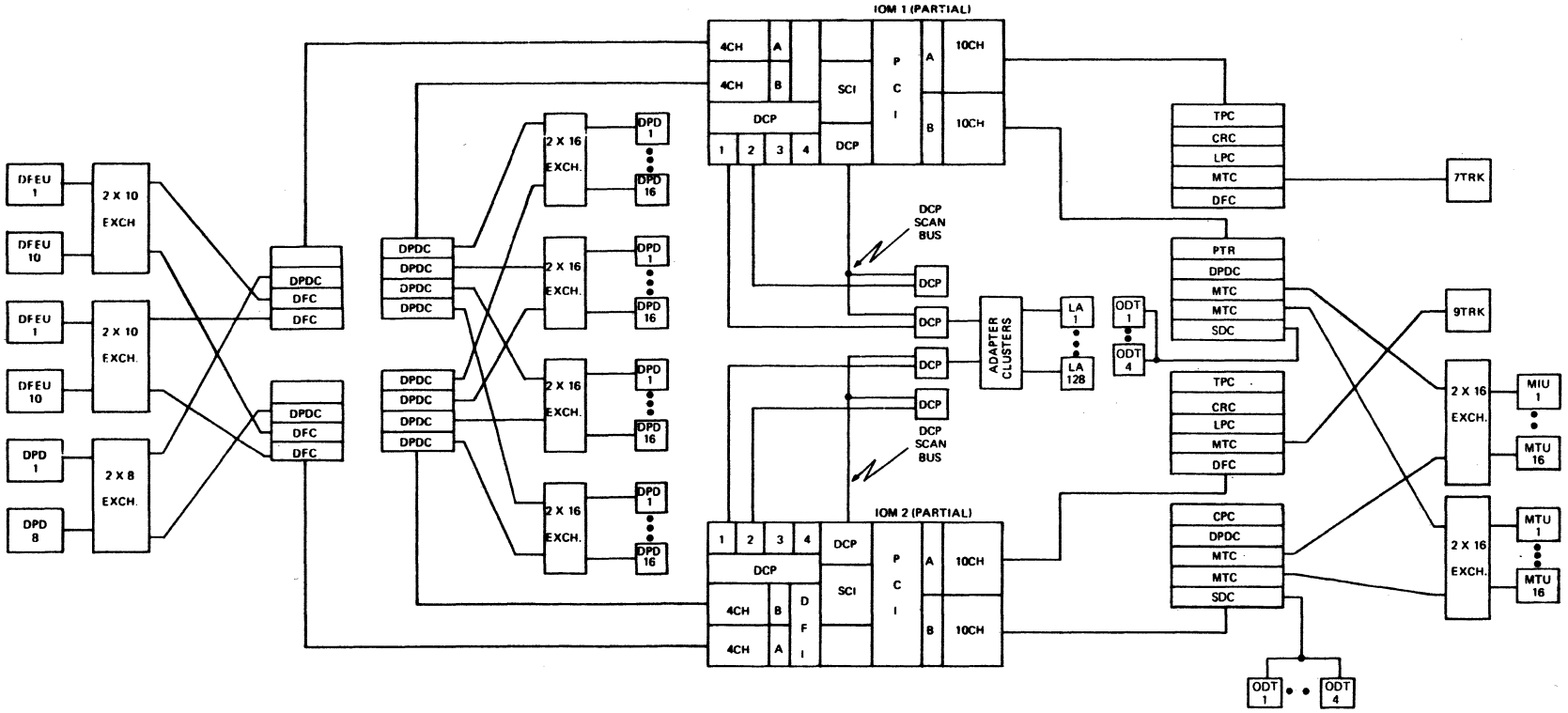
The DCI subsection provides the data and control interface for IOM initiated scan-out operations, and the data interface only for DCP-initiated scan-in operations. Interface is provided for up to four DCP's. As illustrated in figure 4-1-4, an IOM is interfaced with two DCP's.

IOM OPERATIONAL CHARACTERISTICS

The IOM is designed to operate asynchronously with the CPM in the initiation, service, and termination of input/output transfers by use of a *job map* stored in level-1 memory. Basically, the job map consists of five software-constructed elements which define the job request, the peripheral device, and the IOM channel.

Generally, the map elements inform the CPM of its IOM/pripheral resources and their status. When necessary, the CPM alters the queued job requests of the job map to the extent of its interest and interrupts the IOM to request service. The IOM then accesses the job map to determine the input/output job and initiate it. Since the job map is a shared resource of the IOM and CPM, the IOM transfer times are masked by the continual processing and queueing of new requests by the CPM; thus, maximum system throughput is attained with a minimum of CPM time.

The IOM also manages path selection to the requested device (instead of the programmatic preselection of the path which is generally used). This path management eliminates the occurrence of situations whereby: 1) the requested device is free; 2) the preselected path is not free; and 3) an alternate path exists but cannot be used due to the programmatic preselection. These situations generally require involvement of the CPM until the preselected path is free and the job is initiated, which effectively reduces the parallelism of the CPM and IOM. Since the IOM manages the path selection in the B 7800 system, CPM involvement regarding job initiation ends when an interrupt is sent to the IOM. The IOM then initiates the job request when the requested device and any path to that device is available.



LEGEND

DFEU	DISK FILE ELECTRONIC UNIT	TPC	=	TAPE CONTROL
DPD	DISK PACK DRIVE	CRC	=	CARD READER CONTROL
DPDC	DISK PACK DRIVE CONTROL	LPC	=	LINE PRINTER CONTROL
DCP	DATA COMM PROCESSOR	MTC	=	MAGNETIC TAPE CONTROL
LA	LINE ADAPTER	SDC II	=	SUPERVISORY DISPLAY CONTROL II
ODT	OPERATOR DISPLAY TERMINAL	MTU	=	MAGNETIC TAPE UNIT

Figure 4-1-4. Example of IOM Configuration

The design of the IOM incorporates extensive error-detection logic which monitors the flow of control words and data between the IOM and other mainframe modules, within the IOM module itself, and between the IOM module and peripheral devices. Particular emphasis is placed upon preserving the integrity of all memory operations. Generally, the error-detection hardware consists of: parity check and generate circuitry; residue check circuitry; circuitry to detect illegal commands, conditions, and control states; and timeout circuitry for memory transfers, scan bus operations, and internal IOM transfers.

IOM Job Map

The job map, which an IOM accesses from main memory, consists of the following five software-constructed elements:

1. Home Address Words (HA)
2. Unit Table Word (UT)
3. I/O Queue (IOQ)
4. I/O Control Block (IOCB)
5. Status Queue (SQ)

The following four level-1 addresses, which are loaded into the IOM XLATOR at initialize time, enable the IOM to service the job map:

1. Home Address
2. UT Base Address
3. IOQ Header (IOQH) Address
4. SQ Header (SQH) Address

By use of these stored addresses and the contents of previously-fetched map elements, job requests (originally constructed by the CPM) are reconstructed in the IOM and are serviced.

The following basic description of each map element and the sequence in which the job map is serviced is presented in reference to figure 4-1-5. For detailed formats of all words discussed, refer to the appendix of IOM word.

Home Address Word

The 51-bit home address word (HA word) is the first map element fetched by the IOM when interrupted by the CPM. It is fetched by use of the preloaded home address stored in the IOM XLATOR, and contains information which describes the basic command and, as applicable, information which describes the device or channel to be used.

The command to be performed is defined by a code within the HA word, called the home code. In some instances, further definition of the command is provided by additional bits of the HA word. Based on the command decoded, the logic of the IOM is conditioned to perform one of 20 possible control operations. The commands are described under the heading IOM COMMANDS later in this section.

Only one of the 20 commands, the start I/O command, requires immediate further access of other map elements; however, some scan-out commands require access of a second 51-bit HA word. The start I/O command is the basic command used to initiate service of new job requests, whereas the remaining commands are provided for either coldstart/halt load, scan-out control, configuration determination, or diagnostic purposes.

A HA word which contains a start I/O home code also contains a unit designate (UD) number. This number specifies the device to be used for the operation, and is part of the information needed to access the remaining map elements.

Unit Table Word

The unit table (UT) word is the next map element fetched by the IOM in response to a start I/O command. The fetch is performed by use of the UT address preloaded in the IOM XLATOR and the UD number derived from the HA word. The preloaded address serves as a locator for the unit table, and the UD number serves as an index to a particular word of the unit table.

The unit table consists of 256 words, numbered 0 - 255. Word 0 is reserved for use as a fail UT word, and is accessed when an error occurs which cannot be associated with a specific job request. In this instance, a special UD number (000), called a fail UD number, serves as an index to UT word 0. Each of the remaining 255 UT words is assigned to a unique device, and contains information which defines the device and its assignment within the system.

The device-type and assignment information specifically indicates: 1) whether the device is a disk-pack or a magnetic tape unit; 2) if the device is a disk file; 3) whether the device is connected to an exchange; 4) the lowest IOM channel to which the device is connected; and 5) whether this is a high speed device.

For a device connected to an exchange, the UT word contains additional information for use in IOM device/path management. The devices connected to an exchange are described by a linked list of UD numbers in the next unit on exchange (NUD) fields of their UT words. The number (modulo 4) of the last (highest) IOM channel on the exchange (LCEX) is also indicated. The description of the exchange is

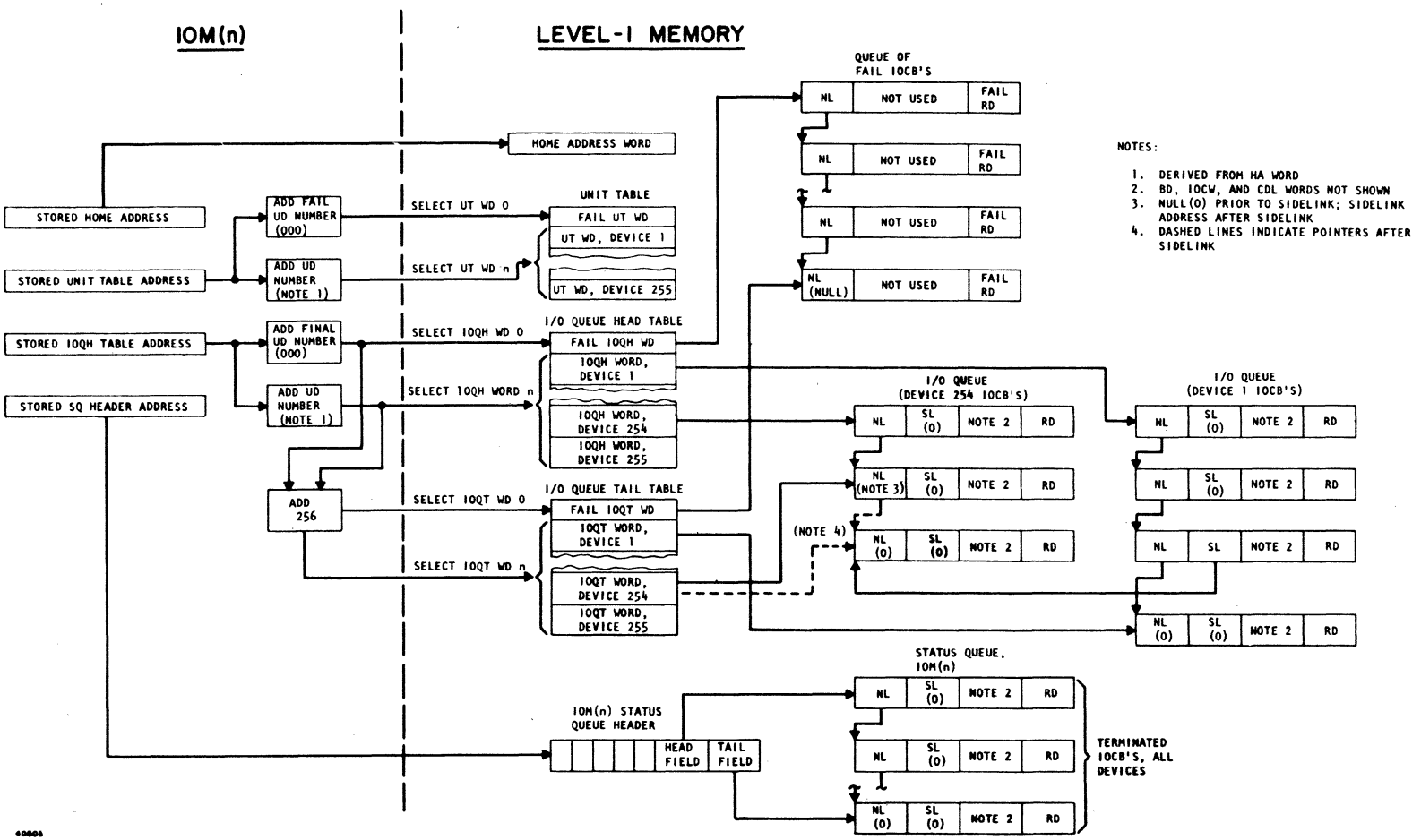


Figure 4-1-5. IOM Job Map

complete because: 1) all IOM's on an exchange must use the same channels; 2) channels on an exchange must be consecutive; and 3) the largest exchanges serve a maximum of four channels. A bit (job bit or JB) is set if a job request for an exchange device must be delayed because a path is not currently available.

IOQ Head (IOQH) and IOQ Tail (IOQT) Tables and Words

The I/O queue (IOQ), which is constructed by the CPM in main memory, contains linked job requests (I/O control blocks) for each device of the system. The extent of the linked job requests for each device is defined by words which indicate the main memory addresses of the first and last of the requests. These words are called the I/O queue head (IOQH) word and the I/O queue tail (IOQT) word, respectively.

The IOQH words for all devices (255 words) are stored in a table called the I/O queue head table; similarly, the IOQT words for the devices (also of 256 words) are stored in a table called the IOQT table, which immediately follows the IOQH table in memory.

The IOQH and IOQT tables contain one special word each (word 0) which is reserved for use by the IOM to report errors that cannot be associated with a specific job request. These words are pointers to a list of the fail I/O control blocks (fail IOCB's) reserved for failure-reporting by the IOM's of the system.

The IOQT table is the element accessed by the CPM to queue additional requests for a device. The IOM also accesses this element when a sidelink operation to another device is specified. This access is required so that the sidelink operation indicated in the job-request queue of one device may be linked to the queue of job requests for the device designated for the sidelink operation. The IOQT word for the sidelink device is altered to reflect the main memory address of the sidelink job, which becomes the last job queued.

The IOQH table is the element accessed by the IOM in order to service job requests. The IOQH word for a device indicates the main memory address of the first job request for that device. Memory addresses of additional jobs for the device are indicated by the next link (NL) word in each job request, thus linking all job requests for a given device.

As is indicated in figure 4-1-5, the last job request for a device is recognized by the IOM when the next link field of a request is found to contain zeroes (null).

The IOQH word is fetched by use of: 1) the IOQH table base address (stored in the XLATOR); and (2) the UD number (derived from the previously-fetched HA word). The UD number indicates which device is to be initiated, and which IOQH word of the IOQH table should be fetched. The UD number is an index to the IOQH table.

When a non-request-related error is detected by the IOM and access to the fail IOQH word (word 0) is required, the word is fetched by use of the fail UD number (000) and the IOQH base address. The memory address of the first available fail IOCB, which is contained in the fail IOQH word, is used to fetch the fail IOCB. The NL field contained in the fetched fail IOCB is then used to update the memory address of the fail IOQH word, so that if a second failure is detected, the next fail IOCB of the queue of fail IOCB's can be accessed. The fail IOQT word, which defines the last IOCB in the queue of ten fail IOCB's, is used only by software; it is not accessed by the IOM.

When a sidelink operation requires a fetch of the IOQT word for a device, 256 is added to the IOQH word address. (The IOQT-word address for a device designated for a sidelink operation equals the IOQH table base address plus the UD number of the device plus 256.)

I/O Control Blocks

The job requests for each device are stored in map elements called I/O control blocks. Each I/O control block (IOCB) contains words which are fetched sequentially starting with the memory address obtained from either: 1) the IOQH word, if the job request is the first for the device; 2) the next link (NL) field of the job request (IOCB) in process, if the job is other than the first for that device; or 3) the side link field of the job request (IOCB) in process, if a sidelink (SL) to another device is indicated. The six IOCB words fetched by the IOM are as follows:

1. Next Link (NL) Word
2. Side Link (SL) Word
3. Buffer Descriptor (BD) Word
4. I/O Control Word (IOCW)
5. Channel Designate Level (CDL) Word
6. Result Descriptor (RD) Word

As previously indicated, the NL word contains the address of the next IOCB for a device, and is the means whereby job requests for a device are linked within the IOQ. When this word contains all zeroes (null), it indicates the request being serviced is the last currently enqueued for the device.

The SL word is used to indicate that a sidelink operation (the service of a job request by a device

other than that presently being serviced, without intervention by the CPM) is required. The SL word contains the memory address of the sidelink job (IOCB), which is started immediately if no other jobs are queued for the designated sidelink device. If queued, the sidelink job is linked to the queue of job requests by insertion of the sidelink memory address in both the IOQT word for the sidelink device and the NL field of the last IOCB previously queued for that device.

The BD word contains the address of the first data location in memory, and the length of the memory area in words.

The IOCW contains the control information necessary to perform the input/output operation, such as: read or write; whether code translation is necessary; backward/forward (tape); frame length (six-bit or eight-bit); etc. The contents of the IOCW and the BD word are used to format the first job word sent to the selected IOM channel.

The CDL word is used to format the second job word sent to the selected IOM channel. This word generally contains information such as: the OP code; the device number; the device variant; and for disk, the segment address.

The RD word is used for storage of the termination status of each request. The RD word is built by the IOM, which then links the terminated request (terminated IOCB) into the status queue.

Fail I/O Control Blocks

A queue of special IOCB's, which are not related to job requests, is also built in memory. These IOCB's, which are called fail IOCB's, are used by the IOM's of the system for reporting errors which cannot be associated with a specific request. The fail IOCB's contain the same six words as job-request IOCB's; however, only the next link word and the result descriptor word have significance.

The result descriptor word is used for storage of a fail result descriptor. The IOM builds the fail result descriptor, inserts it in the fail IOCB RD word, and links the fail IOCB into the status queue.

Status Queue

The status queue (SQ) is a queue of: 1) all job-request related IOCB's which have been serviced and terminated; and 2) any fail IOCB's which have been generated by the IOM. When job-request IOCB's are terminated (or fail IOCB's are generated) and the necessary result descriptor information has been stored in the RD word of the IOCB, the IOCB is unlinked from the job IOQ (or fail IOQ) and is linked into the status queue. The linked IOCB's in the status queue represent a mix of terminated IOCB's for all devices and any fail IOCB's.

The SQ for the system consists of queues of linked IOCB's - one queue for each IOM on the system. The number of queues is dependent on the number of IOM's in the system.

The mechanism by which the status queue is accessed is the SQH address; this is stored in the IOM XLATOR at initialize time. This address is unique for each IOM used, and serves as a pointer to a word in memory which defines the queue of linked IOCB's associated with a particular IOM. This word is called the status queue header (SQH) word.

When a request is terminated, the SQ address of an IOM is used to fetch the SQH word, which contains the following basic information:

1. Null (empty) state of SQ
2. Head field
3. Tail field
4. Status-Change-Vector bit
5. CPM-Interrupt bit
6. CPM Number

The null state of the SQ is checked to determine whether it contains any terminated IOCB's. If the SQ is null (empty), no linkage of the current terminated IOCB to previously terminated IOCB's in the SQ is required. Conversely, if the SQ is not null (contains one or more IOCB's or fail IOCB's), the current terminated IOCB must be linked to the queue of terminated IOCB's in the SQ.

The head field of the SQH word contains the base address of the first terminated IOCB of the SQ. The tail field of the SQH contains the base address of the last terminated IOCB of the SQ, except when the SQ is null or contains only one terminated IOCB. If the SQ is null, the tail field is not used; if the SQ contains only one IOCB, the tail field contains the same address as the head field.

A terminated IOCB is linked to previously terminated IOCB's stored in the SQ by inserting its base address in the next link (NL) word of the terminated IOCB indicated by the SQH tail field. The address in the tail field of the SQH is then replaced with the base address of the currently-terminated IOCB, so that link capability is present when another request is terminated.

If the CPM interrupt bit is on in the SQH word, or the interrupt bit is on in the NL word of a terminated IOCB, a channel interrupt is sent to the CPM specified in SQH when the terminated IOCB is

linked into SQ. An IOM error interrupt is always sent to the designated CPM when a fail IOCB is linked into the SQ.

When the SPO or a DCP requests an input operation, the status-change vector bit in SQH is set, and a channel interrupt is always sent to designated CPM.

IOM Home (HA) Commands

The IOM can be directed to perform 20 home commands. When the IOM receives an interrupt from the CPM, it indicates that a home command has been constructed by the CPM and placed in memory. The home address stored in the IOM is then used to fetch the HA word. A code within HA word 1, home code, is then decoded to determine which command or command group is to be performed.

Table 4-1-1 lists the valid home codes, and the commands and/or command groups defined by them. As indicated, scan commands are defined by: 1) the home code as only scan-in or scan-out groups; 2) determination of type of scan-in or scan-out; and 3) whether DCP is defined by other portions of HA word 1. Similarly, channel busy/channel reserved commands are resolved by other portions of HA word 1. HA word 2 is not used for all commands; when used, it contains information to further define the command.

Table 4-1-1. IOM HA Operations and Corresponding Home Codes

Home Code	IOM Operation
0000	Illegal
0001	Start I/O
0010	Set Channel Busy/Set Channel Reserved
0011	Reset Channel Busy/Reset Channel Reserved
0100	Load Home Address
0101	Load Unit Table Address
0110	Load IOQ Head Table Address
0111	Load SQ Header Address
1000	DCP Scan-out Commands: Initialize Halt Set Attention
1010	Synchronous I/O
1011	Interrogate Peripheral Status
1100	Inhibit IOM
1101	Activate IOM
1111	Illegal

The following brief command descriptions are presented in reference to figure 4-1-6, which depicts the basic contents of the HA words for each command. Detailed formats of the HA word for each command are given in the appendix of IOM word formats.

Start I/O (Home Code 0001)

The start I/O command is the basic command used to initiate input/output servicing of a new job request for a device. The device is defined by a unit designate number contained in bits 28 - 35 of HA word 1; HA word 2 is not used. This command need only be given once in order to service all queued requests for the designated device.

Set Channel Busy/Set Channel Reserved (Home Code 0010)

Home code 0010 may represent one of two commands, depending on the state of bit 39 of HA word 1. If bit 39 is a 0, the set channel busy command has been received; if bit 39 is a 1, the set channel reserved command has been received. Both commands are for exchange channels; the channel number is defined by bits 23 - 27 of HA word 1. HA word 2 is not used.

The set channel busy and set channel reserved commands are used primarily for diagnostic purposes. A start I/O command for an UD, which has the reserved channel bit (RC) set in its UT word, must use a channel that has been set to reserved; otherwise, a reserved channel will not be used. An I/O operation cannot use a channel that has been set to busy. Once either command has been received, the specified channel remains busy (or reserved) until a counter command is received.

Reset Channel Busy/Reset Channel Reserved (Home Code 0011)

Home code 0011 may also represent one of two commands, depending on the state of bit 39 of the HA word (0 defines the reset channel busy command; 1 defines the reset channel reserved command). These commands are the counter commands to the set channel busy/set channel reserved command.

Load Address Commands

1. Load home address (home code 0100)
2. Load unit table address (home code 0101)
3. Load IOQ head table address (home code 0110)
4. Load SQ header address (0111)

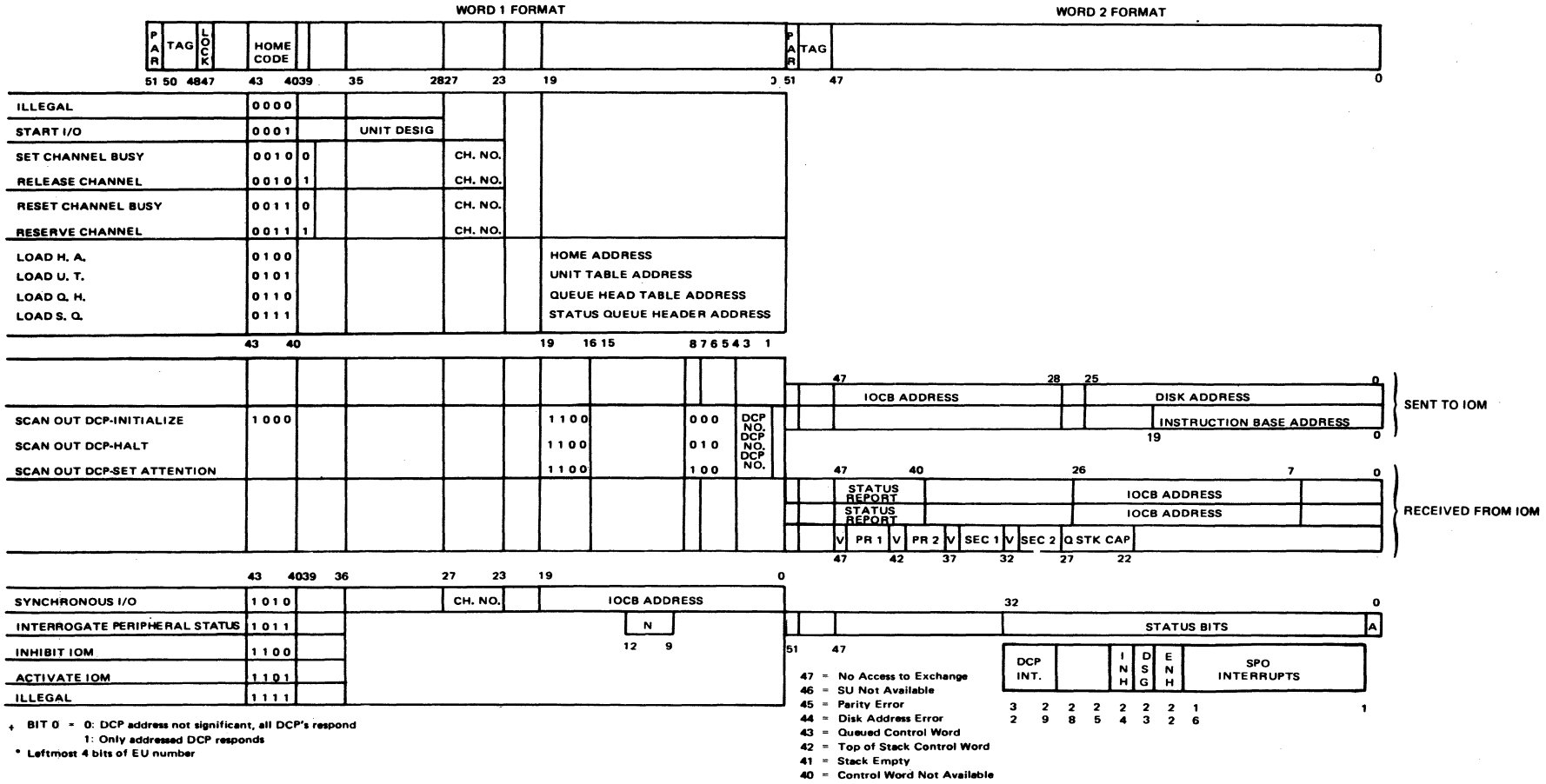


Figure 4-1-6. Home Address Commands

The commands are normally used to load fixed addresses into the IOM XLATOR at initialize time; however, they may also be used to establish new base addresses at any time after initialization. The address to be loaded by each command is contained in bits 0 - 19 of HA word 1; HA word 2 is not used.

DCP Scan-Out Commands (Home Code 1000)

Home code 1000 specifies a scan-out command for a DCP; the specific device for which the scan-out command is intended, as well as the specific type of scan-out command, is defined by other bits of HA word 1.

There are three specific scan-out commands for the DCP; the command type is determined by bits 5, 6, and 7 of HA word 1 as follows:

1. Bits 5, 6, and 7 = 0: Initialize
2. Bit 5 = 0, bit 6 = 1, bit 7 = 0: Halt
3. Bits 5 and 6 = 0, bit 7 = 1: Set Attention

The DCP for which the command is intended is indicated by a DCP number contained in bits 1 - 3 of HA word 1; however, bit 0 is a controlling bit for the DCP address. When it is a 0, the DCP address is not significant, and all DCP's respond. When it is a 1, the addressed DCP is the only DCP to respond.

The initialize command requires access by the IOM of HA word 2, which contains an instruction base address (bits 0 - 19). HA word 2 is not used for the halt and set attention commands.

The initialize and halt commands cause psuedo fault interrupts to occur within the DCP. In the case of the initialize command, the interrupt causes the 20-bit instruction base address to be loaded into the DCP scratch-pad memory. The interrupt generated by the halt command stops DCP operations. In either case, stop actions which would normally occur within the DCP due to fault interrupt occurrence are inhibited.

The set attention command is used to notify the DCP that attention to the B 7800 system is required.

Synchronous I/O Command (Home Code 1010)

The synchronous I/O command provides a means of servicing a single job request during initialization. Only HA word 1 of the job map, which contains the IOCB base address (bits 0 - 19), is accessed; no queue mechanisms are used. When the single job request is terminated, the result descriptor information is stored in HA word 5, and a channel interrupt is sent to the CPM.

Interrogate Peripheral Status Command (Home Code 1011)

The interrogate peripheral status command is used to determine the ready status of all devices assigned to a particular status vector. The status vector to be interrogated is indicated by bits 9 - 12 of HA word 1.

HA word 2 is not accessed for command determination, but is later used for storage of the returned status information. The status information, which is returned in bits 1 - 32 of HA word 2, provides indication of the ready status of up to 32 devices on a vector. Bit 0 of HA word 2 (ATTN) notifies the CPM that the status word has been returned.

Inhibit IOM Command (Home Code 1100)

The inhibit IOM command is used to inhibit all automatic IOM functions, such as data-path management, chaining of linked and side-linked job requests, and ringwalk. If linked job requests for ringwalk devices are being serviced when the command is received, chaining stops after the IOCBs (in progress on each channel) are completed.

The content of HA word 1 consists only of the home code; HA word 2 is not accessed.

Activate IOM Command (Home Code 1101)

The activated IOM command is used to restore automatic functions of the IOM after the inhibit IOM command has been given. The command consists only of the home code in HA word 1; HA word 2 is not used.

AUTOMATIC DISK-PACK OPERATION

The IOM has provisions to automatically re-initiate a disk pack unit after the unit has completed a seek operation. This unit, when issued a conditional I/O command requiring head positioning (seek), must be issued the same command after the seek has been completed in order to accomplish data transfer. The IOM performs this function by examining all device result descriptors received from disk pack units.

DATA TRANSLATION

The PCI section of the IOM has the capability of translating data from one representation code to another during an I/O operation. The data types actually encountered are device dependent, and the translation to be performed (if any) is determined for each individual job request by standard control bits in the IOCW. The IOCW bits used to specify code translations are:

Bit 46. ASCII on for any translation having ASCII input or output.

Bit 44. READ -(READ = 1, WRITE = 0)

Bit 42. TRANSLATE

Bit 41. FRAME LENGTH (eight-bit characters = 1; six-bit characters = 0.)

All possible combinations of these code bits are listed in detail in table 4-1-2 and the specific translation codes used by software for each peripheral devices are given in table 4-1-3.

EBCDIC-BCL Exceptions

Bi-directional translation of corresponding EBCDIC graphics to/from corresponding BCL graphics are provided with the following exceptions:

1. EBCDIC to BCL (output translator)

EBCDIC	BCL
PZ	+
MZ	X (times)
Corresponding graphics	Corresponding graphic
Non-corresponding graphics	(See Note)

NOTE
The following graphics are printed dependent upon whether the printer is equipped for EBCDIC or BCL:

EBCDIC	BCL
' (Apostrophe)	≥
¬ (Logical not)	≥
_ (Underscore)	←
| (Vertical bar)	→

2. BCL to EBCDIC translator (input translator)

BCL X (times)	EBCDIC MZ
Corresponding graphics	Corresponding graphics

Table 4-1-2. General Translation Specification Codes

R 44	T 42	FL 41	A 46	
0	0	0	0	Write 6-bit bytes with no translation
0	0	0	1	(Illegal Code)
0	0	1	0	Write 8-bit bytes with no translation
0	0	1	1	Write EBCDIC from ASCII
0	1	0	0	Write BCL External from BCL Internal
0	1	0	1	Write BCL External from ASCII
0	1	1	0	Write BCL External from EBCDIC
0	1	1	1	Write ASCII from EBCDIC
1	0	0	0	Read 6-bit bytes with no translation
1	0	0	1	(Illegal Code)
1	0	1	0	Read 8-bit bytes with no translation
1	0	1	1	Read EBCDIC into ASCII
1	1	0	0	Read BCL External into BCL Internal
1	1	0	1	Read BCL External into ASCII (See Note 1)
1	1	1	0	Read BCL External into EBCDIC
1	1	1	1	Read ASCII into EBCDIC

Table 4-1-3. Translation Codes by Device

CODE SPECIFIER

Device	R 44	T 42	FL 41	A 46	Description
Card Reader	1	0	0	0	Read binary data (6-bit to 6-bit)
	1	0	1	0	Read EBCDIC data (8-bit to 8-bit)
	1	0	1	1	Read EBCDIC into ASCII
	1	1	0	0	Read BCL External into BCL Internal
	1	1	0	1	Read BCL External into ASCII
	1	1	1	0	Read BCL External into EBCDIC
Card Punch	0	0	0	0	Punch binary (6-bit to 6-bit)
	0	0	1	0	Punch EBCDIC (8-bit to 8-bit)
	0	1	0	0	Punch BCL External from BCL Internal
	0	1	0	1	Punch BCL External from ASCII
	0	1	1	0	Punch BCL External from EBCDIC
	0	0	1	1	Punch EBCDIC from ASCII
Line Printer	0	0	0	0	Write BCL External (6-bit to 6-bit)
	0	1	0	0	Write BCL External from BCL Internal
	0	1	0	1	Write BCL External from ASCII
	0	1	1	0	Write BCL External from EBCDIC
Train Printer	0	0	0	0	Write with no translation (6-bit to 6-bit)
	0	0	1	0	Write with no translation (8-bit to 8-bit)
	0	0	1	1	Write EBCDIC from ASCII
	0	1	0	0	Write BCL External from BCL Internal
	0	1	0	1	Write BCL External from ASCII
	0	1	1	0	Write BCL External from EBCDIC
	0	1	1	1	Write ASCII from EBCDIC
P.T. Reader	1	0	0	0	Read binary (6-bit to 6-bit)
	1	0	1	0	Read EBCDIC or Read ASCM (8-bit to 8-bit)
	1	0	1	1	Read EBCDIC into ASCII
	1	1	0	0	Read BCL External into BCL Internal
	1	1	0	1	Read BCL External into ASCII
	1	1	1	0	Read BCL External into EBCDIC
	1	1	1	1	Read ASCII into EBCDIC
P.T. Punch	0	0	0	0	Punch binary (6-bit to 6-bit)
	0	0	1	0	Punch EBCDIC or Punch ASCII (8-bit to 8-bit)
	0	1	0	0	Punch BCL External from BCL Internal
	0	1	0	1	Punch BCL External from ASCII
	0	1	1	0	Punch BCL External from EBCDIC
	0	1	1	1	Punch ASCII from EBCDIC
	0	0	1	1	Punch EBCDIC from ASCII
7 TR. TAPE	0	0	0	0	Write with no translation (6-bit to 6-bit)
	0	1	0	0	Write BCL External from BCL Internal
	0	1	0	1	Write BCL External from ASCII
	0	1	1	0	Write BCL External from EBCDIC
	1	0	0	0	Read with no translation (6-bit to 6-bit)
	1	1	0	0	Read BCL External into BCL Internal

CODE SPECIFIER

Device	R 44	T 42	FL 41	A 46	Description
9 Tr. Tape	0	0	1	0	Write with no translation (8-bit to 8-bit)
	0	1	1	1	Write ASCII from EBCDIC
	0	0	1	1	Write EBCDIC from ASCII
	1	0	1	0	Read with no translation (8-bit to 8-bit)
	1	1	1	1	Read EBCDIC into ASCII
	1	0	1	1	Read ASCII into EBCDIC
Disk Pack (See Note 1)	0	0	1	0	Write with no translation (8-bit to 8-bit)
	1	0	1	0	Read with no translation (8-bit to 8-bit)

Note 1:
The DFI section of the IOM has no hardware translation capabilities.

IOM-GENERATED INTERRUPTS

The IOM generates the following two interrupts and sends them over individual lines to each central processor:

1. Channel Interrupt
2. IOM Error Interrupt

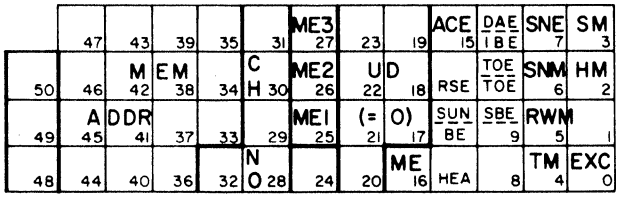
An IOM interrupts exactly one processor; the CPM to be interrupted is determined by the CPM field (bits 44-42) of SQH.

There are three conditions under which the IOM generates an interrupt to a CPM:

1. IO Complete -On job request termination, a channel interrupt is generated when bit 40 of the IOCB NL word is set or when bit 40 of the status queue header is set. These bits are set by software; bit 40 in SQH is reset by the IOM after an interrupt is generated. Unless requested by software, an interrupt is not set for exception conditions (peripheral parity error, end of tape, etc.). The only action taken for an exception condition is that the next request job, if there is more than one request job queued, is not started and bit 0 of the UT word is set.
2. Status Changes -When the inquiry request line for a single line control device changes from off to on state, or when a DCP requests CPM attention, the IOM sets the proper bit in its status change vector (status vector 8). If this is the first such change since the vector was last interrogated by software, the IOM sets bit 45 in SQH to request software to read the status change vector and generates a channel interrupt. (Bit 45 in SQH is reset by software.)
3. IOM Errors -An IOM error interrupt is generated for any error not related to a specific job request (e.g., a memory parity error on the home address word). The generated fail result descriptor is placed in a dummy IOCB from unit 0.

IOM Fail Word

The IOM fail word (figure 4-1-7) is a 48-bit word which contains information regarding errors which cannot be associated with a particular channel or device. (Such errors cause an IOM error interrupt.) When an IOM error interrupt occurs, an IOM fail word is built by the fail mode logic within the IOM translator and placed in the result descriptor word of the "fail IOCB". The fail IOCB is associated with unit designate number 0. The fail IOCB is delinked from the queue of fail IOCB's and linked into the queue of completed IOCB's (defined by the status queue header) in the same manner as a normal I/O termination.



Field	Bits	Error
EXC	0	Exception Bit
MODE WHEN ERROR OCCURRED		(See bits 2 to 6)
HM	2	Home Address Mode
SM	3	Start Mode
TM	4	Terminate Mode
RWM	5	Ring Walk Mode
SNM	6	Scan Mode
SNE	7	(See bits 9 to 14 or bits 10 to 14)
SCAN ERRORS		(If SNE bit 7=1)
SBE	9	Scan Bus Parity Error
TOE	10	Time Out Error (Scan Bus)
DAE	11	Disk Address Error
SUN	13	Storage Unit Not Available
NON-SCAN ERRORS		(If SNE bit 7=0)
TOE	10	Time Out Error (Data Service)
IBE	11	Initiate Busy Channel
HAE	12	Home Address Illegal Command
BE	13	Buffer Register Parity Error
RSE	14	Resident Error (Memory Address)
MEM ADDR	47:20	Memroy Address of 10CB
ME	16	Memory Error (see bits 25 to 27)
ME1, ME2, ME3	27:3	Memory Error Code (If ME bit 16=1)
UD	24:8	Unit Designate (=0)

* IBE is set if:

1. Start Mode. The unit table is either busy (bit 36) or the job bit is set (bit 37) for non-exchange devices. The active channel stack of the IOM has a non-exchange channel marked busy or is in the process of terminating a job, or is reserved.
2. Unit table is not busy (bit 36 = 0) when the IOM job begins to terminate (terminate mode).

Figure 4-1-7. IOM Fail Word

SECTION 2
FUNCTIONAL OPERATION OF INPUT/OUTPUT MODULE SUBSYSTEMS

GENERAL

This section contains a brief description of the operation of each of the IOM subsections described in Section 1 of this chapter. For the formats of the words discussed, refer to the appendix of IOM word formats in this manual.

TRANSLATOR

The translator (figure 4-2-1) is a special-purpose processor capable of performing specific hardwired microsequences. It is the mechanism of the IOM that services I/O requests, generates the request descriptors required to initiate peripheral devices, and reports job termination and failure status conditions to the central processor. The translator is keyed to respond to certain declared flag conditions.

Job Service Initiation

In response to an interrupt from the central processor, the IOM unlocks-fetches the word in memory named by the 20-bit home address (HA) stored in the lower stack. The HA word control fields define the control codes and function details for the request as described in Section 1 of this chapter. When the start I/O command is decoded, the unit designate (UD) field of the HA word is loaded into the UD register (see figure 4-2-1).

The UD field is added to the 20-bit unit table (UT) base address (stored in the UT location of the lower stack) in order to address and lock fetch, from memory, (write with flashback) the unit table word for the device to be started. For devices other than a DFO, the contents of the channel number identification field of the UT word are used to access the active channel stack (ACS). If the device is not connected to an exchange, and if the ACS and UT word busy bits are reset, the I/O queue head word (IOQ base address + UD) is fetched from memory to obtain the base address of the I/O control block (IOCB).

Successive fetches are made of:

1. The IOC base address plus 2 - to obtain the buffer descriptor.
2. The IOCB base address plus 3 - to obtain the I/O control word (IOCW).
3. The IOCB base address plus 4 - to obtain the channel designate level (CDL) field.

The buffer descriptor is comprised of two fields: base address information and buffer-length information. The 20-bit base address field is used to locate the buffer in memory.

The IOCW standard control field (SCF) has information useful to the data service sections such as read/write, translate, and format bits. Information contained in the buffer descriptor, SCF and the IOCW, and CDL word of the IOCB is sent to the data service section to start the selected device. The unit designate number is stored in the ACS and the ACS busy bit is set. The unit table word busy bit is set, and the UT word stored in memory is unlocked. The HA word is unlocked and set to all zeros. Control is transferred to the initial state.

If the device to be started is connected to an exchange and the busy bit of the base channel location in the ACS is set, the translator logic selects the next channel of the exchange and checks its busy bit. If a channel is available, information is fetched from memory and goes to the data service section to start the selected device. If all channels of the exchange are busy, the job bit (JB) in the unit table word is set, and the word is stored unlocked in memory. Control is transferred to the initial state. These conditions are summarized in table 4-2-1.

Job Service Termination

When a device either completes a service or is terminated as a result of an error condition, the data service unit causes the terminate bit to be set for that channel. Terminate bits are located in the active channel stack (ACS) of the translator; one bit for each of the possible 28 channels available. In response to a terminate bit being set, the translator reads the corresponding unit designate information from the ACS. This information is used along with the unit table base address to index and lock-fetch from level-1 memory the UT word for the terminating device. The I/O queue head is then fetched to obtain the base address of the I/O control block (IOCB). The result descriptor (RD) information received from the data service unit is then stored in the sixth word of the IOCB, and the IOCB is linked to the status queue (SQ).

If this is the last request for this unit, the I/O queue head (IOQH) and I/O queue tail (IOQT) are nulled, and the UT word is stored unlocked, to complete the termination. If there are more requests, the

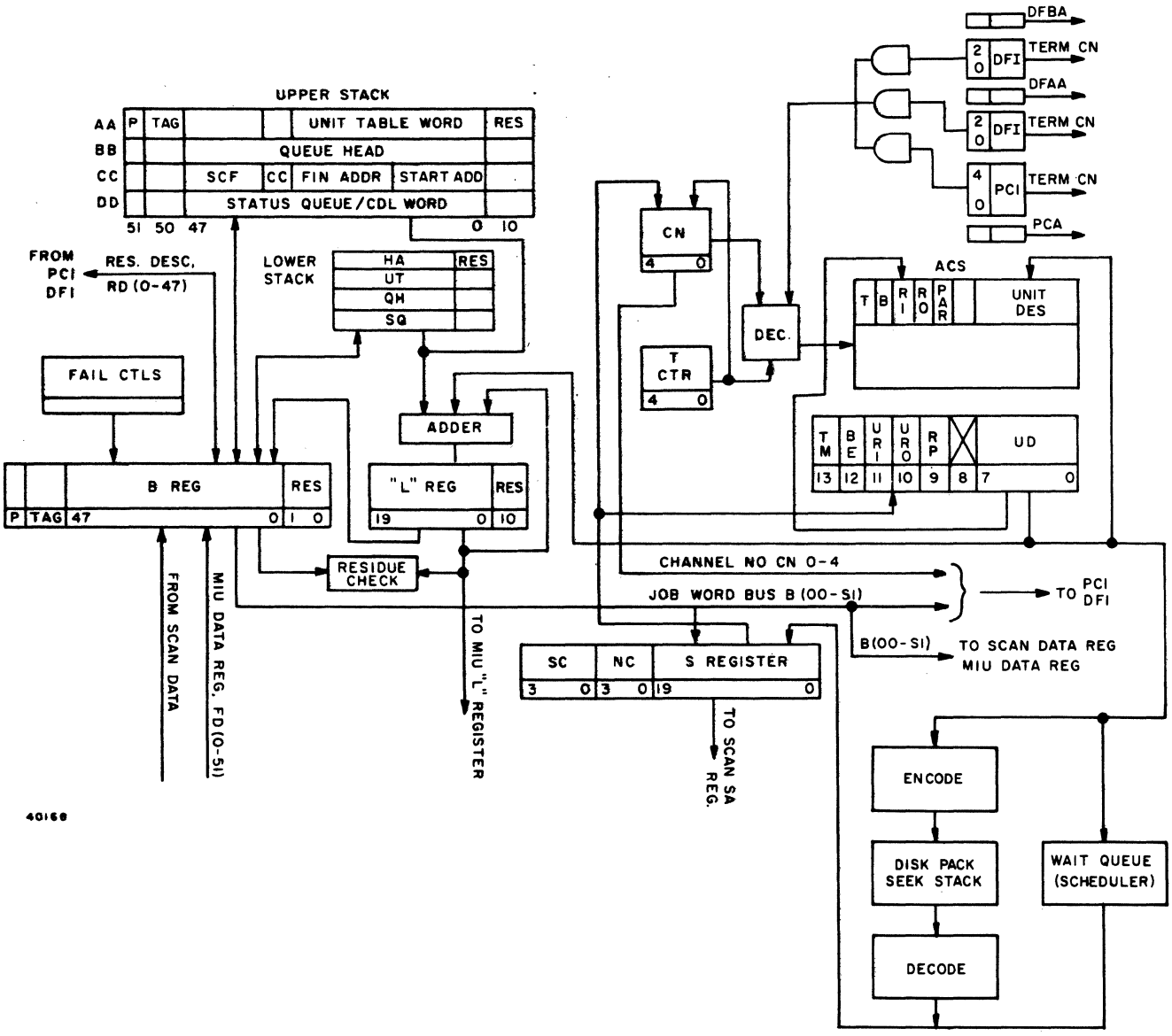


Figure 4-2-1. Translator Component Interface

Table 4-2-1. Unit Table and Active Channel Coded Decisions

Unit Table Word				A.C. Stack			
B20 RC	B38 EX	B37 JB	B36 BZ	CBF	CTF	CRF	Decisions
M	X	0	0	0	0	M	Start job; unlock UT; Set BZ (UT)
X	0	1	X	X	X	X	Error; Set Initiate Busy Channel Error (IBE) in Fail Word
X	0	X	1	X	X	X	Error; Set Initiate Busy Channel Error (IBE) in Fail Word
X	0	X	X	1	X	X	Error; Set Initiate Busy Channel Error (IBE) in Fail Word
X	0	X	X	X	1	X	Error; Set Initiate Busy Channel Error (IBE) in Fail Word
U	0	X	X	X	X	U	Error; Set Initiate Busy Channel Error (IBE) in Fail Word
X	*1	1	0	X	1	X	Unlock UT; Go to initial state
X	*1	1	0	1	X	X	Unlock UT; Go to initial state
X	1	0	0	X	1	X	Unlock UT; Set JB; Go to initial state
X	1	0	0	1	X	X	Unlock UT; Set JB; Go to initial state
M	1	1	0	0	0	M	Start job; Unlock UT; Set BZ, Res JB
U	1	X	0	X	X	U	Unlock UT; Set JB, go to initial state

X = State irrelevant
M = States match
U = States do not match
EX = Exchange Bit
JB = Job to be done
BZ = Job Busy
RC = Use reserved channel only
CBF = Chan Busy FF
CTF = Chan Term FF
CRF = Chan Reserved FF
* - Applicable only when second IOM has set JB

address of the next IOCB is inserted in the 20-bit address field of the IOQH. Control is passed to the start section to initiate the request. If the terminating IOCB is the last request for this unit, and the unit is connected to an exchange, then control is passed to the ring-walk section and a search is made to find a request that is waiting to be initiated.

Exchange Ring-Walk

The following action is taken when an exchange unit terminates and there are no more requests queued for that particular unit.

The present unit designate (PUD) field from UT is saved in the S register. The next unit designate (NUD) field in the UT word of the terminating device points to the next unit (device) of the exchange. The UT word for this unit is fetched from level-1 memory and status of the busy and job bits is checked. If the unit is busy, or no jobs are awaiting service, the information in the NUD field in the UT word points to the next unit of the exchange. This process of looking for a request continues until one is found or the entire exchange has been walked (NUD = saved PUD). When a unit is found with a serviceable request waiting, the IOQH word for that unit is fetched from memory and the job is started.

