operational characteristics of the processors for the Burroughs

B5000

5000—21005
Revision A

The
OPERATIONAL CHARACTERISTICS
of the
PROCESSORS
for the
Burroughs B 5000

SALES TECHNICAL SERVICES
Equipment and Systems Marketing Division

Burroughs Corporation
DETROIT 32, MICHIGAN

TABLE OF CONTENTS

Section	Title	Page
1	INTRODUCTION	1-1
	General	1-1
	System Description	1-1
	Processor Module A	1-1
	Processor Module B	1-2
	Input/Output Channels	1-2
	Memory Modules	1-2
	Storage Drums	1-3
	Card Readers	1-3
	Line Printer	1-3
	Card Punch	1-4
	Magnetic Tape Units	1-4
2	ORGANIZATION	2-1
	General	2-1
	Operation	2-1
	States	2-1
	Levels	2-2
	Modes	2-2
	Programing	2-3
	Syllables	2-3
	Word Mode Syllables	2-3
	Character Mode Syllables	2-3
	Descriptors	2-3
	Program Descriptors	2-4
	Data Descriptors	2-4
	Syllable-Descriptor Operation	2-4
	Operands	2-5
	Control Words	2-5
	Registers	2-6
	Program Registers	2-6
	Primary Stack and Source Registers	2-6
	Secondary Stack and Destination Registers	2-7
	Utility Registers	2-7
	Interrupt System	2-8
	Independent Interrupts	2-8
	Dependent Interrupts	2-9
	Interrupt Register	2-9
	Interrupt Detection	2-10
	Word Mode Interrupt	2-10
	Character Mode Interrupt	2-10
	Return Normal	2-11
	Parallel and Serial Arithmetic	2-11
	Parallel	2-11
	Serial	2-11
	Communication	2-11
	Dual Processors	2-12

TABLE OF CONTENTS (Continued)

Section	Title	Page
3	OPERATION	3-1
	General	3-1
	Storage	3-1
	Stacks	3-2
	Program Reference Table	3-2
	Program Segments	3-3
	Data Storage	3-4
	Input/Output Areas	3-4
	Subroutines	3-4
	Mark Stack Flip-Flop	3-4
	Subroutine Entry	3-5
	Subroutine Exit	3-5
	Addressing	3-6
	Programs	3-6
	Data Addressing	3-6
	Input/Output Addressing	3-8
	Subroutine Addressing	3-8
	Data Editing	3-8
4	DESCRIPTORS	4-1
	General	4-1
	Program Segments	4-2
	Program Descriptor	4-2
	Data and Input/Output	4-3
	Data Descriptor	4-3
	Supervisory Printer Descriptor	4-4
	Keyboard Descriptor	4-5
	Drum Read Descriptor	4-6
	Drum Write Descriptor	4-7
	Card Read Descriptor	4-8
	Card Punch Descriptor	4-9
	Line Printer Descriptor	4-10
	Magnetic Tape Read Descriptor	4-11
	Magnetic Tape Write Descriptor	4-12
	Paper Tape Read Descriptor	4-13
	Paper Tape Write Descriptor	4-14
	External Control Descriptor	4-15
	External Result Descriptors	4-16
5	SYLLABLES	5-1
	General	5-1
	Word Mode	5-1
	Syllable Description	5-2
	Literal	5-2
	Operand Call	5-2
	Descriptor Call	5-2
	Operators	5-2
	Character Mode	5-8
	Syllable Description	5-9
	Operators	5-9
	Control State	5-14
	Syllable Description	5-14

APPENDIXES

APPENDIX A	Program Operators—Functional Listing	A-1
APPENDIX B	Program Operators—Alphabetical Listing by Mnemonic Codes	B-1

LIST OF ILLUSTRATIONS

FIGURE	TITLE	PAGE
1-1	Processor A Configuration	1-2
1-2	System Console	1-2
1-3	B 124 Card Reader	1-3
1-4	B 122 Card Reader	1-3
1-5	B 321 Line Printer	1-3
1-6	B 303 Card Punch	1-3
1-7	B 304 Card Punch	1-4
1-8	B 422 Magnetic Tape Unit	1-4
2-1	Processor Organization	2-1
2-2	State Operation	2-1
2-3	Single Level Operation	2-2
2-4	Word Mode Syllable and Operand Format	2-2
2-5	Character Mode Syllable and Data Word	2-2
2-6	Mode Operation	2-2
2-7	Program Word—Word Mode	2-3
2-8	Program Word—Character Mode	2-3
2-9	Program Descriptor	2-4
2-10	Data Descriptor	2-4
2-11	Referencing A Descriptor	2-4
2-12	Syllable—Descriptor Table	2-4
2-13	Data Word—Word Mode	2-5
2-14	Fixed and Floating Point Representation	2-5
2-15	Data Word—Character Mode	2-5
2-16	Program Registers	2-6
2-17	Primary Stack and Source Registers	2-6
2-18	Secondary Stack and Destination Registers	2-7
2-19	Subroutine Nesting Control	2-7
2-20	Register Configuration	2-8
2-21	Interrupt Conditions	2-9
2-22	Resulting Stack in Memory—Interrupt Control	2-10
2-23	Resulting Stack in Memory—Loop Control	2-10
2-24	Parallel Binary Add Operation	2-11
2-25	Serial Decimal Adder	2-11
2-26	Dual Processor Registers	2-12
3-1	Storage Co-ordination	3-1
3-2	Stack Push-Down	3-2
3-3	Indexing the PRT	3-3
3-4	Program Segment Operation	3-3
3-5	Parameter and Temporary Storage for Subroutines	3-4
3-6	Subroutine Entry Options	3-5

LIST OF ILLUSTRATIONS (Continued)

FIGURE	TITLE	PAGE
3-7	Subroutine Exit Options	3-6
3-8	Constant Indexing	3-7
3-9	Variable Indexing	3-7
3-10	Multilevel Indexing	3-7
3-11	Sub-Program Level Call Syllable Formats	3-8
3-12	Data Word—Character Mode	3-9
3-13	Character Mode—Syllable Format	3-9
3-14	Source—Destination Operation	3-9

SECTION 1
INTRODUCTION

GENERAL

The purpose of this manual is to present the basic internal operations of the Processors for the B 5000 Information Processing System. It is intended to provide a reference for those familiar with the over-all system as set forth in the B 5000 DESCRIPTOR.

This manual describes the internal programing and operation of the processors. However, the primary programing techniques for which the B 5000 was designed do not require the programer to be familiar with the actual functions of the Processor. Highly efficient machine code programs are automatically generated by problem-oriented language compilers which are an integral part of the programing system.

The manual is divided into Sections, each of which discusses a general area of machine functions. The subjects include logical organization, basic concepts, and a description of internal programing codes. Other manuals may be obtained for information regarding the operating system (Master Control Program) and compiler programing techniques (ALGOL and COBOL).

It should be recognized that the B 5000 Processor is basically a problem-language (that is, compiler) oriented unit and, as such, its internal language represents a decided departure from conventionally organized processors. The purpose of this type of internal machine language is to permit the efficient compilation and execution of programs coded as problem statements. In fact, programs which are prepared independently may be processed simultaneously. The simplification of the man-machine communication problem represents a significant advance in the area of data processing as applied to computer systems.

Your local Burroughs Representative should be consulted for additional information concerning the operations of the B 5000.

SYSTEM DESCRIPTION

The B 5000 is a new, modular, high performance, solid state system with a radically different processor organization designed to permit users to use efficiently advanced problem-oriented programing languages. The system consists of truly modular components which provide a high degree of flexibility in tailoring the system to a wide range of applications.

The need for modularity is further emphasized by the tendency of computer systems to assume ever increasing work loads never contemplated during the inception of these systems. The over-all system is more fully described in other literature, but a brief description is included here for review purposes.

Processor Module A

The B 5280 Processor provides a new approach to internal computer organization. It is a high speed parallel computer which contains the registers, internal logic, and a parallel as well as a serial adder. Implementation of problem-oriented languages in an efficient manner is greatly simplified by the logical organization of this unit. The Module A group also includes the following units as shown in Figure 1-1.

A powerful Memory Exchange provides parallel access between the multiple memory module system, the processors, and the I/O Channels. This permits memory modules to communicate with any processor or with any I/O Channel at any time.

Similarly, for the input/output units, an Input/Output Exchange is included to permit individual units to communicate with any I/O Channel without prearranged connections. This allows complete freedom in the assignment of input/output units, simplifies the programing system, and maximizes the simultaneous use of peripheral units.