Disk Pack Control

The following actions are taken upon the receipt and examination of result descriptors from disk pack.

When a result descriptor indicating "seek initiated" is received, the IOM does not de-link the IOCB or store the result descriptor as in normal terminate operations. Instead, the unit number of the disk pack which began a seek operation is stored in a local stack. Contents of this stack are then used to monitor the ready lines of all disk pack units which are currently seeking.

A "seek complete" is detected when the ready line of a seeking disk pack returns to the TRUE state. At this time, the translator performs a start I/O for that unit. Since the original job was not delinked from the job queue when the disk pack initiated its seek, the same job is issued a second time, and the data transfer occurs. After this point, all IOM operations proceed the same as for normal peripheral units. If the disk pack is issued a conditional I/O command which does not require head positioning, data transfer occurs directly, and the automatic disk pack functions of the IOM are not used.

Fail Mode Of Operation

The fail mode lets the IOM report errors that cannot be associated with a specific request. When an error occurs, such as a scan bus error, memory error, home address error, illegal command, etc., control goes to the fail mode (FM) with appropriate error flags.

A fail result descriptor is built in the fail register. This RD indicates:

1. The operational mode when the error occurred.
2. A possible channel number (or memory address, depending on the type of failure).
3. Error flags describing the type of error.

A fail unit designate number (fail UD = zero) is used with the UT word to access a fail UT word. Then the QH and fail UD are used to access the fail I/O queue head. The fail result descriptor is placed in the result descriptor word of the I/O control block (IOCB). The fail IOCB is then delinked from the queue of fail CBs and linked to the status queue as in normal termination. An IOM error interrupt is then sent by the IOM to the central processor designated in the status queue header.

Scheduler

The scheduler stores UDs for certain high speed devices when throughput capacity of the PCI peripheral bus (or the IOM) exceeds a self-limiting range. The PCI peripheral bus throughput is limited to two megabytes; whereas the self-limiting range of the IOM is dynamic, up to six megabytes, because it varies with memory and peripheral bus access time.

When the translator is in idle mode and no higher priority jobs are awaiting service, processing of the stored UDs can begin. Scheduler logic contains two first-in/first-out wait queues for storing UDs, a job counter to record the number of high speed jobs on PCI channels, and limit circuits for monitoring throughput operations.

The translator identifies high speed devices via bit 40 in the UT word (set by software). Then, during start, the job counter is increased by one to record that a job is in progress for that device. Each time a high speed device is to be serviced, the job counter upcounts by one. Similarly, as each device terminates, the job counter is decremented by one.

Whenever the job counter equals a count exceding the two megabyte throughput, a flag is enabled, indicating PCI limit met. Further high speed operations to be started in the PCI section are written to the PCI wait queue. When the job count decrements below the two megabyte limit due to a high speed device terminating, the wait queue is read, if required.

The self-limiting throughput portion of the scheduler monitors valid array and DSB pointer information to determine if limited I/O throughput occurred. Both store and fetch memory operations are monitored to detect this condition. If this has occurred, the next high speed device UD is written to the proper wait queue in order to perform this job during a future start I/O cycle, as described previously.

Time Logger Logic

The time logger logic calculates the time an IOM had taken to service a channel operation. This calculated time, which is in units of 503 microseconds, and the associated channel number are assembled in the B register, as the IOMTIMECELL word. This word is then stored in memory at the IOCB base address plus six.

The counters in this logic are continuously counting and are only interrupted momentarily at the start and termination of an I/O. When a new channel operation is started, the current time (counter) is written into the storage location of this logic, addressed by channel number. When the operation terminates, the stored time (count) is subtracted from the present time and the resultant figure (elapsed time) is stored as the IOMTIMECELL.

Data Service Quiet Logic

Data service quiet logic is used to inform the MCP that an IOM is not servicing any channels and the UD wait queue is empty. Data service quiet logic is a five-bit counter, which is increased by one for each channel operation started and decreased by one as operations terminate. Thus, if the counter equals zero and the UD wait queue is empty, bits 23 (DSQ - data service quiet) and 22 (ENH - enhanced, which serves as a validity bit for 23) are set in the status change vector 8 field of home address word 2.

Zero Length Detection

Zero length detection logic is used to process jobs in which the length of the memory in words for that job equals zero, as indicated by a zero-length field in the buffer descriptor. This is detected during start and the final address field of job word 1 (JW1) is set to zero. JW1 is sent to either the DFI or the PCI subsection. In the DFI, a zero length detection sets the RDL (result descriptor load) flag and terminates the operation. For the PCI, zero length sets the RDP (result descriptor present) flag. No other action is taken by the PCI, except to recognize that no data is to be transferred and to place the job in the queue as a terminate job.

MEMORY INTERFACE UNIT

The memory interface unit (MIU), shown in figure 4-2-2, performs all data and map-word transfers between the IOM and a maximum of four system memory control modules. It detects and reports memory error conditions to the requesting functional unit of the IOM (and to the translator when applicable).

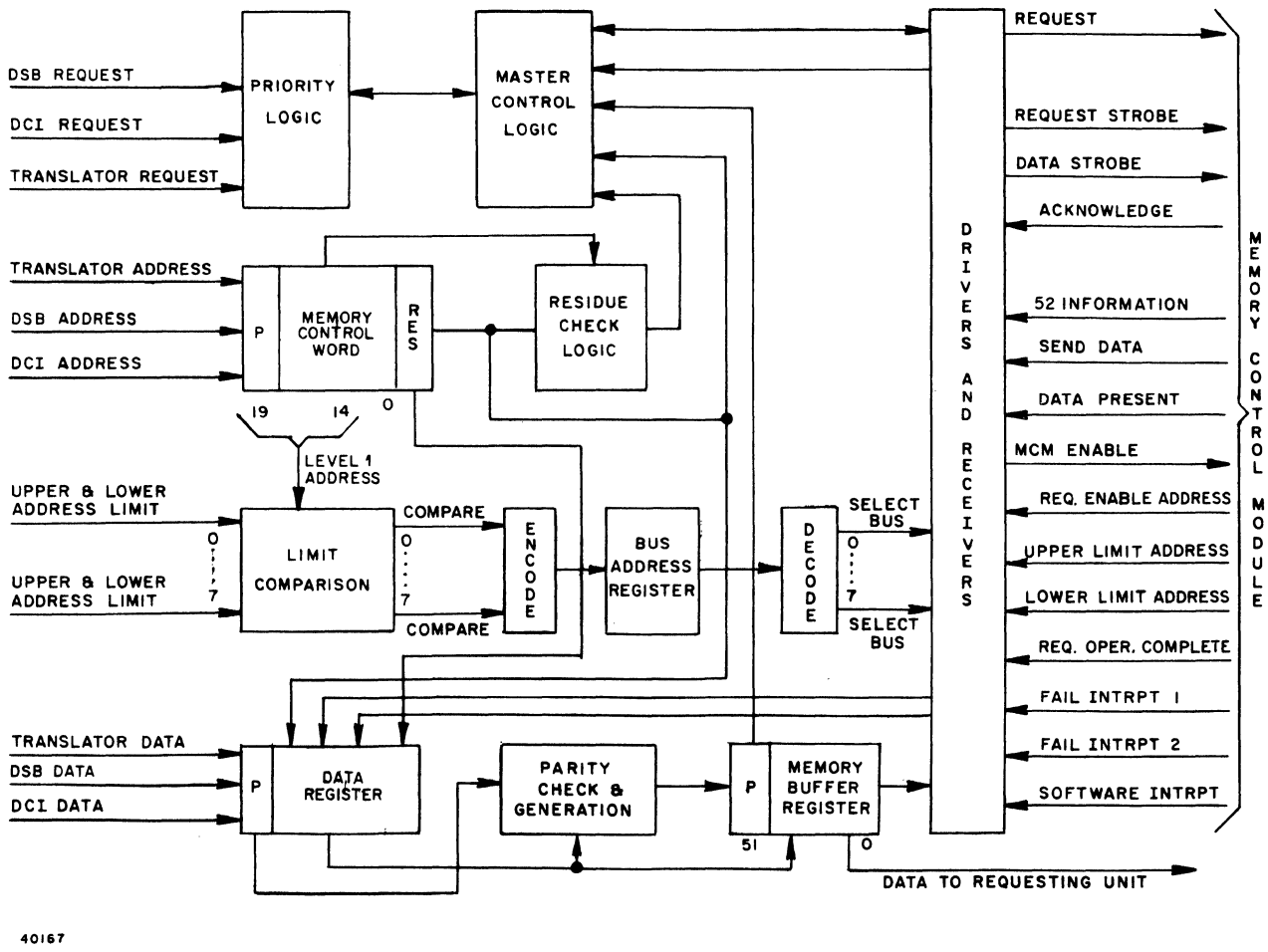


Figure 4-2-2. Memory Interface Unit

The MIU consists of nine functional components as illustrated in figure 4-2-2. These functional components are:

1. Priority Logic. This section is responsible for granting the services of the MIU to the highest priority requesting unit. The order of priority for services is:
   a. Data communications scan interface (DSI)
   b. Data service buffer (DSB)
   c. Translator.
2. Control Logic. This section contains the control logic necessary to execute all MIU operations, including the controls required to complete receiver and driver paths.
3. Residue Check Logic. This section is responsible for checking and verifying the residue bits of the memory addresses transferred from the translator and data service unity.
4. Parity Check and Generate Logic. This section is required to generate odd parity for all words being transferred to memory, and to check for odd parity of all words being fetched from memory.
5. Data Register. This is a 52-bit register and is used to buffer all data transfers between the requesting unit of the IOM and the MIU.
6. Memory Register. This is a 52-bit register and is used to buffer all input data to memory.
7. Control Word Register. This is a 25-bit register and is used to temporarily hold the 24-bit unit control word (UCW) and length transfers from the requesting unit. UCW is comprised of 20 bits of address, two residue bits, a write bit, and memory protect.
8. Receivers and Drivers. There are eight discrete groups of receiver and driver circuits in the MIU - one group per memory control module interface. The state of these groups is determined by the control logic; only one group is active at any one time.
9. Limit Comparison Logic. This section is responsible for comparing the six most significant bits of address in the CWR with the address limits supplied by each MCM in the memory system.
10. Bus Address Register. This register is used to buffer encoded output of the limit comparison logic. The output is decoded to select one of seven memory buses.

When a requesting unit of the IOM needs the MIU for a data transfer, it is required to raise its request lines to the MIU and place a 24-bit UCW and the length information on its interface lines to the MIU.

The MIU manages level-1 memory access requests by the functional units of the IOM on a preassigned priority basis. The access priority scheme for the functional units of the IOM is:

1. First priority: data communications processor interface requests.
2. Second priority: data service buffer requests.
3. Third priority: translator requests.

When a functional unit of the IOM requires the services of the MIU for the purpose of performing a data transfer, it must raise its access request line to the MIU and place a 26-bit unit control word (UCW) on its UCW lines to the MIU. When the requesting unit has priority, the MIU loads the UCW into its control word register and performs one of the following operations:

1. Single data word fetch
2. N-length data word fetch
3. Single-word overwrite with flashback
4. Single-word protected write
5. Single-word protected write with flashback
6. N-length overwrite
7. Single-word overwrite
8. N-word protected write

Upon determining the type of operation requested, the MIU constructs a memory control word (MCW) and transfers it to memory. Upon transferring the MCW to memory, the MIU is required to perform one of the following operations:

1. If a single-word store operation is specified: The MIU raises its request lines to the specified memory control module in order to alternately transmit the MCW and the data word to be stored to the addressed MCM. The MIU continues to transmit the MCW, followed by the data word to be stored, until an acknowledge signal is received from the MCM.
2. If a multiple-word store operation is specified: The MIU raises its request lines to the applicable MCM, and then sends the MCW to the MCM. When the MCM acknowledges receipt of the MCW, the MIU commences the data transfer under the control of the data request signal.
3. If a fetch operation is specified: The MIU raises its request lines and sends the MCW to the applicable MCM. When the MCM acknowledges receipt of the MCW, the MIU enables its memory-bus receiver circuits. Information from the MCM is now accepted by the MIU. However, the MCM is required to transmit a data present strobe pulse to the MIU to cause the information present on the memory bus to be transferred to, and detected by, the requesting IOM. The data present strobe pulse is required for each word transferred from memory to a requesting IOM.

While performing a data transfer, the MIU is required to detect and/or report memory error conditions. Memory errors are divided into two categories by the IOM: MIU-detected errors, and memory-detected errors. Memory errors cause termination of the memory request being processed, and the MIU sends a three-bit error code to the requesting section. The translator reports these errors through the

fail register. Data service section units return them in the result descriptor. A decode of these three bits specifies whether the error is MIU or memory-detected.

Errors detected and/or reported by the MIU and the associated three-bit reporting codes are listed in descending exclusive order as follows:

1. Store Disparity (011). This error condition is declared if a data transfer from an internal unit is received with incorrect parity by the MIU. The data with incorrect parity is transferred to the memory.
2. L1A Address Residue Error (010). This error condition is declared if the MIU receives a UCW with residue bits not agreeing with its memory address field configuration (DCP words are not residue checked).
3. Memory Detected Error (111). This error condition is declared when the addressed memory module responds with a fail 1 (uncorrectable error) indication to a requestor unit.
4. No Access to Memory (101). This error condition is declared if the MIU receives no response from the requestor memory module during a waiting period less than a 25-microsecond writing period. No response is defined as:
   a. Failure to receive, at the MIU, an acknowledged signal from an addressed memory module; or
   b. Incomplete data transfer by an addressed memory module.
5. Fetch Disparity (110). This error condition is declared if a fetch of data from memory is received by the MIU with incorrect parity.
6. Memory Protect Error (100). This error condition is declared when the addressed memory module responds with a protect error signal during a memory protect store operation.
7. Memory Detected Error (001). This error condition is declared when the addressed memory module responds with a fail 2 (one-bit corrected error) indication to a requestor unit. (This error condition does not cause termination of the memory access operation.)

DATA SERVICE BUFFER

The data service buffer (DSB), figure 4-2-3, is a unit within the IOM which allows a continuous peripheral I/O transfer to and from memory by providing a 256 x 52 word buffer. This buffer is divided into eight word buffers for each of the 28 data service unit (DSU) channels.

The DSB has five components, as illustrated in figure 4-2-3. These functional components are described as follows:

1. The request queue (RQ) consists of: a PCI and a DFI input register; read and write pointers; a 32 x 10-bit first-in/first-out request queue; M1 and M2 output registers (not shown); a parity generator/checker (not shown); and a valid request check circuit (not shown). The RQ is used to store, and initiate sequentially, memory requests from either the PCI or DFI, under control of central control.
2. The PCI and DFI arrays consist of the control, status, and valid arrays which contain information necessary to perform various PCI/DSB, DFI/DSB, and central control operations.
   a. The PCI control array consists of 20 x 8 bit locations and the DFI control array consists of 8 x 8 bit locations, one location for each of the 28 DSU channels.
   b. The PCI status array contains 20 x 5 bit locations and the DFI status array has 8 x 5 bit locations.
   c. The PCI valid array uses a 20 x 2 bit location array and the DFI valid array is formed by a 8 x 2 bit array. These determine if an invalid buffer is accessed.
3. The address array has eight functional units:
   a. PCI and DFI UCW Registers. These are 24-bit registers used to buffer unit control words (UCW) from the PCI and DFI into the address array.
   b. Residue Check Logic. This section check and verifies residue of the UCW address.
   c. Address Array. This array consists of 32 x 24 bit word locations, one for each channel of the PCI and DFI, with four locations not used. The address array is used to store the next UCW for the MIU from each channel, while waiting for the RQ to honor the request.
   d. Address Buffer. This is a 24-bit register (not shown) used to temporarily hold the data being read out of the address array and transferred to the UCW register.
   e. 20-Bit Adder/Subtractor. This section (not shown) updates the current address in the unit control word for the next fetch or store operation.
   f. Residue Adder/Subtractor. This logic (not shown) is responsible for generating new residue for all updated addresses being sent to either the UCW register or the address update register.
   g. Unit Control Register. This register is a 24-bit register and is used to buffer the UCW to the MIU.
   h. Address Update Register. This is a 24-bit register and is used to buffer the updated UCW address and residue to the address array.
4. Address logic for DSB main buffer is comprised of eight address lines. These address lines are enable by decoding of three address input gates selecting a particular DSB location.
5. Address logic for the four-word buffers M1 and M2 consists of a three-bit M1 and M2 length counter and associated gate logic. The two least significant bits select which of the four words is addressed.

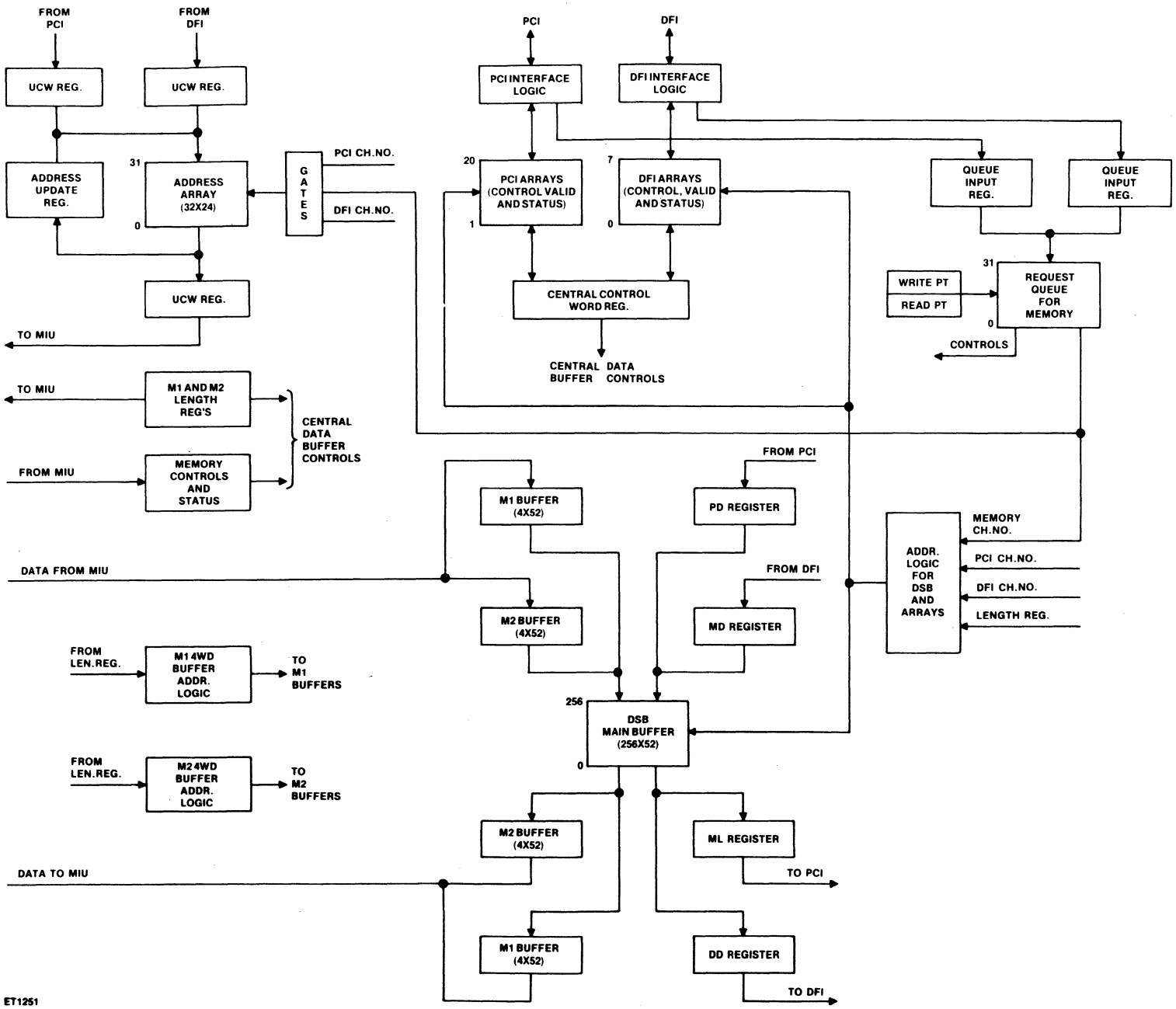


Figure 4-2-3. Data Service Buffer

To control the peripheral I/O transfers, the DSB contains four independently operated controls which are initiated when a request for a particular operation is received. The following subparagraphs describe each DSB control.

PCI/DSB Control

PCSI/DSB control is initiated when a PCI request for PCI control, valid, and status arrays is set in response to a request from the PCI. It is used to control one-word transfers between the PCI and the DSB main buffer.

DFI/DSB Control

DFI/DSB control is initiated when a DFI request for the DFI control, valid, and status arrays is set in response to a unit control word inhibit signal from the DFI. This section controls one or two-word transfers between the DFI and the DSB main buffer.

Because PCI timing and PCI/DSB timing control are one clock cycle ahead of the DFI timing sequence, the PCI request for PCI control, valid, and status arrays always occurs before the DFI request and thus eliminates any request conflicts between the PCI and the DFI.

However, only DFI-A or DFI-B can request the use of the DFI arrays at the same time, by a requesting DFI generating an inhibit signal to inhibit the other DFI section.

Central Control

Central control is initiated when a request for a memory operation is available in the request queue. This request is made as soon as the count in the read and write pointers of the RQ is not equal and the queue is not being written into, thus enabling a memory request. The central control is used to control data transfers between the DSB main buffer and the M1 and M2 four-word buffers.

When the PCI or a DFI is requesting the use of its control array or the address array, central control can not request these arrays for one clock cycle while the PCI or DFI request is acknowledged. Also, if the PCI or DFI requests the use of the DSB main buffer and central control has access to the buffer, the central control loses access for one clock cycle while the data transfer occurs.

MIU/DSB Control

MIU/DSB control begins when central control requests an MIU interface operation. THE MIU/DSB control then responds by sending a data request, along with the UCW and length, to the MIU. The MIU/DSB control regulates data transfers between the M1 and M2 four-word buffers and the MIU.

PERIPHERAL CONTROL INTERFACE

The peripheral control interface (PCI) (figure 4-2-4) lets the IOM interface with from one to 20 peripheral controllers (PCs) and coordinates data transfers between the PCs and the DSB as directed by the translator section of the IOM. The PCI interfaces with memory by one-word transfers via the MIU through the DSB.

The PCI consists of nine functional components (see figure 4-2-4). These functional components are described as follows:

1. PCI Local Memory (PCLM). Comprised of TTL RAM and CTL MULN-type memory chips. PCLM stores the channel descriptors, result descriptors, and data for 20 channels.
2. Descriptor Register (DR). A 79-bit register used to temporarily store the channel descriptor for a job being initiated and the result descriptor while it is being assembled.
3. Shift Logic and Data Buffer Register. The shift logic consists of left and right shift logic to properly position data within the data buffer (DB) register. The DB register can be considered as part of the descriptor register because it holds the CDL and data words for the channel descriptor being processed.
4. Byte Buffer. A 16-bit register used to temporarily store the data byte being transferred between the peripheral control and the PCI, as well as the device result descriptor being transferred from the peripheral control.
5. Code Translator. Consists of the logic circuits necessary to perform various code translations required by some B 7800 peripheral devices. ROM chips are used to perform this function.
6. Stack Section. Contains 20 three-bit stack registers, one for each of the peripheral controls. The three-bit codes are used to address the PCLM in order to perform specific PCI operations, such as writing an updated channel descriptor back into the PCLM for the next service cycle.
7. Queue Section. Contains 20 two-bit queue registers, one per PC. The queue register contains any of three two-bit job codes: initiate, memory, and terminate.
8. Driver and Receiver Section. Has the drivers and receivers necessary to transmit and receive data and control signals between peripheral controls connected to the 20 operational channels.
9. Timing Generator and Control Section. Has the logic used to generate and control timing signals used throughout the logic circuits of the PCI.

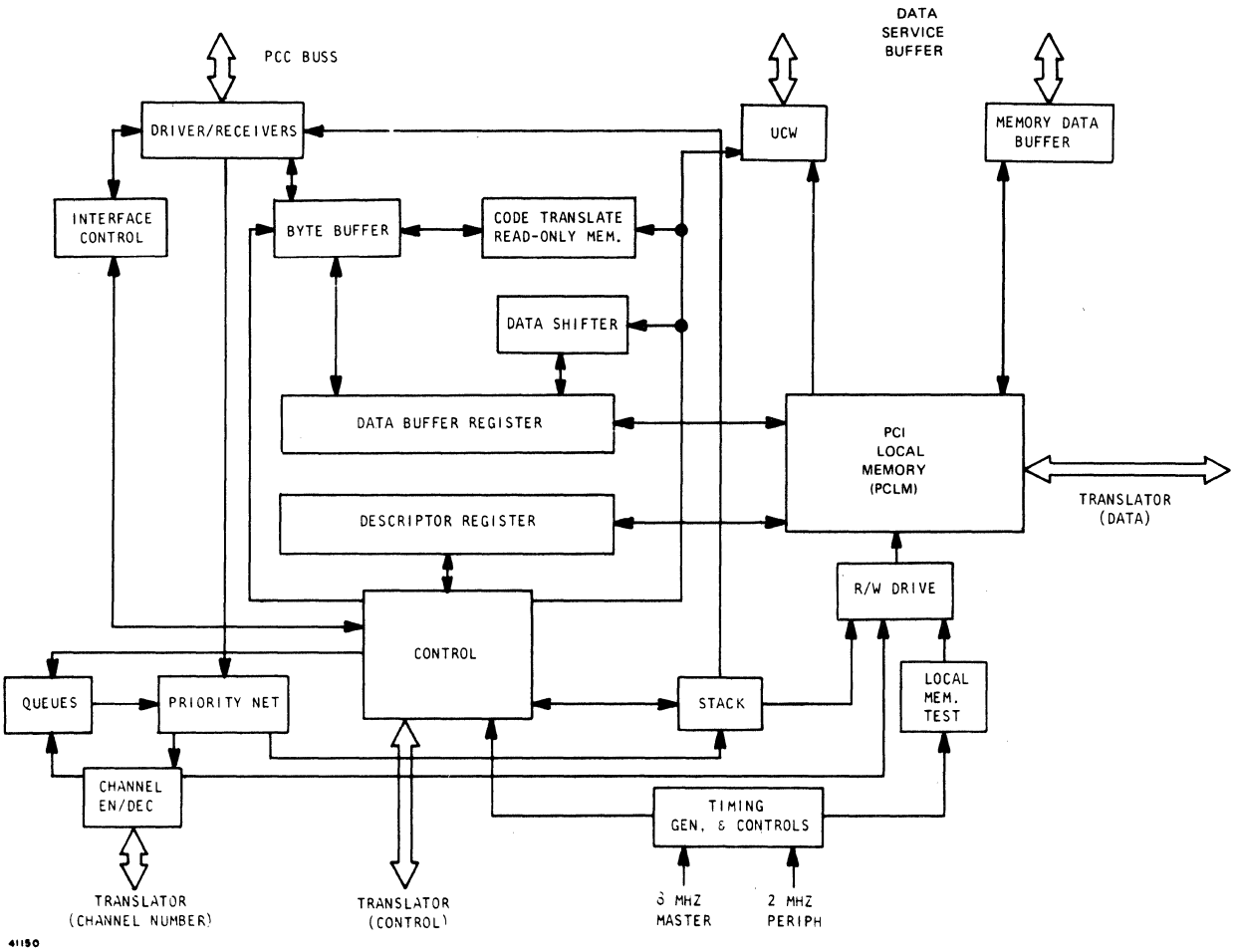


Figure 4-2-4. Peripheral Control Interface

Each PC needs a one-microsecond service cycle to transfer data. By means of overlapping service cycles, and by use of local memory windows (a one-clock period when a particular operation may be performed if no higher priority job exists) all 20 channels are multiplexed.

There are five operational phases of the PCI:

Translator Service.
Channel designate.
Channel data service.
Memory operations.
Result descriptor read.

Each of these phases is described as follows.

Translator Service

This phase of PCI operation is controlled by channel designate level (CDL) control (referred to as CC), and includes all functions required between the PCI and the translator during the initiation phase of an I/O operation. The translator service phase commences when the translator control logic places the first job descriptor word (DSU word) on the translator bus and raises the request line to the PCI. When a local memory window becomes available, the CC strobes the DSU word into the local memory channel allocated to the device to be started, and lowers the PCI-available line to the translator. The translator control logic then lowers the request line and places the second job descriptor word (CDL word) onto the translator bus.

When the next available local memory window occurs, the CC strobes the CDL word into the appropriate location in local memory, strobes the channel number of the new request into the initiate queue stack (INQ), and raises the PCI-available line. This informs the translator that the entire request descriptor has been received and stored in local memory.

Channel Designate

If no channel is currently being initiated, the CC selects the highest priority channel in the INQ, sends this channel number to the CDL stack (CDS) which contains the request descriptor currently being initiated, and resets the INS bit for the selected channel. If no channel requires channel service, the CC checks the busy line of the channel.

If a not busy condition is detected, the CC commences transfer of CDL characters to the appropriate PC at the rate of one character per available service cycle. If the selected channel is busy at initiation, or if the selected channel becomes not busy during transmittal of CDL characters, the request is terminated and appropriate result descriptor information is generated and transferred to the translator (see channel termination). After sending the correct number of CDL characters (four for standard devices and eight for disk file devices), the CC raises the start channel bus line to the PC and resets the CDS, and the initiation operations are completed.

Channel Data Service

After completion of the initiation phase, a channel is serviced upon demand at a rate dependent upon the type of peripheral device involved. The PC requests service by raising the access-request line (ARL) to the PCI. The PCI selects the highest priority channel requesting service and generates the appropriate access granted level (AGL). The presence of this signal grants the next service cycle to the accessed peripheral. The service cycle consists of two T-time periods (T1 and T2) of 500 nanoseconds each; T1 is used for output to the PC. The AGL signal for the next service cycle is generated during the previous service cycle's T2 time period.

Each data transfer is controlled by a channel descriptor which has been generated from information contained in the DSU word of the job descriptor.

If an error is detected at any time during channel service, the PCI generates the appropriate result descriptor information for the translator (see channel termination) and terminates all operations on that channel.

Memory Operations

When the PCI determines that one 52-bit data word is needed from, or is ready to go to memory, the channel number for the transfer goes in the memory queue (MQ). The MQ is a stack which contains the channel numbers of all channels requiring memory access. If no memory operation is currently in progress, the PCI selects the highest priority job in the MQ, and transfers this number to the memory operation stack (MOS). The PCI then resets the MQ bit for the selected channel, transfers the unit control word to the unit control register (UCWR), raises the PCI-memory-request line to the DSB, and, if necessary, transfers data into the memory transfer area (MTA).

Once access to the DSB is granted, the PCI strobe: 1) fetches data from the MTA; 2) strobes the data to the appropriate data buffer in local memory; 3) awaits the release signal which indicates that this memory requests is completed; and 4) resets the MOS bit. The memory operation is thus completed.

If the DSB detects an error at any time during this sequence, the error information is transferred to the PCI. The PCI then causes the request to be terminated, and an appropriate result descriptor is generated.

Result Descriptor Read

After completing the required data transfer, the PCI sends an I/O complete level to the PC. The PC then returns the result descriptor available level and returns a result descriptor. This result descriptor information, plus the channel descriptor information, is used by the PCI to create the result descriptor word to bring about a normal termination.

Abnormal termination result descriptors can occur:

1. During initiation, channel service, or channel memory operations (when errors are detected by the PCI).
2. During channel memory operation (when errors are detected by either the MIU or the MCM).
3. During channel initiation or channel service (when errors are detected by the PC).

No matter what the source, all result descriptors are treated identically.

Once the result descriptor has been generated, it is stored in the local memory location of the channel to be terminated. The channel number of this request is strobed to the translator. Should the translator be unable to accept the channel number, the PCI stores this channel information in the termination queue (TMQ), which contains all the requests to be terminated. Whenever possible, the PCI selects the highest priority request from this stack, transmits the channel number to the translator, and resets the TMQ bit for the selected channel.

The translator replies to the PCI termination with a read-result-descriptor request, which causes the result word to be placed on the translator bus. This completes the termination operation.

DISK FILE INTERFACE UNIT

The disk file interface unit (DFI) in figure 4-2-5 lets the IOM interface up to eight disk file controls (DFCs). It consists of two independent modular sections, each of which is capable of handling four data channels. Each data channel is interfaced to one DFC.

Each DFI section controls data transfers with the DFCs via a 16-bit data bus, at a transfer rate of two eight-bit characters per transfer time. The transfer rate to the DSB is two words (2 x 48 bits) per transfer time.

Each data channel utilizes a four-word data buffer area, the data local memory (LMD), and a 67-bit channel descriptor local memory (LMC). Upon command from the translator, the DFI initiates requests with its associated DFCs. On receipt from memory of a disk file job, the translator requests that a pair of job words be sent to the DFI section assigned to handle that job. The selected DFI section loads the first of the new requests words into the proper channel descriptor location in the LMC. It loads the second job request word into its data buffer area (LMD), from which this word is subsequently sent to the DFC. The DFI then releases the translator.

During data transfer operations, the DFI communicates with the data service buffer (DSB) to get memory accesses. All data transferred by the DFI between the DFCs and memory is temporarily stored in the LMD. Here individual 16-bit bytes are packed or unpacked as they are exchanged with the DFCs.

Figure 4-2-5 illustrates the two DFI sections and their respective interfaces with the translator, DSB, and peripheral control cabinets (PCC). The two sections are identical and contain the following components:

1. Channel Descriptor Local Memory (LMC). Stores four 69-bit channel descriptors, one for each DFC data channel.
2. Descriptor Register (DR). Holds the descriptor of an active channel. The DR contents are used in conjunction with update logic to update the current memory address of the active job and other various control bits.
3. Update Logic. Increases the CA by two words each time a memory access is requested for fetching or storing a pair of data words. It is also used to update the word byte position field (WBP), the residue and phase (FAZ) fields, and various other control bits.
4. Parity Check and Generate Logic. Generates and stores odd parity for each descriptor stored in the LMC, and checks for odd parity on all descriptors read from the LMC.
5. Data Local Memory (LMD). Stores 16 48-bit data words (two double-word locations for each of the four DFC channels). The LMD buffers for data read from, or written onto, disk files. Also, during request initiation, the LMD contains six CDL characters (48 bits total) which are sent to the DFC.
6. Two-Word Buffer (2WB). Acts as a buffer for data being transferred between the LMD and the DSB; can store two 48-bit words.

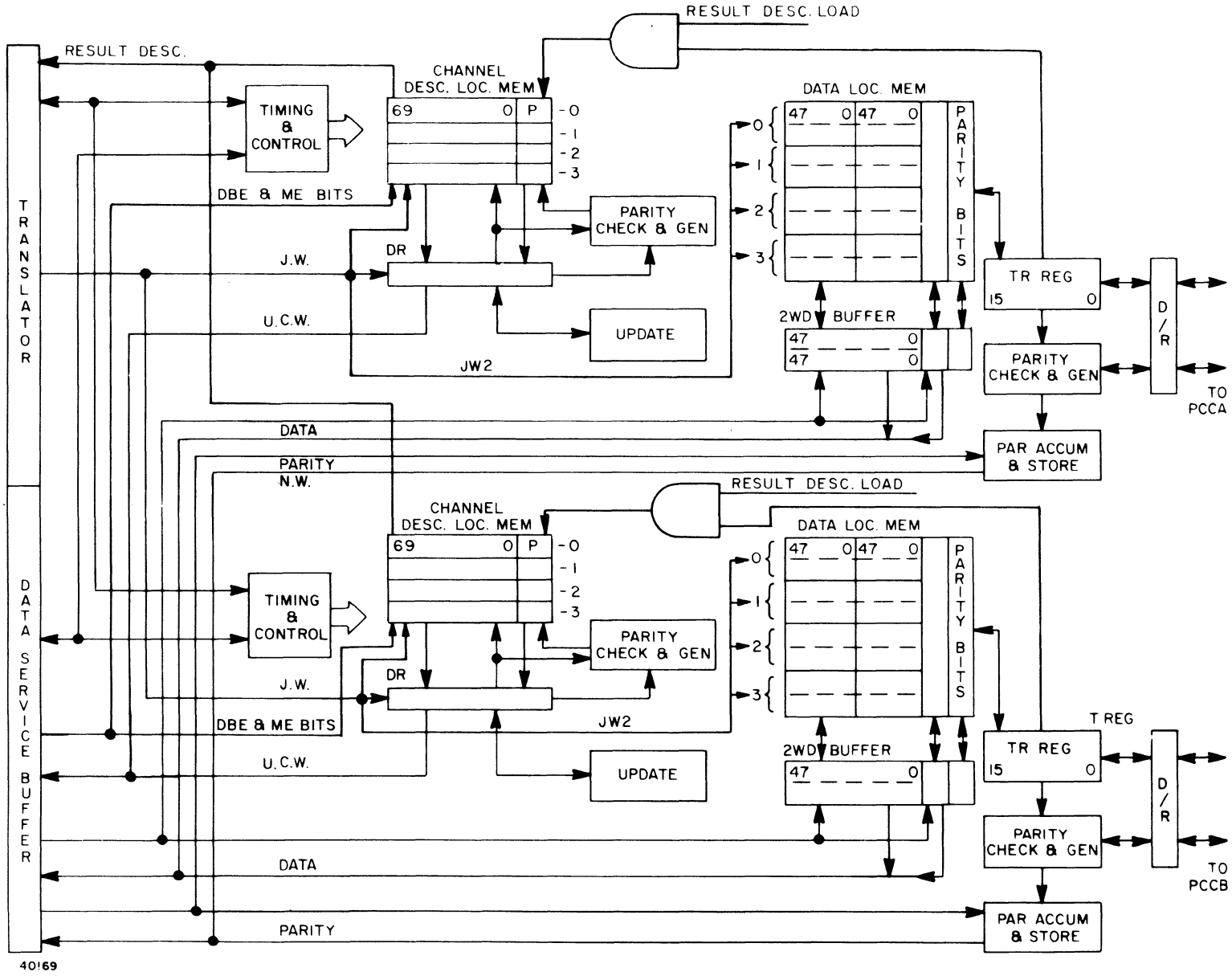


Figure 4-2-5. Disk File Interface

7. Transfer Register (TR). A 16-bit register, used to buffer all data transfers to or from the DFC.

8. Parity Check, Generate, Accumulate, and Store. Checks and generates parity on data transferred from or to the DFC. When data is sent to the DFC, a parity bit (from the MIU, via the DSB) is received with each data word and is stored in the accumulator. The parity bit setting of the accumulator is updated with each 16-bit transfer to the DFC, and is checked against the priority of the last such transfer. When data is received from the DFC, a parity bit is received with each data transfer and is stored in the accumulator. The parity bit setting is updated for each new 16-bit transfer, and a final setting is sent to the MIU, via the DSB, for each new data word. The MIU then checks parity on the full 48-bit word (three 16-bit transfers per 48-bit word).

During normal operation, the DFI section, according to priority, reads the descriptors of its assigned channels in search of an active job to perform. A channel may be in any one of four basic phases of operation. The exact phase is determined by the FAZ field in the channel descriptor. The four basic operations are:

Channel Initiate.
Channel Designate.
Data Service.
Channel Termination.

Upon completion of an operation, the DFI notifies the translator of the termination status and then awaits reinitiation.

Channel and DSB Initiation Operation

The DFI channel initiation operation begins with the receipt of a translator request. When the DFI is ready to service the request for a specified channel, the DFI acknowledges the request. The two job words are now received from the translator.

The portions of the channel descriptor not in the descriptor register (DR) are written directly into the LMC from the job word data lines. The other portions of job word 1 are loaded into the descriptor register. Also, the DFI Busy is enabled.

Next, the first half of job word 2 is loaded into word 3 of the LMD to form the first three CDL characters. Then digits D7-D12 of job word 2 are written into word 4 of the LMD. The JW2 digits are distributed over two LMD words to place the CDL characters in separate bytes. This facilitates the subsequent transfers of these characters to the DFC during the CDL cycle.

Following the above, the DFI is marked not busy, and the channel descriptor is read from LMC into the DR. If errors are detected, a result descriptor load request occurs, the proper code is loaded to the RD, and the channel terminates.

Otherwise, the DFI sends a unit control word (UCW) to the DSB, if the channel job is memory operation, the channel is not already waiting for completion of DSB initialization, and the DSB is not busy with the other DFI. If the DSB is busy, the next time this initiating channel is serviced, the DSB request is attempted again. When the initiating channel is again selected, the UCW is again sent to the DSB. If the UCW is received by the DSB and if the memory operation is a read from disk (store to memory), the phase count is updated to allow the release for the memory operation which is expected from the DSB.

If the memory operation is a write to disk (fetch from memory), the channel job waits until four words from memory are loaded into the DSB main memory. Because delays vary in length in accordance with MIU/MCM access times, the channel must wait so that the data is present for transfer to the device. Meanwhile, the DFI is informed of the DSB release of the channel, allowing other channels to send data to the DSB, while this channel is waiting. Once the four words for this channel are stored in the DSB main storage, the DSB signals the DFI which allows this channel to proceed to the next phase.

Channel Designate Operation

In the channel designate operation (CDL cycle), a control word of 48 bits of information is sent to the addressed DFC channel. The overall channel designate sequence requires eight channel services, consisting of six transfers of eight bits each, and two dummy CDL cycles (service cycles four and five). These 48 information bits are contained in the second request word (job word 2) stored in the LMD.

The word byte pointer (WBP) field of the channel descriptor points to the location in the LMD of the information to be transferred to the DFC. A 16-bit byte, of which only the eight most significant bits are valid, is loaded into the transfer (TR) and is then sent to the DFC. After six such transfers, and two dummy CDL cycles in the middle of the six transfers, a start channel bus signal is sent to the DFC to indicate the end of initiation phase, and the phase field of the channel descriptor is then updated to reflect this.

If the job word indicates an output operation (Memory Read), two fetches of two words each are

requested from memory via the DSB during channel designate operation. The first of these two word fetches is stored in position 00 and 01 of the LMD. The second two word fetch is stored in locations 10 and 11 of the LMD.

Data Service Operation

The data service consists of transferring a 16-bit byte of information to or from the DFC. If the DFC corresponding to the active channel has raised its access request level (ARL), the DFI responds by raising its access granted level (AGL). The DFI will now follow this AGL by granting a one microsecond service cycle. A single 16-bit data byte is transferred for each data service operation.

If the disk file operation is an input operation, the data byte is accepted from the DFC into the transfer register (LMD). If the disk file transfer is an output operation, the data byte is read from the LMD, placed in the transfer register (TR), and applied to the data bus to the DFC.

The LMD location of the 16-bit data byte to be transferred is determined by the word byte pointer (WBP) field of the channel descriptor. With each data byte transfer, WBP is upcounted by one.

After two words are transferred, a memory request is again made via the DSB. The current address (CA) is sent, along with certain control bits, to the DSB. If the request is for a disk read, two words are loaded into the two-word buffer (2WB) from the LMD. If the job request is for a disk write, two words from the DSB are loaded into the 2WB and then into the LMD.

With each DSB request, the current memory address (CA) is counted up and compared to the final memory address (FA). If they are equal, the data service phase is completed and the phase field is incremented by one to advance the channel operation for normal termination. The channel descriptor is then restored into the LMC and the next channel is serviced.

Channel Termination Operations

The channel termination operation consists of two parts: the end of data service and the channel termination interface with the translator.

After data service has been terminated, the DFI sends the terminating channel number to the translator and then awaits the receipt of a read-result-word-request (RDR) from the translator. When this request is received at the DFI, the result descriptor is sent to the translator and the channel termination phase is completed.

The data service operation ends when a normal completion of a data transfer has occurred, or by the detection of an error by either the DFI or the DFC. If the DFI detects the end of data (by CA = FA) or an error, it sends an IOC signal to the DFC. When the DFC detects either an IOC or an error, it sends a result descriptor to the DFI. This result in stored in the FA field of the channel descriptor.

SCAN BUS INTERFACE

The SCI (figure 4-2-6) contains the storage and controls required to provide a scan bus for communication with four DCPs.

The translator initiates scan operations by transmitting a scan control word to the SCI. If a scan-out is required, the translator also transmits the scan-out information to the SCI. Upon completion of the scan operation by the SCI, the translator is notified. In the case of a scan-in operation, the scan-in information is loaded into the translator B register. If an error has been detected by the SCI, error information is loaded into the translator F register.

There are two error conditions which can be reported to the translator by the SCI:

1. Not Ready. If the DCP addressed by the scan bus does not respond with a ready signal within 3 usec, a not ready error is reported.
2. Module Error. If the DCP addressed by the scan bus detects an error on a scan-out or scan-in operation, an error signal is transmitted to the SCI. The SCI then reports a scan error to the translator.

During scan-out operations (only scan-out orders are accepted by a DCP), the scan information lines constitute the scan-out word. The SCI provides a maximum of four DCP memory interfaces (see figure 4-2-7) in a DCI unit.

The DCI unit contains all storage capability and controls required to interface with the DCP memory buses. Memory transfer operations performed are:

1. Fetch (one word)
2. Store with flashback (one word)
3. Protected store with flashback (one word).

If an interface is not used by a DCP, it may be used to accommodate a suitable device.

All errors detected by the DCI or MIU for a DCI memory request go to the DCP which initiated the memory request.