A comprehensive interrupt communication system, which provides automatic and instantaneous recog-

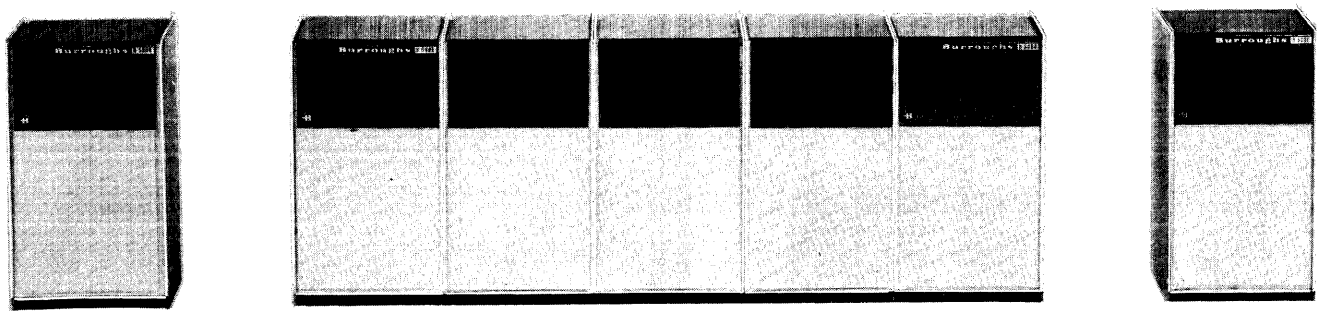


Figure 1-1. Processor A Configuration

nition for a wide range of control conditions, is an outstanding feature. To supplement the internal control of programing operations, an internal timer is included as part of the interrupt system. This provides a method for timing system operations.

The power control and supply equipment is also provided in the Module A grouping.

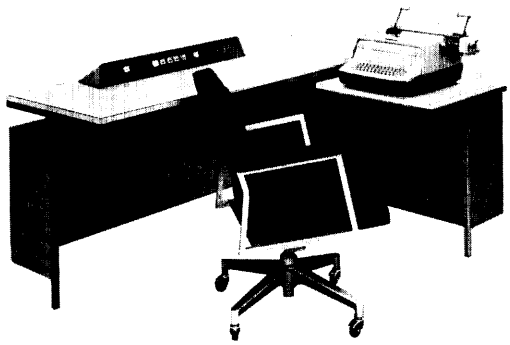


Figure 1-2. System Console

For general system supervision, a System Console (Figure 1-2) is provided as a means for observing the operating status of the system components. Direct communication between the operator and the system is available through the Supervisory Printer and associated keyboard.

Processor Module B

The B 5281 Processor is an optional unit for expanded processing capabilities. It contains logical, arithmetic and editing facilities identical to the B 5280 Processor. Each processor is independent and provides parallel computational and processing facilities. Processor B is contained within one cabinet and is connected to the processor A configuration.

Input/Output Channels

The B 5282 I/O Channel controls and transfers data to and from peripheral input/output equipment.

Up to four I/O channels are available in one B 5000 cabinet. These units permit instantaneous connections to be made between any input/output unit and any memory module. One channel can control and communicate with the maximum number of external devices available for the system. Additional channels provide the ability to perform simultaneous input/output operations.

Memory Modules

The high speed memory for the B 5000 is provided by the B 460 Memory Modules. Each module contains 4096 words of 49 bits each, including parity. Up to eight of these modules can be incorporated in the system. Two cabinets can be used. Each cabinet contains from 1 to 4 memory modules. Thus, a total capacity of 32,768 words of core storage is available to the system.

Each module contains its own addressing and accessing control. Through use of the Memory Exchange and the individual memory access register, multiple memory modules provide parallel access to stored information.

Storage Drums

Auxiliary storage for the B 5000 is available in the form of high speed, high capacity magnetic storage

drums. Each B 430 Storage Drum has a capacity of 32,768 words. These words are recorded parallel by bit, serial by character in frames of 6 bit characters. Each word consists of 48 bits plus a six bit parity frame. The drum organization includes 64 bands each with 512 interlaced words. Two drums can be included in the system. The drum cabinet may contain either one or two drums depending upon the system requirements.

Card Readers

Two card reader models are available for the B 5000. The B 124 Card Reader operates at 800 cards-per-minute and can handle 51, 60, 66, or 80 column cards. The B 122 Card Reader operates at 200 cards-per-minute. Two card readers in any combination can be included in the system. Both units use immediate access clutches and photoelectric reading devices.