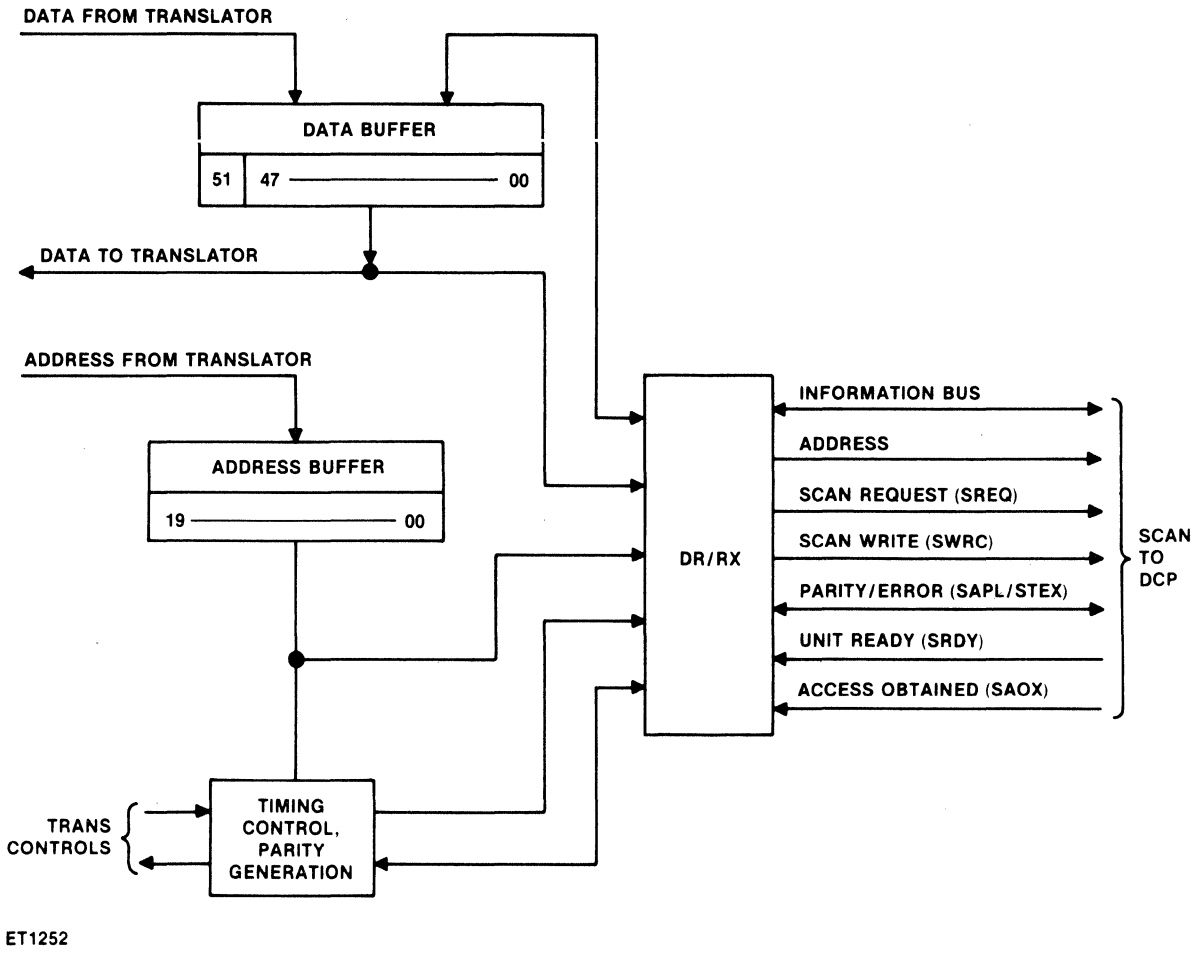


Figure 4-2-6. Scan Bus Interface

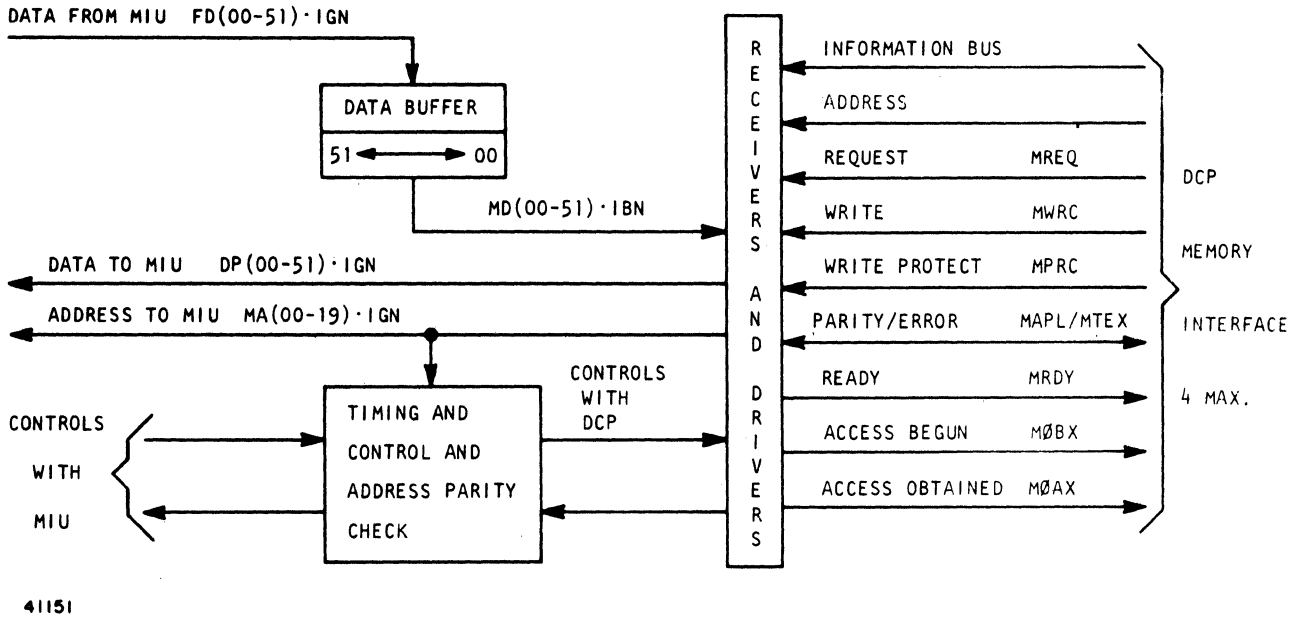


Figure 4-2-7. Data Communications Interface Unit

SECTION 3

PERIPHERAL AND CONTROL WORD FORMATS

INPUT/OUTPUT CONTROL WORD (IOCW)

	LINK 47	MINH 43	B/F 39	35	31	27	23	19	15	11	7	3
0 50	ASC II 46	TRA 42	TEST 38	34	30	26	22	18	14	10	6	2
0 49	SA 45	FML 41	TC 37	33	29	25	21	17	13	9	5	1
0 48	I/O 44	MP 40	TL 36	32	28	24	20	16	12	8	4	0

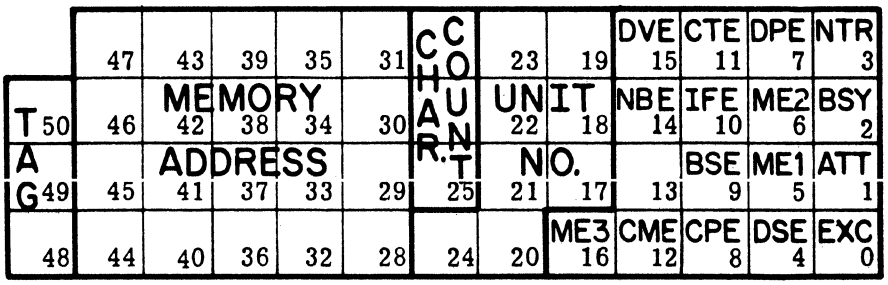
JOB WORD 1 (JBW1) TO DSU

	LINK 47	MINH 43	CT 39	35	31	27	23	19	15	11	7	3
B/F 50	ASCII 46	TRA 42	EXT 38	FINAL 34	30	26	22	18	START 14	10	6	2
TC 49	SA 45	FML 41	T 37	ADDRESS 33	29	25	21	17	ADDRESS 13	9	5	1
TL 48	I/O 44	MP 40	36	32	28	24	20	16	12	8	4	0

FIELD	IOCW BIT S	JBW1 BIT S	DESCRIPTION
LINK	47:1	47:1	SIDELINK. When set, indicates a sidelinked I/O is to be performed. Info about sidelink operation is in second word of IOCB.
ASCII	46:1	46:1	ASCII.
SA	45:1	45:1	SOFTWARE ATTENTION. When set, causes ATT bit in result descriptor to be set.
I/O	44:1	44:1	INPUT/OUTPUT. 1 = READ; 0 = WRITE
MINH	43:1	43:1	MEMORY INHIBIT.
TRA	42:1	42:1	TRANSLATE. Settings of ASCII, I/O, TRA and FML bits, taken together, determine what data translation, if any, is to be done.
FML	41:1	41:1	FRAME LENGTH. 1 = 8 bits; 0 = 6 bits.
MP	40:1	40:1	MEMORY PROTECT.
B/F	39:1	50:1	BACKWARD/FORWARD. 1 = BKWD; 0 = FWD.
TEST	38:1	-	TEST.
TCTL	37:2	49:2	TAG CONTROL. 0 = Store SP tags (0); 1 = Store program tags (3); 2 = Tag field transfer; 3 = Store DP tags (2).
CT EXT	-	39:3	COUNT EXTENSION. Number of characters in fractional word part of data buffer.
FINAL ADDRESS	-	36:17	Memory address of last full word of buffer to be accessed by IOM, low 17 bits only.
START ADDRESS	-	19:20	Memory address of first full word of buffer to be referenced; for a backward tape operation, this is *not* the buffer base address.

Job word 1 is built in the IOM and can be read from the B register (Panel 1, Row 3) in T-time 25 of start mode. Job word 2 (JBW2) passed to the data service unit is (in all cases) the channel designate level (CDL) word from the IOCB. The following are CDL word formats for the various peripheral types:

STANDARD RESULT DESCRIPTOR



Field	BITS	DESCRIPTION
TAG	50:3	0 = Normal; 4 = IOM unable to do sidelinked job.
MEMORY ADDRESS	47:20	Final level 1 memory address (L1A). Hard load RD contains channel used in [32:5].
CHAR COUNT	27:3	No. of last memory word characters validly executed by IOM.
UNIT NO	24:8	Unit No. of device on which job was executed. (Set only for mapped I/O).
ME3,ME2, ME1	16:1 6:1 5:1	MEMORY RELATED ERRORS. Listed in order of decreasing priority.

ME3	ME2	ME1	IOM*	
0	1	1	PER	STORE DISPARITY.
0	1	0	RAE	L1A ADDRESS-RESIDUE ERROR.
1	1	1	F1R	MCM-DETECTED ERROR, FAIL 1 (UNCORRECTABLE).
1	0	1	NOA	NO ACCESS TO MEMORY.
1	1	0	PER	FETCH DISPARITY.
1	0	0	RS1	MEMORY-PROTECT ERROR.
0	0	1	F2R	MCM-DETECTED ERROR, FAIL 2 (CORRECTABLE).

*Panel Light indication in Error Control Register (Panel 2, Row 7).

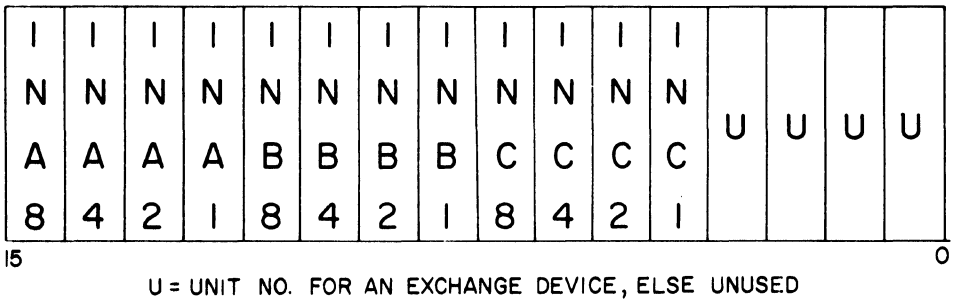
Unit or DSU error bits	15:9	Depends on value of bit 4 (DSE). 0 = unit related errors (in bits 15:9); see below. 1 = DSU error (in bits 15:9), as follows: DVE 15:1 DEVICE-DETECTED ERROR. NBE 14:1 NOT BUSY ERROR. CME 12:1 COMBINATION ERROR. CTE 11:1 COUNTER ERROR. IFE 10:1 INTERFACE ERROR. BSE 9:1 BUS PARITY ERROR. CPE 8:1 CONTROL PARITY ERROR. DPE 7:1 DATA PARITY ERROR (Always set if BSE=1).
DSE	4:1	DATA SERVICE ERROR. 1 = DSU error (in bits 15:9); 0 = Unit related error (in bits 15:9).
NTR	3:1	NOT READY.
BSY	2:1	CHANNEL BUSY ON INITIATE.
ATT	1:1	SOFTWARE ATTENTION.
EXC	0:1	EXCEPTION.

Table 4-3-1. MOD II IOM Data Service Buffer Errors

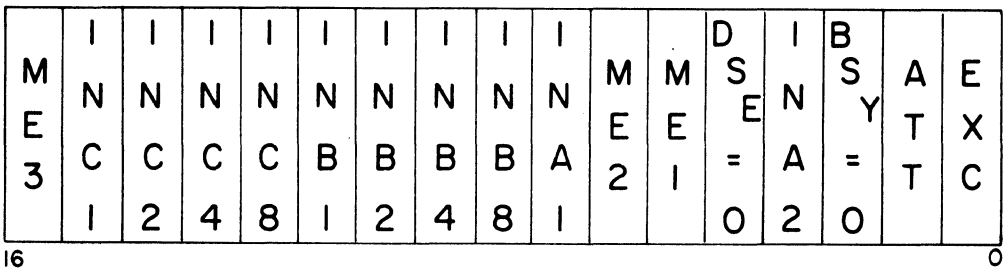
DSE	DBE	ME3	ME2	ME1	
1	1	0	0	0	Illegal data buffer
1	1	0	0	1	No access to data buffer
1	1	0	1	1	Control array Parity
1	1	1	0	0	Address residue error
1	1	1	0	1	Request queue parity
1	1	1	1	0	No response by DSB

Unit Related Errors

The device RD for any type of unit is returned to the IOM as three hexadecimal digits from the control. For a PCI operation, the device RD is in the PCI byte buffer (Panel 1, Row 14), as shown below, when the operation completes.



The INA8 bit should always be on, signifying operation complete. A unit error is signified by INA4=1, and the remaining bits in the device RD are used to make up the RD error field reported to the MCP as shown:



For each PCI channel, the last RD returned to the MCP may be displayed in the RD register (Panel 1, Row 34) of the IOM via local memory operations. Similarly, the last RD from each DFI channel may be displayed (in slightly modified form) in the DFI RD register (Panel 1, Row 10).

Result Descriptors Common To All Peripheral Devices

0003	ATT (Software Attention)
0005	BSY
0009	NOT READY
000D	WRONG LENGTH DATA TRANSFER (Generated by MCP)
0015	BUS PARITY (Reported by MCP for hardware RD=0291)
0021	MEMORY FAIL 2 (Correctable)
0041	RESIDUE ERROR
0061	STORE DISPARITY
0291	BUS PARITY (Changed to 0015 by MCP)
10001	MEMORY PROTECT ERROR
10021	NO ACCESS TO MEMORY
10041	FETCH DISPARITY
10061	MEMORY FAIL 1 (Uncorrectable)

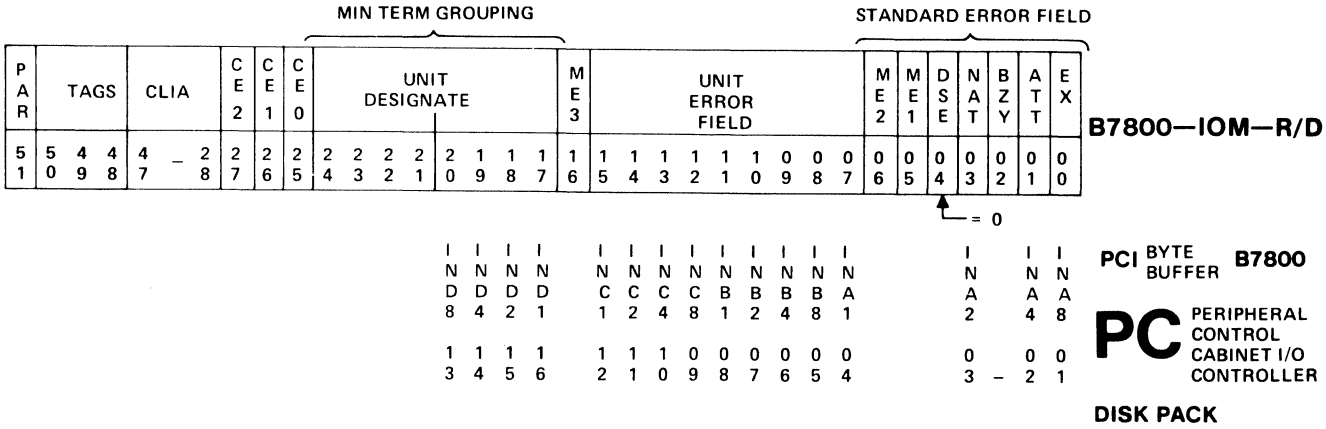
Any result descriptor may also have ATT (bit 1) set.

Internal DSU Error Result Descriptors*

0091	DPE	1591	CME, IFE, CPE, DPE
0111	CPE	1691	CME, IFE, BSE
0191	CPE, DPE	1791	CME, IFE, BSE, CPE
0291	BSE	1811	CME, CTE
0391	BSE, CPE	1891	CME, CTE, DPE
0411	IFE	1911	CME, CTE, CPE
0491	IFE, DPE	1991	CME, CTE, CPE, DPE
0511	IFE, CPE	1A91	CME, CTE, BSE
0591	IFE, CPE, DPE	1B91	CME, CTE, BSE, CPE
0691	IFE, BSE	1C11	CME, CTE, IFE
0791	IFE, BSE, CPE	1C91	CME, CTE, IFE, DPE
0811	CTE	1D11	CME, CTE, IFE, CPE
0891	CTE, DPE	1D91	CME, CTE, IFE, CPE, DPE
0911	CTE, CPE	1E91	CME, CTE, IFE, BSE
0991	CTE, CPE, DPE	1F91	CME, CTE, IFE, BSE, CPE
0A91	CTE, BSE	4011	NBE
0B91	CTE, BSE, CPE	4091	NBE, DPE
0C11	CTE, IFE	4111	NBE, CPE
0C91	CTE, IFE, DPE	4191	NBE, CPE, DPE
0D11	CTE, IFE, CPE	4811	NBE, CTE
0E91	CTE, IFE, BSE	4891	NBE, CTE, DPE
0F91	CTE, IFE, BSE, CPE	4911	NBE, CTE, CPE
1011	CME	4991	NBE, CTE, CPE, DPE
1091	CME, DPE	5011	NBE, CME
1111	CME, CPE	5091	NBE, CME, DPE
1191	CME, CPE, DPE	5111	NBE, CME, CPE
1291	CME, BSE	5191	NBE, CME, CPE, DPE
1391	CME, BSE, CPE	5811	NBE, CME, CTE
1411	CME, IFE	5891	NBE, CME, CTE, DPE
1491	CME, IFE, DPE	5911	NBE, CME, CTE, CPE
1511	CME, IFE, CPE	5991	NBE, CME, CTE, CPE, DPE

* These DSU error result descriptors may end in 3, indicating software attention. Result descriptors in the form 0xxx may also be 8xxx and those in the form 1xxx may also be 9xxx, indicating DVE.

IOM PERIPHERAL RESULT DESCRIPTOR



RESULT DESCRIPTOR LOCATIONS

SCAN R/D @ HA+1 (BIT 42:1=PAR ERR; BIT 43:1=NOT RDY)
COLD START or FAIL in FM RD @ HA+4
SYNC I/O R/D @ HA+5
START I/O @ IOCB+5

ET1268

RESULT DESCRIPTOR LOCATIONS

SCAN R/D @ HA+1 (BIT 42:1=PAR ERR; BIT 43:1=NOT RDY)
COLD START or FAIL in FM→RD @ HA+4
SYNC I/O R/D @ HA+5
START I/O @ IOCB+5

CARD READER

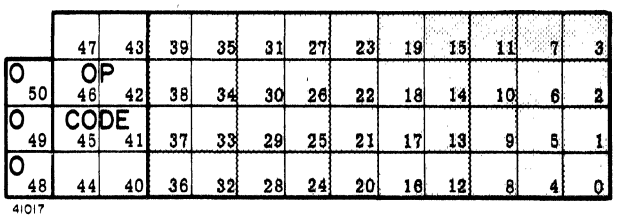
The B 7110 card reader control can be used with either B 9111 (800 cpm) or B 9112 (1400 cpm) card readers. The input hopper and the output stacker have a capacity of 2400 cards each. The card readers accept alpha, binary, or EBCDIC card codes. The card reader converts alpha card code to BCL, which is then converted into internal BCL or EBCDIC by translators in the I/O processor. EBCDIC card code is converted to internal EBCDIC by the B 7110 card reader control. When binary punched cards are read, no translation is made.

The card readers can read 51-, 60-, or 80-column punched cards. Optional features include the ability to read 40-column treasury checks and round holes in postal money orders. Cards of varying thickness are acceptable; however, card thickness and length must be consistent during any one run.

The B 7110-5 card reader control can be used with B 9115 (300 cpm), B 9116 (600 cpm), or B 9117 (800 cpm) card readers. The input hopper has a capacity of 1000 cards. This series of card readers do no internal code manipulation; instead, all 12 rows are sent to the B 7110-5 control. The control then converts alpha code to BCL; EBCDIC is converted to internal EBCDIC; and binary cards are read without translation.

Unlike the B 9111/12 units, the B 9115/16/17 card readers can only read rectangular holes. These card readers may optionally be set up to read 51-column cards with the installation of a B 9915 kit. These series are desk top card readers, but have a stand optionally available (B 9991-2) on which to set the card reader.

CDL Word Format



41017

Field

OP	VAR	Operation
20	IIII	Read BCL
21	IIII	Read Binary
22	IIII	Read EBCDIC
99	IIII	Test

I = ignored

IOCW Information

Operation	IOCW Bits				CDL OP Code
	46	44	42	41	
BCL to Int. BCL	0	1	1	0	20
BCL to ASCII	1	1	1	0	20
BCL to EBCDIC	0	1	1	1	20
Binary (60 bit to 6-bit)	0	1	0	0	21
EBCDIC to EBCDIC	0	1	0	1	22
EBCDIC to ASCII	1	1	0	1	22

Result Descriptor - Unit Error Field

To MCP	From Device	Error Type
0081	D00	Memory-Access Error
0101	C80	Read Check
0281	D40	Validity Check
0381	C80	Read Check and Validity Check
0401	D40	Control Card (generated by IOM)
0881	D10	Bus Parity Error
0889	F10	Bus Parity Error in Initiate Phase*

*3A Control Only

Operations

BCL (OP 20)

Read one card from the card reader. The operation is terminated by reading the specified number of words, or by receiving 80 characters from the reader. The card reader or its control converts BCL card code to BCL code. BCL code is converted to BCL internal control code, ASCII, or EBCDIC by translators in the IOM.

Binary (OP 21)

Read one card from the card reader. The operation is terminated by reading the specified number of words, or by receiving 80 columns of information from the reader. The contents of each card column are divided into two six-bit fields. The upper six bits are stored in memory followed by the lower six bits. There is no code translation or invalid code detection. Tag field transfers are not compatible with this operator and must not be specified.

EBCDIC (OP 22)

Read one card from the card reader. The operation is terminated by reading the specified number of words, or by receiving 80 characters from the reader. The card reader control converts EBCDIC card code to EBCDIC. EBCDIC is stored as received, or is translated to ASCII by the IOM.

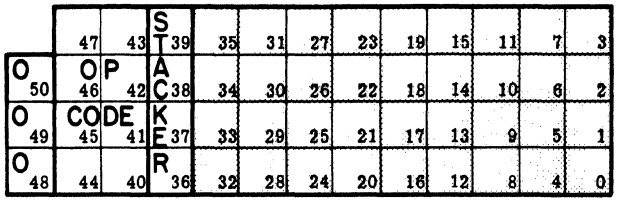
Test (OP 99)

Test the status of the unit and return a result descriptor.

CARD PUNCH

The B 7212 card punch control is used with the B 9213 "300 CPM Punch" which can punch either binary, alpha, or EBCDIC code at a rate of 300 cards per minute. Pre-punched cards may be used, but previously punched columns cannot be repunched. The card punch has a 1000-card capacity input hopper, and three output stackers (primary, auxiliary, and error) which have a capacity of 1200 cards each. Stacker selection is accomplished programmatically.

CDL Word Format



41014

Field

OP	VAR	ADDR	Operation
23	SIII		Punch BCL
24	SIII		Punch Binary
25	SIII		Punch EBCDIC
99	IIII		Test

S = stacker; 0 = normal, 1 = auxiliary, I = ignored

IOCW Information

Operation	IOCW Bits 46	42	41	CDL OP Code
BCL from Int. BCL	0	1	0	23
BCL from ASCII	1	1	0	23
BCL from EBCDIC	0	1	1	23
Binary (6-bit from 6-bit)	0	0	0	24
EBCDIC from EBCDIC	0	0	1	25
EBCDIC from ASCII	1	0	1	25

Result Descriptor - Unit Error Field

To MCP	From Device	Error Type
0081	D00	Punch Check or Memory Access Error
0281	D40	Parity Error
0881	D10	Bus Parity Error

A test-op returns the punch type in bit 10 of the software RD: 1 = Model II & IV

Operations

BCL (OP 23)

Punch one card on the card punch. The operation is terminated by punching the specified number of words or punching 80 columns. The descriptor word count cannot exceed 10 words for punch BCL. BCL internal code, ASCII, or EBCDIC is converted to BCL code by translators in the IOM. The control can include one and only one of the following translators which are used to convert BCL code to BCL card code, ICT card code, or BULL card code:

1. BCL-BCL Card Code Translator.
2. BCL-ICT Card Code Translator.
3. BCL-BULL Card Code Translator.

Binary (OP 24)

Punch one card on the card punch. The operation is terminated by punching the specified number of words or by punching 80 columns. The descriptor word count cannot exceed 20 words for punch binary. A total of 160 six-bit characters of memory are required to punch 80 columns. The contents of each card column are divided into two six-bit characters. The upper six bits are punched from the first six-bit character received and the lower six bits from the next six-bit character. Tag field transfers are compatible with this operator and must not be specified.

Card Punch EBCDIC (OP 25)

Punch one card on the card punch. The operation is terminated by punching the specified number of words or by punching 80 columns. The descriptor word count cannot exceed 13 words for punch EBCDIC. ASCII is translated to EBCDIC by translators in the IOM. The card punch control converts EBCDIC eight-bit code to EBCDIC card code.

Test (OP 99)

Test the status of the unit and return a result descriptor.

Punch Check Error

When punch check is detected by card punch control, the punching of that card is completed and is sent to the error stacker. The punch check bit in the result descriptor is set.

The punch check bit may be present in the result descriptor when addressing a non-present punch, or one that is powered down.

TRAIN PRINTERS

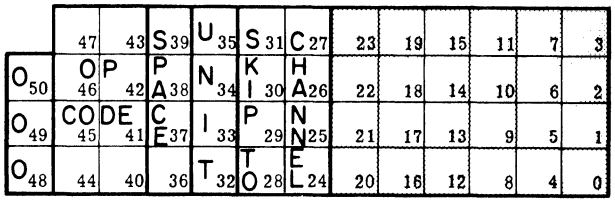
The B 9247 train printers are train-type printers which operate with a buffered I/O control device. The character train module is operator changeable and can be exchanged with a module of a different character set. In the following information there is a list of the character sets available as defined by software on the system. Identification of the character set is performed automatically by the printer; no operator selection is required.

When the printer is in an idle status exceeding one minute, the character train module is automatically shut down, extending the life of the module. Upon receipt of a print instruction the module is automatically restarted.

To assure reliability of the printed output, a positive column dropout detection feature is provided. If a print position fails to print, the printer stops and signals for operator inspection. An override function switch on the operator control panel allows the operator to disable the feature.

The B 7243-1 train printer control is used only for the B 9247-13 (750 LPM) train printer; the B 7247-4 train printer control is for the B 9247-14 (1100 LPM) train printer; and the B 7247-5 train printer control is interfaced to a B 9247-15 (1500 LPM) train printer. The speeds listed are rated with a 48 character set and with the printer operating at six lines per inch, single spacing. The B 9247 printers are all 132 print positions.

CDL Word Format



41032

Field

OP	VAR	ADDR	Operation
10	SUNN		WRITE
11	SUNN		SKIP
29	QQCC		Load Train Image Buffer
99	iuii		Test

NN = Skip to channel (00-11); two decimal digits.
S = Space (, 1, 2); ignored if NN ≠ 0.
QQ = 1D of train buffer being loaded in low 6 bits; high 2 bits must be zero.
(See table 4.3.2 for list of train ID No's.)
CC = 8 bit buffer image designated to be printed for any invalid character.
U = Unit Designate (0 or 1)
i = ignored

IOCW Information

Operation	IOCW Bits 46	43	42	41	39	CDL OP Code
PRINT	A	0	A	A	0	10
Space or Skip	X	1	X	X	0	11
Load TIB	A	0	A	A	1	29

A = Any IOM-allowed output translation; if 6 bit characters result, the IOM pads to 8 bits with leading zeroes. Train printer operations always use all 8 bits.

Result Descriptor - Unit Error Field

To MCP	From Device	Error Type
0181	D80	Print Check Internal Control
0281	D40	Memory Parity Check
0481	D20	Print Check - Synchronization
0881	D10	Bus Parity Error during Data Transfer
0889	F10	Bus Parity Error in Initiate Phase
1001	C08	End-of-Page
4001	C02	Train Image Buffer not loaded.
8001	C01	Incorrect Train on Printer

A Test OP returns the following information in the RD to the MCP:

bit 8 on =Train Image Buffer Not Loaded.
Bit 9 on = 1000 LPM adapter installed.
bits 10-15 = Train ID (15 is LSB)

Operation

Print (OP 10)

Print one line on the printer. The length of the line is determined by the number of printer columns (132) or by printing the specified number of words. Spacing or skipping takes place after the printing.

Space (OP 11)

Space, as specified by CDL word bits 37:2 (bits 31:8 must equal zero), moves paper either 00 (zero), 01 (one), or 10 (two) lines per operation without any printing occurring.

Skip (OP 11)

Skip, as specified by CDL word bits 31:8, slews paper to channel 01 to 12 on the format tape (channel 12 is reserved for end-of-page, but may be skipped to) without any printing occurring.

Load Train Image Buffer (OP 29)

This command is used to load the train image buffer (TIB) with the character set that must be on the train module. If they are not alike, an error occurs and the printer is declared not ready. In addition to loading the TIB, this command also loads bits 39:8 (QQ) of the CDL word into the train identification register, and bits 31:8 of the CDL word into the invalid character code register which is the character used whenever an invalid character is encountered in the print field data.

Test (OP 99)

This I/O command instructs the control to interrogate the printer and control status and return the following information:

1. Readiness status of both control and printer.
2. Type of printer connected (400/750 LPM or 1000/1500 LPM).
3. Load status of the train image buffer.
4. Identification of train module mounted on the printer.

Table 4-3-2. Train ID Numbers

ID NO	Train Table Names
0	NOT SPECIFIED
1	EBCDIC18
2	FORTRAN48
3	B300B500
4	EBCDIC48
5	EBCDIC72
6	UK3500
7	UK6500
8	LATINPORTUGAL
9	LATINSPAIN3
10	SWEDENFINLAND3
11	DENMARK
12	BCL64
13	TURKEY
14	EBCDIC16
15	ANSCII72
16	EBCDIC96
17	KATAKANA
18	ALPHAEBCDIC
19	NUMERICEBCDIC
20	RPG48
21	OCRANUMERIC
22	OCRBNUMERIC
23	FRANCEBELGIUM
24	UK
25	GERMANYAUSTRIA
26	ITALY
27	SWEDENFINLAND2
28	LATINSPAIN2
29	ANSCII64A
30	BRAZIL
31	DENMARKNORWAY
32	YUGOSLAVIA
33	EBCDIC64A
34	ANSCII96A
35	EBCDIC64B
36	ANSCII64B
37	ANSCII96B

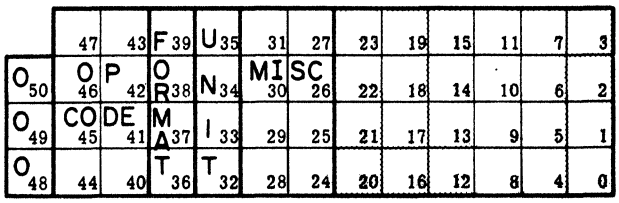
MAGNETIC TAPE SUBSYSTEM

The B 9495 magnetic tape subsystem consists of a Master Electronics Control Unit (MEC) and from one to sixteen tape drive units. The MEC consists of a main cabinet and, if required, an auxiliary cabinet. The MEC contains from one to four input/output channels and from one to eight exchanges (each exchange capable of operating two magnetic tape drive units). The subsystem is capable of operating in the NRZ or phase encoded mode of recording or a combination of both depending upon the options installed in the MEC and tape units.

The tape units feature automatic loading of tape when using a 10.5 inch reel of tape with or without a cartridge band. Semi-automatic loading is provided by the tape unit when using incomplete reels or smaller reels of tape. Semi-automatic loading requires the operator to set a switch in the tape unit and place the leading edge of tape in an area beyond the first air threading guide prior to pressing the load button.

The subsystem can be from one to four controls, and from one to sixteen tape units, depending upon customer requirements. Four controls give the system the ability to do tape operations on four drives at the same time. The recording mode is phase encoded (PE) or a combination of PE and NRZ. The NRZ capabilities are options installed in the MEC and tape units as required by the customer.

CDL Word Format



41035

OP	VAR	ADDR	Operation
01	iUVi		Rewind
02	DUVC		Read Forward
03	DUVC		Read Backward
04	DUVi		Erase
06	DUVi		Write
08	DUNN		Space Forward
09	DUNN		Space Backward
99	iUii		Test

i = Ignored
U = Unit Designate (LSD)
NN = No. of Records (2 decimal digits; 00 spaces 100 records)
C = CRC Correction, if bit 27 is on (9 track NRZ only); track in error is in [26:3].
V4 = (Read/Write) = Special BCL translation in PC if bit 30 is on (7 track EVEN parity only).
V2, V1 (Read) = Maintenance variants if [29:2] ≠ 0 (ignored for PE tape).
V2 (Write) = Write tapemark if bit 29 is on.
V8 (Read) = Do not store information if bit 31 is on.
V1 (Rewind) = Unload tape if bit 28 is on (6A control only).
V8 (Erase) = Backward pseudo-erase if bit 31 is on.
D= Density and Parity

density -		800	556@	200	1600*	#
parity	EVEN @	0	2	4	6	8
	ODD	1	3	5	7	9

@ = 7 track only
* = 9 track PE only
# = Unit selected density

IOCW Information

	Operation	IOCW Bits 46	44	43	42	41	39	CDL Op Code
7 track	Read Binary (6-bit to 6-bit)	0	1	0	0	0	0/1	02/03
	Read BCL into Int. BCL	0	1	0	1	0	0/1	02/03
	Read BCL into EBCDIC	0	1	0	1	1	0/1	02/03
	Read BCL into ASCII	1	1	0	1	0	0/1	02/03
	Write Binary (6-bit to 6-bit)	0	0	0	0	0	x	06
	Write BCL from Int. BCL	0	0	0	1	0	x	06
	Write BCL from EBCDIC	0	0	0	1	1	x	06
	Write BCL from ASCII	1	0	0	1	0	x	06
	Erase	x	0	1	x	0	0	04

IOCW Information

	Operation	IOCW Bits 46	44	43	42	41	39	CDL OP Code
9 track	Read Binary (8-bit to 8-bit)	0	1	0	0	1	0/1	02/03
	Read EBCDIC into ASCII	1	1	0	0	1	0/1	02/03
	Read ASCII into EBCDIC	1	1	0	1	1	0/1	02/03
	Write Binary (8-bit to 8-bit)	0	0	0	0	1	x	06
	Write EBCDIC from ASCII	1	0	0	0	1	x	06
	Write ASCII from EBCDIC	1	0	0	1	1	x	06
	Erase	x	0	1	x	1	0	04
Both	Rewind	x	0	1	x	x	1	01
	Space	x	1	1	x	x	0/1	08/09
	Write tapemark	x	0	x	x	x	x	06

x = not used

Result Descriptor - Unit Error Field

To MCP	From Device	Error Type
0000	800	Normal Termination by PC on Non-Read Operation
	C20	Normal Termination by PC on Read Operation
0801		Long Block } may be subsequently generated by 1OM from final
0401		Short Block } L1A address comparison with buffer length
0081	D00	Memory-Access Error
0101	C80	Beginning-of-Tape or End-of-Tape
0109	E80	Not Ready During Operation*
0201	C40	Write Lockout or End-of-File
0481	D20	Peripheral Interface Parity During Data Transfer*
0481	D20	Read Memory Access Error**
0489	F20	Peripheral Interface Parity in Initiate Phase*
0881	D10	Bus Parity Error (System Interface Parity during Data Transfer*)
0889	F10	System Interface Parity in Initiate Phase*
0C01	-	Tape Positioning uncertain during Retry***
0C81	D30	Parity Error

To MCP	From Device	Error Type
0D81	D80	Parity Error and End-of-Tape
0E81	D70	Parity Error and End-of-File
2001	C04	Non-Present Option (Incorrect Density)
4009	E02	Not Ready, Rewinding
8001	C01	Blank Tape Timeout
8101	C81	Blank Tape Timeout and Beginning-of-Tape
Y001 (Y odd)	COZ (Z ≥ 8)	CRC Correction Requested (9 track NRZ only); track in error is in Z4, Z2 and Z1 (LSB); in Y2, Y4 and Y8 (LSB)

* Model 6A Control only
** Model 5A Control only
*** Generated by Software

A test op returns the unit density in [11:2] of the RD to the MCP as follows:
0 = 800 BPI, 1 = 556 BPI, 2 = 200 BPI, 3 = 1600 BPI.

Operations

Rewind (OP 01)

Rewind the designated tape unit. The control is released and a result descriptor returned after rewind is initiated.

Read OP 02 (Forward) or OP 03 (Reverse)

Read a record from the designated tape unit. The operation is terminated by detection of an interrecord gap. Information transfer is terminated after reading the specified number of words or by sensing an internal DSU error.

Erase (OP 04)

Erase in the forward direction on the designated tape unit. The operation is terminated by erasing the number of words specified. No memory cycles are used.

Write (OP 06)

Write a record on the designated tape unit. The operation is terminated by writing the specified number of words or a delimiter in the data stream.

Write Tape Mark (OP 06)

Write a tape mark record on the unit designated, when V = 2.

Space OP 08 (Forward): OP 09 (Reverse)

Space 1 to 100 records as specified by the BCD value of NN of the CDL word. If bits NN are all 0's, space 100 records.

Test (OP 99)

Test the status of the designated unit and return a result descriptor.

BCL Alpha Operation (7-Track Tape with Even Parity)

When the six-bit frame size and even parity are selected, BCL internal code is converted to BCL code on write, and BCL code is converted to BCL internal code on read. The BCL "?" code is written (001111).

Exception Conditions

End-of-tape does not terminate an operation. The end-of-tape bit is set in the result descriptor after the operation is completed.

On read operations, when a vertical parity error is detected and the six-bit frame size is selected, a BCL "?" code is stored by the control in memory in place of the code in error.

CRC Correction (9-Track, 800 BPI Only)

CDL bits V enables CRC correction. The three LSB's define the track to be corrected (0-7). The parity track cannot be corrected.

DISK PACK DRIVE SUBSYSTEM

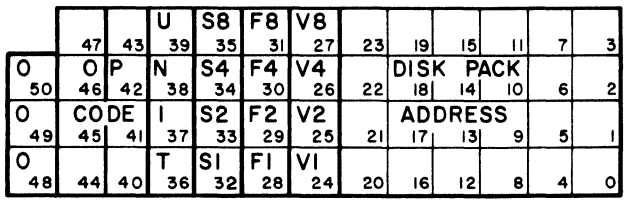
The disk pack drive subsystems are high speed, modular, random information storage systems. A basic disk pack drive subsystem includes the disk pack drive controller, dual disk pack drive, and the interconnecting cables. The subsystem is interfaced to the system via a host transfer control.

The controller acts upon I/O instructions from the IOM, powers the pack drives, and transfers information between the drives and the IOM. The controller performs the operation specified by the OP code (and variants) of the CDL word, and, at the completion of the operation, generates a result descriptor which contains operation and/or error status information.

The disk pack drive controller without an exchange allows one controller to be used with up to eight disk pack spindles (four dual drives) in a "one-by" configuration. The disk pack drive controller with an exchange configuration allows two controllers access to up to 16 disk pack spindles (eight dual drives) in a "two-by" configuration. This allows the I/O module to execute two simultaneous operations (two reads, two writes, or a read and a write). The above configurations are applicable to the B 7385 pack controller and to the B 7387 disk pack controller/exchange.

In addition, the B 7385 and the B 7387 pack controllers allow an option of dual access to the same controller for redundancy purposes in a "continuous processing" environment.

CDL Word General Format



ET1253

Op	Var	Addr	Operation
50	USFV	cccccc	Write
51	USFV	cccccc	Read
56	USFV	cccccc	Initialize
57	USFV	cccccc	Verify
58	USFV	cccccc	Relocate
99	USFV	N/N	Test

Some variables are common to all commands. Their functions are as follows:

U8 - U1. The hexadecimal value of the unit desired.

S1. A ONE state indicates an unconditional operation; that is, for any operation, the controller will do whatever head positioning is required to locate the sector desired, and then perform the operation requested, during which time the I/O channel waits. For a ZERO state, a conditional operation is indicated; that is, the controller checks the unit and, if the unit is presently seeking, sends back a unit seeking R/D. If a seek is required, the controller issues a seek to the selected drive and returns a seek initiated R/D. If a seek is not required, the command is done and, when completed, returns an appropriate R/D.

F1. A ONE state is used to access a drive which had been previously placed in maintenance mode. Note that this variable applies only to the 206/207 drives interfaced with a B 9387/Bx 387 controller.

CDL Word Format, Write (OP 50)

Variables that apply to this command:

S8. In a ZERO state, indicates a normal write; in a ONE state, indicates a controller controlware load.

Write

Data sent from the host system is written onto the disk pack starting at the designated "C"-Address. Partial-sector writes result in a fill of nulls (null is equivalent to hex 00) in the remainder of the sector. Upon detection of any error, the operation is immediately terminated and the appropriate R/D is returned to the host system.

Load Host

The load host operation is used to load operational and diagnostic controlware from the host system into the disk pack drive controller.

CDL Word Format, Read (OP 51)

Variables that apply to this command are as follows:

No variable. Read with retry and error correction enable.

S8. Read with retry and error correction disabled.

S4. Read absolute (refer to V4, V2, V1).

S2. Read unit ID.

F4. Subsystem poll (online only).

V8. Read memory.

V4 -V1. With S4 set, V4 -V1 is the "C-Address" increment.

Read

Information is sent to the host system from the selected "C"-Address. A partial sector read results in the termination of data transfer, but controller release waits for completion of the total sector read. Error checking is performed on the total sector.

Upon detection of any read data error not related to erroneous data received from the disk pack, the operation is immediately terminated and the appropriate R/D is returned to the host system.

If erroneous data is detected from the disk pack and retry/correction is enabled, the controller invokes the appropriate retry procedure using track offset and PLO early/late, where applicable. If the error persists after retry, the controller: 1) corrects the data (if possible) using the sector data read from disk having the smallest magnitude data error; 2) logs the event; and 3) continues the read operation. If a data error is corrected by retry, then a data error retry R/D is returned to the host, if no other errors occurred. If controlware correction is attempted on a data error, the data error correction R/D is returned to the host, if correction is successful. If correction is unsuccessful, the data error R/D is returned to the host when an HTC1A interface is in use, and the data error correction R/D (without successful recovery bit) is returned to the host when an HT-LCP interface is in use. If retry/correction is disabled, the data error R/D is returned to the host.

In the event of multiple data errors per read operation, the data error R/D takes precedence over the data error correction R/D, which takes place over data error retry R/D.

Error log data accumulated for one read operation is destroyed by the next operation.. The log information may be retrieved through the read memory command.

Read Absolute

The address field (two words), data field (50 or 90 words), and the error protection code syndrome (two words) of the specified sector are returned to the host. The sector is located by its actual expected position relative to the index mark.

Read Unit ID

A three-word (16-bit word) data field which indicates the controlware loaded and the type of drive selected, is returned to the host followed by an appropriate result descriptor indicating the condition of the selected unit.

The format for the three words returned is as follows:

1. Digits 0, 1, 2, and 3 are used to indicate the controlware release code.
2. Digits 4 and 5 are reserved.
3. Digits 6 and 7 are used to indicate the unit ID. The following is a list of applicable values for large systems applications:
    a. 30 = 225 Disk Pack Drive.
    b. 40 = 235 Disk Pack Drive.
    c. 68 = 206 (Sequential) Disk Pack Drive.
    d. 78 = 206 (Interlaced) Disk Pack Drive.
    e. A8 = 207 (Sequential) Disk Pack Drive.
    f. B8 = 207 (Interlaced) Disk Pack Drive.
4. Digits 8 through 11 are reserved.

Subsystem Poll

When this C/D is received the least four digits of the "C"-Address represent a 16-bit field designating what units are to be polled. Each bit set in this field represents a different unit, the most significant bit being unit 15 and the least significant bit being unit 0.

All units represented in the 16-bit field are checked to see if there are any in seek-ready or in an error state. If all the designated units are busy or on-line and seeking, they are all checked again. This process continues until at least one of the designated units becomes seek-ready or goes into an error state. As soon as this happens, 48 bits of data are returned to the host; the first 16 represent the designated units that are seek-ready or are in an error state. The most significant bit of data represents unit 15 and the least significant bit (of the first 16) represents unit 0. After the data is returned to the host, the operation is terminated with an operation complete R/D.

NOTE

A conditional cancel from the host is accepted by the controller until it is ready to send back the results of the checking.

Read Memory

The read memory command transfers the contents of the controller's buffer memory to the host system. The 256 word buffer memory is divided into two logical areas: control memory (first 76 words) and data memory (next 180 words). The controlware listing for each set of controlware details the contents of all 256 locations. Execution of this command releases the controller from the controller lock state.

The significance of the first 23 words made available by the command are as follows:

Bit Word Location	Description
0	C/W Release Code
1	Result descriptor tag from last operation
2	Micro-program memory address if DPDC installation timeout or buffer memory parity error (Bx 383/384/385). Processor, or DMC channel, buffer memory address for buffer memory parity error (B9387).
3	Disk DDP Status
4	Reserved for disk DDP diagnostic information
5	Reserved for disk DDP diagnostic information
6	Disk drive status
7	Reserved for extended drive status
8	Host DDP status
9	Reserved
10	Reserved
11	Disk address 0 0 01 02
12	Disk address C3 C4 C5 C6
13	Number of retries on above address
14	Total number of retries on last command description
15	Op code (HTC1A)
16	US variants (HTC1A)
17	FV variants (HTC1A)
18	Command descriptor 0 0 C1 C2
19	Command descriptor C3 C4 C5 C6
20	Result descriptor (word 1)
21	
22	

CDL Word Format, Initialize (OP 56)

Variables that apply to this command are as follows:

V4. In a ZERO state, causes a full initialize; in a ONE state, causes a data only initialize.

Initialize

The controller writes sector addresses and gaps in all tracks of the cylinder specified by the "C"-Address. The controller also writes a predefined data pattern in the data field of each sector. The "C"-Address must point to the first sector of a cylinder.

Initialize Data Only

The controller receives a one word pattern from the host and writes it to the data field of each addressable sector on the cylinder specified by the "C"-Address. The "C"-Address must point to the first sector of a cylinder.

CDL Word Format, Verify (OP 57)

Variables that apply to this command are as follows:

V8. In a ZERO state, enables EPC checking only; in a ONE state, enables data checking also.

Verify

The controller reads and checks for address errors and data field error protection code errors in all addressable sectors on the designated cylinder. If the data compare variant bit is set (V8), the data field is also compared against a predetermined pattern. The position of each sector is checked relative to the index mark. During verification, a relocated sector is verified in the same apparent manner as any other sector. The "C"-Address must point to the first sector of a cylinder.

The "C"-Address of a sector in error is returned as the first two words of a three word data transfer when an HTC1A interface is in use.

CDL Word Format, Relocate (OP 58)

No variable, other than the common variables covered previously.

Relocate

The controller flags the address area of the sector specified in the "C"-Address with an error configuration, selects an unused spare and writes the original sector's address into the address area, and fills the spare sector's data field with a predefined pattern.

CDL Word Format, Test Commands (OP 99)

The following variables are shown as applicable:

S4. In a ONE state, this variable generates a controller lock disable (online only).

S2. In a ONE state, this variable generates a controller lock enable.

S1. Set to a 1, causes a non-busy drive to be powered off.

V8. Provides a POWER UP signal to a powered-down drive, when set to 1 (available on the B 9387/Bx 387 -206/207 drives only).

V2. Take selected drive out of maintenance mode if this bit is set (available on the B 9387/Bx 387 -206/207 drives only).

V1. Place the selected unit in maintenance mode (available on the B 9387/Bx 387 -206/207 drives only).

No variables. Normal test operation to be performed.

Controller Lock Disable

The controller lock disable option places the DPDC into an unlockable state such that a lockable R/D causes the controller not to lock. For dual host configuration, each host can disable controller lock independent of the other host.

Controller Lock Enable

The controller lock enable option places the DPDC into a lockable state such that a lockable R/D causes the controller to lock, thereby preventing the loss of pertinent error information stored in its memory. For dual host configuration, each host can enable controller lock independent of the other host. When in the locked state, the disk pack drive controller returns a controller locked R/D to the locking host for all commands requiring a drive access until:

1. A read memory command is received from the locking host.
2. A controller lock disable command is received from the locking host.
3. A predetermined time interval has expired (no greater than two seconds).

Power Unit Down

Upon selecting a non-busy spindle, a power down command is issued to the spindle and an operation complete R/D is immediately sent to the host.

Power Unit Up

Upon selecting a powered down spindle, a power up command is issued to the spindle and an operation complete R/D is immediately sent to the host.

Place Unit Into Maintenance Mode

The controller attempts to place the designated spindle into maintenance mode. Once in maintenance mode, the spindle can only be accessed by descriptors with the variable F1 set to a ONE state.

Release Unit From Maintenance Mode

The controller attempts to take the designated spindle out of maintenance mode.

Test Operation

The controller checks the status of the designated spindle, and returns an appropriate result descriptor indicating the condition of the unit.

File Addressing

The "C"-Address designates a particular track and sector on a disk pack and is the starting point for all operations having that file address. Addresses are numbered sequentially, starting with sector 0 (first sector on a track), on surface 0 (head 0), track 0 (cylinder 0), and continuing through all sectors, heads, and cylinders (including the maintenance cylinder) respectively.

Result Descriptors

Result information is generated by the controller and the host transfer control in the form of a result descriptor (R/D). One R/D is returned to the host per I/O initiate. Conditions reported in the result descriptor are listed in table 4-3-3 and are described in the following paragraphs, along with corresponding action taken by the controller.

Operation Complete

When the specified operation has been successfully performed with no exceptions, this R/D is returned to the host.

Seek Error

If, during a read, write, verify, or relocate operation, a seek has been initiated (which results in the heads being positioned over the wrong cylinder or the proper head is not selected by the drive, both as verified by the controller reading the address field from a track on the cylinder in question), the operation is terminated.

Table 4-3-3. Controller and Host Transfer Result Descriptor Information

MCP	Device	Description	Lock Controller
0000	8000	Op Complete, No Exception	No
0801	C100	Seek Error	Yes
8801	C110	Seek Timeout	Yes
C801	C130	Data Error Retry	Yes
0401	C200	Unit Busy	No
0C01	C300	Data Error Correction	Yes
0201	C400	Unit Seeking	No
0A01	C500	Seek Initiated	No
0101	C800	Address EPC Error	Yes
4101	C820	Address Position Error	Yes
0901	C900	Address Timeout	Yes
0501	CA00	Write Lockout	No
0301	CC00	First Action With Unit	No
0081	D000	Memory Access Error	Yes
8881	D110	Host Parity Error	Yes
0481	D200	Speed Error	Yes
0981	D900	Link Parity Error **	N/A
0581	DA00	Data Error	Yes
0009	E000	Not Ready	Yes
0809	E100	HTC Timeout **	N/A
0109	E800	Controller In Local **	N/A
0509	EA00	Controller Locked	No *
0889	F100	Controller Failure	Yes

* Controller Already Locked

** Generated by the HTC

Seek Time-Out

If a disk pack drive fails to complete an initiated seek within one second, it goes into a seek time-out state. The controller, upon detecting a disk pack drive which is accessed in this state, or upon recognizing that a similar time period has elapsed for an initiated seek, terminates the operation.

Data Error Retry

If, during a read operation all data errors were successfully corrected by retry, the appropriate bits are set (1) in the result descriptor upon the completion of the operation.

Unit Busy

At the initiation of any operation, if the unit accessed is being used by another controller, and does not become available within a preselected period (one second nominal), the operation is terminated.

Data Error Correction

If, during a read operation with no uncorrectable data errors, a data error was detected and corrected by the controlware after the specified number of retries had been unsuccessful, the designated bits are set (1).

Unit Seeking

At the initiation of any operation without the unconditional variant set (1), if the drive is found to be seeking, the operation is terminated.

Seek Initiated

At the initiation of any operation without the unconditional variant set (1), if the drive is caused to begin seeking, the operation is terminated.

Address EPC Error

If a sector address after the initial sector of an operation is found to have its error protection code (EPC) in error and the sector is not found in the spares, the operation is terminated.

Address Position Error

If a sector address after the initial sector of an operation is not in sequence and is not found in the spares, the operation is terminated.

Address Time-Out

If there is a failure to find a specified sector address (first sector of operation) or to find a spare sector into which that address has been relocated, the operation is terminated.

Write Lockout

At the beginning of a write, initialize, or relocate operation, if it is detected that the unit to be accessed is in a write lockout state, the operation is not initiated. For a read unit ID or test operation, if the unit is detected to be in a write lockout state and no other exceptions are detected, then this R/D is returned to the host at the end of the operation.

First Action with Unit

At the initiation of any operation, if the controller detects that this is the first action to be performed with the designated unit since it was last powered up, the operation is terminated. The determination of the first action status is performed by hardware as a large system option for the subsystem.

Memory Access Error

If a memory access is requested by the DPDC but not used (memory request error), or if a memory request is not honored by the host in time, the operation is terminated at the end of the sector being processed.

Host Parity Error

During any operation, if a parity error occurs (as determined in host DDP status) between the host system and the disk pack drive controller, the operation is terminated. All operations are terminated immediately if the error occurs in the transmission of the command descriptor information. All operations are terminated at the end of the sector being read from or written to disk if the error occurred in the transmission of data.

Speed Error

During an initialize operation, if a failure to write a full track of information between index pulses is detected, the operation is terminated.

Link Parity Error

Parity error on HTC to remote device interface.

Data Error

During a read operation, if an uncorrectable disk pack data error is detected, the operation is terminated at the end of the word count.

If an error protection code error or data non-compare error is detected during a verify operation, the operation is terminated at the end of the sector being processed at the time of error detection.

Not Ready

The controller will terminate the operation in progress under any one of the following conditions:

1. A disk pack drive is not ready, or reports abnormal status.
2. The maximum file address of the drive is exceeded.
3. A non-existent unit is addressed.
4. An undigit is found in the "C"-Address.
5. An invalid command descriptor is received.
6. No unused spares were available for a relocate command.

HTC Time-Out

Indicates no host activity with the HTC for a predetermined time. The operation is terminated.

Local

Indicates that the remote device is off-line.

Controller Locked

This indicates that the controller is locked, and the command descriptor (C/D) is not:

1. Read Memory.
2. Controller Lock Disable.

Controller Failure

During any operation, if a disk pack controller detects any one of several serious errors (various internal parity or timeout conditions), the operation in progress is terminated. If the controller was writing to the disk at the time the error occurred, it could destroy the information on the track.

Result Descriptor (R/D) Tags

In order to facilitate maintenance and increase the ease of debugging the controller, a method of generating result information (similar to what the R/D is to the host system) is incorporated in the controlware. Associated with each unique place in the controlware that an R/D may be generated is a label called an R/D TAG. Since there may be several places in the controlware which generate the same R/D, the R/D TAG provides a high degree of visibility as to the exact state of the controller when the R/D was generated. Therefore, the R/D TAG number is stored in buffer memory where it can be obtained by the host system. Each R/D TAG is of the form: R/D TA- -XXX, where XXX is: 000 -999.

R/D TAGS as displayed.

0XXX - lockable R/D TAG.

1XXX - unlockable R/D TAG.

2XXX - MPM address for instruction time-out.

4XXX - MPM address for buffer memory parity error.

Any given R/D TAG may appear in more than one set of controlware, but always maintains its unique meaning.

DISK FILE SUBSYSTEM (TYPE 5N)

The 5N disk file peripheral subsystem (referred to as the 5N disk file or the 5N subsystem) is a major peripheral unit designed to operate with all computer systems that require very fast information access and transfer. The 5N subsystem uses a head-per-track unit that has been designed for complete field maintainability, including replacement of heads, disk, and spindle assemblies under normal field conditions. The minimum subsystem configuration is one cabinet containing one disk electronics (DE) module and one disk storage (DS) module. A maximum subsystem configuration is four cabinets containing one disk electronics (DE) and four disk storage (DS) modules.

The dual needs of multiple storage capacities and expandability are provided by the modular cabinet design. Two types of cabinets are used. A primary cabinet contains a power supply and a disk storage module (DS), and is wide enough to accommodate a disk electronics (DE) module. An add-on cabinet contains only a DS and must be configured with a primary cabinet. A primary cabinet with a DE installed constitutes a minimum subsystem. A maximum subsystem has two primaries, each with a power supply, one with the DE, and two add-ons. Only one add-on unit can be supported by a primary.

The DS contains a 14 inch disk which rotates at 6,000 rpm, providing an average access time of five milliseconds and a maximum transfer rate of 1.25 megabytes. The data is organized into fixed format segments of 100 or 180 byte lengths, giving the disk a storage capacity of approximately six million bytes. The primary function of the DS is to write and read the information passed to and from the DE. Two segment interlace options are provided which affect the average transfer rate. A 1 by 2 interlace option gives a 656 kB average transfer rate; an alternate 1 by 4 interlace option gives a 328 kB transfer rate.

The DE is the interface controller linking the DS(s) to the I/O portion of the host system. The DE contains the control logic and necessary buffering to provide precise synchronous command, status, and data management between the selected DS and the I/O of the host system. Two ports are available in the DE for system interconnection. This allows the failsoft mode of operation, if desired. The DE receives information from the host system command which includes segment address, and then initiates a search for the selected segment. After address coincidence, the DE then controls the active data transfer and terminates the operation upon command. Odd parity is assigned and checked on all status, data, and address transmissions within the disk file subsystem. Result status is available in the DE following every operation completed. Data and control signals are managed internally by the DE and DS in a bit serial mode.

Segment Organization

A DS contains one 14.5 inch diameter disk. The read/write head assemblies, called head modules, are positioned along the radius of the disk surface, eight on each side. Each head module has 32 active tracks (actual individual heads) and three possible (one guaranteed) spares. The eight head modules with 32 tracks give a total of 256 active addressable data tracks for each side of the disk. Activating a spare track to replace an active track which has failed can be accomplished in approximately 10 minutes without opening the enclosure.

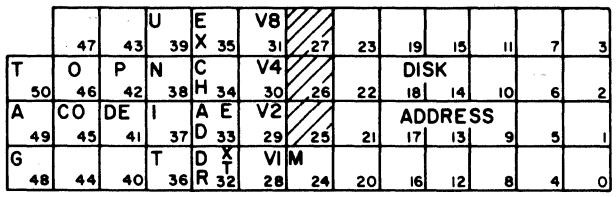
The data tracks are formatted into fixed segment lengths of either 100 or 180 bytes. Each segment contains 100 or 180 bytes of data plus four bytes of error checking and correction (ECC) code. Segment format is a function of the DE and is established by a local operation called an *initialize*. Initialize causes a write operation to be performed on the selected DS in which the segment addresses are written in the data tracks.

Interlace Options

A segment interlace feature is used on the 5N subsystem to minimize the possible band-pass saturation of the host system caused by the very high transfer rate of the data. Two interleave ratios are available: a 1-to-2 segment interleaving or a 1-to-4 segment interleaving, which means that the consecutively addressed segments are located every second or every fourth segment position along a data track. Selection of the interleave ratio permits adjustment of the total system I/O channel through-put loading. The bit rate between the DE and I/O control during data segment transfers is at the full DS disk rate of 10 megabits per second; however, during multi-segment transfers the average transfer rate is reduced by the segment interlacing.

Selection of the active interlace option is a function of the DE. It is controlled by a PROM chip in a manner similar to the segment option. A local initialize function writes the segment addresses to conform to the interlace parameter installed in the DE.

CDL Word Format



ET1270

OP	VAR	ADDR	Operation
50	UFVM	AAAAAA	Write
51	UFVM	AAAAAA	Read
52	UFVM	AAAAAA	Check
99	UFii		Test

U = Unit Designate (LSD)
F : (35:2) = MSB of position on exchange
(33:2) = reserved for disk address extension
V8 : 1 = Read extender status
V4: 1 = Enable extra revolution result descriptor
V1 : 1 = Maintenance segment
A = Disk segment address (6 BCD digits)
i = Ignored

IOCW Information

Operation	IOCW Bits					CDL
	46	44	43	42	41	OP Code
WRITE	0	0	0	0	1	50
READ	0	1	0	0	1	51
CHECK	0	1	1	0	1	52

Subsystem Commands

The basic command set executed by the 5N subsystem is described below. The commands are READ NORMAL, READ MAINTENANCE, READ STATUS, WRITE, WRITE MAINTENANCE, TERMINATE, SEGMENT REPEAT, SEGMENT WAIT, and TEST. Channel parity is also received and checked by the DE with every command byte. If a parity error is detected, the command is not executed, and a result status byte is returned to the I/O.

Two commands are executed by the DE which can be used only when the DE is in local. They are the INITIALIZE and the VERIFY commands. The OP byte is loaded with the DE maintenance panel switches, and the operation takes place under the control of the DE local logic.

Read Normal

The READ NORMAL command is received over the CSO line as a one byte operation code followed by three bytes of file address. The DE decodes the file address bytes for disk, track, and segment corresponding to the address received. The DE then initiates a search for the addressed segment. Upon segment coincidence the segment data is transmitted from the DE to the I/O over the DI line at the disk bit rate. The data transfer operation continues through consecutive segments in a track and through consecutive tracks and disks until the operation is terminated by the I/O. A time delay of up to one disk revolution can occur when the operation continues from disk to disk.

During READ NORMAL, if any errors or warning conditions are detected in the subsystem, the result status is transmitted over the CSI line by the DE. However, the read operation continues until terminated by the I/O.

Read Maintenance

The READ MAINTENANCE command is identical to the READ NORMAL command except that only the maintenance segment of each track is read.

Read Status

The READ STATUS command is received from the I/O over the CSO line as a one byte operation code followed by three address bytes. The command causes the DE to transfer 64 bits of extended status message over the DI line. The operation is terminated by the DE upon completion of the message transfer.

Write Normal

The WRITE command is received over the CSO line as a one byte operation code followed by three bytes of file address. The DE decodes the file address bytes, selects the disk and track, and searches for the segment corresponding to the file address received. Upon segment coincidence, the segment data is transferred to the DE over the DO line at the disk bit rate until the operation is terminated by the I/O. The data rate is regulated in the I/O by the DE clock it receives on the DI line. The writing operation continues through consecutive segments in a track, through consecutive tracks, and then consecutively from disk to disk. A time delay of up to one disk revolution can occur when a write operation crosses over from disk to disk. The write operation is terminated when the I/O issues a TERMINATE command over the CSO line. Should the I/O prematurely terminate the operation, the DE completes recording the current segment. The DE stops the transmission of clock pulses over the DI line between segments. When the last addressable segment is written, the DE independently terminates the operation.

Write Maintenance

The WRITE MAINTENANCE command is identical to the WRITE command in format and operation except that only the maintenance segments are written.

Test Command

A TEST command is received over the CSO line as a one byte operation code followed by three bytes of file address. The DE decodes the command to select the disk and responds with result status on the CSI line. The TEST command permits the I/O to check the status of the selected disk.

Initialize

The INITIALIZE command is executed by the DE in local as a maintenance operation. It is a one byte command loaded into the command control via the DE maintenance panel switches. This command is a write

function which writes the addresses in the tracks of the DS units associated with that DE. The command must be used following installation or when the segment or interlace option is changed.

Verify

The VERIFY command is executed by the DE in local as a maintenance operation. It is a one byte command loaded into the command control via the DE maintenance panel switches. This command is a read function which reads the addresses written in the tracks of the DS, checking the sequence and validity of the addresses. Use of this command is a good local test of the read function.

Result Descriptors

To MCP	From Device	Error Type
0009	E00	Not Ready
0101	C80	DE Busy (Timeout)
0109	E80	Warning
0181	D80	Address Error from DE
0201	C40	Write Lockout
0281	D40	Command Parity Error (System to DFC)
0481	D20	Transmission Error between DFC and DE
0801	C10	Extra Revolution
4081	D02	Controller (Data Buffer) Parity Error
8081	D01	Data Parity Error (System to DFC) on Write Data Error Correction Not Done on Read

A test operation returns a different type of 2 in [11:2] of the RD to software.

Extended Status Message (ESM)

The normal result status returned to the system indicates only a few specific errors and general classes of errors. This generality is desirable for normal operational software; however, the 5N subsystem has an additional feature called an extended status message (ESM) which is available on request from the DE following any operation. Information contained in the ESM may be used to:

1. Initiate break-out or precautionary routines.
2. Build an audit trail for failure analysis.
3. Construct a unit performance profile.
4. Reduce unscheduled maintenance by detailing the extent of a reported failure.
5. Provide diagnostic capability without interrupting customer operation.

The extended status information is stored in the DE registers after each operation and is transferred to the system as data in response to the read status command. If the next command is not a read status command, the information is replaced by new result status from that command. The 64-bit extended status message has its own special format and is transmitted to the I/O.

The internal operating conditions and performance parameters of the subsystem are monitored automatically and continually. The DE identifies detected errors and reports the identification in the 64 bit extended status message (ESM). Disk subsystem detected errors, with their fault locations identified where possible, are reported more specifically in the ESM than was transmitted in the eight-bit RSB. The performance parameters monitored denote subsystem performance deterioration and show possible failure conditions. The ESM is intended as system input for logging and analysis.

The data in the extended result descriptor is arranged in 20 words of 12 hexadecimal digits. The first 64 bits are used on the 5N subsystem. Bit 63 is the first bit of word zero; bit 0 is least significant bit of digit nine in word one. The following describe the significance of each bit:

Bit 63 — EXCESSIVE TEMPERATURE. Indicates excessive temperature in a disk enclosure of the subsystem. This causes that DS to retract the heads. If an affected DS is selected, "not ready" is indicated in the RSB.

Bit 62 — HEAD LOAD PRESSURE. Indicates that the primary air pressure has dropped below 45 psi and the DS heads have been retracted. If an affected DS is selected, "not ready" is indicated in the RSB.

Bit 61 — HEAD TOUCH. Indicates a DS with an uncleared head touch condition. Selection of the DS causes "not ready" to be reported in the RSB.

Bit 60 — DISK SPEED LOW. Indicates that the disk is rotating at less than 5700 rpm and the heads have been retracted. Selection of the DS causes "not ready" to be reported in the RSB.

Bit 59 — WRITE DRIVER OVERHEAT. Indicates that a write driver overheat condition exists in the DS head module. The heads are retracted because this is an unsafe condition. Selection of the DS causes "not ready" to be reported in the RSB.

Bit 58 — NEGATIVE HOUSING PRESSURE. Indicates that the air pressure inside the DS enclosure has fallen below a safe operating level. The heads are retracted because this is an unsafe condition. Selection of the DS causes "not ready" to be reported in the RSB.

Bits 53-50 — HIGH TEMPERATURE. Indicates a DS enclosure temperature above the acceptable limit in any of the DS's on the subsystem. Normal operation can continue. A "Warning" bit is set in the RSB. Each DS is assigned an identification bit in the ESM as follows:

Bit 50 = DS 0
Bit 51 = DS 1
Bit 52 = DS 2
Bit 53 = DS 3

Bit 49 — AC POWER LOW. Indicates that the input ac line voltage has been sensed at 90 percent of nominal. The subsystem will go "not ready" if the condition continues for over 0.5 second. If the input line voltage drops below 70 percent of nominal, the subsystem will shut down.

Bit 46 — ADDRESS REDUNDANCY CHECK ERROR. Indicates comparison failure of the selected segment address held in the DE and the repeated address read from the selected track. The address error bit is in the RSB.

Bit 45 — PHI BIT TIMEOUT. Indicates a data read failure. The DE was unable to locate data on the selected track within 15 ms. The address error bit is set in the RSB.

Bit 44 — ADDRESS SEARCH TIMEOUT. Indicates an address search failure. The DE was unable to locate the segment address requested by the system within 15 ms. The address error bit is set in the RSB.

Bit 42 — INTERNAL ADDRESS PARITY ERROR. Indicates that a parity error was detected on the address byte transfer from the DE to the DS. The command transmission error bit is set in the RSB.

Bit 41 — CONTROL CHANNEL PARITY ERROR. Indicates that a parity error was detected on the OP and address byte transfer from the I/O control to the DE. The command transmission error bit is set in the RSB. In the case of segment repeat, segment wait, or terminate commands, the DE continues through its current segment before returning the RSB.

Bit 40 — INVALID REQUEST. Indicates that the bit pattern of the OP code received by the DE is not valid. The command transmission error bit is set in the RSB. If the invalid command is received during a read or write, the DE continues through its current segment before returning the RSB.

Bit 35 — DATA READ ERROR, ECC. Indicates a comparison failure between the accumulated ECC in the DE and the ECC read from the track. This comparison is made following the read of each segment. The operation will continue to completion regardless of when the comparison failed. The data transmission error bit is set in the RSB.

Bit 34 — DATA WRITE ERROR, ECC. Indicates a comparison failure between the accumulated ECC in the DE and the ECC received from the I/O. This comparison is made as the ECC is being written for each segment. The operation will continue to completion regardless of when the comparison failed. The data transmission error bit is set in the RSB.

Bit 33 — CLOCK SYNC ERROR. Indicates a clock pulse count failure. During a write operation the DE counts the number of clocks received from the I/O. If the number of clocks received is not equal to the number of clocks required for a full segment at the end of the segment, an error is indicated. The data transmission error bit is set in the RSB and the operation continues to completion.

Bit 32 — WRITE FAILURE. Indicates that an open circuit in the DS head module was detected during a write operation. The data transmission error bit is set in the RSB and the operation continues to completion.

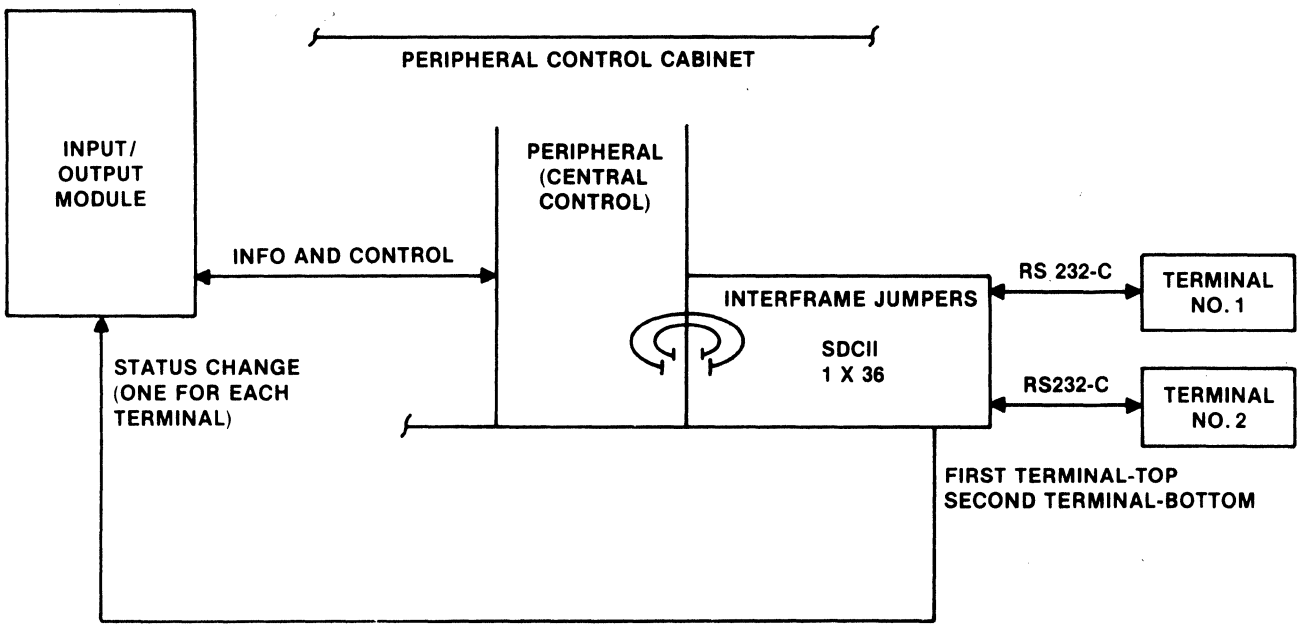
Bits 22-00 — DISK ADDRESS. Indicates the address of the last segment accessed, divided into four fields. The bits are identified below (some bits are reserved for expansion).

Bits 22-21 = DS Number, 0 to 3 binary, lsb.
Bits 17-14 = Head Number, 0 to 15 binary, lsb.
Bits 13-09 = Track Number, 0 to 31 binary, lsb.
Bits 06-00 = Segment Number, 0 to 107 binary, lsb.

SUPERVISORY DISPLAY CONTROL II

The Supervisory Display Control II (SDC II) is a type A control used to interface the system to up to two operator display terminals. The SDC figure 4-3-1 performs the following functions:

1. Message heading insertion for messages transmitted to the terminals.
2. Message heading deletion for messages received from the terminals.
3. Generation of vertical and longitudinal parity (even).
4. Checking of vertical and longitudinal parity.
5. Internal code conversion between EBCDIC to/from the system and ASCII (7 bit) from/to the terminals.
6. Generation of input request interrupts.



ET1254

Figure 4-3-1. SDC II in B 7800 Systems

CDL Word Format

	47	43	U 39	35	31	27	23	19	15	11	7	3
O 50	O 46	P 42	N 38	34	30	26	22	18	14	10	6	2
O 49	CO 45	DE 41	I 37	33	29	25	21	17	13	9	5	1
O 48	44	40	T 36	32	28	24	20	16	12	8	4	0

OP	VAR	ADDR	Operation
32	Uiii		Read
34	Uiii		Write
99	Uiii		Test

i - ignored
U = Unit Designate

IOCM Information

Operation	IOCW Bits 46	44	42	41	CDL OP Code
Read EBCDIC	0	1	0	1	32
Write EBCDIC	0	0	0	1	34

Result Descriptor - Unit Error Field

TO MCP	From Device	Error Type
0009	E000	Not Ready
0A81	D500	I/O Parity Error*
0281	D400	Memory Access Error*
0201	C400	Data Parity Error*
0401	C200	Control Character
0801	C100	Read Overflow
8001	C010	Time Out/Invalid Character
0301	CC00	Internal Parity Error*

Test Op R/D to MCP

2000	Unit is B9348 (TD804)
4000	Unit is B9352
6000	Unit is B9348-34 (TD830-1)

IOM/SDC II interface format is shown in figure 4-3-2. Read and write message formats are defined in figure 4-3-3 and 4-3-4.

The SDC is designed to interface with the following operator display terminals (ODT):
1. B 9348: Supervisory Input and Display.
2. B 9352: Input and Display Terminal.
3. B 9348-34: Operator Display Terminal.

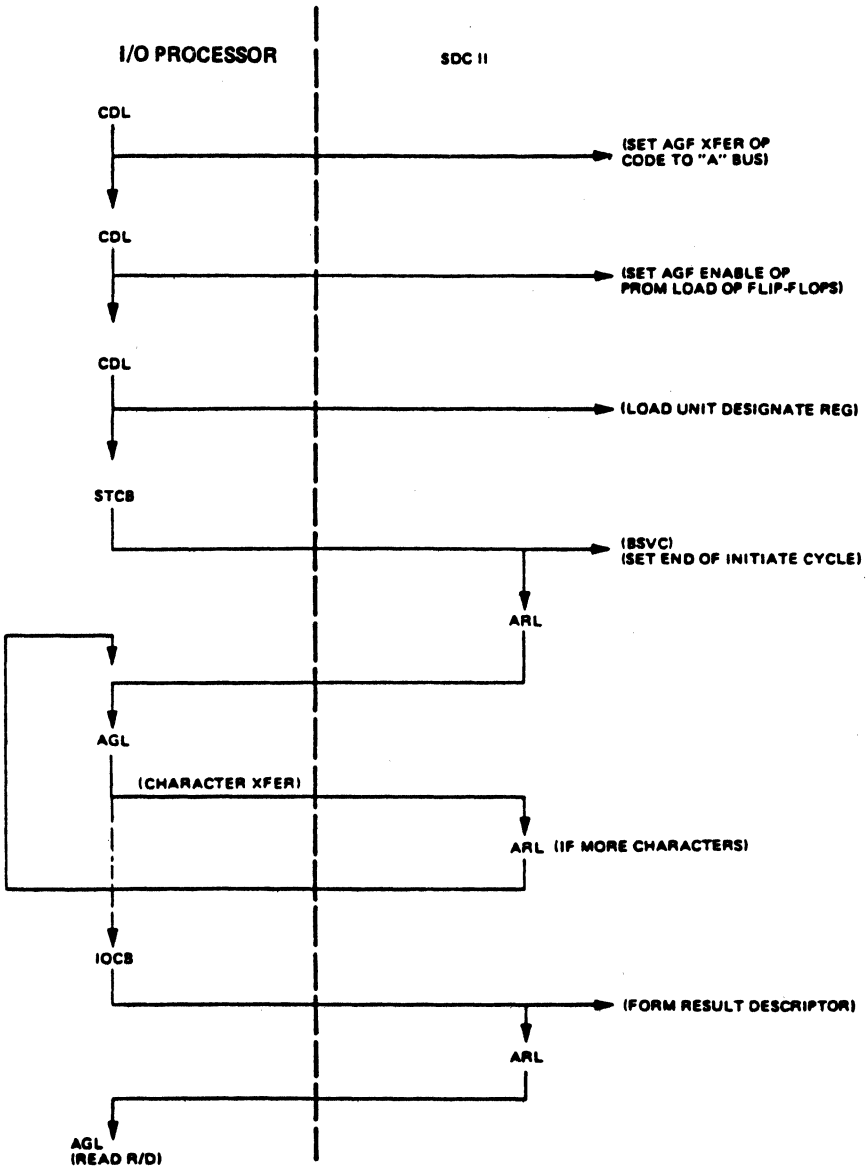
A jumper wire on the backplane in the control is used to identify the type device installed to the test operation command by way of the test operation's result descriptor. Baud rate may be set to one of three values via strapping on printed circuit board. These values are 2.4K, 9.6K, and 19.2K baud.

The interface to the terminals is in accord with EIA RS232-C standard asynchronous (with a modification of the data terminal ready line redefined to be true when the terminal is in receive mode and false at all other times).

Operation

Read (OP 32)

Read an input message from the designated terminal unit until an end of text character is detected or until the area descriptor word count is exhausted, whichever comes first. The IOM stores this data after having received it in EBCDIC due to the translator in the control.



ET1256

Figure 4-3-2. IOM/SDC II Format

Write (OP 34)

Send a message to the designated terminal until an end of text character is detected or until the area descriptor word count is exhausted, whichever comes first. The IOM only sends EBCDIC to the control and the control translates it to ASCII (7 bit) for use by the terminal.

Test (OP 99)

Return a result descriptor indicating the type of both terminal units connected to the control. If there are no type bits in the result descriptor returned, it means there is no terminal unit connected to the supervisory display control with that unit designate.

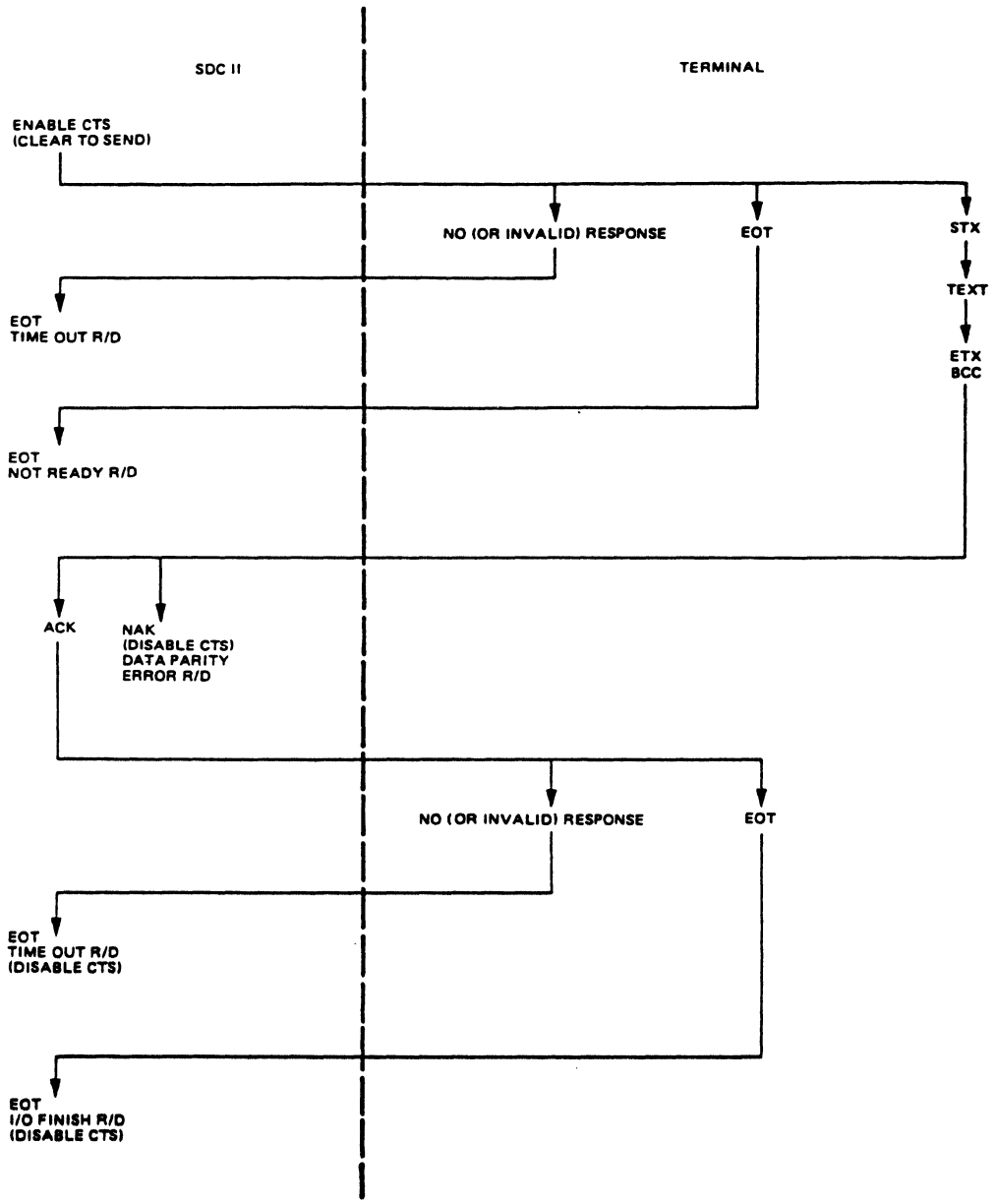


Figure 4-3-3. Message from Terminal (Read)

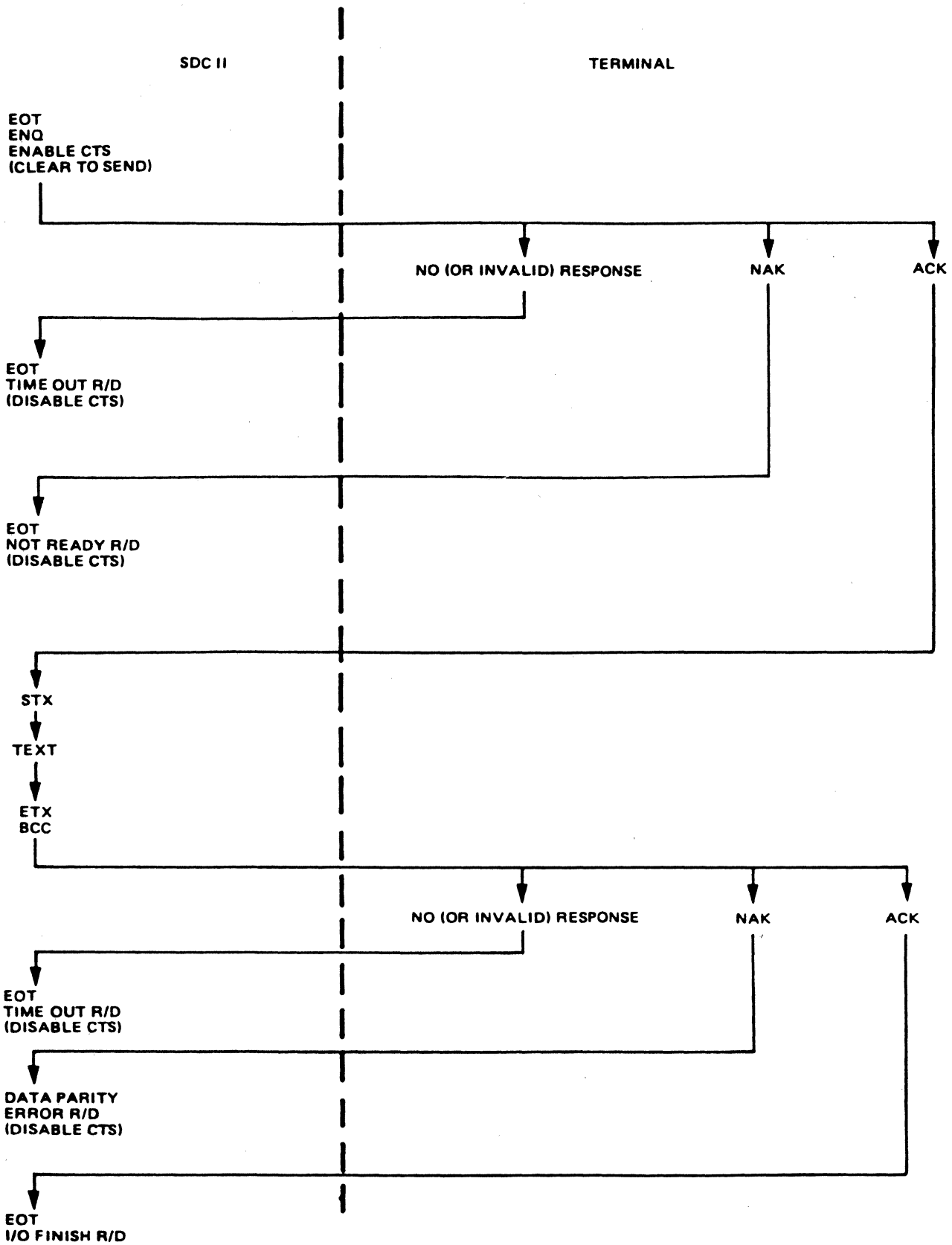


Figure 4-3-4. Message to Terminal (Write)

CHAPTER 5

GENERAL DESCRIPTION OF MEMORY SUBSYSTEM

INTRODUCTION

The B 7800 Memory Subsystem provides the main storage for the B 7800 Data Processing System. The memory subsystem stores or supplies words of information as directed by either of two types of requestor: a central processor or an input/output module.

A B 7800 Memory Subsystem consists of one to four model II memory control modules (or one to two model III memory control modules) coupled through a memory requestor switch-interlock network to a maximum of eight memory requestors. (See figures 5-1-1 and 5-1-2.) The memory subsystem can service each requestor in the same manner so that any operation performed for one requestor may be performed for any other requestor.

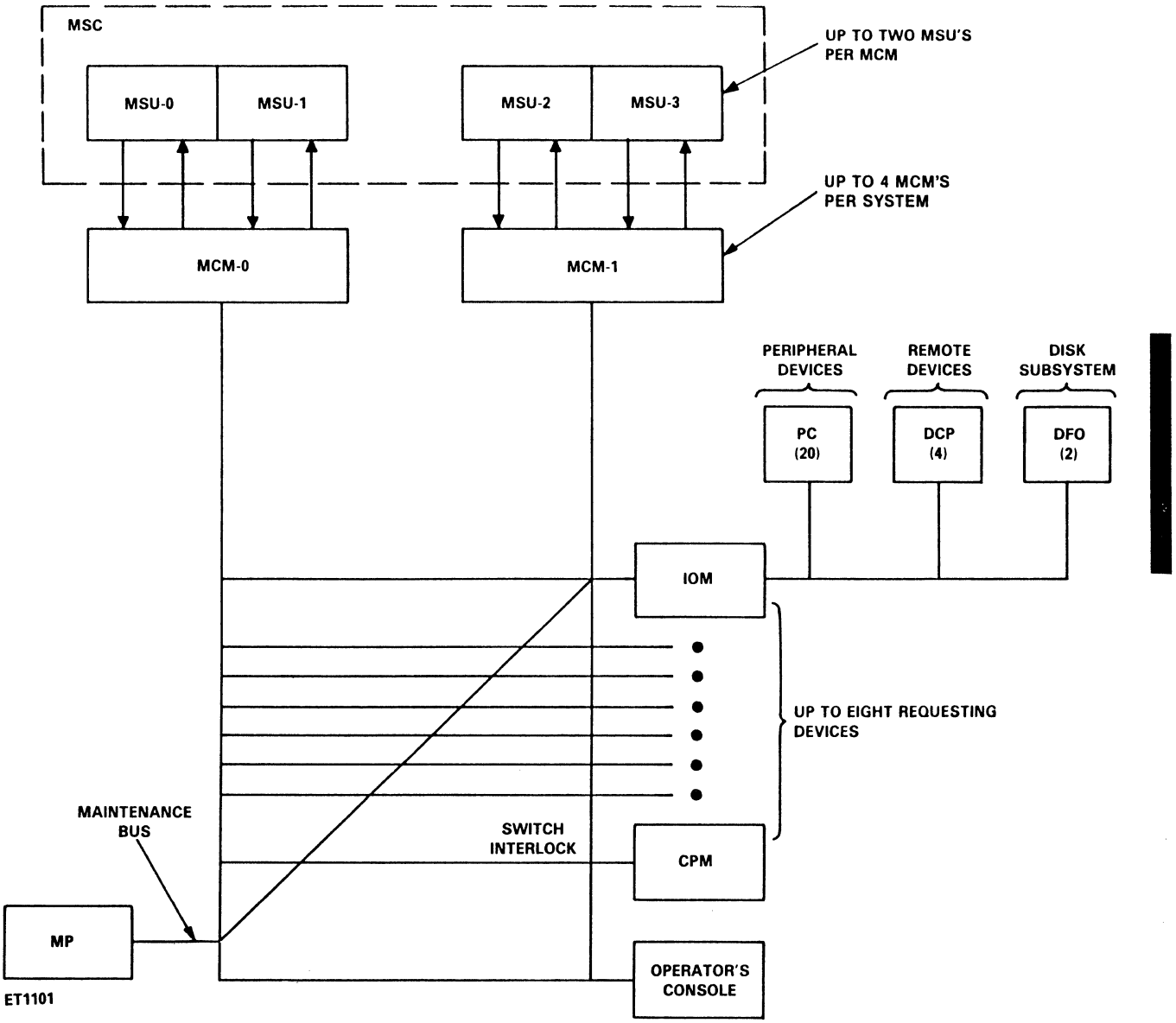


Figure 5-1-1. B 7800 Memory Subsystem with Model II Memory Control Modules Diagram

A model II MCM can control either one or two MSUs. The model III MCM can control up to eight MSUs: however, one MCM will normally be configured to access either two or, as shown in figure 5-1-2, four MSUs.

Each MSU has the storage capacity of 131,072 words.

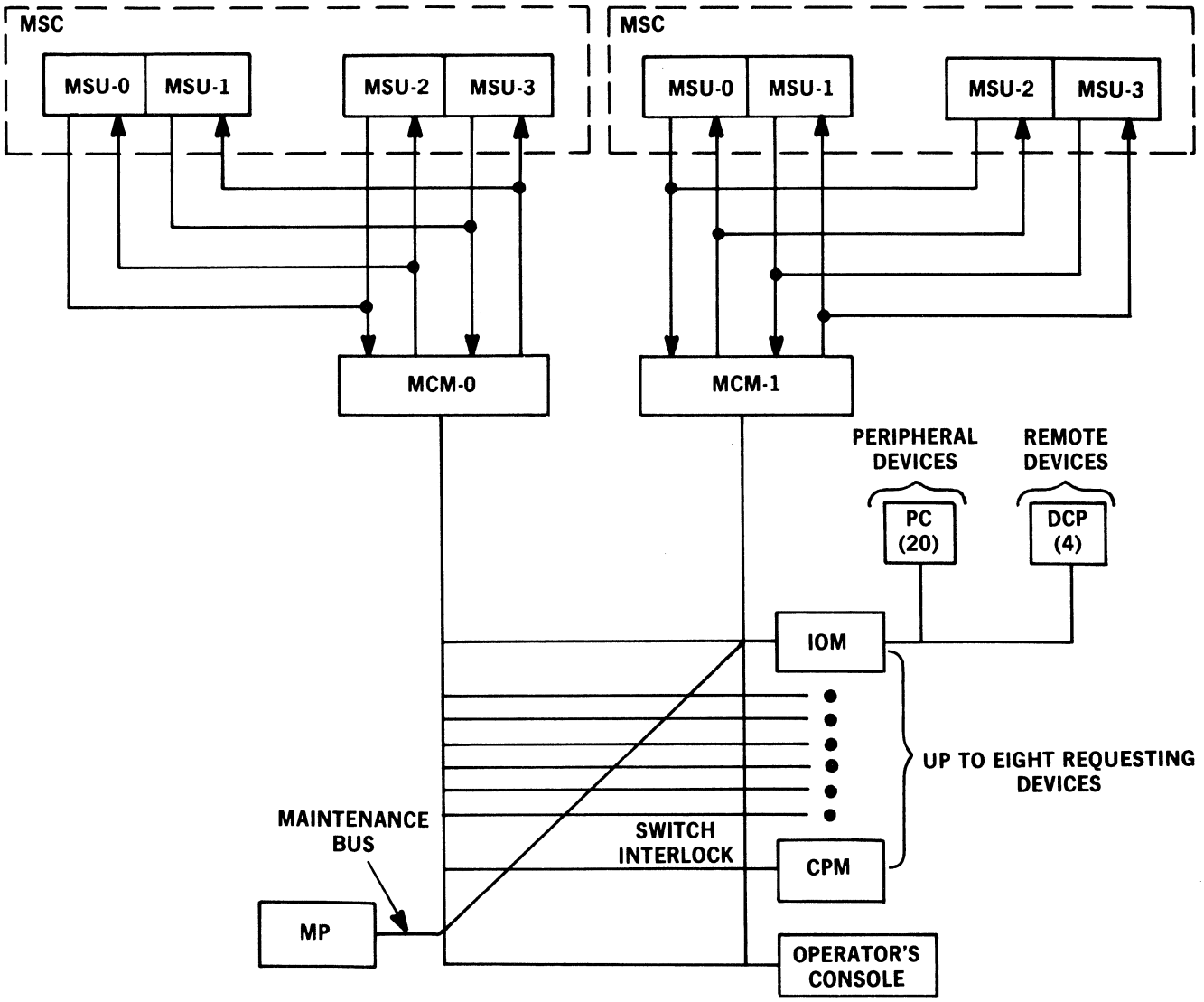
NOTE
16K IC MSUs and 4K IC MSUs cannot by mixed on any MCM. All stacks must be the same type.

MEMORY CAPACITY

A B 7800 Memory System may be built with various combinations of the two configurations of memory modules to achieve the desired total memory capacity. Table 5-1-1 lists the possible combination of memory sizes.

Minimum Memory Size

The minimum memory size is one MSU memory module of 131,072 words (786,432 bytes). This would be one MCM controlling one MSC containing one MSU. For optimum system performance the minimum B 7800 Memory Subsystem recommended is four MSU memory modules.



ET 1686

Figure 5-1-2. B 7800 Memory Subsystem with Model III Memory Control Modules Diagram

Table 5-1-1. B 7800 Memory Subsystem Configurations

Memory Size Words	Bytes	Number of MSUs	Number of MSCs	Number of MCMs	
				Model II	Model III
131,072	,786,432	1	1	1	1
262,144	1,572,864	2	1	1	1
393,216	2,359,296	3	1	2	1
524,288	3,145,728	4	1	2	1
655,360	3,932,160	5	2	3	2
786,432	4,718,592	6	2	3	2
917,504	5,505,024	7	2	4	2
1,048,576	6,291,456	8	2	4	2

Maximum Memory Size

The maximum memory size is 1,048,576 words (6,291,456 bytes), packaged as follows:

1. Eight MSU modules
2. Two MSC cabinets
3. Four model II MCM modules (or two model III MCM modules).

MSU Reconfiguration

The B 7800 Memory Subsystem is designed with high reliability to minimize the occurrence of failure. Extensive error detection and reporting logic permits early detection and definition of failures. Automatic correction of single-bit parity errors minimizes interruption to the system. The modular design, separate power supplies, and independent interface concept permit soft reconfiguration. In case of an MSU failure, the system can be manually or programmatically configured to operate with only one MSU available to the MCM.

Address Allocation

There is no specific assignment order within the system for particular MCM configurations. Memory module address range assignments are based on system requirements and are assigned through use of the memory limits word. For example, any MCM in the system can be assigned the lower (memory zero) address range by setting the memory limits register.

Subsystem Allocation

The memory capacity can be manually or programmatically allocated into subsystems with respect to designated requestors. For example, in a model III MCM subsystem, MCM 0 can be dedicated to requestors 0 and 7 while MCM 1 can be dedicated to requestors 1 and 6.

CLOCK RATE AND READ ACCESS TIMES

The B 7800 Memory Subsystem operates at a clock rate of 8.13874 megahertz. Effective read access time for the MCM is as follows:

1. Single-word access is 1.500 us or 0.250 us per byte.
2. Two-word access is 1.625 us, or 0.813 us per word, or 0.136 us per byte.
3. Four-word access is 1.875 us, or 0.469 us per word, or 0.078 us per byte.
4. Eight-word access is 2.375 us, or 0.296 us per word, or 0.050 us per byte.

Multiple Word Transfer (Phasing)

In a multiple word transfer (called phasing) words are transferred in bursts of up to eight; one word is transferred at each clock cycle.

If the requested address is less than eight words from the upper address limit of the MCM, the memory operation is limited to a single-word transfer.

These phasing limits do not have to be taken into consideration when a requestor sends a memory request. The requestor simply requests the desired number of words to be transferred, and then decreases the number of words each time a memory transfer is completed. If at the end of the operation the word count is not equal to zero, then another request is made until all the words are transferred.

WORD FORMATS

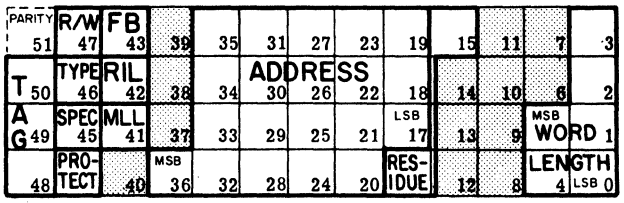
All words used by the B 7800 Mainframe System are 52 bits in length. The 52-bit word consists of 48 bits of information, three tag bits, and an overall parity bit.

When information is passed from a requestor to an MCM, the requestor adds a parity bit which produces odd parity on the resultant 52-bit word being transferred. The MCM checks the word it receives for odd parity to verify that an error was not made during transmission.

When an MCM receives a 52-bit word from a requestor, the MCM adds seven bits of error correction code and adds another bit for maintaining odd parity on the overall 60-bit word to the MSU. If a word should be accidentally altered while residing in an MSU, the seven check bits in conjunction with the overall parity bit allows for the detection of the error and provide a means for the automatic correction of errors in which a single bit has been altered. The MCM then sends the original 52-bit word to the requestor.

MCM Control Word

At the start of every memory operation, an MCM control word is transmitted from the requestor to the memory control module. The control word format, bits and fields as received at the memory control module are described below. Table 5-1-2 lists the operation codes for the MCM.



Field	Bits	Description
PARITY	51:1	The requestor generates odd parity for the 52-bit control word.
TAG	50:3	The tag bits are not used in the control word.
R/W	47:1	R/W bit: 0 = read (fetch) operation; 1 = write operation.
TYPE	46:1	The type bit is set for fail word fetch (if bit 36=1), box ID word fetch (if bit 36=0) and N-word protected write operations. NOTE Box ID word fetch and use of bit 36 are only used in the control word for model III MCM.
SPEC	45:1	The SPEC (specifier) bit indicates either a single word or multi-word operation: 0 = multi-word operation; 1 = single word operation or a box ID word fetch if bit 46 = 1 and bit 36 = 0.
PROTECT	44:1	The protect bit, when "1" indicates that a protected-write operation is to be performed. Protected write only allows the write operation to be performed when bit 48 of the original memory word is off. When bit 48 is on, the write operation is terminated and the contents of the original memory word are not changed.

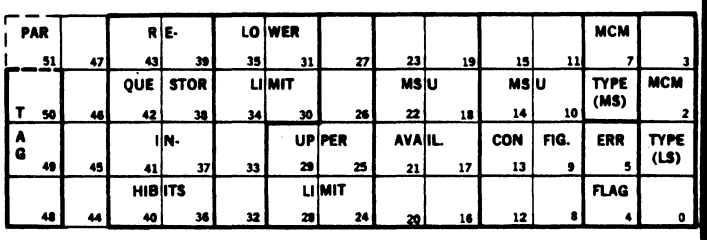
Table 5-1-2. Operation Codes for the MCM

Operation	R/W 47	TYPE 46	SPEC 45	PROT 44	FB 43	FIL 42	MLL 41
Fetch, Single word	0	0	1	0	0	0	0
Fetch, Multiple word	0	0	0	0	0	0	0
Box ID Word Fetch (Bit 36=0)	0	1	1	0	0	0	0
Fetch, Fail register (Bit 36=1)	0	1	1	0	0	0	0
Write, Single word overwrite	1	0	1	0	0	0	0
Write, Multiple word overwrite	1	0	0	0	0	0	0
Write, Single word overwrite with flashback	1	0	1	0	1	0	0
Write, Single Word Protected Write	1	0	1	1	0	0	0
Write, Single word protected write flashback	1	0	1	1	1	0	0
Write, Multiple word protected write	1	1	0	1	0	0	0
Load, Requestor inhibits	1	0	1	0	0	1	0
Load, Memory limits	1	0	1	0	0	0	1

Field	Bits	Description
FB	43:1	The FB (flashback) bit, when a "1" indicates that the original contents of the memory location are to be transferred to the requestor.
RIL	42:2	The RIL (requestor inhibits load) bit is used to specify that a load requestor-inhibit operation is to be performed.
MLL	41:1	The MLL (memory limits load) bit is used to specify that the upper and lower address registers and the MSU available register are to be loaded.
	40:4	Unused.
ADDRESS	36:20	The address bits specify the starting-memory address of the memory operation.
RESIDUE	16:2	The address residue bits indicate the residue value of the 20-bit memory address within the control word.
WORD LENGTH	5:6	The word-length bits indicate the number of words to be transferred during multi-word operations.

Box ID Word (For Model III MCM)

The box ID word is locked in a 52-bit fail register until a box ID word fetch request (OP code 30 and bit 36 reset in the MCM control word) is made by the requestor or a manual clear operation is performed. The format, bits, and fields of the box ID word are described below.



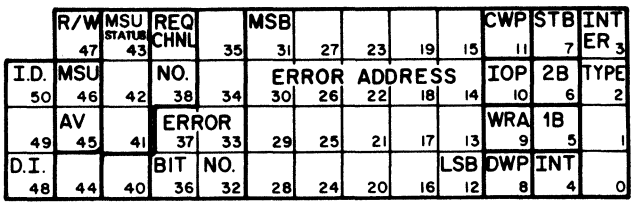
ET 1687

Field	Bits	Description
Tag	50:3	The tag bits are always 0.
Requestor Inhibits	43:8	This field indicates which requestors currently have
Lower Limits	35:6	This field defines the lowest address that can be handled by the MCM within the total memory subsystem.
Upper Limits	29:6	This field defines the highest address that can be handled by the MCM within the total memory subsystem.
MSU Availability	23:8	These bits indicate which of the MSUs are powered up.
MSU Configuration	15:8	This field indicates the number of 128K MSUs that functionally connected with the MCM.
MCM Type (MS)	7:2	These bits are not used in current configuration of the memory subsystem; they are reserved for future expansion.
ERR (Error)	5:1	When set, this bit indicates that an error has been reported and a second fetch of the fail register is necessary to obtain the MCM fail word (OP code 30 and bit 36 set in the MCM control word).
MCM Type Flag	4:1	This bit is always 0 in the box ID word.
MCM Type (LS)	3:4	These bits identify the MCM as a model III; they are configured as 0011.

MCM Fail Word

The MCM fail word contains all pertinent information necessary to identify a hardware failure. The fail word information is locked in a 52-bit fail register until a fetch-the-fail register operation request (OP code 30 and, if model III MCM bit 36 set in the MCM control word) is made by the requestor or a manual clear operation is performed. The MCM sends a fail 1 interrupt signal to the requestor when an irrecoverable error has occurred. If one of the bits in a memory word was incorrect, the specific bit is corrected by the MCM. The correct memory word is then sent to the requestor which allows the requestor to continue processing with correct data.

The format, bits, and fields of the MCM fail word are described below.



ET1271

Field	Bits	Description
I.D.	50:2	These bits are always configured as 110 for the fail word.
D.I. (Model II MCM only)	48:1	The delayed interrupt (D.I.) bit, when set, indicates that an internal error was detected during the previous memory cycle and had occurred after the requestor operation complete (ROC) signal was sent to the requestor. This interrupt does not occur until the next memory operation is performed.
R/W	47:1	The R/W bit indicates that either a read or write operation was being executed when the error was detected. (Read = 0; write = 1.)
MSU AV (Model II MCM only)	46:2	The MSU AV (MSU available) field indicates the number of MSUs in use by the MCM when the error was detected. The field indication is as follows: Bit 45 / Bit 45 0 0 No MSU is available 0 1 One MSU is available 1 0 Two MSUs are available 1 1 Four MSUs are available
MSU STATUS (Model II MCM only)	44:4	The MSU STATUS bits are loaded from the MSU status register to indicate stack availability as follows: 1. 1 AV - sectors 0-3 in MSU-1 are available 2. 2AV - sectors 4-7 in MSU-1 are available 3. 3AV - sectors 0-3 in MSU-2 are available 4. 4AV - sectors 4-7 in MSU-2 are available
REQ CHNL NO.	40:3	The REQ CHNL NO. (requestor-channel-number field) contains the number of the requestor who was using the MCM when the failure occurred. This field is not locked in the fail register if detection of a one-bit error occurs.
ERROR BIT NO.	37:6	The error bit number field is only valid when bit 5 (1-bit error) is set. This field is the binary number of the bit that failed in memory.
ER ADDRS	31:20	The error-address field contains the address of the location that was being accessed if a one-bit or two-bit error occurred. The address is related to one-bit or two-bit errors as follows: Error Indication: 2-Bit / 1-Bit — Error Address Belongs to: 0 1 — 1-Bit Error 1 0 — 2-Bit Error 1 1 — 1-Bit Error
CWP*	11:1	The CWP (control word parity) bit when set, indicates that the MCM has detected incorrect parity on the control word received from the requestor.
IOP*	10:1	The IOP (illegal operation) bit, when set, indicates that the MCM has detected an illegal operation in the operations are as follows: (1) Word length = 0 (2) Single-word operation word length greater than 1 (3) Special-request strobe is not sent by the requestor when either memory-limits load or requestor-inhibit load is to be performed. (4) Illegal-operation code (refer to table 5-1-2).

* Fail 1 interrupt condition

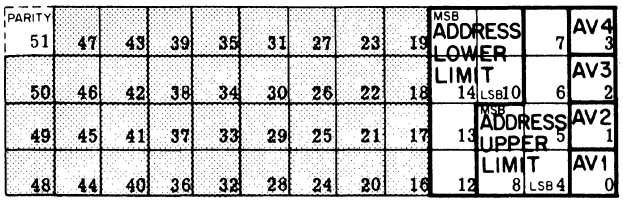
Field	Bits	Description
WRA*	9:1	The WRA (wrong address) bit, when set, indicates that the address in the control word did not fall within the upper and lower address limits assigned to the MCM.
DWP*	8:1	The DWP (data-word parity) bit, when set, indicates that a data word containing even parity was received from the requestor.
STB*	7:1	The STB (data-strobe) error bit when set, indicates that the MCM has detected an error in the number of words sent by a requestor during a multiple word transfer.
2B*	6:1	A 2B (2-bit) error, when set, indicates that the MCM detected a (non-correctable) multiple bit error from the MSU. If this error occurs, the data transfer to the requestor is completed.
1B	5:1	A 1B (1-bit) error, when set, indicates that the MCM detected a (correctable) 1 bit error from the MSU.
INT*	4:1	The INT (internal) error bit, when set, indicates an error occurred within the MCM or MSU. This error is further defined by bits 3:4.
INT ERR TYPE	3:4	The internal error type bits define the type of internal error identified by bit 4. The internal error bits are defined as follows:

Fail Word Bit 3210	Error Type
0000	Sector Busy - indicates that the selected sector for this operation remained unavailable past a preset time limit.
0001	Read Available-indicates that the MSU has failed to respond with a Read Available signal during a read operation.
0010	Checker/Generator-indicates that an error had occurred in the MCM Parity Checker/Generator (data) circuits.
0011	Address Residue-indicates that either a bad address residue was detected in the Control Word received from a requestor, or was generated by the MCM Address Counter during a multiple word transfer operation.
0100	Configuration Error-indicates that the MSU status register contains either 0 or 3 MSUs available, which is illegal.
0101	MSU Availability-indicates that MSUs actually available does not agree with the MSU status register.
0110	Data transfer control (DTC) Failure-indicates that a failure occurred in the DTC circuit.

Memory Address Limits Word

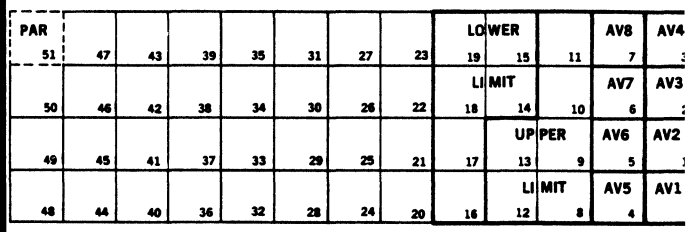
The memory address limits word changes the MCM and MSU configuration to reflect the number of MSUs available to the MCM as well as the upper and lower address limits. This word follows the special request signal and the memory address limits load word during memory control operations between the MCM and requestor. The format, bits, and fields of the memory address limits are described below.

For model II MCM:



Field	Bits	Description
PARITY	51:1	The MCM examines the memory address limits word for odd parity.
	50:35	Unused.
ALL	15:6	The address lower limit is the most significant 6 bits of the lowest 20-bit memory address available to this MCM.
AUL	9:6	The address upper limit is the most significant 6 bits of the highest 20-bit memory address available to this MCM.
AV4	3:1	When AV4 is a "1", MSU-2, sectors 4-7 are available to this MCM.
AV3	2:1	When AV3 is a "1", MSU-2, sectors 0-3 are available to this MCM.
AV2	1:1	When AV2 is a "1", MSU-1, sectors 4-7 are available to this MCM.
AV1	0:1	When AV1 is a "1", MSU-1, sectors 0-3 are available to this MCM.

For model III MCM:



Field	Bits	Description
Parity	51:1	The MCM examines the memory address limits word for odd memory.
Address Lower Limit	19:6	The address lower limit is the most significant 6 bits of the lowest 20-bit memory address available to this MCM.
Address Upper Limit	13:6	The address upper limit is the most significant 6 bits of the highest 20-bit memory address available to this MCM.
AV8 thru AV1	7:8	Eight bits for MSU availability.

Memory Requestor Inhibits Word

The memory requestor inhibits word loads the requestor inhibit register with new data to indicate which requestors now have access to the MCM. This word follows the special request signal and the memory requestor inhibits load control word during memory control operations between the MCM and requestor. The format, bits and fields of the memory requestor inhibits word are described below.

PARITY 51	47	43	39	35	31	27	23	19	15	11	RI7 7	RI3 3
50	46	42	38	34	30	26	22	18	14	10	RI6 6	RI2 2
49	45	41	37	33	29	25	21	17	13	9	RI5 5	RI1 1
48	44	40	36	32	28	24	20	16	12	8	RI4 4	RI0 0

Field	Bits	Description
PARITY	51:1	The MCM examines the requestor inhibit word for odd parity.
	50:44	Unused
R17	7:1	When bit R17 is a "1", the requestor who is designated requestor 7 is inhibited from access to the MCM.
R16	6:1	When bit R16 is a "1", the requestor who is designated requestor 6 is inhibited from access to the MCM.
R16	5:1	When bit R15 is a "1", the requestor who is designated requestor 5 is inhibited from access to the MCM.
R14	4:1	When bit R14 is a "1", the requestor who is designated requestor 4 is inhibited from access to the MCM.
R13	3:1	When bit R13 is a "1", the requestor who is designated requestor 3 is inhibited from access to the MCM.
R12	2:1	When bit R12 is a "1", the requestor who is designated requestor 2 is inhibited from access to the MCM.
R11	1:1	When bit R11 is a "1", the requestor who is designated requestor 1 is inhibited from access to the MCM.
R10	0:1	When bit R10 is a "1", the requestor who is designated requestor 0 is inhibited from access to the MCM.

SIGNAL INTERFACE BETWEEN REQUESTOR, MCM, AND MSU

The control and information flow between the requestor, MCM, and MSU is described in the following paragraphs and shown in figure 5-1-3.

Signal Interface Between MCM and Requestor

1. Data and Parity. Data and parity are transferred between a requestor and an MCM via a unique set of 52-bidirectional data lines. These lines are also used for the transmission of the control word.

2. Special Request Signal (RQSN). A special request signal (RQSN) is used by a CPM to gain access to a memory control module (regardless of the state of the requestor inhibits register). The RQSN signal goes true in coincidence with the request signal (REQ) whenever a memory address limit load or requestor inhibits load operation is performed.

3. Request Signal (REQN). A request signal (REQN) is sent by a requestor to select a specific MCM. REQ goes true one clock period prior to the request strobe (RSTB) and remains true until the receipt of an acknowledge signal (ACK) from the MCM.

4. Data Strobe Signal (DSTB). A data strobe signal (DSTB) is sent to inform the MCM that data is to be transmitted over the data lines. The signal is used only in the N-length overwrite and the N-word protected write operations. The data strobe precedes the data word by one clock and its width indicates the number of data words following it.

5. Request Strobe Signal (RSTB). A request strobe signal (RSTB) is sent to inform the MCM that a control word is being transferred over the data lines. It is true initially one clock period following the start of the request signal (REQ). The control word is transmitted in coincidence with the request strobe.

   a. For single word protected write and single word overwrite: the request strobe (RSTB) will cycle true and false during successive clock periods. During the false period, the data word to be stored is placed on the data lines.
   b. For all other operations: the request strobe (RSTB) is true one clock period following the request signal (REQN) and remains true until the acknowledge signal (ACK) is received.

6. Data Available Signal (DAV). A data available signal (DAV) is transmitted to the requestor to indicate that data is available and will be transmitted in the following clock period.

7. Acknowledge Signal (ACK). An acknowledge signal (ACK) of one clock period duration is sent to

the requestor to signify that the MCM has accepted the control word and is processing the request.

8. Send Data Signal (SND). A send data signal (SND) is sent to the requestor during an N-length overwrite and may be sent during an N-word protected write. The send data signal indicates the number of data words that must be transmitted to the MCM. The number of words to be transmitted is equal to the number of clock periods the send data signal is true.

NOTE

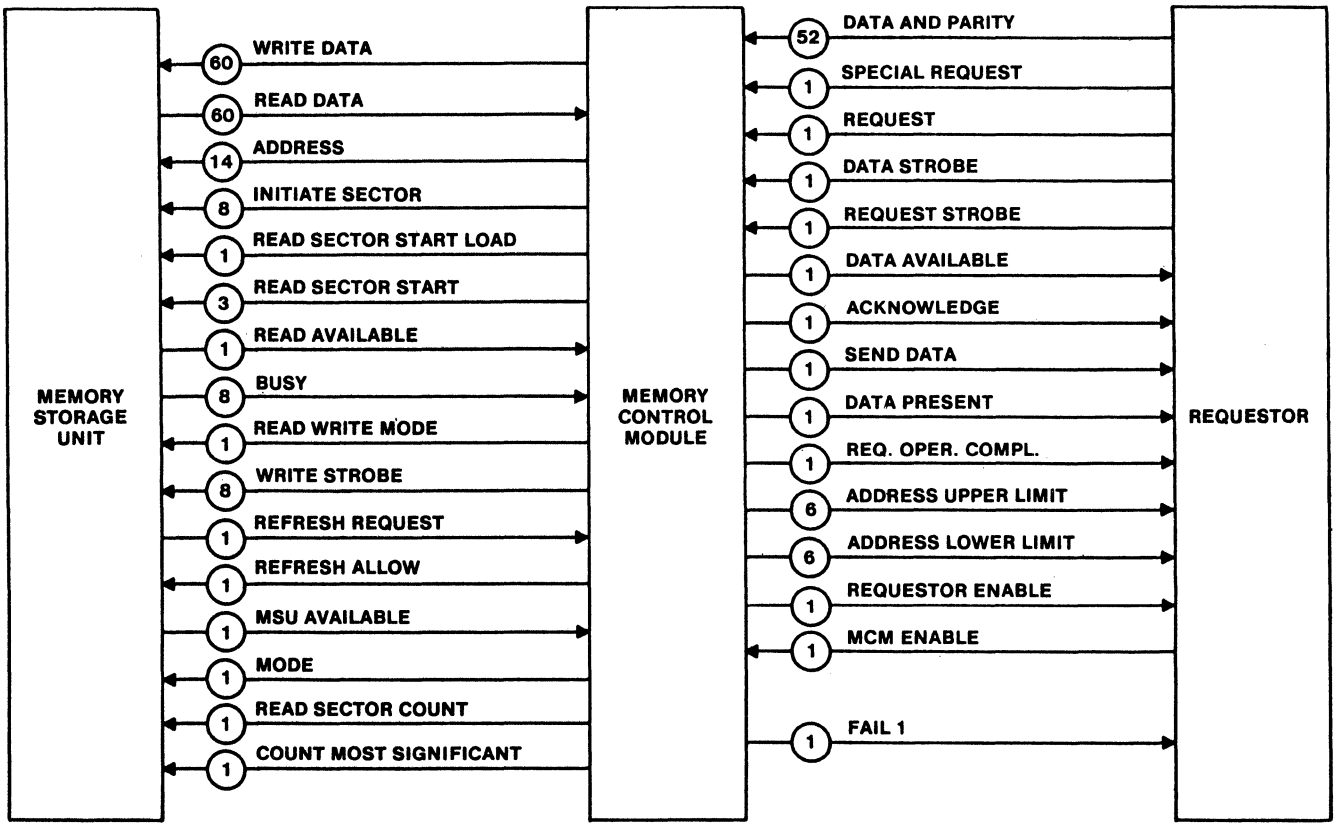
The send data signal will not be transmitted if an attempt is made to write into a protected area during an N-word protected write. Also, the number of data words requested by the MCM must be transferred before a requestor ends operation.

9. Data Present Signal (DAPB). Signal DAPB is sent to the requestor to indicate that a valid data word (or words) is being transmitted from the MCM. The DAPB is transmitted in coincidence with the data word. A word is transmitted each clock period that the DAPB is true.

10. Requestor Operation Complete Signal (RQOC). The MCM sends a one clock period requestor operation complete signal (RQOC) to signify the end of the requestor's part of the memory operation. The following variations apply:

a. Single or N-length fetches; single word overwrite with flashback: the RQOC is sent coincident with the final clock period of the data present signal (DAPB).
b. Single word overwrite; N-length overwrite or N-word protected write: the RQOC signal is sent following the check of parity on the final data word received by the MCM.
c. Single or N-word protected write: The RQOC signal is sent with or following FALS signal if word(s) are protected in N-word protected write.
d. Single word overwrite without flashback: an RQOC is generated following the check of parity on the data word received by the MCM.

11. Address Upper Limit. The address upper limit is the most significant six bits of the highest 20-bit memory access available to this MCM (the least significant 14 bits are assumed to be "1's").



ET1261

Figure 5-1-3. Requestor-MCM-MSU Interface

12. Address Lower Limit. The address lower limit is the most significant six bits of the lowest 20-bit memory address available to this MCM (the least significant 14 bits are assumed to be"0's").

13. Requestor Enable Signal. The MCM sends to the requestor an enable signal which is used under the following conditions to enable or disable communications between the MCM and the requestor:

a. Whenever the MCM is power cycling up or down.
b. Whenever the appropriate requestor inhibit FF is set.

14. MCM Enable Signal. The requestor sends to the MCM an enable signal which is used to enable or disable communications between the requestor and the MCM. This signal is a steady state signal which disables communications whenever the requestor is power cycling up or down.

15. Failure Interrupt 1 Signal (FAL1). The MCM transmits a one-clock period FAIL 1 interrupt signal to the requestor if any of the following errors occur:

a. Control word parity
b. Illegal operation code
c. Wrong MCM
d. Data strobe error
e. Two-bit error
f. Internal error

The MCM fail register will then be loaded with information to facilitate error analysis.

Signal Interface Between MCM and MSU

1. Write Data and Read Data Lines. The data lines are comprised of one overall parity bit, seven check bits used in single bit error correction, a parity bit on just the data word, and a 51-bit data word passed to and from the requestor.

2. Address Lines. These lines are used to transfer a 14-bit address to the MSU to specify the RAM (random access memory) chip locations to be accessed.

3. Initiate Sector. This group of control signals is sent to the MSU to start either a read or write operation.

4. Read Sector Start Load. This control signal loads the starting sector number into the read sector address register within the MSU.

5. Read Sector Start. These signals are the three bit binary sector starting address of the operation.

6. Read Available. The read available informs the MCM of the availability of the data from the selected address.

7. Busy. The MSUs signal to the MCM whether the sectors are busy or idle.

8. Read Write Mode. When high indicates to the MSU that a write operation is to be performed, and when low indicates that a read is desired.

9. Write Strobe. These signals strobe the data from the MCM into the write register of the designated sector.

10. Refresh Request. A signal to the MCM to indicate that a refresh cycle is required.

11. Refresh Allow. A control signal that informs the MSU(s) that a refresh cycle can be performed.

12. MSU Available. This signal indicates to the MCM that power is up in the MSU.

13. Count Most Significant. This signal allows the most significant bit of the sector address to be counted, which allows eight-word phasing.

14. Mode, Read Sector Count (RSC). These control signals are sent to the MSU to enable an eight-megahertz operation in the MSU.

DEFINITION OF MCM OPERATIONS

The various MCM operations are briefly described in the following paragraphs.

1. Data Word Fetch (Single or Multiple Word). This operation is a standard fetch of data. If a multiple word fetch is initiated, the data words are transferred to the requestor at the clock rate and within the limits discussed previously.

2. Fail Word Fetch. This operation is a fetch of the fail register within the MCM. The fail register is cleared as a result of this operation.

3. Single Word Overwrite with Flashback. This operation is a standard write/read operation. The data from the requestor is written into the addressed location. The original data read out of the address location is transferred back (or flashed back) to the requestor.

4. Single Word Protected Write (with/without flashback). This operation is a conditional write of data into memory. The data word transferred by the requestor is written into memory only if the address is not protected (i.e., bit 48 of the original word is "0"). The requestor may indicate whether he requires flashback; however, the MCM will unconditionally flash back data to the requestor.

5. Overwrite (Single or Multiple Word). This operation is a standard write of data into memory. If the operation is an overwrite, the rate of data transfer to the MCM will be controlled by the MCM.

6. Multiple Word Protected Write (1>N>4). This operation is a conditional write of data into memory. The data is written into memory as long as none of

the addresses are protected (i.e., bit 48 = 0 for each address). The requestor will transmit the data only upon request of the MCM. The MCM will transmit a Fail S signal to the requestor if any of the addresses were protected, and it will unconditionally flashback data to the requestor.

7. Load Requestor Inhibit Register. This operation is similar to a single word overwrite with the exception that the data word is transferred to the requestor inhibit register instead of to the MSU. The state of the requestor inhibit register determines which requestors may communicate with the MCM.

8. Load Memory Limit. This operation is similar to a single word overwrite with the exception that the limits field within the data word is transferred to the memory limit register instead of to the MSU. The memory limits consist of the lower and upper MCM memory addresses and the MSUs available for use by the MCM.

MCM LOGIC FUNCTIONS

The basic logic functions of the MCM are priority resolution, data transfer and control, and error protection. (See the block diagram in figure 5-1-4.)

Priority Resolution Logic

Priority resolution logic controls communications between each requestor and the MCM. Lower numbered requestors are given the highest-priority access into memory. Only those requestors selected by the state of the requestor inhibit register are allowed to be serviced by the MCM. The exception to this rule is that through the use of the special request signal, CPMs are able to override the state of the requestor inhibit register. A requestor is not serviced if the requestor interface has failed so that other requestors are not locked out. The highest priority requestor is prevented from obtaining consecutive service if a lower priority requestor is waiting to be serviced.

Data Transfer And Control Logic

The data transfer and control logic provides the sequential control signals required to route the data through the four main data registers (input, output, control word, and memory buffer registers). A brief description of these registers is provided below:

1. Input Register. A 52-bit register used as a temporary buffer register for control words and data words received from memory.

2. Memory Buffer Register. A 60-bit register used as a temporary buffer register for data words transferred to or from MSUs. During a fetch, the fail register information, except bit FR51, is transferred to the memory buffer register before being placed in the output register.

3. Control Word Register. A 52-bit register used to contain the control word transmitted by the requestor.

4. Output Register. A 52-bit register used to buffer data words that are being transmitted to a requestor during a fetch operation. The output register also contains the bit correction logic required to correct one-bit errors detected by the error correction logic.

Error Detection Logic

The error detection logic detects errors in requestor and MSU data and control interface; detects multiple bit errors; corrects one-bit errors that occur in the MSU during a fetch operation; and detects an internal error if a failure occurs in the check/generator logic.

4K and 16K MSU OPERATIONS

The 4K and 16k MSU performs the following operations:

1. Read Cycle. The MSU reads out data (nondestructively) from the memory address defined by the MCM and places the data on the bus to the MCM.

2. Write Cycle. The MSU accepts information from the MCM and stores it into the addressed location.

3. Refresh Cycle. The MSU refreshes all eight sectors simultaneously on receipt of a refresh allow signal from the MCM.

Each MSU contains 128K words of 64 bits each arranged into eight separately controlled, independently operated storage sectors of 16K words.

Each of these sectors is multiplexed in sequence (starting at any one of eight sectors) to perform either a read or write operation. Read data from the sectors is sequentially transmitted from a starting sector over the read data bus via a read data output register. Write data is sent over a write data bus to the sectors within the MSU. The sectors share the following:

1. Address bus -14 bits
2. Write data bus -64 bits
3. Read data bus -64 bits
4. Read data output register -64 bits
5. Read data multiplexer -64 bits
6. Read sector counter -three bits
7. Refresh mechanism.

4K MSU LOGIC FUNCTIONS

The basic logic functions of the 4K MSU are: data transfer and control, data register/multiplex, timing and address, refresh, and storage area. (See block diagram in figure 5-1-5.)

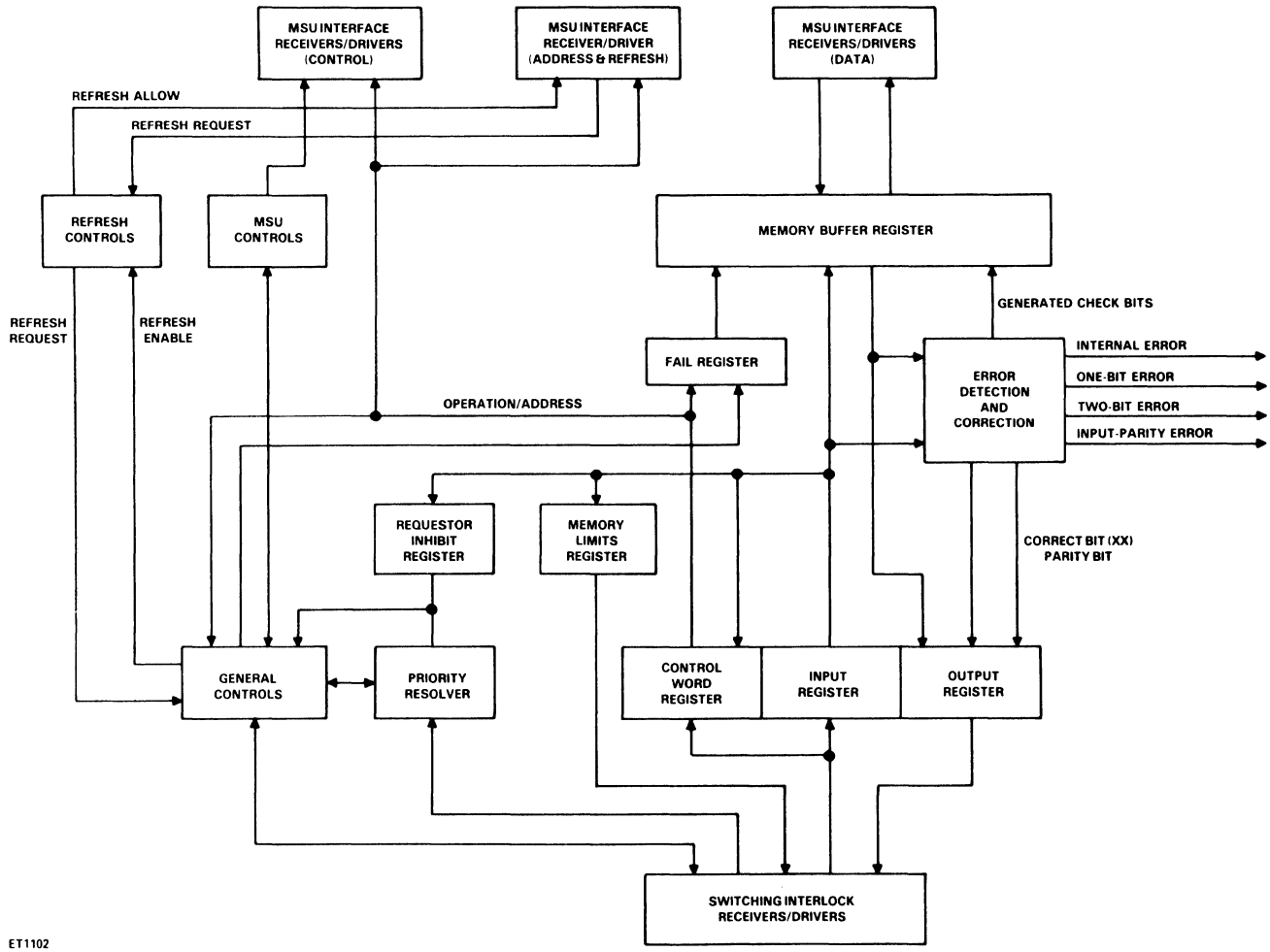


Figure 5-1-4. Memory Control Module Block Diagram

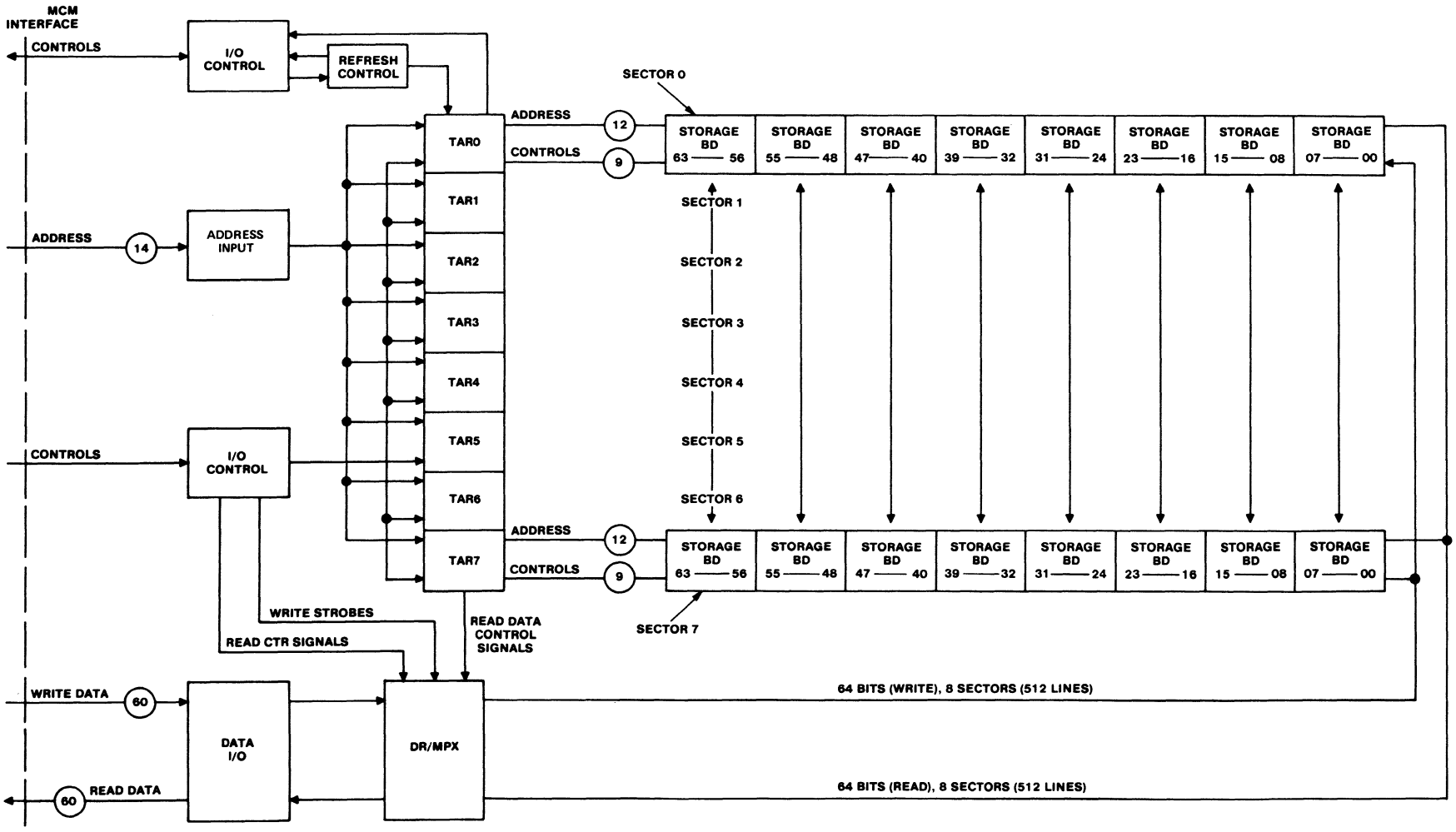
Data Transfer and Control Logic

The data transfer and control logic provides the buffers and latches required to interface the data and control signals between the 4K MSU and MCM. A brief description of these logic areas is given below:

1. Data I/O logic. This logic contains write data buffers, final read data latches and read data cable drivers for transferring data words to or from MCM.

2. I/O Control logic. This logic latches the interface controls necessary to enable write data to the requested sector in storage, to enable read data from the gating, row decoding and address multiplexing circuits. The write or refresh operation.

3. Address Input logic. This logic is used to buffer a 14-bit address word from the MCM.



NOTES

1. 64 STORAGE BOARDS, EACH OF WHICH IS 16K X 8 BITS.
2. 16 DR/MPX BOARDS, EACH OF WHICH LATCHES 4 BITS, 8 SECTORS OF READ AND WRITE DATA.
3. FOR B7800/B7700 CONFIGURATIONS, ONLY A 60 BIT WORD IS STORED IN THE MSU. AS A RESULT, STORAGE BOARD BITS 60 THRU 63 ARE NOT USED.
4. 3 DATA I/O BOARDS.
5. 1 I/O CONTROL BOARD.

EP1226

Figure 5-1-5. 4K Memory Storage Unit Block Diagram

Data Register/Multiplex Logic

Sixteen data register/multiplex (DR/MPX) logic boards are contained in the 4K MSU. Each board latches four bits of write data for each of the eight sectors and then multiplexes data of selected sector to the data latches in the data I/O logic.

Timing and Address Logic

Timing and address (TAR) logic is provided for each sector. The logic is used to generate timing and to latch a 14-bit address word for its sector. The timing section generates various control signals required to perform a read, write or refresh cycle for the selected sector. The address word is routed to chip address and row select circuitry on the eight associated IC storage boards.

Storage Area

The storage area is composed of 64 storage boards. Each storage board contains a four by eight array of 4Kx1 RAM (random access memory) chips. In addition to these RAM chips, each board contains write data buffers, read data gating, row decoding and address multiplexing circuits. The address from TAR is separated into two addresses:

1. Address bits 0 through 11 are used as chip addresses to select one of 4K bit locations in each of the chips.
2. Address bits 11 and 12 are used as a row address to select one of the four rows of chips.

Each row contains eight RAM chips. Each RAM chip is organized as a 64 row by 64 column array and is addressed by a chip row address (6 bits) and a chip column address (6 bits) respectively.

Refresh Logic

The refresh logic is enabled by refresh allow from the MCM. If sectors 0 through 7 are not cycling, a refresh address is accepted by the eight TAR circuits to initiate a refresh cycle. During a refresh cycle, all four chip rows are enabled by a refresh signal. This refresh signal enables all 32 RAM chips to do a refresh in the locations specified by the chip row address. When the RAM chip is in refresh cycle, the entire 64 column (or bits) of the selected row is refreshed. Because the RAM chip has 64 rows, 64 refresh cycles are required to refresh the entire storage board or the entire memory, as all sectors (0 through 7) are enabled during a refresh cycle.

16K MSU LOGIC FUNCTIONS

The basic logic functions of the 16K MSU are: data transfer and control, timing, address and refresh, and storage. (See block diagram, figure 5-1-6.)

DATA TRANSFER AND CONTROL LOGIC

The data transfer and control logic provides the buffers and latches required to interface the data and control signals between the 16K MSU and MCM. A brief description of these logic areas follows:

1. Data I/O logic. This logic contains write data buffers, final read data latches and read data cable drivers for transferring data words to or from the MCM.
2. Control I/O logic. This logic latches the interface controls necessary to enable write data to the requested sector in storage, to enable read data from the selected sector onto the read data bus, and to initiate a read, write, or refresh operation.

TIMING LOGIC

Timing logic is provided for both the odd and even storage boards; it is used to generate address strobes and latching and enabling signals required to perform a read, write, or refresh cycle for the selected sector. The timing logic also controls the sector counter in the control I/O logic, and provides clocks to the address register board.

ADDRESS AND REFRESH LOGIC

The address logic accepts a 14-bit address word from the MCM, latches the address, and routes it to the even or odd storage boards, as determined by the timing logic. The refresh logic is enabled by a refresh allow signal from the MCM. The column and row addresses are multiplexed by the address register board for application to the RAM chips on the storage boards.

STORAGE LOGIC

The storage logic consists of 16 storage boards, each of which contains a four-by-eight array of 16K x 1 RAM (random access memory) chips. In addition to these RAM chips, each board contains read and write data latches and read data output multiplexers. The 14-bit address from the address register is applied in two seven-bit segments to the selected storage boards (row address first, followed by column address). When a refresh cycle is performed, the row address strobes are sent to all eight sectors of the memory and the column addresses are not used.

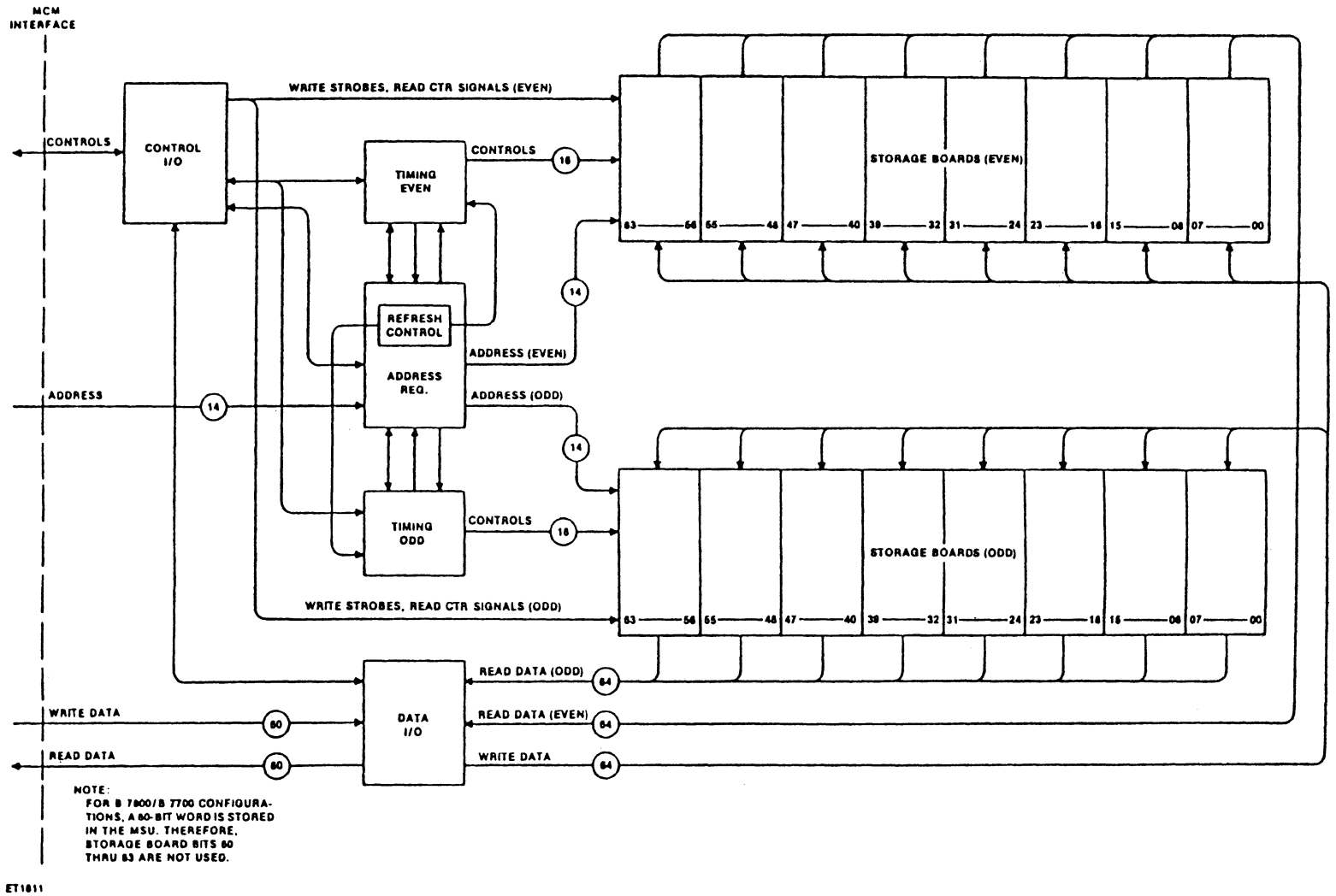


Figure 5-1-6. 16K Memory Storage Unit Block Diagram

CHAPTER 6
MAINTENANCE DIAGNOSTIC PROCESSING

INTRODUCTION

Maintenance diagnostic processing (MDP) is a programmatically controlled maintenance system which is part of the B 7800 Master Control Program. The programs provided by MDP perform the following operations:

1. Real-time static and dynamic testing of the B 7800 mainframe modules (CPM, IOM, and MCM.)
2. Verification and diagnostic testing of the B 7800 logic cards.
3. PROM programming and verification operations.

These programs can be run in the mix along with user programs.

The hardware portion of the MDP is structured to allow any CPM to interface with any other module (CPM, IOM, or MCM) or the PROM programmer and card test facility in order that the module and PROM programmer and card test facility can execute the MDP operations.

In addition to system MDP, a maintenance processor (MP) is provided to perform the same maintenance operations as the system MDP. Whereas, the system MDP makes use of an on-line CPM to test an off-line module, the MP is used off line to test a module and operate the PROM programmer and card test facility.

The MP is completely independent of the B 7800 operating system. The source of the master control program, MDP data base, and MDP test programs for the MP consists of dedicated disks and magnetic tape systems. The MP forms a central testing facility which is permanently connected by use of a maintenance bus to the MDP hardware of a CPM.

An important capability of the MDP allows the field engineer to perform any module panel operation from the supervisory display console or to develop special test operations for use in testing of module functions. This type of testing is established by use of the module interrogation and command interpreter (MICI) program, thus providing the field engineer with an additional aid in isolating the failed logic. The MICI program uses the MDP hardware circuitry of the B 7800 system as the MDP software, but is run independently of the MDP software controls.

In addition to the module testing capabilities, the MDP provides the facility for producing a formatted printout of the CPM and IOM operations. This printout, commonly known as a panel dump, is useful in analyzing module operation at time of failure.

MDP CONFIGURATION

As shown in figure 6-1-1, the CPMs and MP are interconnected via a maintenance bus. The IOMs and MCMs are interfaced with the CPMs via parallel buses through distribution cards to a common daisy-chain bus. Each CPM contains up to four distribution cards, each of which can drive two modules.

The maintenance test logic (MTL) of the CPM is connected with the MDP via a foreplane cable from a master distribution card (MDC) to the MTL control interface card (MTLCIN), as shown in figure 6-1-1. The MDC card is interfaced with maintenance bus through an interface control card (MCIC). This same card is interfaced with IOMs and MCMs by means of distribution cards. It is over these paths that MDP sends data and instructions between various combination of modules.

The card tester and PROM programmer are housed in a maintenance console. Included in the maintenance console is a supervisory console display for use in the operation of the MP system. The maintenance console is interfaced with a CPM in the same manner as an IOM and MCM.

MDP OPERATIONS

All MDP operations are performed by use of the B 7800 operators SPRR 29 and RPRR 29 and are defined by a software-constructed control word. This control word is transferred as literal data to the MDC card of a CPM.

Any operation in which data information is returned from the module under test requires a RPRR 29 operator to transfer the data from the MDC card to the top-of-the-stack.

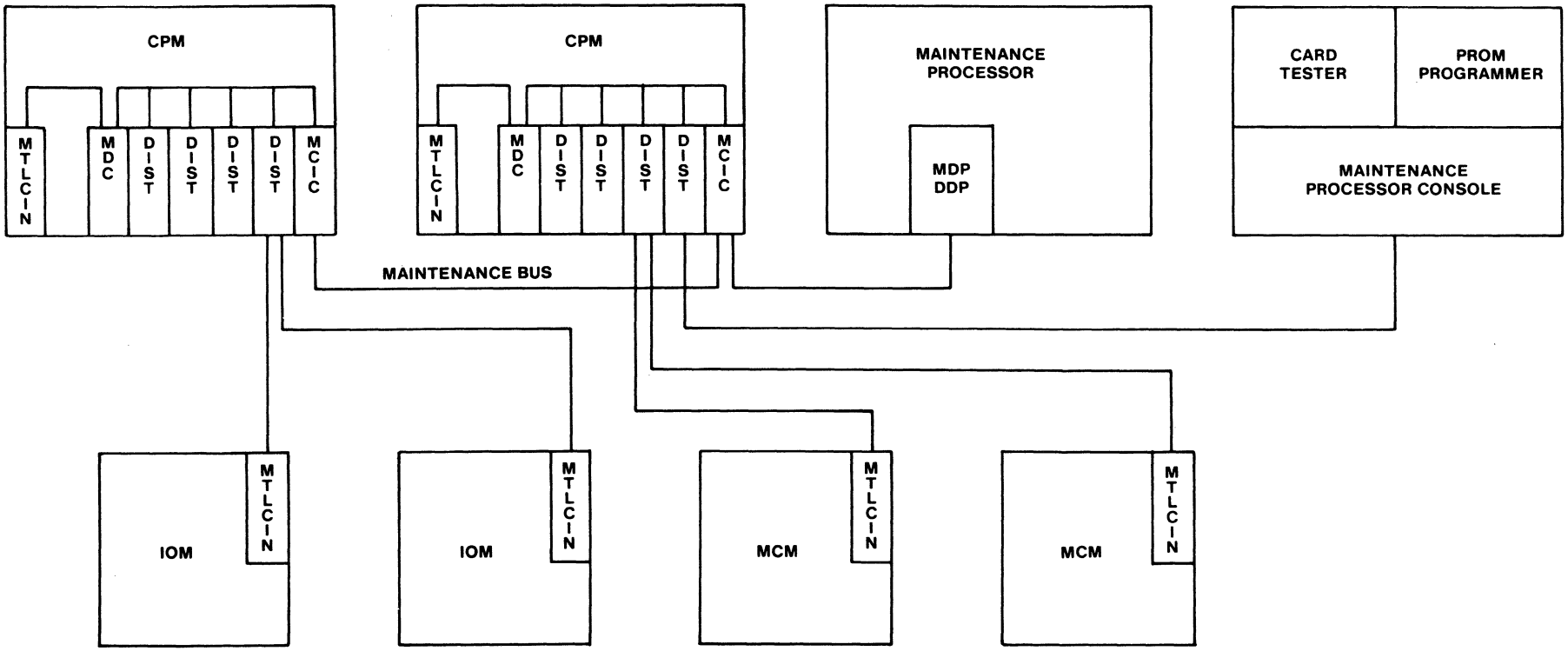
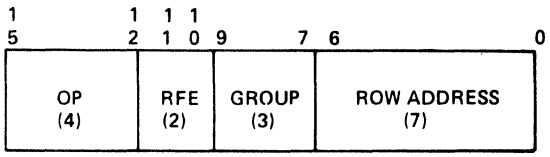


Figure 6-1-1. Typical MDP Configuration

The format of the control word is shown below.



OP: Specifies the operation that is to be performed.

RFE: Reserved for expansion.

GROUP: Indicates a particular group within a row.

- 0 = bits 00 - 09
- 1 = bits 10 - 19
- 2 = bits 20 - 29
- 3 = bits 30 - 39
- 4 = bits 40 - 49
- 5 = bits 50 - 51

ROW ADDRESS: Specifies a panel row address.

Other fields used with various operations are defined as follows:

MID: When a module ID needs to be specified (bits 4:5).

COUNT: When a value has to be given with the operation (bits 6:7).

The following three MDP operations, which are transferred as a control word to an MDC card in a CPM, enable the module under test to execute the operation.

1. Bus operations.
2. Data type operations.
3. Control type operations.

The following paragraphs describe each MDP operation.

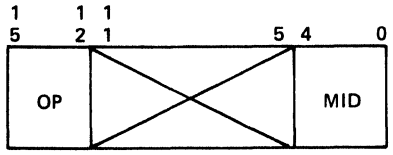
Bus Operations

Because modules are connected to a common MDP bus and can be configured as a split system, the software performs a special bus request operation, prior to initializing normal MDP functions. This bus request operation eventually results in capturing the bus for use in testing a module or in operating the PROM programmer and card test facility. The bus remains captured until explicitly released by the MDP program.

While an MDP program is testing a module, the bus is available for other maintenance operations, such as card testing. To accomplish this, each MDP program is periodically time-sharing the bus while it is executing an operation.

Bus Request Operation

To capture a specific module onto the bus, a SPRR 29 followed by a request control word is performed by the MDP. The control word format for a bus request is shown below.



OP = 7

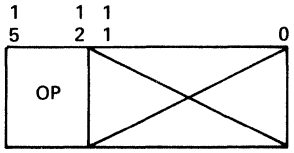
MID = module ID

- 0 - 7 = Requestors (CMP/IOM)
- 8 - 15 = MCM
- 16 = PROM programmer
- 17 = card test

Next an RPRR 29 is executed to determine if the bus has been captured. A Boolean 1 returned indicates that the bus has been captured. A Boolean 0 means that the bus has not been captured and the software must loop back to the SPRR 29 until the capture has been accomplished. Failure to perform this read will cause an error to be generated.

Bus Release Operation

To release the bus, a SPRR 29 followed by a release control word is performed by the MDP. The control word format for a bus release is shown below:



OP = 15

Data Type Operations

All data type operations (fetch, store, and transmit data) are performed or initialized by transferring an MDP control word to the MDC card of the CPM. This transfer is accomplished by executing a SPRR 29 followed by a control word as the argument. Any operation, which results in a return of data from the module under test, requires a RPRR 29 to transfer a ten bit data word from the MDC card to the top-of-the-stack.

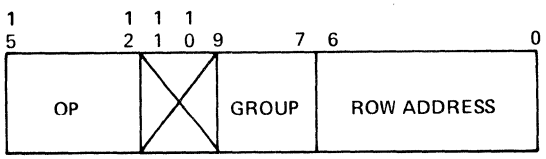
Fetch Operation

The fetch operation is used to fetch a range of flip-flops in a module. The fetch occurs in ten-bit groups in sequential group order.

The instruction sequence requires the operation to be initialized with a single SPRR 29 to transfer the control word, followed by an RPRR 29 for each group to be fetched. The control word address fields

can have any start address, but would normally be initialized to 0. Software must keep count of the groups transferred in order to identify the groups as they are fetched, but can terminate the operation at any time by initiating a new operation. All fetched data is returned to the top-of-stack as a ten-bit right-justified operand. Sequencing of addresses proceeds by counting the group address until group 5 is obtained at which time the group address resets to 0 and the row address is incremented.

The control word format for a fetch operation is shown below:



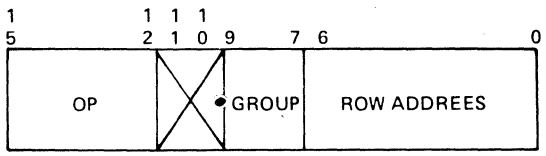
OP = 1
GROUP = starting group (binary)
ROW ADDRESS = starting row (binary)

Store Operation

The store operation is used to store a range of ten-bit data words to a group of ten contiguous panel flip-flops within a module under test. Only binary 1's can be stored; therefore, flip-flops can only be cleared to 0's with a module clear or a row clear operation.

Store is similar to fetch with respect to address sequencing. The control word contains the address of the first group to be stored. All following SPRR words contain a ten-bit data word. Software must record the number of groups stored to determine the termination point of the operation. All data is stored as a ten-bit right-justified operand. The data is not transferred to the module under test when the SPRR 29 of a store op code is executed. Data is transferred to the MDC logic and subsequently to the module under test when the SPRR 29 followed by XMIT data control word is executed.

The control word format for a store operation is shown below:

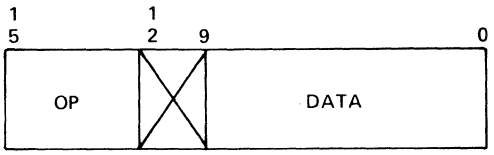


OP = 6
GROUP = starting group (0 - 5) (binary)
ROW ADDRESS = starting row (binary)

XMIT Data Operation

When the MDC card receives an operation code of 6 (STORE OP), the subsequent XMIT DATA operations cause the least significant ten bits of the top-of-stack to be loaded into the MDC card and transmitted to the module under test. If the MDC card does not receive a STORE OP when the XMIT DATA op code is issued, then a fail interrupt will be generated. The store address is incremented on each XMIT DATA operation.

The control word format for a XMIT DATA operation is shown below:



OP = 0

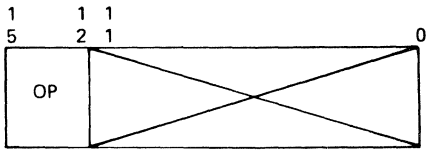
Control Type Operations

The control type operations are used to perform specific module operations. These operations are clear module, clear row, and issue clocks. To execute one of these operations, a SPRR 29 is issued with the control word for that operation.

Clear Module Operation

The clear module operation is used to initialize a module under test by clearing all flip-flops within that module.

The control word format is shown below:

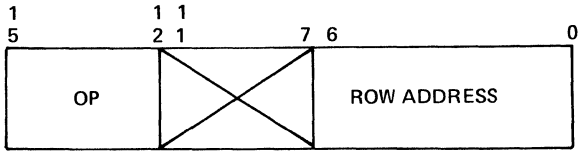


OP = 3

Clear Row Operation

The clear row operation is used to initialize an addressed row of flip-flops by clearing all flip-flops within the row.

The control word format is shown below:

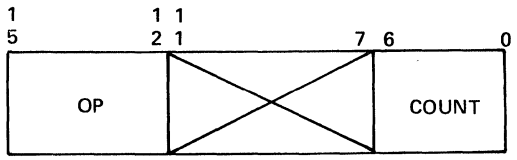


OP = 5

ROW ADDRESS = address of row to be cleared (binary).

Issue Clock(s) Operation

The issue clock(s) operation is used to single pulse a module under test by enabling a variable number of system 8mHz clocks to be generated. This operation requires that the module under test is in single pulse mode.



OP = 4

COUNT = number of clocks to be generated.

MAINTENANCE PROCESSOR

The maintenance processor (MP) is an independent maintenance system which is added to the B 7800 system to augment the system MDP operation capability. The MP performs MDP operations in the same manner as the CPM.

The MP is a modular CPU in terms of subsystems and is configured as shown in figure 6-1-2. The essential components of the MP are the firmware store section used for storing microprogram instructions, a memory used for temporarily storing test data, a processor used for performing operations defined by instructions stored in the firmware, and four I/O control ports (device dependent ports, DDPs) used for interfacing external systems with the processor.

Following is a brief description of the interface of each DDP.

1. Maintenance diagnostic control (MDC-DDP) provides the data and instruction interface for maintenance processing operation.
2. Magnetic tape control (MTC-DDP) provides the tape control interface for MDP data base.
3. Fixed control disk (FCD-DDP) provides the Burroughs Super Mini Disk (BSMD) interface for master control program (MCP) of the MP, MDP test programs, and panel dump programs.

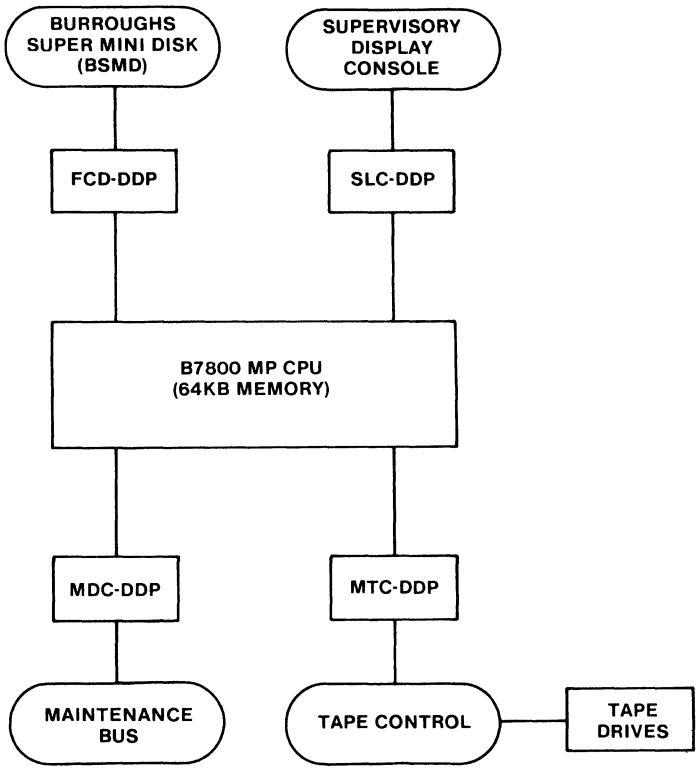


Figure 6-1-2. Maintenance Processor Configuration

4. Single line control (SLC-DDP) provides the supervisory console display interface for operation of the maintenance processing system.

By use of the supervisor display console, the field engineer can request the system supervisor to execute the appropriate MP operating mode. Once an MP operation mode is executed, that mode has control of the system until it is completed. Only certain specified supervisor commands can supersede an MP operating mode.

MP Operating Modes

The MP may be operated in one of 11 modes as follows:

1. Test mode. This causes the MP to capture the specified module and execute the indicated test on the designated device. If tape is used, the system unit number must be supplied. If the file is on disk, the file name is supplied. If the test function fails to capture the specified module, the field engineer will be notified and control will return to the system supervisor. The MP is not capable of removing a module from a running system. The responsibility is upon the field engineer to configure the module out of the system and place it in test status.

2. Module panel dump mode. This causes a panel dump to be taken from the designated module and stored on MP disk. The name of the file must be enclosed in quotes. It is necessary to copy this file onto a tape to be analyzed on a B 7800. If a file already exists with the specified file name, the user is notified and the request is ignored.
3. Copy mode. This transfers the files between MP disk and B 7800 tape. The "FROM" option loads files to disk from tape and the "TO" option dumps files from disk onto tape.
4. PROM programmer mode. This implements the system PROM support capabilities.
5. Remove mode. This allows the field engineer to remove the specified files from MP disk.
6. Print directory mode. This allows the field engineer to request a display of the files currently stored on MP disk.
7. Change mode. This allows the field engineer to change the name of a specified file on MP disk.
8. Dump mode. This modifies the MP restart procedures so that a memory dump may be taken. Invoking this mode causes a DMPFL file to be created on disk.
9. Data Reset mode. This causes the MP date word to be adjusted. The new date is given in the form MM/DD/YY.
10. Diskmap Update mode. This updates the system disk map, and displays both the number of available sectors and the size of the largest available segment.
11. Time Reset mode. This causes the MP time word to be set to a new value. The new time is represented on input as military time (24-hour clock).

Supervisor Commands

The following supervisor commands can be used during execution of the various MP operating modes.

1.Discontinue command unconditionally terminates the function that is being executed and returns control to the system supervisor.
2. Test Option command allows the field engineer to display or modify test options at any time. The "DO" form will display the condition of each option. The "DO+" form will set the specified options and the "DO–" form will reset the specified options.
3. What Date command causes the value of the MP date word to be displayed.
4. What Time command, which causes the value of the MP time word to be displayed.
5. What MCP command causes the version number of the system executive and of the system interpreter to be displayed.
6. Why command displays the current operating status of the MP.

CARD TESTER

The card tester is housed in the maintenance console along with the promburner and is connected to the B 7800 MDP system though the card test/PROM programmer distribution card in the CPM. Once the field engineer has located the suspected circuit card by use of the MDP module testing, the card tester can be used to diagnose and/or verify a faulty component on the card. The essential components of the card tester are three card test logic boards, a card test fixture, and a set of test points used for testing logic which is inaccessible via the card connector pins.

Functional Interface

A simplified diagram of the card test data flow is presented in figure 6-1-3.

The card tester is interfaced with a CPM distribution card via a 40 conductor cable. This interface consists of the following:

1. Ten test data lines used for writing into the card test (CT) register during MDP store operation.
2. Ten scan return lines used for returning output of the card under test to the MDP system during MDP fetch operation.
3. Two row address lines used for selecting 50 bits of the 150-bit CT register.
4. Three group address lines used for selecting a group of 10 bits within the addressed 50 bits of the 150-bit CT register.
5. Card test active line used for enabling card test interface logic. This line is made active by decoding of OP code 17 in the bus request control word. (Refer to paragraph headed Bus Operations for discussion of bus request control word.)
6. CT register enable line used for enabling CT register output to the connector pins of the card under test.
7. Strobe line used for strobing MDP data into the selected bits of the CT register.
8. Four clock mask lines used for selecting the card pin numbers to which clocks are to be applied.
9. Single pulse line used for routing clocks to card tester.

General Operation

The MDP card test contains various test data patterns and special instructions which are processed by the CPM and passed to the card tester.

To capture card tester onto the maintenance bus, the MDP issues a bus request control word with an OP code of 17 to the CPM. When the card tester is captured onto the bus, the row and group address

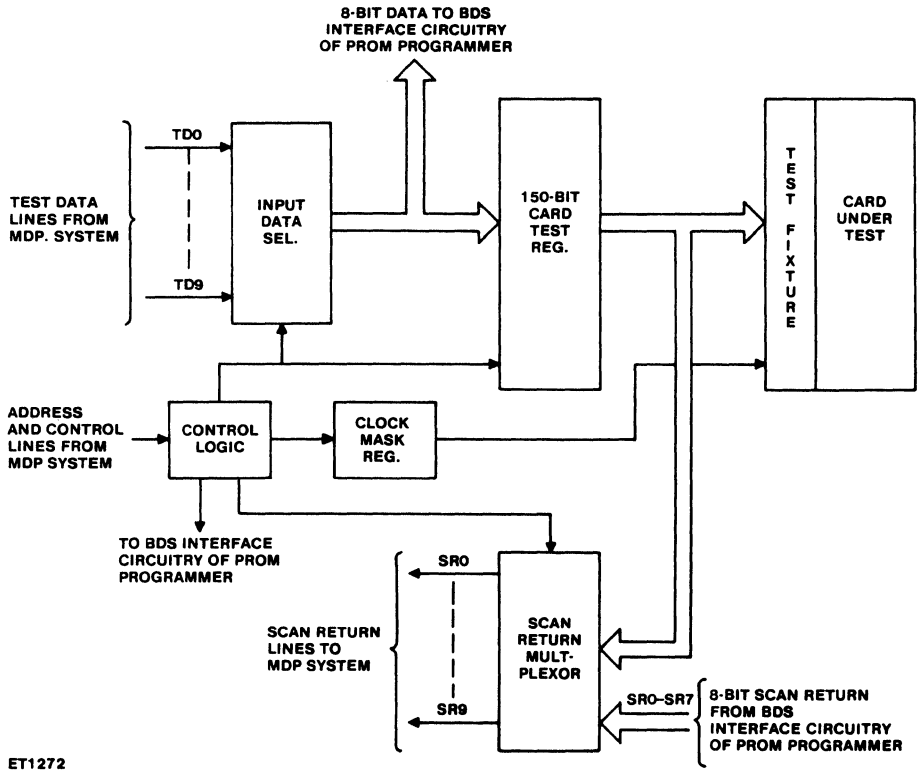


Figure 6-1-3. Card Tester Data Flow, Simplified Diagram

field of a store control word is extracted by the CPM and used to enable desired address lines to the card tester. Then, the test data contained in the transmit data control word is enabled onto the test data lines. This data is loaded into the addressed portion of the CT register by the strobe interface signal. The CPM continues to process MDP control words until desired test data is loaded into the CT register. After CT register has been set up, the register contents are applied to the card under test.

To sample card output pins, the row and group address field of MDP fetch control words is extracted by the CPM and passed to the card tester as an address to the multiplexer. The selected card pin outputs are then routed through the multiplexer to the MDP system where pin verification process begins.

In addition to the test data pattern capabilities, the MDP can supply up to 127 clocks to specific clock pins on the card under test. These pins can be enabled to receive clocks by setting a clock mask bit in the clock mask register. The clock mask bits are set by processing an MDP store and transmit data control word in the CPM. Subsequently, the CPM recognizes the bit value contained in the transmit data control word and activates the corresponding clock mask interface line to the clock mask register. Then, an issue clock control word is processed to enable clocks to be applied to the selected clock pin on the card under test.

PROM PROGRAMMER

Like the card tester, the PROM programmer is housed in the maintenance console and is connected to the B 7800 MDP system through the card test/prom programmer distribution card in the CPM. The PROM programmer portion of this card contains the necessary logic to interface between the MDP bus and the BDS (basic data system) backplane interface of the PROM programmer. This is accomplished by synchronizing the MDP strobe to an internal free-running clock, performing a translation between MDP operations and BDS operations, and generating one cycle of a two-phase clock for each operation to be executed by the PROM programmer.

Basically, the PROM programmer is divided into digital and analog circuitry. The digital circuitry

functions as a programmable controller for analog circuitry. The analog circuitry consists of two power supplies, each capable of being switched on and off to its own programmable voltage level. One voltage supply provides chip supply voltage for both programming and verification. The other supplies the voltage to be applied to the PROM output pin during programming.

Figure 6-1-4 is a simplified block diagram of the PROM programmer showing the general interconnections between the major components. Table 6-1-1 contains a listing of these components and also provides the function of each component.

Functional Interface

There are 21 signals which interface between the distributor card in the CPM and the PROM programmer. These signals are routed through the card tester cards where a CTL to TTL level conversion is performed, as shown in figure 6-1-3. Sixteen of these are data lines: eight lines each for input and output data. Other lines are for control purposes, as shown in figure 6-1-4. Two of these lines (PH-14C-1 and PH-24C-1) are dedicated to the two-phase clock system. Two more lines (CONT4C-1 and D1R-4C-1) are enabled by decoding 2 LSBs in the MDP store control word for PROM programmer and are used to encode the four basic control operations described under paragraph headed MDP/PROM Programming Operations. One line (APRM4BRN) enables the PROM programmer internal logic when it is being addressed by MDP bus request control word with a module ID of 16.

General Operation

The hardware of the PROM programmer has two functions: (1) PROM programming and (2) PROM verification.

To program a PROM, the software executes a sequence of basic PROM programmer operations. Each basic operation consists of a series of BDS operations, each generated by executing a strictly defined set of MDP instructions. (Refer to paragraph headed MDP/PROM Programming Operations.)

Table 6-1-1. PROM Programmer Components

Component	Function
Command register and decoder	To store and decode the OP code field of a PROM programmer control word.
Address register	1. To address a panel LED after execution of CLEAR OP. 2. To address parameter registers for loading of digital programming values prior to LDREG command. 3. To supply PROM address during programming (PROM/NOOP) and read operations.
Parameter Registers	
1. V_{CC} register	Stores digital value of V_{CC} to be applied to the PROM supply voltage pin during programming or verification. This digital value is an input to a D/A converter to provide an analog voltage level.
2. V_P register	Stores digital value of V_P to be applied to the PROM output pin during programming.
3. T_{CC} register	a. Stores digital value of pulse width of the V_{CC} pulse to be applied to the PROM during programming. b. Stores digital value of multiplying factor to be applied to both the T_{CC} and T_P counts.
4. T_P register	Stores digital value of pulse width of V_P pulse to be applied to the PROM output pin during programming.
Bit select register	To drive a group of eight relays which connect V_P output to the PROM output pin to be programmed.
Program register	To store the values of four selectable options during PROM bit programming.
Data output register	To store the PROM output data for interface to data bus.
Clock divider and selector (part of BDS interface logic)	To divide the phase 2 clock by digitally selectable values of 1, 8, 32, or 64 for use by the T_{CC} and T_P counters.
Read flip-flop	To activate turn-on of the V_{CC} supply voltage for continuous application of V_{CC} to the PROM for verification.

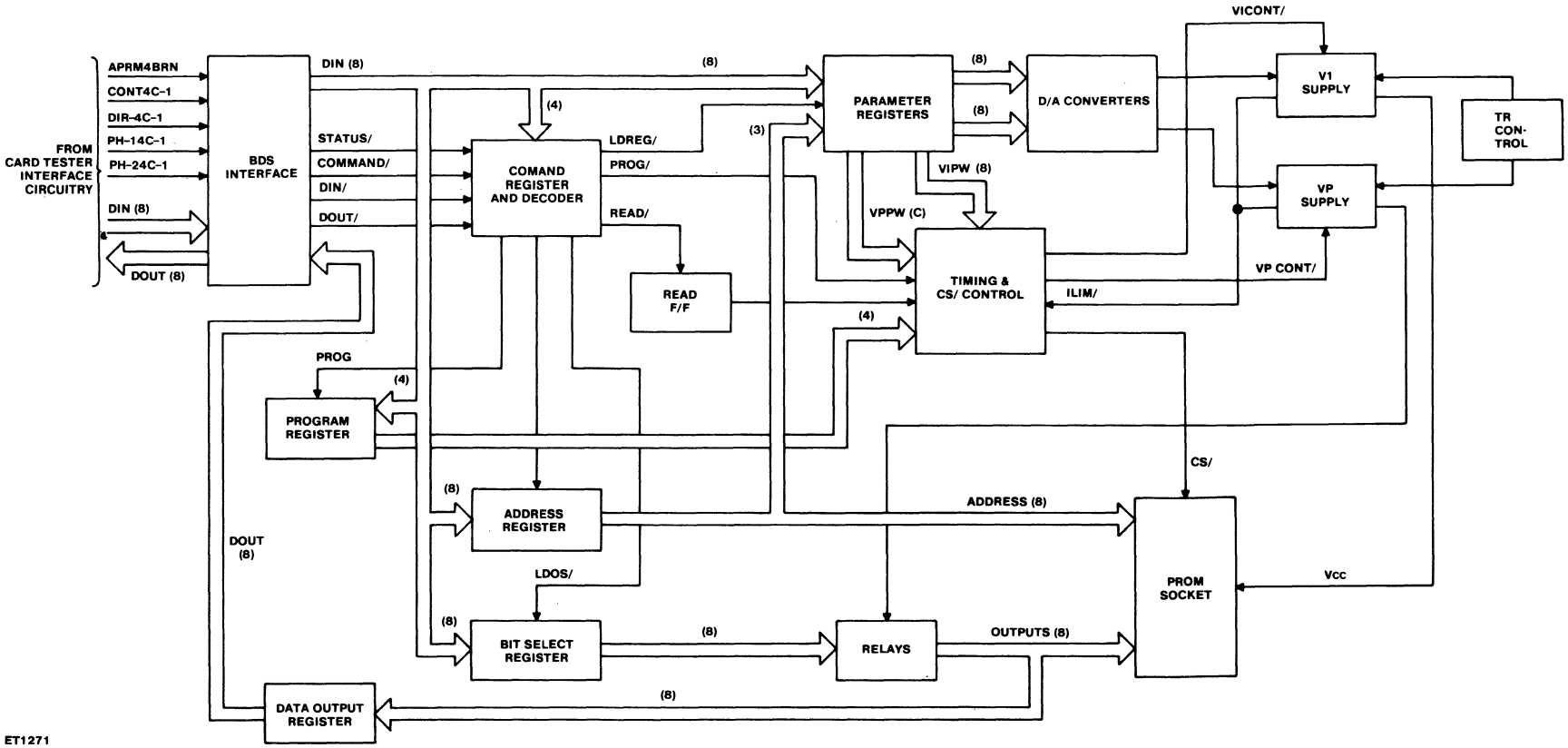


Figure 6-1.4. PROM Programmer Block Diagram

The first operation in programming is to clear the PROM Programmer and turn on the LED identifying the proper front panel socket. At this point in the operation, there is a pause in the software and the user is requested to insert the PROM in the selected socket and acknowledge this action by means of the SPO. Then the parameter registers are loaded with the values applicable to the particular PROM. (The information in these registers remains valid throughout the programming cycle.)

The next operation is to load the bit select register. This register drives a bank of relays which connect the Vp supply to the PROM output pin to be programmed. Because of the mechanical nature of these relays, the PROM is programmed by bit columns (rather than by words of data) to minimize operation of the relays.

Following the load of the bit select register, the PROM address is loaded, and a PROG instruction followed by a NO-OP is issued. These instructions activate the timing and programming pulse generation circuitry and burn the selected bit. A STATUS READ is performed to detect when the timing is complete and the programming of the next bit may proceed. The sequence of load address, PROG, NO-OP, and STATUS READ continues until the bit column is complete. Then the bit select register is loaded with the next column and the addresses are cycled through again. This process continues for all output bits of the PROM until the programming is complete.

To verify a PROM, the PROM data is checked for validity with the chip supply voltage at both its high and low limits. The basic sequence consists of clearing the unit, loading the desired value of Vcc into its parameter register, issuing a READ command to apply Vcc continuously to the PROM, and finally, cycling through the addresses to obtain the data from the PROM.

MDP/PROM Programmer Operations

There are four basic control operations performed by the BDS interface of the PROM programmer. These operations are set up by the processing of certain MDP control words in the CPM. The four control operations are listed below.

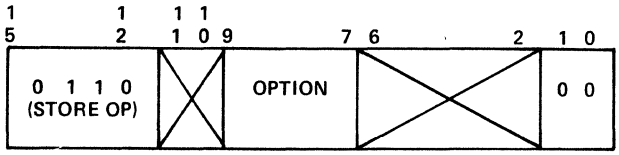
1. Control Word In sequence (CWI) causes a prescribed control word to be loaded into the PROM programmer.
2. Data Word In sequence (DWI) causes an eight-bit data word to be transferred to the PROM programmer.
3. Read Status sequence returns the low order data bit from the programmer to the MDP as a READY status condition.
4. DATA OUT sequence returns the PROM data to the MDP system for verification purposes.

The following paragraphs describe each operation and the sequence in which the MDP control words are executed.

Control Word In Sequence

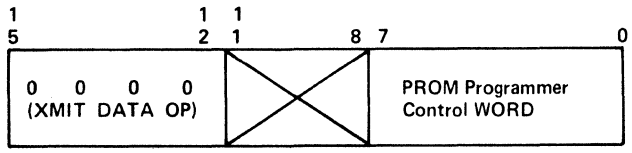
To load a control word into the PROM programmer command register, the following two MDP control words are executed in sequence:

1.MDP STORE



The standard MDP store control word redefined, as shown above, causes the proper PROM programmer/MDP interface control signals to be generated to prepare for the loading of a control word into the PROM programmer command register. The actual loading of the control word is accomplished with an MDP XMIT data control word. Option Field: 000 or 101 - (Refer to note under paragraph headed Data Word In Sequence.)

2. MDP XMIT DATA

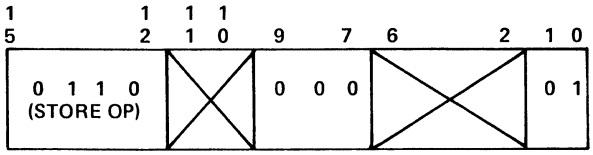


The MDP XMIT DATA control word following the MDP STORE causes the PROM programmer control word to be loaded into the PROM programmer command register. The PROM programmer control word utilizes the low order eight bits of the data field of the MDP XMIT data control word.

Data Word In Sequence (DWI)

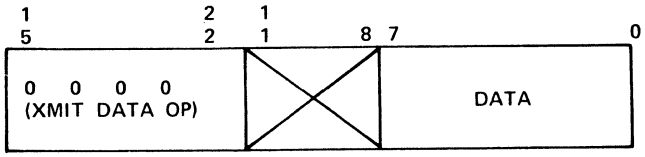
To load a data word into a PROM programmer register, address register, or bit select register, the following MDP commands are executed in order. The destination register for the data is defined by the preceding CWI sequence.

1. MDP STORE



The standard MDP STORE control word redefined as shown above causes the proper PROM programmer/MDP interface control signals to be generated to prepare for the loading of a data word into a destination register (parameter register, address register or bit select register) previously selected by the last CWI operation. The actual loading of the data word is accomplished with an MDP XMIT data control word.

2. MDP XMIT DATA



The MDP XMIT data control word following the previously defined MDP STORE will accomplish the desired data load. The data field in this case is the low order eight bits of the standard MDP XMIT data control word.

NOTE

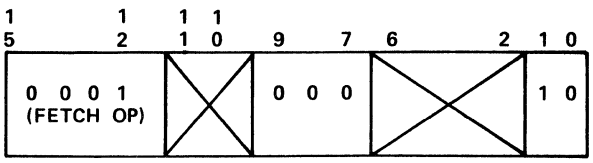
Many PROM programmer operations require a CWI followed by a DWI which indicates the following sequence of MDP commands:
STORE - XMIT
DATA - STORE - XMIT DATA.

If the option field is 101, the following sequence of MDP commands will perform the same function: STORE - XMIT DATA - XMIT DATA.

Read Status Operation

Execution of the following MDP command causes a status read to be performed. This operation is performed after every load program register/NO-OP sequence to determine when the internal PROM bit burning cycle is complete so that programming of the next bit may proceed.

MDP FETCH



Execution of an MDP FETCH containing a control word formatted as shown above returns the state of the PROM programmer ready bit into position 0 of the returned data word. All other bits in the returned data (status) word are invalid. Only the low order eight bits of the returned data word are used.

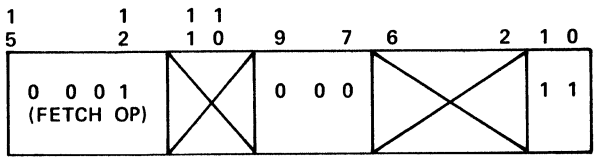
NOTE

No more than six RPRR 29 operators are issued for a given fetch so that the low order bits of the MDP FETCH control word remain valid.

Data Out Operation

Execution of the following MDP command performs an MDP read of PROM data output. This operation is used to retrieve data for verify operations.

MDP FETCH



MODULE INTERROGATION AND COMMAND INTERPRETER PROGRAM

The module interrogation and command interpreter (MICI) program allows the manipulation, control, interrogation, and display of B 7800 mainframe modules from a standard system ODT. MICI also has all the capabilities provided by the front panel switches of the B 7800 mainframe modules.

Where possible the reserved names used for flip-flop and register identification have been derived from the actual name of the element. MICI obtains these names, along with other miscellaneous data, from an information file titled MDP/CIF. This file must be present for MICI to be able to run. Also contained in this file are a series of displays for the various modules.

The MICI commands are grouped into three categories as follows:

1. Module interrogation group.
2. Module command group.
3. Interpreter directive group.

The following paragraphs describe each category and the various commands from which the module functions are executed.

Module Interrogation Group

The module interrogation group of commands causes the state of a module to be displayed. Available displays range from the complete module panel dump which goes to the printer to a simple request to display only one item to the screen. The interrogation mode is a multi-item display set up to display some functional portion of the subject module. This display allows up to sixty (60) elements of the module being interrogated to be displayed at a time. This is accomplished by dividing the last twenty (20) lines of the ODT screen into three equal columns, thereby providing sixty "slots" in which a module element may be displayed along with its identifier.

Display Command

The DISPLAY command causes the items listed to be displayed. If module is not specified, the default will be used. If module cannot be determined, a syntax error will occur.

Items to be displayed can be storages, flip-flops, registers, switches, literal strings, blanks or predefined displays. The displays can be created by using the MAKEDISPLAY command. In addition, a predefined display can be specified if it is contained in MDP/CIF file.

The option of displaying data contained within local storages of the various modules is provided. The display of storage data mixed with other elements is specifically not allowed. The syntax for displaying storage data is structured such that a single index within a storage array may be specified, or a series of storage arrays all using the same storage index may be used, or both types may be mixed. An example of a storage array is the IOM active channel stack or the CPM display registers. The storage index is a hex number for the CPM and MCM and a decimal number for the IOM. The capability also exists to display a portion (field) of a storage element.

Dump Command

The DUMP command causes the state of the module under test to be dumped, analyzed and printed.

This command is mechanized so that the printout is actually created by SYSTEM/MDUTILITY. The default MDUTILITY printout will be provided, or the "KWIK" form may be requested.

Module Command Group

The module command group of commands causes the state of the subject module to be altered.

When a command is directed to a module, no special steps of preparation are performed upon that module. Therefore, the responsibility is upon the user to properly condition the module to achieve the desired results (it is meaningless to issue 15 clocks to a module which is not in single pulse). It also should be noted that when the interpreter ceases operation, no special shutdown is performed upon the module.

Clear Command

The CLEAR command resets all flip-flops, which are capable of being reset, in the module under test. This command provides the same function as the CONTROL CLEAR button on the module panel. The SWITCH form of the command turns off all switches which are capable of being turned-off in the module under test. Because the SWITCH form of the command turns off (disables) the switch register (SWRGE) (for the IOM and MCM only), any future display of switches is of the panel switches. SWRGE can be set to display the soft switches.

Load Command

The LOAD command causes the indicated file to be loaded to the module specified (for MCM only). The file to be loaded may be specially formatted code file or a SYSTEST Confidence file.

Pulse Command

The PULSE command causes one or more clocks to be sent to the module under test. This command provides the same function as the SINGLE PULSE pushbutton on the module panel.

It should be noted that if the module being "pulsed" is not in a single pulse state (its clock has stopped) this command is essentially a NO-OP.

A decimal value may be specified as a parameter to the PULSE command. This number should be in the range of 1 through 127, and is interpreted as the quantity of clocks to be issued. The absence of a value implies a quantity of one.

Set Command

The SET command causes the indicated elements to be set; that is, a flip-flop is set to TRUE, a register is set to a specified value, and a switch is set to ON.

In the case of a register, it is possible that an implied RESET will be performed, because of the manner in which the Maintenance Test Logic (MTL) functions. For example, if a register, which has the value "99", is to be set to "0", the operation would be performed as follows:

1. Read the MTL row(s) to obtain the existing state.
2. Clear the MTL row.
3. Modify the local copy.
4. Set the MTL row to the modified local copy.

Because of the manner in which the MTL is implemented on the various modules, the SET command cannot always perform the requested operation. Any such restriction is treated as a syntax error. Thus, the processing of the SET command is terminated. A group of items can be processed as if each item is contained in its own command.

For example, the command:
SET PINT, EUP, CM1
is processed as:
SET PINT SET EUP SET CM1

When a register is set to a value, that value is interpreted as a string of hex characters. If the hex string is less than the capacity of the register, the string is applied right-justified with leading zeroes inserted into the unspecified portion of the register. Thus, a 20-bit register containing 3FFFF is set to 00321 upon receipt of a SET command with a parameter of 321.

A special variation of the SET command allows data to be written to memory. It should be noted that no adjustments are made for address limit registers, so the address given must be consistent with the module limits. There is no requirement to have the MCM inhibit any special condition. The data format is represented as a thirteen-digit hex word, with the most significant digit the tag. If fewer than thirteen digits are given, the data is entered right-justified and a tag of zero is provided.

Reset Command

The RESET command causes the indicated storage elements to be reset; that is, a flip-flop is set to false, a register is set to zero, and a switch is set to OFF.

Because of the manner in which the MTL is implemented, a register or flip-flop RESET could involve the following sequence of operations:

1. Read the entire MTL row.
2. Clear the MTL row.
3. Reset the indicated terms in the local copy.
4. Set the MTL row to the local modified copy.

In some cases, only step 2 is required. The hardware restrictions, discussed under the heading Set Command, also apply to the RESET command. As in the case of the SET command, the inability to perform an operation results in a syntax error. Thus, the processing of the RESET command is terminated.

Test Command

The TEST command provides a means of making a conditional transfer of control, based on the state or value of an element. If the state or value is a true condition, the MICI command specified is processed; if not, the next input command is processed.

Interpreter Directive Group

The interpreter directive group of commands establishes the operating mode of the interpreter and the various options available. These commands are the following:

1. AUTODISPLAY command causes MICI to automatically initiate an input that has caused action to be performed on a module.

2. CANCEL command is used to terminate a command that has been partially processed.

3. DEFAULTBOX command causes MICI to save the indicated module information. This information can then be utilized for any module-oriented command where the module information has been omitted.

4. EXECUTE command causes execution of a sequenced data file of a series of MICI commands.

5. HELP command displays the available commands, display identifiers, execute identifiers, storage identifiers, and sub-module identifiers.

6. MAKEDISPLAY command specifies to MICI a particular view of a module which is to be made available for use. Up to fifty (50) views can be specified. The created display is only valid for that particular execution of MICI. The DISPLAY command is used to invoke the created display.

7. RECALL command displays a specified number of lines of input; up to 20 lines may be recalled.

8. RELEASE command causes any printer backup files created by the MICI to be printed.

9. REPEAT command causes a specified MICI command(s) to be repeated a specified number of times.

10. SHOW DISPLAY command allows for the interrogation and possible subsequent modification of a previously "made" or "loaded" display. Execution of this command causes MICI to present all the items specified for that display.

11. STOP command causes the MICI interpreter to terminate operation.

12. TERM command is used to control the format of displays that appear on the supervisory console.

13. TRANSFER command provides a means of transferring control to a specified software label in order to bypass the execution of MICI commands.

14. VERSION command is used to display the Maintenance Diagnostic Test (MDT) level, the CIF file level, and MICI level on the input device.

15. WAIT command provides a means of waiting a fixed number of seconds, or waiting for the operator to provide a null input, before processing is continued.

APPENDIX A
COLLATING INFORMATION

All characters are collated according to their internal binary value. Because the B 7800 has the capability of representing characters internally in BCL, EBCDIC, or USASCII, and because characters are collated according to their internal representation (not necessarily the same as their external mode) a variety of collating sequences is possible. The following table may be used to determine the applicable collating sequence.

Input Mode	Output Mode	Internal Mode	Collating Sequence
BCL	BCL	BCL	BCL (BCL internal)
BCL	EBCDIC	EBCDIC	BCL Translated to EBCDIC
BCL	BCL	EBCDIC	BCL Translated to EBCDIC
EBCDIC	EBCDIC	EBCDIC	EBCDIC
EBCDIC	BCL	EBCDIC	BCL Translated to EBCDIC
EBCDIC	USASCII	EBCDIC	USASCII Translated to EBCDIC
USASCII	USASCII	USASCII	USASCII
USASCII	EBCDIC	EBCDIC	USASCII Translated to EBCDIC
USASCII	BCL	USASCII	BCL Translated to USASCII
USASCII	EBCDIC	USASCII	USASCII Translated to EBCDIC

CHARACTER REPRESENTATION

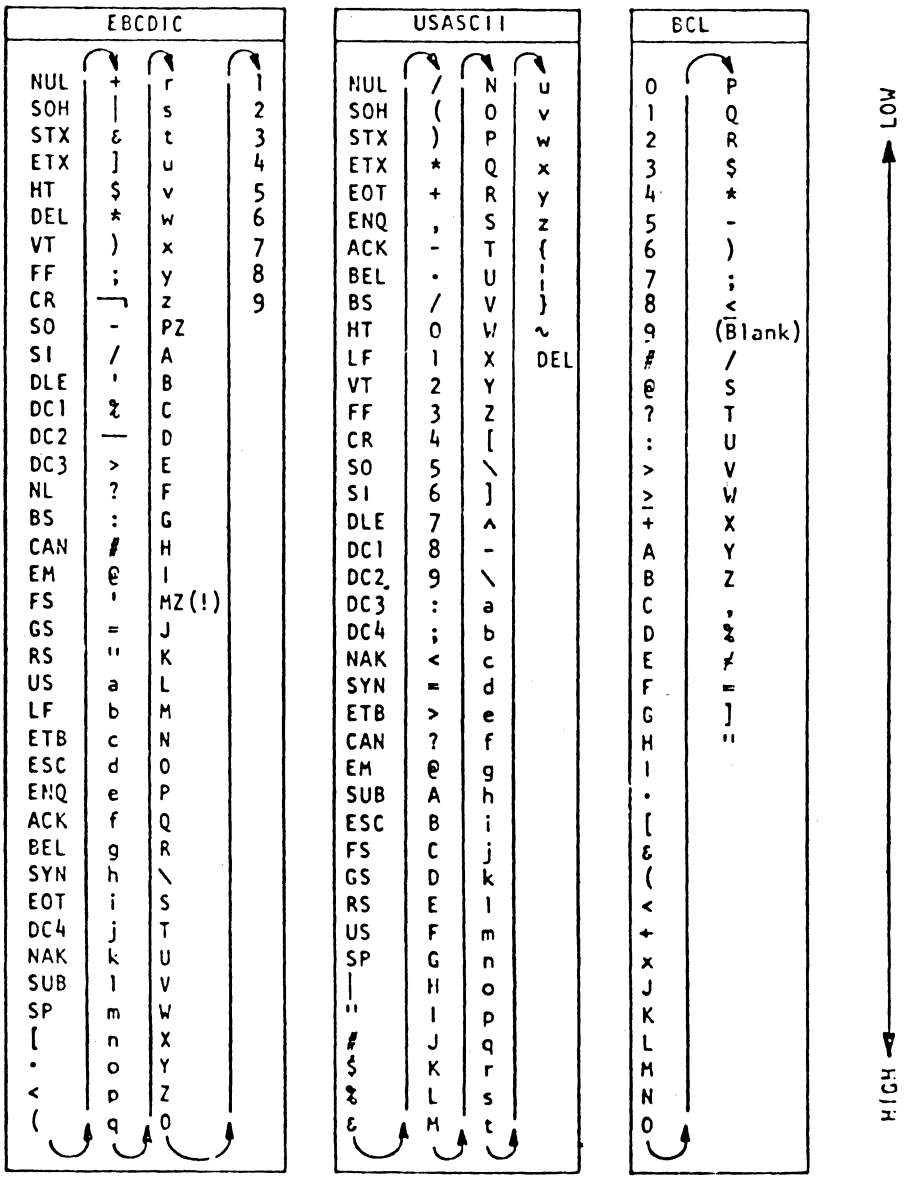
The BCL, EBCDIC, and USASCII graphics are the same except as follows:

BCL	EBCDIC	USASCII
≥	' (single quote)	'
x (multiply)	! or | or MZ	}
≤	¬ (not)	^
≠	_ (underscore)	_
←	| (or)	!
+	PZ (+)	{
↓	<	
↑	>	
<	+	
>	-	

A BCL plus sign is never translated to an EBCDIC PZ (plus zero) sign, although the EBCDIC PZ is translated to a BCL plus sign.

EBCDIC 1110 0000 is translated to BCL 00 0000 with an additional flag bit on the next to most significant bit line (7th bit). As the print drums have 64 graphics and space this signal can be used to print the 64th graphic. The 64th graphic is a "CR" for BCL drums and a "¢" for EBCDIC drums.

COLLATING SEQUENCES



EBCDIC COLLATING SEQUENCE

EBCDIC Character	Hex. Code	Internal Code	Card Code Zone Number	
NUL	00	0000 0000	12-0-9-	8-1
SOH	01	0000 0001	12-9-	1
STX	02	0000 0010	12-9-	2
ETX	03	0000 0011	12-9-	3
HT	05	0000 0101	12-9-	5
DEL	07	0000 0111	12-9-	7
VT	0B	0000 1011	12-9-	8-3
FF	0C	0000 1100	12-9-	8-4
CR	0D	0000 1101	12-9-	8-5
SO	0E	0000 1110	12-9-	8-6
SI	0F	0000 1111	12-9-	8-7
DLE	10	0001 0000	12-11-9-	8-1
DC1	11	0001 0001	11-9-	1
DC2	12	0001 0010	11-9-	2
DC3	13	0001 0011	11-9-	3
NL	15	0001 0101	11-9	5
BS	16	0001 0110	11-9-	6
CAN	18	0001 1000	11-9-	8
EM	19	0001 1001	11-9-	8-1
FS	1C	0001 1100	11-9-	8-4
GS	1D	0001 1101	11-9-	8-5
RS	1E	0001 1110	11-9-	8-6
US	1F	0001 1111	11-9-	8-7
LF	25	0010 0101	0-9-	5
ETB	26	0010 0110	0-9-	6
ESC	27	0010 0110	0-9-	7
ENQ	2D	0010 1101	0-9-	8-5
ACK	2E	0010 1110	0-9-	8-6
BEL	2F	0010 1111	0-9-	8-7
SYN	32	0011 0010	9-	2
EOT	37	0011 0111	9-	7
DC4	3C	0011 1100	9-	8-4
NAK	3D	0011 1101	9-	8-5
SUB	3F	0011 1111	9-	8-7
SP	40	0100 0000	(No Punches)	
[	4A	0100 1010	12-	8-2
.	4B	0100 1011	12-	8-3
<	4C	0100 1100	12-	8-4
(	4D	0100 1101	12-	8-5
+	4E	0100 1110	12-	8-6
| (¬)	4F	0100 1111	12-	8-7

LOW ▲ HIGH

EBCDIC Character	Hex. Code	Internal Code	Card Code Zone Number	
&	50	0101 0000	12-	-
]	5A	0101 1010	11-	8-2
$	5B	0101 1011	11-	8-3
*	5C	0101 1100	11-	8-4
)	5D	0101 1101	11-	8-5
;	5E	0101 1110	11-	8-6
¬ (≤)	5F	0101 1111	11-	8-7
- (Dash)	60	0110 0000	11-	-
/	61	0110 0001	0-	1
, (Comma)	6B	0110 1011	0-	8-3
%	6C	0110 1100	0-	8-4
_ (≠)	6D	0110 1101	0-	8-5
>	6E	0110 1110	0-	8-6
?	6F	0110 1111	0-	8-7
:	7A	0111 1010	-	8-2
#	7B	0111 1011	-	8-3
@	7C	0111 1100	-	8-4
' (≥)	7D	0111 1101	-	8-5
=	7E	0111 1110	-	8-6
"	7F	0111 1111	-	8-7
a	81	1000 0001	12-0-	1
b	82	1000 0010	12-0-	2
c	83	1000 0011	12-0-	3
d	84	1000 0100	12-0-	4
e	85	1000 0101	12-0-	5
f	86	1000 0110	12-0-	6
g	87	1000 0111	12-0-	7
h	88	1000 1000	12-0-	8
i	89	1000 1001	12-0-	9
j	91	1001 0001	12-11-	1
k	92	1001 0010	12-11-	2
l	93	1001 0011	12-11-	3
m	94	1001 0100	12-11-	4
n	95	1001 0101	12-11-	5
o	96	1001 0110	12-11-	6
p	97	1001 0111	12-11-	7
q	98	1001 1000	12-11-	8
r	99	1001 1001	12-11-	9

LOW ▲ HIGH

EBCDIC COLLATING SEQUENCE (Cont)

EBCDIC Character	Hex. Code	Internal Code	Card Code Zone Number
s	A2	1010 0010	11-0- 2
t	A3	1010 0011	11-0- 3
u	A4	1010 0100	11-0- 4
v	A5	1010 0101	11-0- 5
w	A6	1010 0110	11-0- 6
x	A7	1010 0111	11-0- 7
y	A8	1010 1000	11-0- 8
z	A9	1010 1001	11-0- 9
PZ (+)	C0	1100 0000	12-0
A	C1	1100 0001	12- 1
B	C2	1100 0010	12- 2
C	C3	1100 0011	12- 3
D	C4	1100 0100	12- 4
E	C5	1100 0101	12- 5
F	C6	1100 0110	12- 6
G	C7	1100 0111	12- 7
H	C8	1100 1000	12- 8
I	C9	1100 1001	12- 9
MZ (!)	D0	1101 0000	11- 0
J	D1	1101 0001	11- 1
K	D2	1101 0010	11- 2
L	D3	1101 0011	11- 3
M	D4	1101 0100	11- 4
N	D5	1101 0101	11- 5
O	D6	1101 0110	11- 6
P	D7	1101 0111	11- 7
Q	D8	1101 1000	11- 8
R	D9	1101 1001	11- 9
\ (CR) (¢)	E0	1110 0000	0- 8-2
S	E2	1110 0010	0- 2
T	E3	1110 0011	0- 3
U	E4	1110 0100	0- 4
V	E5	1110 0101	0- 5
W	E6	1110 0110	0- 6
X	E7	1110 0111	0- 7
Y	E8	1110 1000	0- 8
Z	E9	1110 1001	0- 9

LOW ▲ ▼ HIGH

EBCDIC Character	Hex. Code	Internal Code	Card Code Zone Number
0	F0	1111 0000	- 0
1	F1	1111 0001	- 1
2	F2	1111 0010	- 2
3	F3	1111 0011	- 3
4	F4	1111 0100	- 4
5	F5	1111 0101	- 5
6	F6	1111 0110	- 6
7	F7	1111 0111	- 7
8	F8	1111 1000	- 8
9	F9	1111 1001	- 9

LOW ▲ ▼ HIGH

BCL COLLATING SEQUENCE (BCL INTERNAL)

BCL Character	BCL Octal	BCL Hex	BCL Internal BA 8421	BCL External BA 8421	Card Code Zone Number
0	00	00	00 0000	00 1010	- 0
1	01	01	00 0001	00 0001	- 1
2	02	02	00 0010	00 0010	- 2
3	03	03	00 0011	00 0011	- 3
4	04	04	00 0100	00 0100	- 4
5	05	05	00 0101	00 0101	- 5
6	06	06	00 0110	00 0110	- 6
7	07	07	00 0111	00 0111	- 7
8	10	08	00 1000	00 1000	- 8
9	11	09	00 1001	00 1001	- 9
#	12	0A	00 1010	00 1011	- 8-3
@	13	0B	00 1011	00 1100	- 8-4
?	14	0C	00 1100	00 0000	All other card codes
:	15	0D	00 1101	00 1101	- 8-5
>	16	0E	00 1110	00 1110	- 8-6
≥	17	0F	00 1111	00 1111	- 8-7
+	20	10	01 0000	11 1010	12 0
A	21	11	01 0001	11 0001	12 1
B	22	12	01 0010	11 0010	12 2
C	23	13	01 0011	11 0011	12 3
D	24	14	01 0100	11 0100	12 4
E	25	15	01 0101	11 0101	12 5
F	26	16	01 0110	11 0110	12 6
G	27	17	01 0111	11 0111	12 7

LOW ▲ ▼ HIGH

BCL COLLATING SEQUENCE (BCL INTERNAL) (Cont)

BCL Character	BCL Octal	BCL Hex	BCL Internal BA 8421	BCL External BA 8421	Card Code Zone	Number
H	30	18	01 1000	11 1000	12	8
I	31	19	01 1001	11 1001	12	9
.	32	1A	01 1010	11 1011	12	8-3
[	33	1B	01 1011	11 1100	12	8-4
&	34	1C	01 1100	11 0000	12	-
(	35	1D	01 1101	11 1101	12	8-5
<	36	1E	01 1110	11 1110	12	8-6
+	37	1F	01 1111	11 1111	12	8-7
x(Mult.)	40	20	10 0000	10 1010	11	0
J	41	21	10 0001	10 0001	11	1
K	42	22	10 0010	10 0010	11	2
L	43	23	10 0011	10 0011	11	3
M	44	24	10 0100	10 0100	11	4
N	45	25	10 0101	10 0101	11	5
O	46	26	10 0110	10 0110	11	6
P	47	27	10 0111	10 0111	11	7
Q	50	28	10 1000	10 1000	11	8
R	51	29	10 1001	10 1001	11	9
S	52	2A	10 1010	10 1011	11	8-3
*	53	2B	10 1011	10 1100	11	8-4
-	54	2C	10 1100	10 0000	11	-
)	55	2D	10 1101	10 1101	11	8-5
;	56	2E	10 1110	10 1110	11	8-6
≤	57	2F	10 1111	10 1111	11	8-7
Blank	60	30	11 0000	01 0000	-	-
/	61	31	11 0001	01 0001	0	1
S	62	32	11 0010	01 0010	0	2
T	63	33	11 0011	01 0011	0	3
U	64	34	11 0100	01 0100	0	4
V	65	35	11 0101	01 0101	0	5
W	66	36	11 0110	01 0110	0	6
X	67	37	11 0111	01 0111	0	7
Y	70	38	11 1000	01 1000	0	8
Z	71	39	11 1001	01 1001	0	9
,	72	3A	11 1010	01 1011	0	8-3
%	73	3B	11 1011	01 1100	0	8-4
≠	74	3C	11 1100	01 1010	0	8-2
=	75	3D	11 1101	01 1101	0	8-5
]	76	3E	11 1110	01 1110	0	8-6
"	77	3F	11 1111	01 1111	0	8-7

LOW ▲ ▼ HIGH

COLLATING SEQUENCE - USASCII X3.4-1968

USASCII Character	Hex Code	Internal Code
NUL	00	0000 0000
SOH	01	0000 0001
STX	02	0000 0010
ETX	03	0000 0011
EOT	04	0000 0100
ENQ	05	0000 0101
ACK	06	0000 0110
BEL	07	0000 0111
BS	08	0000 1000
HT	09	0000 1001
LF	0A	0000 1010
VT	0B	0000 1011
FF	0C	0000 1100
CR	0D	0000 1101
SO	0E	0000 1110
SI	0F	0000 1111
DLE	10	0001 0000
DC1	11	0001 0001
DC2	12	0001 0010
DC3	13	0001 0011
DC4	14	0001 0100
NAK	15	0001 0101
SYN	16	0001 0110
ETB	17	0001 0111
CAN	18	0001 1000
EM	19	0001 1001
SUB	1A	0001 1010
ESC	1B	0001 1011
FS	1C	0001 1100
GS	1D	0001 1101
RS	1E	0001 1110
US	1F	0001 1111
SP	20	0010 0000
! (or)	21	0010 0001
"	22	0010 0010
#	23	0010 0011
$	24	0010 0100
%	25	0010 0101
&	26	0010 0110
'	27	0010 0111
(	28	0010 1000
)	29	0010 1001
*	2A	0010 1010
+	2B	0010 1011
,	2C	0010 1100

LOW ▲ ▼ HIGH

USASCII Character	Hex Code	Internal Code
-	2D	0010 1101
.	2E	0010 1110
/	2F	0010 1111
0	30	0011 0000
1	31	0011 0001
2	32	0011 0010
3	33	0011 0011
4	34	0011 0100
5	35	0011 0101
6	36	0011 0110
7	37	0011 0111
8	38	0011 1000
9	39	0011 1001
:	3A	0011 1010
;	3B	0011 1011
<	3C	0011 1100
=	3D	0011 1101
>	3E	0011 1110
?	3F	0011 1111
@	40	0100 0000
A	41	0100 0001
B	42	0100 0010
C	43	0100 0011
D	44	0100 0100
E	45	0100 0101
F	46	0100 0110
G	47	0100 0111
H	48	0100 1000
I	49	0100 1001
J	4A	0100 1010
K	4B	0100 1011
L	4C	0100 1100
M	4D	0100 1101
N	4E	0100 1110
O	4F	0100 1111
P	50	0101 0000
Q	51	0101 0001
R	52	0101 0010
S	53	0101 0011
T	54	0101 0100
U	55	0101 0101
V	56	0101 0110
W	57	0101 0111
X	58	0101 1000

LOW ▲ ▼ HIGH

COLLATING SEQUENCE - USASCII X3.4-1968 (Cont)

USASCII Character	Hex Code	Internal Code
Y	59	0101 1001
Z	5A	0101 1010
[	5B	0101 1011
\	5C	0101 1100
]	5D	0101 1101
^ (¬)	5E	0101 1110
-	5F	0101 1111
`	60	0110 0000
a	61	0110 0001
b	62	0110 0010
c	63	0110 0011
d	64	0110 0100
e	65	0110 0101
f	66	0110 0110
g	67	0110 0111
h	68	0110 1000
i	69	0110 1001
j	6A	0110 1010
k	6B	0110 1011
l	6C	0110 1100

LOW ▲ ▼ HIGH

USASCII Character	Hex Code	Internal Code
m	6D	0110 1101
n	6E	0110 1110
o	6F	0110 1111
p	70	0111 0000
q	71	0111 0001
r	72	0111 0010
s	73	0111 0011
t	74	0111 0100
u	75	0111 0101
v	76	0111 0110
w	77	0111 0111
x	78	0111 1000
y	79	0111 1001
z	7A	0111 1010
{	7B	0111 1011
|	7C	0111 1100
}	7D	0111 1101
~	7E	0111 1110
DEL	7F	0111 1111

LOW ▲ ▼ HIGH

COLLATING SEQUENCE - BCL TRANSLATED TO EBCDIC

COLLATING SEQUENCE - BCL TRANSLATED TO EBCDIC

BCL Character	BCL External BA 4321	BCL Hex.	BCL Octal	Translated EBCDIC Code	EBCDIC Hex	Card Code Zone	Number
(Blank)	01 0000	10	20	0100 0000	40	-	-
[	11 1100	3C	74	0100 1010	4A	12	8-4
.	11 1011	3B	73	0100 1011	4B	12	8-3
<	11 1110	3E	76	0100 1100	4C	12	8-6
(	11 1101	3D	75	0100 1101	4D	12	8-5
+	11 1010	3A	72	0100 1110	4E	12	0
+	11 1111	3F	77	0100 1111	4F	12	8-7
&	11 0000	30	60	0101 0000	50	12	-
]	01 1110	1E	36	0101 1010	5A	0	8-6
$	10 1011	2B	53	0101 1011	5B	11	8-3
*	10 1100	2C	54	0101 1100	5C	11	8-4
)	10 1101	2D	55	0101 1101	5D	11	8-5
;	10 1110	2E	56	0101 1110	5E	11	8-6
≤	10 1111	2F	57	0101 1111	5F	11	8-7
-	10 0000	20	40	0110 0000	60	11	-
/	01 0001	11	21	0110 0001	61	0	1
,	01 1011	1B	33	0110 1011	6B	0	8-3
%	01 1100	1C	34	0110 1100	6C	0	8-4
≠	01 1010	1A	32	0110 1101	6D	0	8-2
>	00 1110	0E	16	0110 1110	6E	-	8-6
?	00 0000	00	00	0110 1111	6F	All other card codes	
:	00 1101	0D	15	0111 1010	7A	-	8-5
#	00 1011	0B	13	0111 1011	7B	-	8-3
@	00 1100	0C	14	0111 1100	7C	-	8-4
≥	00 1111	0F	17	0111 1101	7D	-	8-7
=	01 1101	1D	35	0111 1110	7E	0	8-5
"	01 1111	1F	37	0111 1111	7F	0	8-7
A	11 0001	31	61	1100 0001	C1	12	1
B	11 0010	32	62	1100 0010	C2	12	2
C	11 0011	33	63	1100 0011	C3	12	3
D	11 0100	34	64	1100 0100	C4	12	4
E	11 0101	35	65	1100 0101	C5	12	5
F	11 0110	36	66	1100 0110	C6	12	6
G	11 0111	37	67	1100 0111	C7	12	7
H	11 1000	38	70	1100 1000	C8	12	8
I	11 1001	39	71	1100 1001	C9	12	9

LOW ▲ ▼ HIGH

COLLATING SEQUENCE - BCL TRANSLATED TO EBCDIC (Cont)

BCL Character	BCL External BA 4321	BCL Hex.	BCL Octal	Translated EBCDIC Code	EBCDIC Hex.	Card Zone	Card Number
x(mult)	10 1010	2A	52	1101 0000	D0	11	0
J	10 0001	21	41	1101 0001	D1	11	1
K	10 0010	22	42	1101 0010	D2	11	2
L	10 0011	23	43	1101 0011	D3	11	3
M	10 0100	24	44	1101 0100	D4	11	4
N	10 0101	25	45	1101 0101	D5	11	5
O	10 0110	26	46	1101 0110	D6	11	6
P	10 0111	27	47	1101 0111	D7	11	7
Q	10 1000	28	50	1101 1000	D8	11	8
R	10 1001	29	51	1101 1001	D9	11	9
S	01 0010	12	22	1110 0010	E2	0	2
T	01 0011	13	23	1110 0011	E3	0	3
U	01 0100	14	24	1110 0100	E4	0	4
V	01 0101	15	25	1110 0101	E5	0	5
W	01 0110	16	26	1110 0110	E6	0	6
X	01 0111	17	27	1110 0111	E7	0	7
Y	01 1000	18	30	1110 1000	E8	0	8
Z	01 1001	19	31	1110 1001	E9	0	9
0	00 1010	0A	12	1111 0000	F0	-	0
1	00 0001	01	01	1111 0001	F1	-	1
2	00 0010	02	02	1111 0010	F2	-	2
3	00 0011	03	03	1111 0011	F3	-	3
4	00 0100	04	04	1111 0100	F4	-	4
5	00 0101	05	05	1111 0101	F5	-	5
6	00 0110	06	06	1111 0110	F6	-	6
7	00 0111	07	07	1111 0111	F7	-	7
8	00 1000	08	10	1111 1000	F8	-	8
9	00 1001	09	11	1111 1001	F9	-	9

LOW

HIGH

COLLATING SEQUENCE - BCL TRANSLATED TO USASCII

BCL Character	BCL External BA 8421	BCL Hex.	BCL Octal	Translated USASCII Code	USASCII Hex.	Card Code Zone Numbers	
Blank	01 0000	10	20	0010 0000	20	-	-
+	11 1111	3F	77	0010 0001	21	12	8-7
"	01 1111	1F	37	0010 0010	22	0	8-7
#	01 1011	0B	33	0010 0011	23	-	8-3
$	10 1011	2B	53	0010 0100	24	11	8-3
%	01 1100	1C	34	0010 0101	25	0	8-4
&	11 0000	30	60	0010 0110	26	12	-
'	01 1111	0F	37	0010 0111	27	-	8-7
(	11 1101	3D	75	0010 1000	28	12	8-3
)	10 1101	2D	55	0010 1001	29	11	8-5
*	10 1100	2C	54	0010 1010	2A	11	8-4
+	11 1010	3A	72	0010 1011	2B	12	0
,	01 1011	1B	33	0010 1100	2C	0	8-3
-	10 0000	20	40	0010 1101	2D	11	-
.	10 1011	3B	53	0010 1110	2E	12	8-3
/	01 0001	11	21	0010 1111	2F	0	1
0	00 1010	0A	12	0011 0000	30	-	0
1	00 0001	01	01	0011 0001	31	-	1
2	00 0010	02	02	0011 0010	32	-	2
3	00 0011	03	03	0011 0011	33	-	3
4	00 0100	04	04	0011 0100	34	-	4
5	00 0101	05	05	0011 0101	35	-	5
6	00 0110	06	06	0011 0110	36	-	6
7	00 0111	07	07	0011 0111	37	-	7
8	00 1000	08	08	0011 1000	38	-	8
9	00 1001	09	09	0011 1001	39	-	9
:	10 1101	0D	55	0011 1010	3A	-	8-5
;	10 1110	2E	56	0011 1011	3B	11	8-6
<	11 1110	3E	76	0011 1100	3C	12	8-6
=	01 1101	1D	35	0011 1101	3D	0	8-5
>	00 1110	0E	16	0011 1110	3E	-	8-6
?	00 0000	00	00	0011 1111	3F	All other card codes	
@	00 1100	0C	14	0100 0000	40	-	8-4
A	11 0001	31	61	0100 0001	41	12	1
B	11 0010	32	62	0100 0010	42	12	2
C	11 0011	33	63	0100 0011	43	12	3
D	11 0100	34	64	0100 0100	44	12	4
E	11 0101	35	65	0100 0101	45	12	5
F	11 0110	36	66	0100 0110	46	12	6
G	11 0111	37	67	0100 0111	47	12	7
H	11 1000	38	70	0100 1000	48	12	8
I	11 1001	39	71	0100 1001	49	12	9
J	10 0001	21	41	0100 1010	4A	11	1
K	10 0010	22	42	0100 1011	4B	11	2
L	10 0011	23	43	0100 1100	4C	11	3

LOW ▲ ▼ HIGH

COLLATING SEQUENCE - BCL TRANSLATED TO USASCII (Cont)

BCL Character	BCL External BA 8421	BCL Hex.	BCL Octal	Translated USASCII Code	USASCII Hex.	Card Code Zone	Card Code Number
M	10 0100	24	44	0100 1101	4D	11	4
N	10 0101	25	45	0100 1110	4E	11	5
O	10 0110	26	46	0100 1111	4F	11	6
P	10 0111	27	47	0101 0000	50	11	7
Q	10 1000	28	50	0101 0001	51	11	8
R	10 1001	29	51	0101 0010	52	11	9
S	01 0010	12	22	0101 0011	53	0	2
T	01 0011	13	23	0101 0100	54	0	3
U	01 0100	14	24	0101 0101	55	0	4
V	01 0101	15	25	0101 0110	56	0	5
W	01 0110	16	26	0101 0111	57	0	6
X	01 0111	17	27	0101 1000	58	0	7
Y	01 1000	18	30	0101 1001	59	0	8
Z	01 1001	19	31	0101 1010	5A	0	9
[	11 1100	3C	74	0101 1011	5B	12	8-4
]	01 1110	1E	36	0101 1101	5D	0	8-6
<	10 1111	2F	57	0101 1110	5E	11	8-7
≠	01 1010	1A	32	0101 1111	5F	0	8-2
x(Mult)	10 1010	2A	52	0111 1101	7D	11	0

LOW ▲ ▼ HIGH

COLLATING SEQUENCE - USASCII X3.4-1968 TRANSLATED TO EBCDIC

LOW

USASCII Character	USASCII Hex. Code	Translated EBCDIC Code	EBCDIC Hex. Code	USASCII Character	USASCII Hex. Code	Translated EBCDIC Code	EBCDIC Hex. Code
NULL	00	0000 0000	00	&	26	0101 0000	50
SOH	01	0000 0001	01	]	5D	0101 1010	5A
STX	02	0000 0010	02	$	24	0101 1011	5B
ETX	03	0000 0011	03	*	2A	0101 1100	5C
HT	09	0000 0101	05	)	29	0101 1101	5D
DEL	7F	0000 0111	07	;	3B	0101 1110	5E
VT	0B	0000 1011	0B	^ (¬)	5E	0101 1111	5F
FF	0C	0000 1100	0C				
CR	0D	0000 1101	0D	-	2D	0110 0000	60
SO	0E	0000 1110	0E	/	2F	0110 0001	61
SI	0F	0000 1111	0F	¦	7C	0110 1010	6A
				,	2C	0110 1011	6B
DLE	10	0001 0000	10	%	25	0110 1100	6C
DC1	11	0001 0001	11	_	5F	0110 1101	6D
DC2	12	0001 0010	12	>	3E	0110 1110	6E
DC3	13	0001 0011	13	?	3F	0110 1111	6F
BS	08	0001 0110	16				
				`	60	0111 1001	79
				:	3A	0111 1010	7A
				#	23	0111 1011	7B
CAN	18	0001 1000	18	@	4C	0111 1100	7C
EM	19	0001 1001	19	'	27	0111 1101	7D
FS	1C	0001 1100	1C	=	3D	0111 1110	7E
GS	1D	0001 1101	1D	"	22	0111 1111	7F
RS	1E	0001 1110	1E				
US	1F	0001 1111	1F	a	61	1000 0001	81
				b	62	1000 0010	82
LF	0A	0010 0101	25	c	63	1000 0011	83
ETB	17	0010 0110	26	d	64	1000 0100	84
ESC	1B	0010 0111	27	e	65	1000 0101	85
ENQ	05	0010 1101	2D	f	66	1000 0110	86
ACK	06	0010 1110	2E	g	67	1000 0111	87
BEL	07	0010 1111	2F	h	68	1000 1000	88
				i	69	1000 1001	89
SYN	16	0011 0010	32				
EOT	04	0011 0111	37	j	6A	1001 0001	91
DC4	14	0011 1100	3C	k	6B	1001 0010	92
NAK	15	0011 1101	3D	l	6C	1001 0011	93
SUB	1A	0011 1111	3F	m	6D	1001 0100	94
				n	6E	1001 0101	95
SP	20	0100 0000	40	o	6F	1001 0110	96
[	5B	0100 1010	4A	p	70	1001 0111	97
.	2E	0100 1011	4B	q	71	1001 1000	98
<	3C	0100 1100	4C	r	72	1001 1001	99
(	28	0100 1101	4D				
+	2B	0100 1110	4E	~	7D	1010 0001	A1
| (or)	21	0100 1111	4F	s	73	1010 0010	A2

HIGH

COLLATING SEQUENCE - USASCII X3.4-1968 TRANSLATED TO EBCDIC (Cont)

USASCII Character	USASCII Hex. Code	Translated EBCDIC Code	EBCDIC Hex. Code
t	74	1010 0011	A3
u	75	1010 0100	A4
v	76	1010 0101	A5
w	77	1010 0110	A6
x	78	1010 0111	A7
y	79	1010 1000	A8
z	7A	1010 1001	A9
{	7B	1100 0000	C0
A	41	1100 0001	C1
B	42	1100 0010	C2
C	43	1100 0011	C3
D	44	1100 0100	C4
E	45	1100 0101	C5
F	46	1100 0110	C6
G	47	1100 0111	C7
H	48	1100 1000	C8
I	49	1100 1000	C9
}	70	1101 0000	D0
J	4A	1101 0001	D1
K	4B	1101 0010	D2
L	4C	1101 0011	D3
M	4D	1101 0100	D4
N	4E	1101 0101	D5
O	4F	1101 0110	D6
P	50	1101 0111	D7
Q	51	1101 1000	D8
R	52	1101 1001	D9
\	5C	1110 0000	E0
S	53	1110 0010	E2
T	54	1110 0011	E3
U	55	1110 0100	E4
V	56	1110 0101	E5
W	57	1110 0110	E6
X	58	1110 0111	E7
Y	59	1110 1000	E8
Z	5A	1110 1001	E9
0	30	1111 0000	F0
1	13	1111 0001	F1
2	32	1111 0010	F2
3	33	1111 0011	F2
4	34	1111 0100	F4
5	35	1111 0101	F5
6	36	1111 0110	F6
7	37	1111 0111	F7
8	38	1111 1000	F8
9	39	1111 1001	F9

XALGOL COLLATING SEQUENCE (B 5700 BCL)

BCL Character	BCL Octal	BCL Hex	BCL Internal BA 8421	BCL External BA 8421	Card Code Zone	Card Code Number
Blank	60	30	11 0000	01 0000	-	-
.	32	1A	01 1010	11 1011	12	8-3
[	33	1B	01 1011	11 1100	12	8-4
(	35	1D	01 1101	11 1101	12	8-5
<	36	1E	01 1110	11 1110	12	8-6
+	37	1F	01 1111	11 1111	12	8-7
&	34	1C	01 1100	11 0000	12	-
$	52	2A	10 1010	10 1011	11	8-3
*	53	2B	10 1011	10 1100	11	8-4
)	55	2D	10 1101	10 1101	11	8-5
;	56	2E	10 1110	10 1110	11	8-6
≤	57	2F	10 1111	10 1111	11	8-7
-	54	2C	10 1100	10 0000	11	-
/	61	31	11 0001	01 0001	0	1
,	72	3A	11 1010	01 1011	0	8-3
%	73	3B	11 1011	01 1100	0	8-4
=	75	3D	11 1101	01 1101	0	8-5
]	76	3E	11 1110	01 1110	0	8-6
"	77	3F	11 1111	01 1111	0	8-7
#	12	0A	00 1010	00 1011	-	8-3
@	13	0B	00 1011	00 1100	-	8-4
:	15	0D	00 1101	00 1101	-	8-5
>	16	0E	00 1110	00 1110	-	8-6
≥	17	0F	00 1111	00 1111	-	8-7
+	20	10	01 0000	11 1010	12	0
A	21	11	01 0001	11 0001	12	1
B	22	12	01 0010	11 0010	12	2
C	23	13	01 0011	11 0011	12	3
D	24	14	01 0100	11 0100	12	4
E	25	15	01 0101	11 0101	12	5
F	26	16	01 0110	11 0110	12	6
G	27	17	01 0111	11 0111	12	7
H	30	18	01 1000	11 1000	12	8
I	31	19	01 1001	11 1001	12	9
x	40	20	10 0000	10 1010	11	0
J	41	21	10 0001	10 0001	11	1
K	42	22	10 0010	10 0010	11	2
L	43	23	10 0011	10 0011	11	3
M	44	24	10 0100	10 0100	11	4
N	45	25	10 0101	10 0101	11	5

LOW ↑ COLLATING SEQUENCE ↓ HIGH

XALGOL COLLATING SEQUENCE (B 5700 BCL) (Cont)

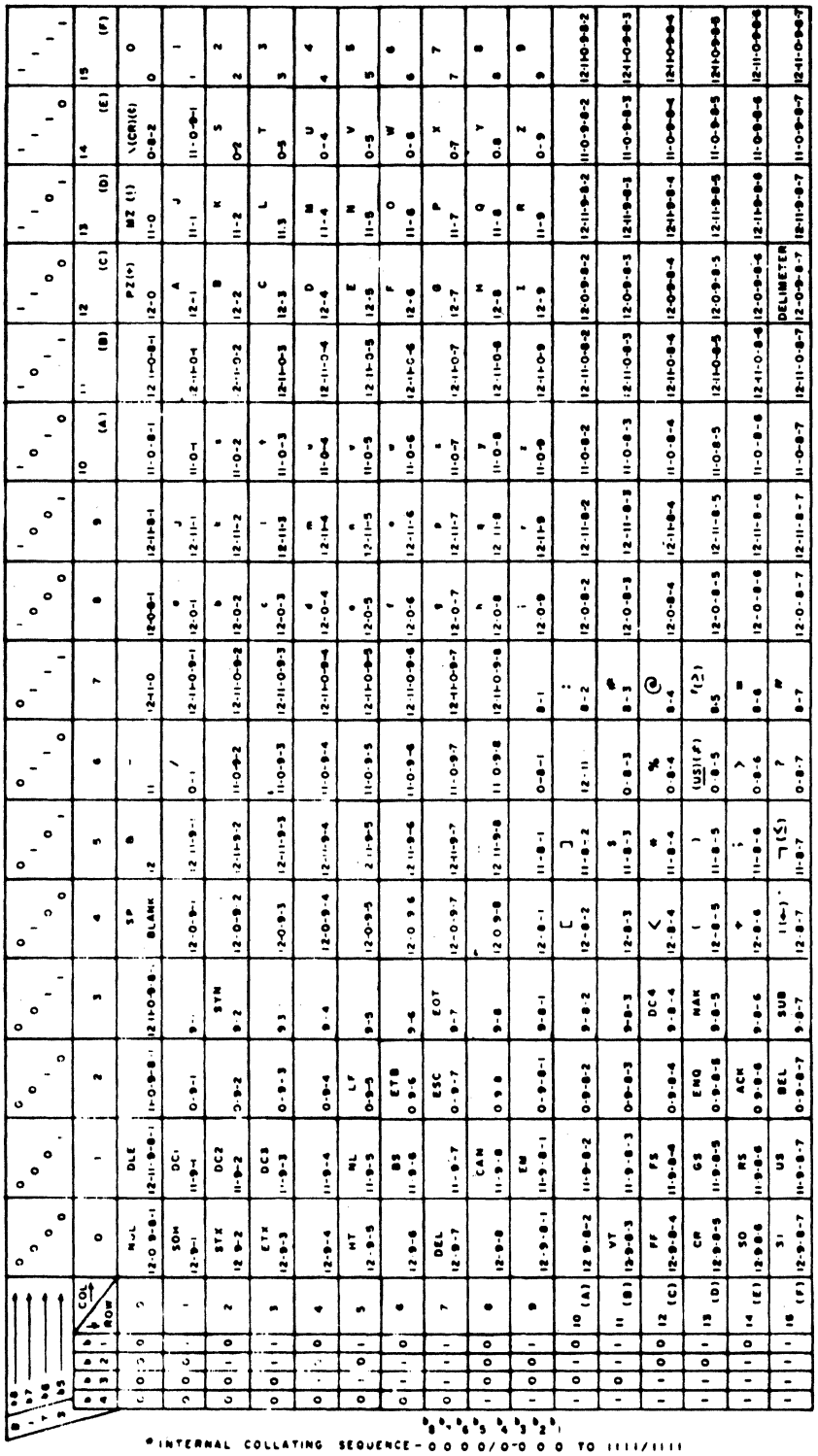
BCL Character	BCL Octal	BCL Hex	BCL Internal BA 8421	BCL External BA 8421	Card Code Zone Number	
0	46	26	10 0110	10 0110	11 6	
P	47	27	10 0111	10 0111	11 7	LOW
Q	50	28	10 1000	10 1000	11 8	
R	51	29	10 1001	10 1001	11 9	
≠	74	3C	11 1100	01 1010	0 8-2	
S	62	32	11 0010	01 0010	0 2	
T	63	33	11 0011	01 0011	0 3	
U	64	34	11 0100	01 0100	0 4	
V	65	35	11 0101	01 0101	0 5	
W	66	36	11 0110	01 0110	0 6	
X	67	37	11 0111	01 0111	0 7	
Y	70	38	11 1000	01 1000	0 8	
Z	71	39	11 1001	01 1001	0 9	
0	00	00	00 0000	00 1010	- 0	
1	01	01	00 0001	00 0001	- 1	HIGH
2	02	02	00 0010	00 0010	- 2	
3	03	03	00 0011	00 0011	- 3	
4	04	04	00 0100	00 0100	- 4	
5	05	05	00 0101	00 0101	- 5	
6	06	06	00 0110	00 0110	- 6	
7	07	07	00 0111	00 0111	- 7	
8	10	08	00 1000	00 1000	- 8	
9	11	09	00 1001	00 1001	- 9	
?	14	0C	00 1100	00 0000	ALL OTHER CARD CODES	

FORTRAN BCD COLLATING SEQUENCE

BCD Character	Internal Representation Hex	Internal Representation Binary	Internal Translation Binary	Internal Translation Hex	Card Code Zone	Card Code Number
. (period)	1A	01 1010	0100 1011	4B	12	8-3
)	1B	01 1011	0100 1100	4C	12	8-4
+	1C	01 1100	0101 0000	50	12	
$	2A	10 1010	0101 1011	5B	11	8-3
*	2B	10 1011	0101 1100	5C	11	8-4
;	2E	10 1110	0101 1110	5E	11	8-6
≤	2F	10 1111	0101 1111	5F	11	8-7
- (minus)	2C	10 1100	0110 0000	60	11	
/	31	11 0001	0110 0001	61	0	1
, (comma)	3A	11 1010	0110 1011	6B	0	8-3
(	3B	11 1011	0110 1100	6C	0	8-4
≠	3D	11 1101	0110 1101	6D	0	8-5
>	3E	11 1110	0110 1110	6E	0	8-6
?	3F	11 1111	0110 1111	6F	0	8-7
=	0A	00 1010	0111 1011	7B		8-3
@	0B	00 1011	0111 1100	7C		8-4
≥	0D	00 1101	0111 1101	7D		8-5
≠	0E	00 1110	0111 1110	7E		8-6
"	0F	00 1111	0111 1111	7E		8-7
A	11	01 0001	1100 0001	C1	12	1
B	12	01 0010	1100 0010	C2	12	2
C	13	01 0011	1100 0011	C3	12	3
D	14	01 0100	1100 0100	C4	12	4
E	15	01 0101	1100 0101	C5	12	5
F	16	01 0110	1100 0110	C6	12	6
G	17	01 0111	1100 0111	C7	12	7
H	18	01 1000	1100 1000	C8	12	8
I	19	01 1001	1100 1001	C9	12	9
J	21	10 0001	1101 0001	D1	11	1
K	22	10 0010	1101 0010	D2	11	2
L	23	10 0011	1101 0011	D3	11	3
M	24	10 0100	1101 0100	D4	11	4
N	25	10 0101	1100 0101	D5	11	5
O	26	10 0110	1101 0110	D6	11	6
P	27	10 0111	1101 0111	D7	11	7
Q	28	10 1000	1101 1000	D8	11	8
R	29	10 1001	1101 1001	D9	11	9
S	31	11 0010	1110 0010	E2	0	2
T	33	11 0011	1110 0011	E3	0	3
U	34	11 0100	1110 0100	E4	0	4
V	35	11 0101	1110 0101	E5	0	5
W	36	11 0110	1110 0110	E6	0	6
X	37	11 0111	1110 0111	E7	0	7
Y	38	11 1000	1110 1000	E8	0	8
Z	39	11 1001	1110 1001	E9	0	9
0	00	00 0000	1111 0000	F0		0
1	01	00 0001	1111 0001	F1		1
2	02	00 0010	1111 0010	F2		2
3	03	00 0011	1111 0011	F3		3
4	04	00 0100	1111 0100	F4		4
5	05	05 0101	1111 0101	F5		5
6	06	00 0110	1111 0110	F6		6
7	07	00 0111	1111 0111	F7		7
8	08	00 1000	1111 1000	F8		8
9	09	00 1001	1111 1001	F9		9

Low ↓ High

EXTENDED BINARY CODED DECIMAL INTERCHANGE CODES (EBCDIC)



*INTERNAL COLLATING SEQUENCE - 0 0 0 0 0/0-0 0 0 TO 1111/1111

APPENDIX B
DATA REPRESENTATION

EBCDIC GRAPHIC	BCL	DECIMAL VALUE	EBCDIC INTERNAL	HEX. GRAPHIC	EBCDIC CARD CODE	BCL CARD CODE	OCTAL	BCL INTERNAL	BCL EXTERNAL
BLANK		64	0100 0000	40	No Punches	No Punches	60	11 0000	01 0000
[		74	0100 1010	4A	12 8 2	12 8 4	33	01 1011	11 1100
.		75	0100 1011	4B	12 8 3	12 8 3	32	01 1010	11 1011
<		76	0100 1100	4C	12 8 4	12 8 6	36	01 1110	11 1110
(		77	0100 1101	4D	12 8 5	12 8 5	35	01 1101	11 1101
+		78	0100 1110	4E	12 8 6				11 1010
!	←	79	0100 1111	4F	12 8 7	12 8 7	37	01 1111	11 1111
&		80	0101 0000	50	12	12	34	01 1100	11 0000
]		90	0101 1010	5A	11 8 2	0 8 6	76	11 1110	01 1110
$		91	0101 1011	5B	11 8 3	11 8 3	52	10 1010	10 1011
*		92	0101 1100	5C	11 8 4	11 8 4	53	10 1011	10 1100
)		93	0101 1101	5D	11 8 5	11 8 5	55	10 1101	10 1101
;		94	0101 1110	5E	11 8 6	11 8 6	56	10 1110	10 1110
¬	≤	95	0101 1111	5F	11 8 7	11 8 7	57	10 1111	10 1111
-		96	0110 0000	60	11	11	54	10 1100	10 0000
/		97	0110 0001	61	0 1	0 1	61	11 0001	01 0001
,		107	0110 1011	6B	0 8 3	0 8 3	72	11 1010	01 1011
%		108	0110 1100	6C	0 8 4	0 8 4	73	11 1011	01 1100
_	≠	109	0110 1101	6D	0 8 5	0 8 2	74	11 1100	01 1010
>		110	0110 1110	6E	0 8 6	8 6	16	00 1110	00 1110
?		111	0110 1111	6F	0 8 7	*	14	00 1100	00 0000
:		122	0111 1010	7A	8 2	8 5	15	00 1101	00 1101
#		123	0111 1011	7B	8 3	8 3	12	00 1010	00 1011
@		124	0111 1100	7C	8 4	8 4	13	00 1011	00 1100
'	≥	125	0111 1101	7D	8 5	8 7	17	00 1111	00 1111
=		126	0111 1110	7E	8 6	0 8 5	75	11 1101	01 1101
"		127	0111 1111	7F	8 7	0 8 7	77	11 1111	01 1111
(+)PZ	+	192	1100 0000	CO	12 0	12 0	20	01 0000	11 1010
A		193	1100 0001	C1	12 1	12 1	21	01 0001	11 0001
B		194	1100 0010	C2	12 2	12 2	22	01 0010	11 0010
C		195	1100 0011	C3	12 3	12 3	23	01 0011	11 0011
D		196	1100 0100	C4	12 4	12 4	24	01 0100	11 0100
E		197	1100 0101	C5	12 5	12 5	25	01 0101	11 0101
F		198	1100 0110	C6	12 6	12 6	26	01 0110	11 0110
G		199	1100 0111	C7	12 7	12 7	27	01 0111	11 0111
H		200	1100 1000	C8	12 8	12 8	30	01 1000	11 1000
I		201	1100 1001	C9	12 9	12 9	31	01 1001	11 1001
(!)MZ	MULT x	208	1101 0000	D0	11 0	11 0	40	10 0000	10 1010
J		209	1101 0001	D1	11 1	11 1	41	10 0001	10 0001
K		210	1101 0010	D2	11 2	11 2	42	10 0010	10 0010
L		211	1101 0011	D3	11 3	11 3	43	10 0011	10 0011
M		212	1101 0100	D4	11 4	11 4	44	10 0100	10 0100
N		213	1101 0101	D5	11 5	11 5	45	10 0101	10 0101
O		214	1101 0110	D6	11 6	11 6	46	10 0110	10 0110
P		215	1101 0111	D7	11 7	11 7	47	10 0111	10 0111

*All other codes

DATA REPRESENTATION

EBCDIC GRAPHIC	BCL	DECIMAL VALUE	EBCDIC INTERNAL	HEX. GRAPHIC	EBCDIC CARD CODE	BCL CARD CODE	OCTAL	BCL INTERNAL	BCL EXTERNAL
Q		216	1101 1000	D8	11 8	11 8	50	10 1000	10 1000
R		217	1101 1001	D9	11 9	11 9	51	10 1001	10 1001
¢		224	1110 0000	E0	0 8 2				00 0000
S		226	1110 0010	E2	0 2	0 2	62	11 0010	01 0010
T		227	1110 0011	E3	0 3	0 3	63	11 0011	01 0011
U		228	1110 0100	E4	0 4	0 4	64	11 0100	01 0100
V		229	1110 0101	E5	0 5	0 5	65	11 0101	01 0101
W		230	1110 0110	E6	0 6	0 6	66	11 0110	01 0110
X		231	1110 0111	E7	0 7	0 7	67	11 0111	01 0111
Y		232	1110 1000	E8	0 8	0 8	70	11 1000	01 1000
Z		233	1110 1001	E9	0 9	0 9	71	11 1001	01 1001
0		240	1111 0000	F0	0	0	00	00 0000	00 1010
1		241	1111 0001	F1	1	1	01	00 0001	00 0001
2		242	1111 0010	F2	2	2	02	00 0010	00 0010
3		243	1111 0011	F3	3	3	03	00 0011	00 0011
4		244	1111 0100	F4	4	4	04	00 0100	00 0100
5		245	1111 0101	F5	5	5	05	00 0101	00 0101
6		246	1111 0110	F6	6	6	06	00 0110	00 0110
7		247	1111 0111	F7	7	7	07	00 0111	00 0111
8		248	1111 1000	F8	8	8	10	00 1000	00 1000
9		249	1111 1001	F9	9	9	11	00 1001	00 1001

NOTES

1. EBCDIC 0100 1110 also translates to BCL 11 1010.
2. EBCDIC 1100 1111 is translated to BCL 00 0000 with an additional flag bit on the most significant bit line (8th bit). This function is used by the unbuffered printer to stop scanning.
3. EBCDIC 1110 0000 is translated to BCL 00 0000 with an additional flag bit on the next to most significant bit line (7th bit). As the print drums have 64 graphics and space this signal can be used to print the 64th graphic. The 64th graphic is a "CR" for BCL drums and a "¢" for EBCDIC drums.
4. The remaining 189 EBCDIC codes are translated to BCL 00 0000 (? code).
5. The EBCDIC graphics and BCL graphics are the same except as follows:

BCL		EBCDIC
≥		' (single quote)
x	(multiply)	!
≤		¬ (not)
≠		¯ (underscore)
←		

APPENDIX C

PROCESSOR OPERATORS, BY HEXADECIMAL CODE

MODE ID& HEX CODE	OPERATOR NAME	MNEMONIC
(P)00→3F	VALUE CALL	VALC
(P)40→7F	NAME CALL	NAMC
(V)42	SET TWO SINGLES TO DOUBLE	JOIN
(V)43	SET DOUBLE TO TWO SINGLES	SPLT
(V)44	IDLE UNTIL INTERRUPT	IDLE
(V)45	SET INTERVAL TIMER	SINT
(V)46	ENABLE EXTERNAL INTERRUPTS	EEXI
(V)47	DISABLE EXTERNAL INTERRUPTS	DEXI
(V)48	IGNORE PARITY	IGPR
(V)4A	SCAN IN	SCNI
(V)4E	READ PROCESSOR IDENTIFICATION	WHOI
(P)80	ADD	ADD
(P)81	SUBTRACT	SUBT
(P)82	MULTIPLY	MULT
(P)83	DIVIDE	DIVD
(P)84	INTEGER DIVIDE	IDV
(V)84	PAUSE UNTIL INTERRUPT	PAUS
(P)85	REMAINDER DIVIDE	RDIV
(V)85	OCCURS INDEX	OCRX
(P)86	INTEGERIZE, TRUNCATED	NTIA
(P)87	INTEGERIZE, ROUNDED	NTGR
(V)87	INTEGERIZE, ROUNDED, DOUBLE PRECISION	NTGD
(P)88	LESS THAN	LESS
(P)89	GREATER THAN OR EQUAL	GREQ
(P)8A	GREATER THAN	GRTR
(P)8B	LESS THAN OR EQUAL	LSEQ
(V)8B	LEADING ONE TEST	LOG2
(P)	EQUAL	EQUL
(P)8D	NOT EQUAL	NEQL
(P)8E	CHANGE SIGN BIT	CHSN
(P)8F	EXTENDED MULTIPLE	MULX
(V)8F	INTERRUPT CHANNEL N	INCN
(P)90	LOGICAL AND	LAND
(P)91	LOGICAL OR	LOR
(P)92	LOGICAL NEGATE	LNOT
(P)93	LOGICAL EQUIVALENCE	LEQV
(P)94	LOGICAL EQUAL	SAME
(P)95	ESCAPE TO 16-BIT INSTRUCTION	VARI

MODE ID & HEX CODE	OPERATOR NAME	MNEMONIC
(P)96	BIT SET	BSET
(P)97	DYNAMIC BIT SET	DBST
(P)98	FIELD TRANSFER	FLTR
(P)99	DYNAMIC FIELD TRANSFER	DFTR
(P)9A	FIELD ISOLATE	ISOL
(P)9B	DYNAMIC FIELD ISOLATE	DISO
(P)9C	FIELD INSERT	INSR
(P)9D	DYNAMIC FIELD INSERT	DINS
(P)9E	BIT RESET	BRST
(P)9F	DYNAMIC BIT RESET	DBRS
(P)A0	BRANCH FALSE	BRFL
(P)A1	BRANCH TRUE	BRTR
(P)A2	BRANCH UNCONDITIONAL	BRUN
(P)A3	EXIT	EXIT
(P)A4	STEP AND BRANCH	STBR
(P)A5	INDEX AND LOAD NAME	NXLN
(P)A6	INDEX	INDX
(P)A7	RETURN	RETN
(P)A8	DYNAMIC BRANCH FALSE	DBFL
(V)A8	SET MEMORY INHIBITS	SINH
(P)A9	DYNAMIC BRANCH TRUE	DBTR
(P)AA	DYNAMIC BRANCH UNCONDITIONAL	DBUN
(V)AA	SET MEMORY LIMITS	SLMT
(P)AB	ENTER	ENTR
(P)AC	EVALUATE DESCRIPTOR	EVAL
(V)AC	FETCH MEMORY FAIL	FMFR
(P)AD	INDEX AND LOAD VALUE	NXLV
(P)AE	MARK STACK	MKST
(P)AF	STUFF ENVIRONMENT	STFF
(V)AF	MOVE TO STACK	MVST
(P)B0	LIT CALL ZERO	ZERO
(P)B1	LIT CALL ONE	ONE
(P)B2	LIT CALL 8 BITS	LT8
(P)B3	LIT CALL 16 BITS	LT16
(P)B4	PUSH DOWN STACK REGISTERS	PUSH
(V)B4	SET TAG FIELD	STAG
(P)B5	DELETE TOP OF STACK	DLET
(V)B5	READ TAG FIELD	RTAG
(P)B6	EXCHANGE	EXCH
(V)B6	ROTATE STACK UP	RSUP
(P)B7	DUPLICATE TOP OF STACK	DUPL

MODE ID & HEX CODE	OPERATOR NAME	MNEMONIC
(V)B7	ROTATE STACK DOWN	RSDN
(P)B8	STORE DESTRUCTIVE	STOD
(V)B8	READ PROCESSOR REGISTER	RPRR
(P)B9	STORE NON-DESTRUCTIVE	STON
(V)B9	SET PROCESSOR REGISTER	SPRR
(P)BA	OVERWRITE DESTRUCTIVE	OVRD
(V)BA	READ WITH LOCK	RDLK
(P)BB	OVERWRITE NON-DESTRUCTIVE	OVRN
(V)BB	COUNT BINARY ONES	CBON
(V)BC	LOAD TRANSPARENT	LODT
(P)BD	LOAD	LOAD
(V)BD	LINKED LIST LOOKUP	LLLU
(P)BE	LIT CALL 48 BITS	LT48
(V)BE	MASKED SEARCH FOR EQUAL	SRCH
(P)BF	MAKE PROGRAM CONTROL WORD	MPCW
(V)BF	STOP	STOP
(P)C0	SCALE LEFT	SCLF
(P)C1	DYNAMIC SCALE LEFT	DSLF
(P)C2	SCALE RIGHT TRUNCATE	SCRT
(P)C3	DYNAMIC SCALE RIGHT TRUNCATE	DSRT
(P)C4	SCALE RIGHT SAVE	SCRS
(P)C5	DYNAMIC SCALE RIGHT SAVE	DSRS
(P)C6	SCALE RIGHT FINAL	SCRF
(P)C7	DYNAMIC SCALE RIGHT FINAL	DSRF
(P)C8	SCALE RIGHT ROUND	SCRR
(P)C9	DYNAMIC SCALE RIGHT ROUND	DSRR
(P)CA	INPUT CONVERT, DESTRUCTIVE	ICVD
(P)CB	INPUT CONVERT, UPDATE	ICVU
(P)CC	SET TO SINGLE PRECISION TRUNCATE	SNGT
(P)CD	SET TO SINGLE PRECISION, ROUNDED	SNGL
(P)CE	SET TO DOUBLE PRECISION	XTND
(P)CF	INSERT MARK STACK	IMKS
(E)D0	MOVE WITH INSERT	MINS
(P)D0	TABLE ENTER EDIT, DESTRUCTIVE	TEED
(V)D0	UNPACK SIGNED, DESTRUCTIVE	USND
(E)D1	MOVE WITH FLOAT	MFLT
(P)D1	PACK DESTRUCTIVE	PACD
(V)D1	UNPACK ABSOLUTE, DESTRUCTIVE	UABD
(E)D2	SKIP FORWARD SOURCE CHARACTERS	SFSC
(P)D2	EXECUTE SINGLE MICRO, DESTRUCTIVE	EXSD
(V)D2	TRANSFER WHILE FALSE, DESTRUCTIVE	TWFD
(E)D3	SKIP REVERSE SOURCE CHARACTERS	SRSC
(P)D3	TRANSFER WORDS, DESTRUCTIVE	TWSD
(V)D3	TRANSFER WHILE TRUE, DESTRUCTIVE	TWTD
(E)D4	RESET FLOAT	RSTF
(P)D4	TRANSFER WORDS, OVERWRITE DESTRUCTIVE	TWOD
(V)D4	SCAN WHILE FALSE, DESTRUCTIVE	SWFD
(E)D5	END FLOAT	ENDF
(P)D5	STRING ISOLATE	SISO
(V)D5	SCAN WHILE TRUE, DESTRUCTIVE	SWTD
(E)D6	MOVE NUMERIC UNCONDITIONAL	MVNU
(P)D6	SET EXTERNAL SIGN	SXSN
(E)D7	MOVE CHARACTERS	MCHR
(P)D7	READ AND CLEAR OVERFLOW FLIP-FLOP	ROFF
(V)D7	TRANSLATE	TRNS
(E)D8	INSERT OVERPUNCH	INOP
(P)D8	TABLE ENTER EDIT, UPDATE	TEEU
(V)D8	UNPACK SIGNED UPDATE	USNU
(E)D9	INSERT DISPLAY SIGN	INSG
(P)D9	PACK UPDATE	PACU
(V)D9	UNPACK ABSOLUTE, UPDATE	UABU
(E)DA	SKIP FORWARD DESTINATION CHARACTERS	SFDC
(P)DA	EXECUTE SINGLE MICRO, UPDATE	EXSU
(V)DA	TRANSFER WHILE FALSE, UPDATE	TWFU
(E)DB	SKIP REVERSE DESTINATION CHARACTERS	SRDC
(P)DB	TRANSFER WORDS, UPDATE	TWSU
(V)DB	TRANSFER WHILE TRUE, UPDATE	TWTU
(E)DC	INSERT UNCONDITIONAL	INSU
(P)DC	TRANSFER WORDS OVERWRITE UPDATE	TWOU
(V)DC	SCAN WHILE FALSE, UPDATE	SWFU
(E)DD	INSERT CONDITIONAL	INSC
(P)DD	EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE	EXPU
(V)DD	SCAN WHILE TRUE, UPDATE	SWTU
(E)DE	END EDIT	ENDE

MODE ID & HEX CODE	OPERATOR NAME	MNEMONIC
(P)DE	READ TRUE/FALSE FLIP-FLOP	RTFF
(E)DF	CONDITIONAL HALT	HALT
(P)DF	CONDITIONAL HALT	HALT
(V)DF	CONDITIONAL HALT	HALT
(P)E0	TRANSFER WHILE LESS, DESTRUCTIVE	TLSD
(P)E1	TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE	TGED
(Z)E1		
(P)E2	TRANSFER WHILE GREATER, DESTRUCTIVE	TGTD
(P)E3	TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE	TLED
(P)E4	TRANSFER WHILE EQUAL, DESTRUCTIVE	TEQD
(P)E5	TRANSFER WHILE NOT EQUAL, DESTRUCTIVE	TNED
(P)E6	TRANSFER UNCONDITIONAL, DESTRUCTIVE	TUND
(P)E7	MULTIPLE-WORD VECTOR MODE	VMOM
(P)E8	TRANSFER WHILE LESS, UPDATE	TLSU
(P)E9	TRANSFER WHILE GREATER OR EQUAL, UPDATE	TGEU
(P)EA	INCREMENT TRANSFER WHILE GREATER, UPDATE	TGTU
(P)EB	TRANSFER WHILE LESS OR EQUAL, UPDATE	TLEU
(P)EC	INCREMENT TRANSFER WHILE EQUAL, UPDATE	TEQU
(P)ED	TRANSFER WHILE NOT EQUAL, UPDATE	TNEU
(P)EE	TRANSFER UNCONDITIONAL, UPDATE	TUNU
(P)EF	SINGLE-WORD VECTOR MODE	VMOS
(P)F0	COMPARE CHARACTERS LESS, DESTRUCTIVE	CLSD
(V)F0	SCAN WHILE LESS, DESTRUCTIVE	SLSD
(P)F1	COMPARE CHARACTERS GREATER OR EQUAL, DESTRUCTIVE	CGED
(V)F1	SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE	SGED
(P)F2	COMPARE CHARACTERS GREATER, DESTRUCTIVE	CGTD
(V)F2	SCAN WHILE GREATER, DESTRUCTIVE	SGTD
(P)F3	COMPARE CHARACTERS LESS OR EQUAL, DESTRUCTIVE	CLED
(V)F3	SCAN WHILE LESS OR EQUAL, DESTRUCTIVE	SLED
(P)F4	COMPARE CHARACTERS EQUAL, DESTRUCTIVE	CEQD
(V)F4	SCAN WHILE EQUAL, DESTRUCTIVE	SEQD
(P)F5	COMPARE CHARACTERS NOT EQUAL, DESTRUCTIVE	CNED
(V)F5	SCAN WHILE NOT EQUAL, DESTRUCTIVE	SNED
(P)F8	COMPARE CHARACTERS LESS, UPDATE	CLSU
(V)F8	SCAN WHILE LESS, UPDATE	SLSU
(P)F9	COMPARE CHARACTERS GREATER OR EQUAL, UPDATE	CGEU
(V)F9	SCAN WHILE GREATER OR EQUAL, UPDATE	SGEU
(P)FA	INCREMENT COMPARE CHARACTERS GREATER, UPDATE	CGTU
(V)FA	SCAN WHILE GREATER, UPDATE	SGTU
(P)FB	COMPARE CHARACTERS LESS OR EQUAL, UPDATE	CLEU
(V)FB	SCAN WHILE LESS OR EQUAL, UPDATE	SLEU

MODE ID & HEX CODE	OPERATOR NAME	MNEMONIC
(P)FC	COMPARE CHARACTERS EQUAL, UPDATE	CEQU
(V)FC	SCAN WHILE EQUAL, UPDATE	SEQU
(P)FD	COMPARE CHARACTERS NOT EQUAL, UPDATE	CNEU
(V)FD	SCAN WHILE NOT EQUAL, UPDATE	SNEU

MODE ID & HEX CODE	OPERATOR NAME	MNEMONIC
(E)FE	NO OPERATION	NOOP
(P)FE	NO OPERATION	NOOP
(V)FE	NO OPERATION	NOOP
(E)FF	INVALID OPERATION	NVLD
(P)FF	INVALID OPERATION	NVLD
(V)FF	INVALID OPERATION	NVLD

APPENDIX D
PROCESSOR OPERATORS BY MNEMONICS

Mnemonic	Mode ID & Hex Code	Operator Name	Page
		SCALE RIGHT ROUNDED	3-4-14
DSRS	(P)C5	DYNAMIC SCALE RIGHT SAVE	3-4-14
DSRT	(P)C3	DYNAMIC SCALE RIGHT TRUNCATE	3-4-14
DUPL	(P)B7	DUPLICATE TOP OF STACK	3-4-15
		E	
EEXI	(V)46	ENABLE EXTERNAL INTERRUPTS	3-4-25
ENDE	(E)DE	END EDIT	3-4-30
ENDF	(E)D5	END FLOAT	3-4-30
ENTR	(P)AB	ENTER	3-4-20
EQUL	(P)8C	EQUAL	3-4-13
EVAL	(P)AC	EVALUATE DESCRIPTOR	3-4-20
EXCH	(P)B6	EXCHANGE	3-4-14
EXIT	(P)A3	EXIT	3-4-20
EXPU	(P)DD	EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE	3-4-10
EXSD	(P)D2	EXECUTE SINGLE MICRO DESTRUCTIVE	3-4-9
EXSU	(P)DA	EXECUTE SINGLE MICRO, UPDATE	3-4-10
		F	
FLTR	(P)98	FIELD TRANSFER	3-4-20
FMFR	(V)AC	FETCH MEMORY FAIL REGISTER	3-4-28
		G	
GREQ	(P)89	GREATER THAN OR EQUAL	3-4-13
GRTR	(P)8A	GREATER THAN	3-4-13
		H	
HALT	(P)DF	CONDITIONAL HALT	3-4-22
HALT	(V)DF	CONDITIONAL HALT	3-4-22
HALT	(E)DF	CONDITIONAL HALT	3-4-22
		I	
ICVD	(P)CA	INPUT CONVERT, DESTRUCTIVE	3-4-11
ICVU	(P)CB	INPUT CONVERT, UPDATE	3-4-11
IDIV	(P)84	INTEGER DIVIDE	3-4-6
IDLE	(V)44	IDLE UNTIL INTERRUPT	3-4-25
IGPR	(V)48	IGNORE PARITY	3-4-28
IMKS	(P)CF	INSERT MARK STACK	3-4-19
INCN	(V)8F	INTERRUPT CHANNEL N	3-4-28
INDX	(P)A6	INDEX	3-4-10
INOP	(E)D8	INSERT OVERPUNCH	3-4-29
INSC	(E)DD	INSERT CONDITIONAL	3-4-29
INSG	(E)D9	INSERT DISPLAY SIGN	3-4-29
INSR	(P)9C	FIELD INSERT	3-4-21
INSU	(E)DC	INSERT UNCONDITIONAL	3-4-29
ISOL	(P)9A	FIELD ISOLATE	3-4-20
		J	
JOIN	(V)42	SET TWO SINGLES TO DOUBLE	3-4-21
		L	
LAND	(P)90	LOGICAL AND	3-4-12
LEQV	(P)93	LOGICAL EQUIVALENCE	3-4-12
LESS	(P)88	LESS THAN	3-4-13
LLLU	(V)BD	LINKED LIST LOOKUP	3-4-26
LNOT	(P)92	LOGICAL NEGATE	3-4-12

Mnemonic	Mode ID & Hex Code	Operator Name	Page
LOAD	(P)BD	LOAD	3-4-11
LODT	(V)BC	LOAD TRANSPARENT	3-4-11
LOG2	(V)8B	LEADING ONE TEST	3-4-7
LOR	(P)91	LOGICAL OR	3-4-12
LSEQ	(P)8B	LESS THAN OR EQUAL	3-4-13
LT16	(P)B3	LIT CALL 16 BITS	3-4-11
LT48	(P)BE	LIT CALL 48 BITS	3-4-11
LT8	(P)B2	LIT CALL 8 BITS	3-4-11
M			
MCHR	(E)D7	MOVE CHARACTERS	3-4-29
MFLT	(E)D1	MOVE WITH FLOAT	3-4-30
MINS	(E)D0	MOVE WITH INSERT	3-4-29
MKST	(P)AE	MARK STACK	3-4-19
MPCW	(P)BF	MAKE PROGRAM CONTROL WORD	3-4-12
MULT	(P)82	MULTIPLY	3-4-6
MULX	(P)8F	EXTENDED MULTIPLY	3-4-6
MVNU	(E)D6	MOVE NUMERIC	3-4-29
MVST	(V)AF	MOVE TO STACK	3-4-27
N			
NAMC	(P)40 →7F	NAME CALL	3-4-19
NEQL	(P)8D	NOT EQUAL	3-4-13
NOOP	(P)FE	NO OPERATION	3-4-22
NOOP	(V)FE	NO OPERATION	3-4-22
NOOP	(E)FE	NO OPERATION	3-4-22
NTGD	(V)87	INTEGERIZE, ROUNDED, DOUBLE PRECISION	3-4-6
NTGR	(P)87	INTEGERIZE, ROUNDED	3-4-6
NTIA	(P)86	INTEGERIZE, TRUNCATED	3-4-6
NVLD	(P)FF	INVALID OPERATOR	3-4-22
NVLD	(V)FF	INVALID OPERATOR	3-4-22
NVLD	(E)FF	INVALID OPERATOR	3-4-22
NXLN	(P)A5	INDEX AND LOAD NAME	3-4-10
NXLV	(P)AD	INDEX AND LOAD VALUE	3-4-10
O			
OCRX	(V)85	OCCURS INDEX	3-4-27
ONE	(P)B1	LIT CALL ONE	3-4-11
OVRD	(P)BA	OVERWRITE DESTRUCTIVE	3-4-15
OVRN	(P)BB	OVERWRITE NON-DESTRUCTIVE	3-4-15
P			
PACD	(P)D1	PACK DESTRUCTIVE	3-4-12
PACU	(P)D9	PACK UPDATE	3-4-13
PAUS	(V)84	PAUSE UNTIL INTERRUPT	3-4-28
PUSH	(P)B4	PUSH DOWN STACK REGISTERS	3-4-15
R			
RDIV	(P)85	REMAINDER DIVIDE	3-4-6
RDLK	(V)BA	READ WITH LOCK	3-4-15
RETN	(P)A7	RETURN	3-4-20
ROFF	(P)D7	READ AND CLEAR OVERFLOW FLIP-FLOP	3-4-22
RPRR	(V)B8	READ PROCESSOR REGISTER	3-4-25
RSDN	(V)B7	ROTATE STACK DOWN	3-4-14
RSTF	(E)D4	RESET FLOAT	3-4-30
RSUP	(V)B6	ROTATE STACK UP	3-4-15
RTAG	(V)B5	READ TAG FIELD	3-4-24
RTFF	(P)DE	READ TRUE/ FALSE FLIP-FLOP	3-4-22
S			
SAME	(P)94	LOGICAL EQUAL	3-4-13
SCLF	(P)C0	SCALE LEFT	3-4-13

Mnemonic	Mode ID & Hex Code	Operator Name	Page
SCNI	(V)4A	SCAN IN (TOD ONLY)	3-4-23
SCRF	(P)C6	SCALE RIGHT FINAL	3-4-14
SCRR	(P)C8	SCALE RIGHT ROUNDED	3-4-14
SCRS	(P)C4	SCALE RIGHT SAVE	3-4-14
SCRT	(P)C2	SCALE RIGHT TRUNCATE	3-4-14
SEQD	(V)F4	SCAN WHILE EQUAL DESTRUCTIVE	3-4-23
SEQU	(V)FC	SCAN WHILE EQUAL, UPDATE	3-4-23
SFDC	(E)DA	SKIP FORWARD DESTINATION CHARACTERS	3-4-30
SFSC	(E)D2	SKIP FORWARD SOURCE CHARACTERS	3-4-30
SGED	(V)F1	SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE	3-4-23
SGEU	(V)F9	SCAN WHILE GREATER OR EQUAL, UPDATE	3-4-23
SGTD	(V)F2	SCAN WHILE GREATER, DESTRUCTIVE	3-4-23
SGTU	(V)FA	SCAN WHILE GREATER, UPDATE	3-4-23
SINH	(V)A8	SET MEMORY INHIBITS	3-4-28
SINT	(V)45	SET INTERVAL TIMER	3-4-25
SISO	(P)D5	STRING ISOLATE	3-4-16
SLED	(V)F3	SCAN WHILE LESS OR EQUAL, DESTRUCTIVE	3-4-23
SLEU	(V)FB	SCAN WHILE LESS OR EQUAL, UPDATE	3-4-23
SLMT	(V)AA	SET MEMORY LIMITS	3-4-28
SLSD	(V)F0	SCAN WHILE LESS, DESTRUCTIVE	3-4-24
SLSU	(V)F8	SCAN WHILE LESS, UPDATE	3-4-24
SNED	(V)F5	SCAN WHILE NOT EQUAL, DESTRUCTIVE	3-4-24
SNEU	(V)FD	SCAN WHILE NOT EQUAL, UPDATE	3-4-24
SNGL	(P)CD	SET TO SINGLE PRECISION, ROUNDED	3-4-21
SNGT	(P)CC	SET TO SINGLE PRECISION, TRUNCATED	3-4-21
SPLT	(V)43	SET DOUBLE TO TWO SINGLES	3-4-21
SPRR	(V)B9	SET PROCESSOR REGISTER	3-4-25
SRCH	(V)BE	MASKED SEARCH FOR EQUAL	3-4-27
SRDC	(E)DB	SKIP REVERSE DESTINATION CHARACTERS	3-4-30
SRSC	(E)D3	SKIP REVERSE SOURCE CHARACTER	3-4-30
STAG	(V)B4	SET TAG FIELD	3-4-24
STFF	(P)AF	STUFF ENVIRONMENT	3-4-22
STOD	(P)B8	STORE DESTRUCTIVE	3-4-15
STON	(P)B9	STORE NON-DESTRUCTIVE	3-4-15
STOP	(V)BF	STOP	3-4-28
SUBT	(P)81	SUBTRACT	3-4-5
SWFD	(V)D4	SCAN WHILE FALSE, DESTRUCTIVE	3-4-24
SWFU	(V)DC	SCAN WHILE FALSE, UPDATE	3-4-24
SWTD	(V)D5	SCAN WHILE	

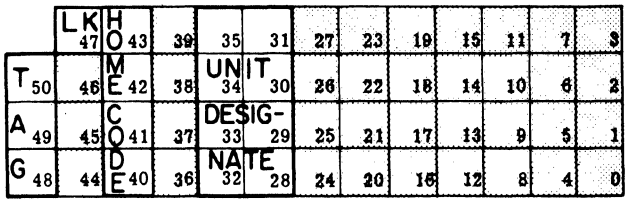
Mnemonic	Mode ID & Hex Code	Operator Name	Page
		TRUE, DESTRUCTIVE	3-4-24
SWTU	(V)DD	SCAN WHILE TRUE, UPDATE	3-4-24
SXSN	(P)D6	SET EXTERNAL SIGN	3-4-22
	T		
TEED	(P)D0	TABLE ENTER EDIT, DESTRUCTIVE	3-4-9
TEEU	(P)D8	TABLE ENTER EDIT, UPDATE	3-4-9
TEQD	(P)E4	TRANSFER WHILE EQUAL, DESTRUCTIVE	3-4-17
TEQU	(P)EC	TRANSFER WHILE EQUAL, UPDATE	3-4-17
TGED	(P)E1	TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE	3-4-17
TGEU	(P)E9	TRANSFER WHILE GREATER OR EQUAL, UPDATE	3-4-17
TGTD	(P)E2	TRANSFER WHILE GREATER, DESTRUCTIVE	3-4-17
TGTU	(P)EA	TRANSFER WHILE GREATER, UPDATE	3-4-17
TLED	(P)E3	TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE	3-4-17
TLEU	(P)EB	TRANSFER WHILE LESS OR EQUAL, UPDATE	3-4-17
TLSD	(P)E0	TRANSFER WHILE LESS, DESTRUCTIVE	3-4-17
TLSU	(P)E8	TRANSFER WHILE LESS, UPDATE	3-4-17
TNED	(P)E5	TRANSFER WHILE NOT EQUAL, DESTRUCTIVE	3-4-18
TNEU	(P)ED	TRANSFER WHILE NOT EQUAL, UPDATE	3-4-18
TRNS	(V)D7	TRANSLATE	3-4-22
TUND	(P)E6	TRANSFER UNCONDITIONAL DESTRUCTIVE	3-4-18
TUNU	(P)EE	TRANSFER UNCONDITIONAL UPDATE	3-4-18
TWFD	(V)D2	TRANSFER WHILE FALSE, DESTRUCTIVE	3-4-18
TWFU	(V)DA	TRANSFER WHILE FALSE, UPDATE	3-4-18
TWOD	(P)D41	TRANSFER WORDS OVERWRITE, DESTRUCTIVE	3-4-16
TWOU	(P)DC	TRANSFER WORDS OVERWRITE, UPDATE	3-4-17
TWSD	(P)D3	TRANSFER WORDS, DESTRUCTIVE	3-4-16
TWSU	(P)DB	TRANSFER WORDS, UPDATE	3-4-16
TWTD	(V)D3	TRANSFER WHILE TRUE, DESTRUCTIVE	3-4-18
TWTU	(V)DB	TRANSFER WHILE TRUE, UPDATE	3-4-18
	U		
UABD	(V)D1	UNPACK ABSOLUTE, DESTRUCTIVE	3-4-25
UABU	(V)D9	UNPACK ABSOLUTE, UPDATE	3-4-25
USND	(V)D0	UNPACK SIGNED, DESTRUCTIVE	3-4-25

Mnemonic	Mode ID & Hex Code	Operator Name	Page
USNU	(V)D8	UNPACK SIGNED, UPDATE	3-4-26
		V	
VALC	(P)00→3F	VALUE CALL	3-4-19
VARI	(P)95	ESCAPE TO 16-BIT INSTRUCTION	3-4-22
		W	
WHOI	(V)4E	READ PROCESSOR IDENTIFICATION	3-4-25
		X	
XTND	(P)CE	SET TO DOUBLE PRECISION	3-4-21
		Z	
ZERO	(P)B0	LIT CALL ZERO	3-4-11

APPENDIX E

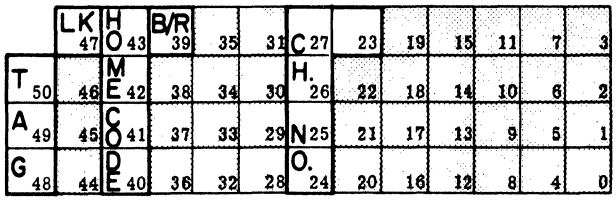
IOM WORD FORMATS

HA WORD 1, START I/O COMMAND



FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
-	46:3	Not used.
HOME CODE	43:4	Defines Start I/O command (0001).
-	39:4	Not used.
UNIT DESIG-NATE	35:8	A unique 8-bit code-used with the UT base address to index and lock fetch from memory the UT word for the device to be started, and used with the QH base address to unlock fetch from memory the QH word, which points to the IOCB base address.
-	27:28	Not used.

HA WORD 1, SET CHANNEL BUSY/RESERVED

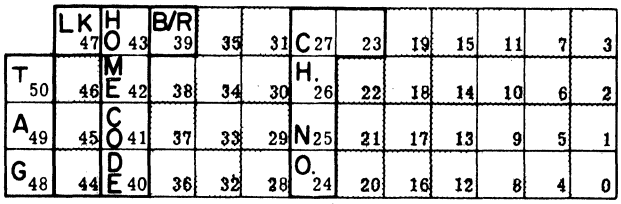


FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
-	46:3	Not used.
HOME CODE	43:4	Defines Set CH Busy/Set CH Reserved Commands (0010).
B/R	39:1	When reset, further defines command as Set CH Busy; when set, further defines command as Set CH Reserved.
-	38:11	Not used.
CH. NO.	27:5	Identifies one of the 28 possible IOM channels.
-	22:23	Not used.

HA WORD 1, SET CHANNEL BUSY/RESERVED

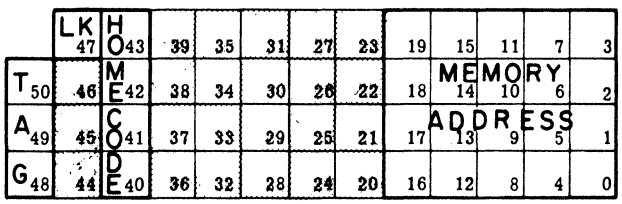
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
-	46:3	Not used.
HOME CODE	43:4	Defines Set CH Busy/Set CH Reserved Commands (0010).
B/R	39:1	When reset, further defines command as Set CH Busy; when set, further defines command as Set CH Reserved.
-	38:11	Not used.
CH. NO.	27:5	Identifies one of the 28 possible IOM channels.
-	22:23	Not used.

HA WORD 1, RESET CHANNEL RESERVED



FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
—	46:3	Not used.
HOME CODE	43:4	Defines Reset CH Busy/Reset CH Reserved Commands (0011).
B/R	39:1	When reset, further defines command as Reset CH Busy; when set, further defines command as Reset CH Reserved.
—	38:11	Not used.
CH. NO.	27:5	Identifies one of the 28 possible IOM channels.
—	22:23	Not used.

HA WORD 1, LOAD BASE ADDRESS (HA, UT, UOQH, SQ) COMMANDS

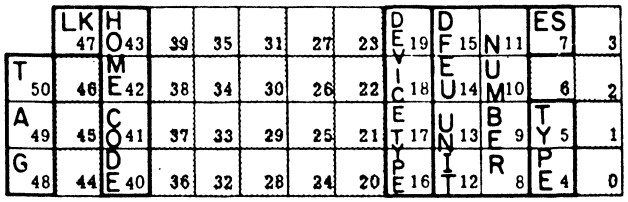


FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.

HA WORD 1, LOAD BASE ADDRESS (HA, UT, UOQH, SQ) COMMANDS

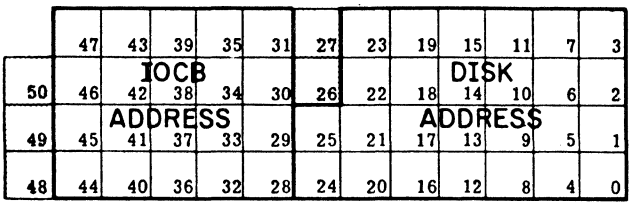
—	46:3	Not used.
HOME CODE	43:4	Defines: (1) Load Home Address Command (0100) (2) Load Unit Table Address Command (0101) (3) Load I/O Queue Head Address Command (0110) (4) Load Status Queue Address (0111).
—	39:20	Not used.
MEMORY ADDRESS	19:20	The memory address to be stored in the Translator of the IOM to enable access of the IOM Job Map.

HA WORD 1, DFO SCAN-OUT COMMANDS (CLEAR STACK AND STORE CW REQUEST)



FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes the word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
—	46:3	Not used.
HOME CODE	43:4	Defines the command as Scan Out (DFO or DCP) when 1000.
—	39:20	Not used.
DEVICE TYPE	19:4	Defines the Scan Out command is for a DFO (1001).
DFEU UNIT NMBR and ES (EXCH SELECT)	15:8, 7:1	Together define the DFO by specifying a DFEU unit number and whether it is directly (bit 7=0) or indirectly (bit 7=1) connected to the DFO.
—	6:1	Not used.
TYPE	5:2	Defines the DFO Scan-Out command as Clear the Stack (10) or Store Control Word Request (01).
—	3:4	Not used.

HA WORD 2, DFO SCAN OUT/STORE CONTROL WORD REQUEST COMMAND

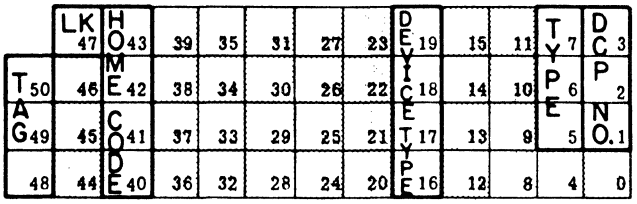


FIELD	BITS	DESCRIPTION
—	50:3	Not used.
IOCB ADDRESS	47:20	The base address of the job in memory.
—	27:2	Not used.
DISK ADDRESS	25:26	The disk address to be used for the job.

NOTE

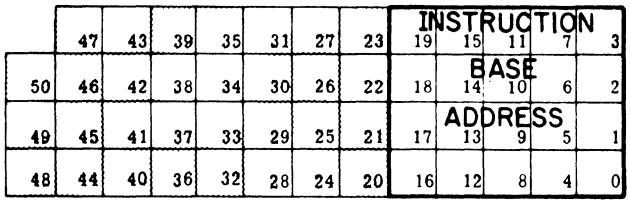
This format also represents the format of the Scan Information word sent to the DFO.

HA WORD 1, DCP SCAN-OUT COMMANDS (INITIATE, HALT, SET ATTENTION)



FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes the word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
—	46:3	Not used.
HOME CODE	43:4	Defines the command as Scan Out (DFO or DCP) when 1000.
—	39:20	Not used.
DEVICE TYPE	19:4	Defines the Scan Out command is for a DCP (110).
—	15:8	Not used.
TYPE	7:3	Defines the DCP Scan-Out command as Initiate (000), Halt (010), or Set Attention (100).
—	4:1	Not used.
DCP NO.	3:3	Defines the DCP for which the command is intended.
—	0:1	Not used.

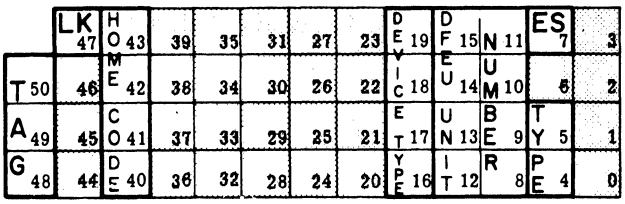
HA WORD 2, DCP SCAN-OUT/INITIATE COMMAND



FIELD	BITS	DESCRIPTION
—	50:31	Not used.
INSTRUCTION BASE ADDRESS	19:20	Define the memory base addres the DCP code.

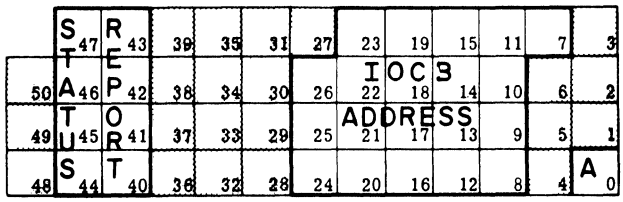
NOTE
this format also represents the format of the Scan Information word sent to the DCP.

HA WORD 1, DFO SCAN-IN COMMANDS (QUEUED CONTROL WORD, TOP OF STACK, REPORT)



FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes the word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
—	46:3	Not used.
HOME CODE	43:4	Defines the command as Scan in (1001).
—	39:20	Not used.
DEVICE TYPE	19:4	Defines the command as for a DFO (1001).
DFEU UNIT NMBR and ES (EXCH SEL)	15:8, 7:1	Together define the DFO by specifying a DFEU unit number and whether it is directly or indirectly connected to the DFO (via an exchange). These fields are not used for the Scan-In DFO Report Command.
—	6:1	Not used.
TYPE	5:2	Defines the DFO Scan-Out command as either Queued Control Word (01), Top of Stack (10), or Report (11).
—	3:4	Not used.

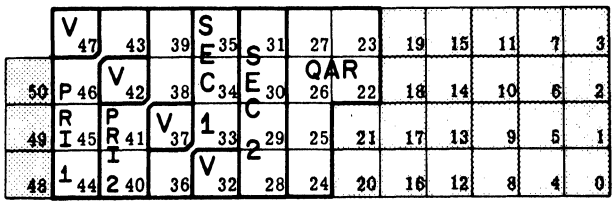
HA WORD 2 (SCAN-IN WORD), SCAN-IN DFO QUEUED CONTROL WORD AND TOP OF STACK COMMANDS



FIELD	BITS	DESCRIPTION
—	50:3	Not used.
STATUS REPORT	47:8	Describes the nature of the DFO by bits set as follows: (1) 47 set = No Access to Exchange (2) 46 set = SU Not Available (3) 45 set = Parity Error (4) 44 set = Disk Address Error (5) 43 set = Queded Control Word (6) 42 set = Top of Stack Control Word (7) 41 set = Stack Empty (8) 40 set = Control Word Not Available
—	39:13	Not used.
IOCB ADDR	26:20	Defines the memory address of the IOCB.
—	6:6	Not used.
A (ATTEN)	0:1	When set, alerts the IOM to examine the STATUS REPORT FIELD.

NOTE:
This format also represents the format of the Scan Information word received from the DFO.

HA WORD 2 SCAN-IN DFO REPORT COMMAND



FIELD	BITS	DESCRIPTION
-	50:3	Not used.
V	47:1	When true indicates connection of an EU/DFO bus at port 1 and the EUs connected to this bus are referenced by the EUD code present on lines 43-46 of the Scan Information Lines.
PRI 1	46:4	EUD code for port 1 bus (bit 46=MSB).

V	42:1	When true indicates connection of an EU/DFO bus at port 2 and the EUs connected to this bus are referenced by the EUD code present on lines 38-41.
PRI 2	41:4	EUD code for port 2 bus (bit 41=MSB).
V	37:1	When true indicates connection of an EU/DFO bus at port 3, and the EUs connected to this bus are referenced by the EUD code present on lines 33-36.
SEC 1	36:4	EUD code for port 3 bus (bit 36=MSB).
V	32:1	When true indicates connection of an EU/DFO bus at port 4, and the EUs connected to this bus are referenced by the EUD code present on lines 28-31.
SEC 2	31:4	EUD code for port 4 bus (bit 31=MSB).

NOTE

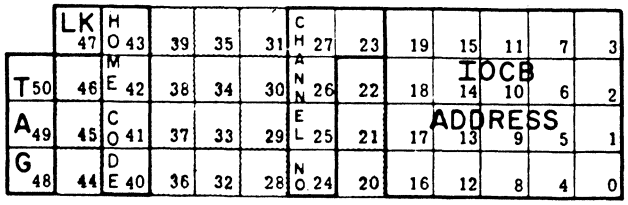
If a given EUD code appears on Scan Information lines 38-41 or 43-46, then the EUs referenced by the code are connected to the responding DFO in a direct manner, but if the EUD code appears on lines 28-31 or 33-36, then the EUs referenced by the EUD code are connected to the responding DFO indirectly (that is, via the other DFO of the DFO-pair).

QAR	27:6	Indicates capacity of memory stack (bit 27=MSB).
-	21:22	Not used.

NOTE

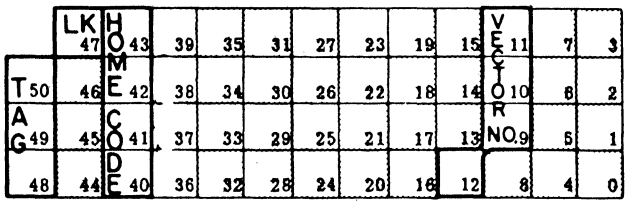
This format also represents the format of the Scan Information word received from the DFO.

HA WORD 1, SYNC I/O COMMAND



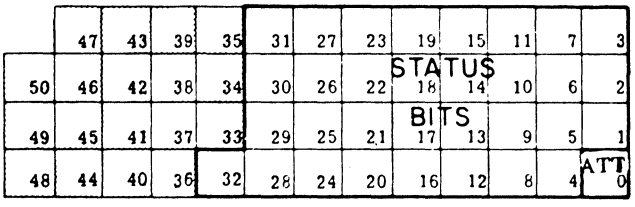
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes the word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
—	46:3	Not used.
HOME CODE	43:4	Defines the command as Sync I/O.
—	29:12	Not used.
CHANNEL NO.	27:5	Identifies one of the 28 possible IOM channels.
—	22:3	Not used.
IOCB ADDRESS	19:20	The address of the job request in memory.

HA WORD 1, INTERROGATE PERIPHERAL STATUS COMMAND



FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes the word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM use. Resets when IOM services HA words.
—	46:3	Not used.
HOME CODE	43:4	Defines Interrogate Peripheral Status Command (1011).
VECTOR NO.	12:4	Defines the number of the status vector to be interrogated.
—	8:9	Not used.

HA WORD 2 (STATUS WORD RETURNED), INTERROGATE PERIPHERAL STATUS COMMAND



FIELD	BITS	DESCRIPTION
—	50:18	Not used.
STATUS BITS	32:32	Each bit of this field, when on, indicates the ready status of the associated unit on the vector. (Refer to table I-1 for referencing the ready status vector, ready status bit, and device number of any peripheral device.)
ATT	0:1	When set, alerts the IOM to examine the STATUS BITS field.

Table E-1. Status Vector Cross Reference

VECTOR BIT NO.	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		VECTOR
"B" REGISTER	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
31 - 0																																	ATTENTION BIT	0
63 - 32																																		1
95 - 64																																		2
UNIT 127 - 96																																		3
DEST. 159 - 128																																		4
191 - 160																																		5
223 - 192																																		6
255 - 224																																		7

TO FIND THE STATUS VECTOR FOR A UD NUMBER, DIVIDE THE UD NUMBER BY 32. THE STATUS VECTOR IS THE INTERGER QUOTIENT AND THE VECTOR BIT IS THE REMAINDER PLUS ONE.

EXAMPLE: UD NUMBER = 95

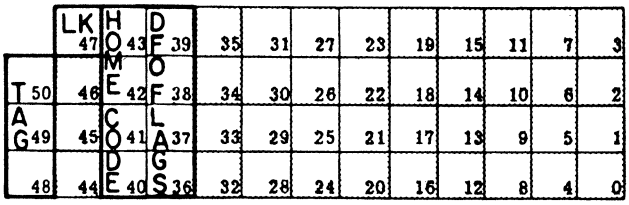
$$32\overline{)95}\quad 2\ \text{(STATUS VECTOR)},\quad 64,\quad 31 + 1 = 32\ \text{(VECTOR BIT NUMBER)}$$

TO FIND THE UD NUMBER, MULTIPLY THE STATUS VECTOR NUMBER BY 32 AND ADD TO THE RESULT THE VECTOR BIT NUMBER MINUS ONE.

EXAMPLE: STATUS VECTOR NUMBER = 2

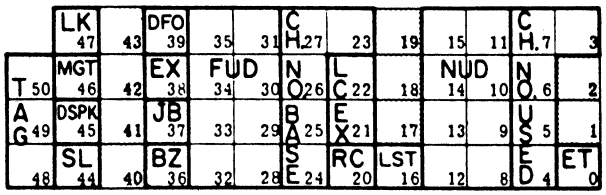
$2 \times 32 = 64 + (32 - 1) = 95$ (UD NUMBER)

HA WORD 1, INHIBIT IOM, ACTIVATE IOM, AND LOAD DFO FLAGS COMMANDS



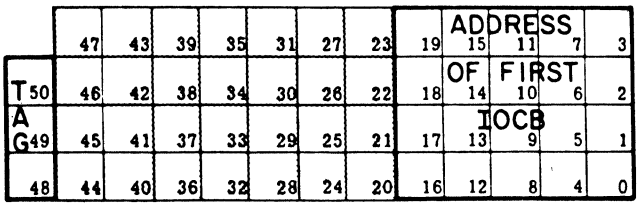
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set by software indicates the HA words are available for IOM services.
HOME CODE	43:4	Defines the command as follows: (1) 1100 = Inhibit IOM (2) 1101 = Activate IOM (3) 1110 = Load DFO Flags
	39:4	DFO Flags (for LOAD DFO FLAGS command only).
	35:36	Not Used.

UNIT TABLE WORD



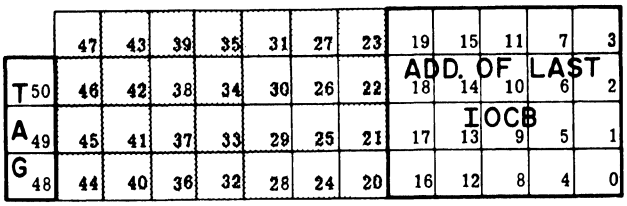
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set, indicates the UT word is being operated on.
MGT	46:1	When set, indicates this job request is for a magnetic tape. (Set by software.)
DSPK	45:1	When set, indicates this job request is for a disk pack. (Set by software.)
SL	44:1	When set, indicates the presence of a side link in IOCB+1.
—	43:4	Not used.
DFO	39:1	When set, indicates unit is under control of a DFO. A ring walk will not be performed with this bit set. (Set by software.)
EX	38:1	When set, indicates the unit is connected to an exchange. A ring walk will be performed (if the job bit is set) with this bit set. (Set by software.) Not used if bit 39 is set.
JB	37:1	When set, indicates that all channels associated with this request were busy, and when a channel becomes free and no further request are queued for that device, this job is to be done. (Set by IOM.) Used only with exch. devices (Bit 38=1). Not used with DFO (Bit 39).
BZ	36:1	When set, indicates that this unit is busy. (Set by IOM.)
FUD	35:8	Points to the First Unit Designate Number connected to the exchange.
CH. NO. BASE	27:5	For units not on an exchange, the number of the channel to which this unit is connected. For units on an exchange, the lowest numbered channel to which the exchange is connected. NOTE: CN 0 and 21 through 23 are unassigned and will cause a fail.
LCEX	22:2	Indicates the 2 least significant bits of the last channel number of the exchange, for the device to be used.
RC	20:1	When set, permits this unit to use a reserved channel.
—	19:3	Not used.
LST	16:1	When set, indicates this is the last Unit Designate on the exchange.
NUD	15:8	Points to the Next Unit Designate number connected to the exchange.
CH. NO. USED	7:5	These bits specify the channel that was used to service the device. (Set by IOM.)
—	2:2	Not used.
ET	0:1	When set, indicates that an error condition has been reported in the current Result Descriptor, and therefore additional jobs should not be initiated for the unit. This bit is normally reset by software.

IOQH WORD



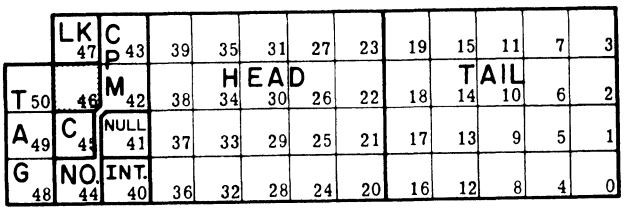
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
—	47:28	Not used.
ADDRESS OF FIRST IOCB	19:20	Address of 1st IOCB in the IOQ. If bits 19-0 are null (zero), the UT word is unlocked and restored to memory.

IOQT WORD



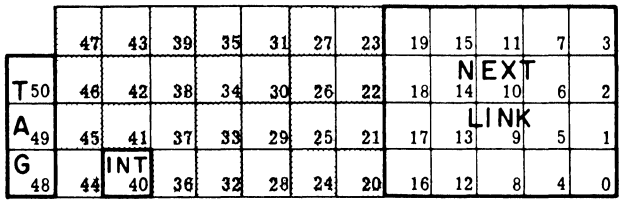
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
—	47:28	Not used.
ADD OF LAST IOCB	19:20	Address of last IOCB in the IOQ.

SQH WORD



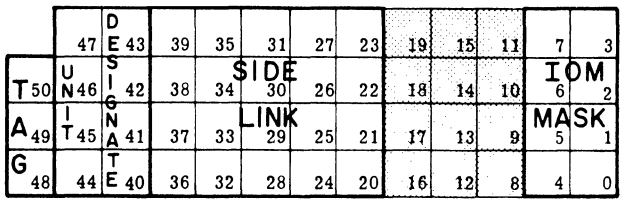
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
LK	47:1	When set, indicates the SQH word is being operated on.
—	46:1	Not used.
C	45:1	Notifies software, when set, that a status change vector has occurred.
CPM NO.	44:3	Points to the CPM that will be interrupted by either channel interrupt or error interrupt.
NULL	41:1	When a 0, indicates that the queue is empty; when a 1, indicates terminated jobs are under queue.
INT	40:1	When set, (set by software) indicates that the CPM number field shall be interrupted upon job termination. (Reset by IOM)
HEAD	39:20	A 20-bit address pointing to the IOCB of the first device terminated. (Not used if bit 41 = 0)
TAIL	19:20	A 20-bit address pointing to the IOCB of the last device terminated. (Not used if 41 = 0)

IOCB WORD 0 (IOCB I/O LINKAGE (N/L) WORD)



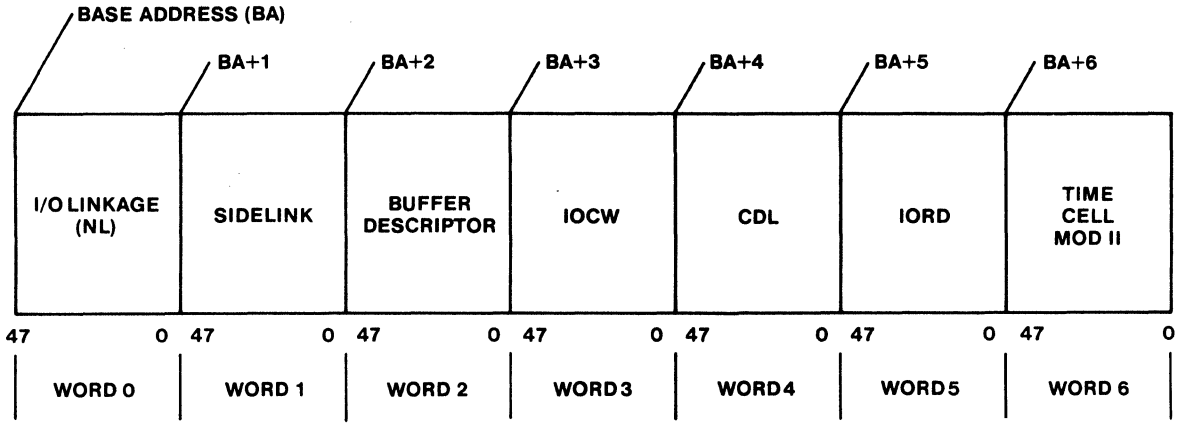
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
—	47:7	Not used.
INT	40:1	When set, notifies the IOM to interrupt the CPM specified in the SQ word upon completion of this job.
—	39:20	Not used.
NEXT LINK	19:20	Memory address of the next job (IOCB) queued for this device.

IOCB WORD 1 (IOCB SIDELINK (SL) WORD)



FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes the word is single precision (000).
UNIT DESIGNATE	47:8	Defines the device which is to perform this sidelinked job.
SIDE LINK	39:20	Memory address of the sidelinked job.
-	19:12	Not used.
IOM MASK	7:8	Defines an IOM channel number and thus defines the IOM (or IOM's) which can perform the sidelinked job.

I/O CONTROL BLOCK (IOCB)



* WORDS 7 THRU N ARE RESERVED FOR SOFTWARE USE ONLY

ET1286

IOCB WORD 2 (IOCB BUFFER DESCRIPTOR (BD) WORD)

	47	43	CT EXT 39	35	31	27	23	19	15	11	7	3
T 50	46	42	38	LENGTH 34	30	26	22	18	BASE 14	10	6	2
AG 49	45	41	T 37	33	29	25	21	17	ADDRESS 13	9	5	1
48	44	40	36	32	28	24	20	16	12	8	4	0

FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes word is single precision (000).
-	47:8	Not used.
CT EXT	39:3	If the length of the buffer includes a fractional part of a word, this field describes the number of characters in that fractional part.
LENGTH	36:17	Describes the length of the buffer in words. (Excess characters are described by the CT EXT field.)
BASE ADDRESS	19:20	Describes the memory address of the first data word of the buffer.

IOCW (IOCB WORD 3)

	ASC 47	MINH 43	B/F 39	35	31	27	23	19	15	11	7	3
T 50	SL 46	TRA 42	T 38	34	30	26	22	18	14	10	6	2
AG 49	SA 45	FML 41	TAG 37	33	29	25	21	17	13	9	5	1
48	I/O 44	MP 40	CTL 36	32	28	24	20	16	12	8	4	0

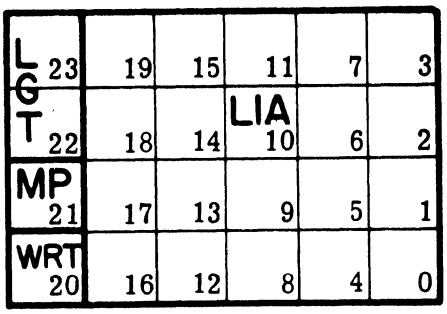
FIELD	BITS	DESCRIPTION
TAG	50:3	Denotes the word is single precision (000).
ASC	47:1	When set, indicates that ASCII translation is required.
SL	46:1	When set, indicates that a sidelink to another IOCW is required. (The address of the new IOCW is stored in bits 0 thru 19 of the IOCB SL word.)
SA	45:1	When set, will cause bit 1 of the result descriptor word (the Exception bit) to be set.
I/O	44:1	When set, indicates that the transfer is to be an input operation. When reset, indicates that the transfer is to be an output operation.
MINH	43:1	When set, indicates that data will not be transferred to/from memory.
TRA	42:1	When set, indicates that internal IOM translation is needed.
FML	41:1	When set, indicates that the frame length is to be 8-bits. When reset, indicates that the frame length is to be 6-bits.

MP	40:1	When set, indicates that a memory protect interrupt will occur if an attempt is made to store into a word in memory which has bit 48 = 1. The store will not occur.
B/F	39:1	When set, indicates a backward operation on a tape unit. When reset, indicates a forward operation on a tape unit.
T	38:1	When set, indicates a test operation.
TAG CTL	37:2	Indicates the following:

37	36	
0	0	Store single precision t:
1	1	Store double precision tags
0	1	Store program tags
1	0	Tag field transfer

35:36 Not used.

UNIT CONTROL WORD (UCW)

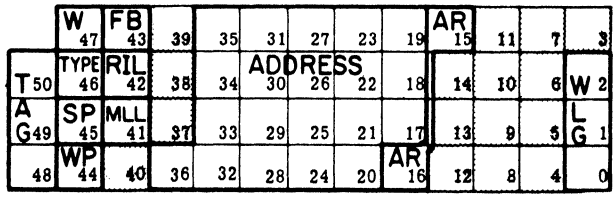


FIELD	BITS	DESCRIPTION
LGT	23:2	Specify the total length of the field being transferred as follows:

23	22	
0	1	= Transfer 1 word
1	0	= Transfer 2 words

MP	21:1	On a one or two word store, if bit 48 of the information word already stored in that memory location is a one (protected word), memory shall not perform the store but shall send an error signal to the requestor.
WRT	20:1	Shall identify the service request as a Read (WRT=0) or Write (WRT=1) operation.
L1A	19:20	Shall specify the absolute starting memory address of the transfer.

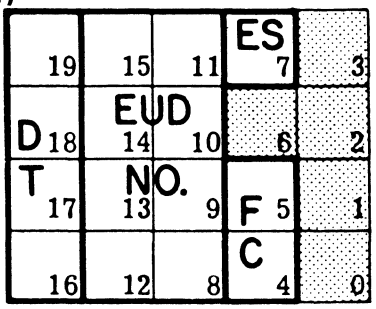
MEMORY CONTROL WORD



FIELD	BITS	DESCRIPTION
TAG	50:3	Not significant for control purposes; examined only for generation of parity.
W (Write)	47:1	When a 0 specifies that a read/restore operation is to be performed. When a 1, specifies that one of the write variations, as defined by the TYPE field, is to be performed.
TYPE	46:1	When the W field is a 1, specifies which write variation is to be performed as follows: when 46=0, a Clear/Write operation shall be performed (the Overwrite and Single-Word protected Write operations use this variation). When 46=1, a Read/Modify/Restore operation shall be performed (the N-Word Protected Write operation uses this variation). When the field is a 0 and 46=1, the contents of the Fail Register are fetched.

SP (Specifier)	45:1	When a 1, indicates that a single-word operation is to be performed. When a 0, indicates that an N-word operation is to be performed.
WP (Write Protect)	44:1	When a 1, indicates that a Protected Write operation is to be performed. It is a 0 if any other type of operation is specified.
FB (Flashback)	43:1	When a 1, specifies that the original contents of the memory location are to be sent to the requestor.
RIL (Requestor Inhibit Load)	42:1	Used in a Single-Word Overwrite operation to specify that a Load Requestor operation is to be performed. When a 1, specifies that the next data word sent to the MCM be loaded into the Requestor Inhibit Register instead of into memory.
MLL (Memory Limits Load)	41:1	When a 1, specifies that the next data word sent to the MCM be loaded into the Memory Limit Registers and the Available Register, instead of into memory.
ADDRESS	36:20	Specify the starting address for the memory operation.
AR (Address Residue)	16:2	Indicate the proper value (00, 01, or 10) that result from changes in the ADDRESS field.
—	14:12	Not used.
WLG (Word Length)	2:3	Indicates the number of words to be transferred during memory operations (2 words maximum).

DFO SCAN ADDRESS WORD (SCAN-IN AND SCAN-OUT)

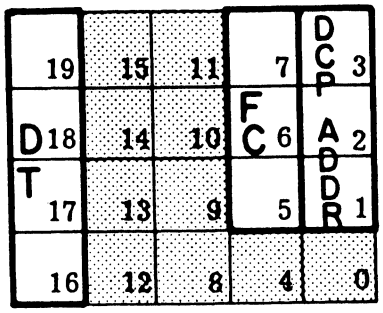


FIELD	BITS	DESCRIPTION
DT	19:4	Identifies the information as for a DFO (1001).
EUD NO. AND ES (EXCHANGE SELECT)	15:8, 7:1	Together define the DFO by specifying a DFEU unit designate number and whether it is directly or indirectly connected to DFO (via an exchange). These fields are not used if Scan-In DFO Report is the job to be implemented.
-	6:1	Not used.
FC	5:2	Function code which defines the operation as follows: (1) During Scan-Out: 5 4 0 1 = Store CW Request 1 0 = Clear-the-Stack (2) During Scan-In: 5 4 0 1 = Queued CW Request 1 0 = Top-of-Stack Request 1 1 = Report Request
	3:4	Not used.

NOTE

The format of the DFO Scan Address word may be related directly to bits 0 through 19 of HA word 1, when HA word 1 contains a command for DFO scan-out or scan-in.

DCP SCAN ADDRESS WORD

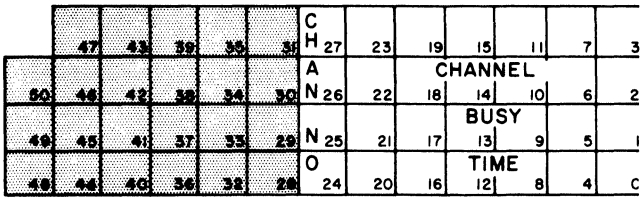


FIELD	BITS	DESCRIPTION
DT	19:4	Defines the Scan-Out command is for a DCP (1100).
—	15:8	Not used.
FC	7:3	Defines the DCP Scan-Out command as Initiate (000), Halt (010), or Set Attention (100).
—	4:1	Not used.
DCP ADDR	3:3	Defines the DCP for which the command is intended.
—	0:1	Not used.

NOTE

The format of the DCP Scan Address word may be related directly to bits 0 through 19 of HA word 1, when HA word 1 contains a DCP scan-out command.

IOM TIMECELL WORD



ET1287

NOTE

The channel busy time is in units of 503 microseconds for a total of 527.72 seconds.

Burroughs

PUBLICATION CHANGE NOTICE

PCN No.: 5010796-001 Date: February 26, 1981
Publication Title: B 7800 Information Processing Systems Reference Manual (August, 1979)
Other Affected Publications: —
Supersedes: —

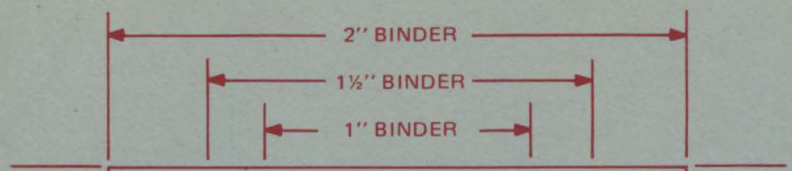
Description

This PCN contains changes and additions to the B 7800 Information Processing Systems Reference Manual, form 5010796, dated August 1979. Revisions to the text are indicated by black vertical bars on the affected pages.

Replace these pages			Add these pages
Title	3-3-1	4-4-1	3-2-45 thru 3-2-49
iii thru xi	3-3-3	5-1-1 thru 5-1-15	3-3-4A
1-1-1	3-3-13	6-1-1	3-4-19 thru 3-4-29
2-2-3	3-3-15	6-1-5	D-5
2-2-5	3-4-1	6-1-7	
3-1-1	3-4-3	6-1-11	
3-2-1 thru 3-2-43	3-4-9	6-1-13	
	3-4-11	D-1	
	3-4-15	D-3	
	3-4-17		

Retain this PCN cover page as a record of changes made to the basic publication.

The above pages covering
PCN 5010796-001



B 7800 Information Processing Systems

REFERENCE MANUAL

5010796

Printed in U.S.A.