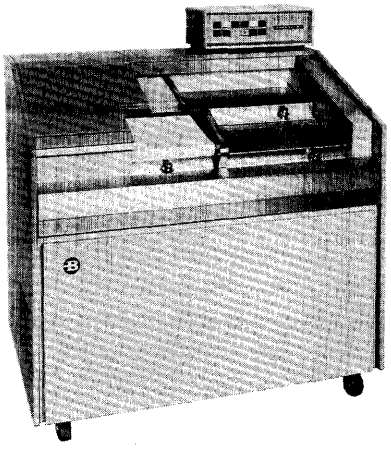


Figure 1-3. B 124 Card Reader

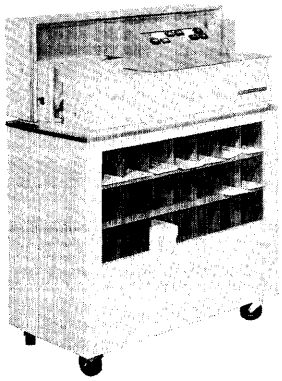


Figure 1-4. B 122 Card Reader

Line Printer

The B 321 Line Printer produces alphanumeric output at a rate of over 700 lines-per-minute. Each print line consists of 120 positions, spaced at 10 characters-per-inch horizontally. Vertical spacing may be either six or eight lines-per-inch. Two of these units can be included in the system.

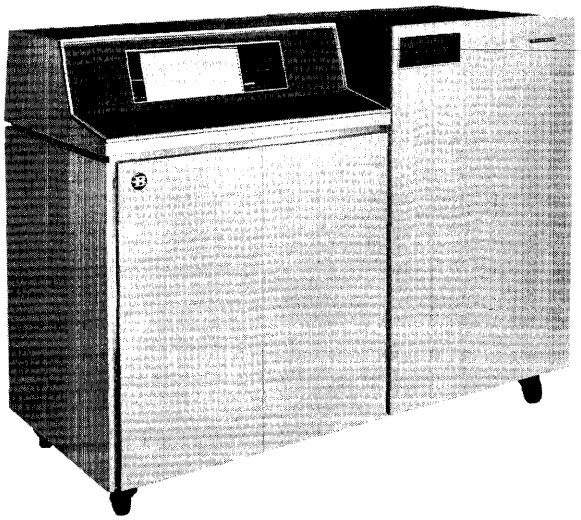


Figure 1-5. B 321 Line Printer

Card Punch

Two card punch models are available for the B 5000. The B 303 operates at 100 cards-per-minute and the B 304 operates at 300 cards-per-minute. Both punches contain an internal row buffer as standard equipment and perform a check on all information punched. The system will accommodate one punch unit.

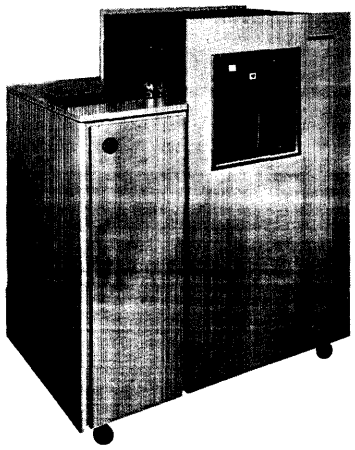


Figure 1-6. B 303 Card Punch

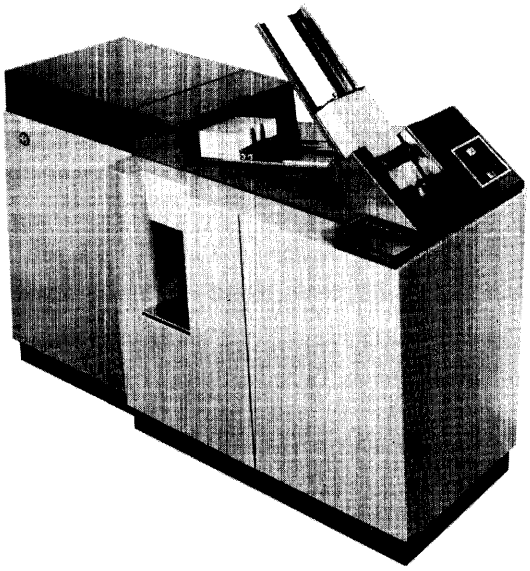


Figure 1-7. B 304 Card Punch

Magnetic Tape Units

The B 422 Magnetic Tape Unit provides high speed input/output and auxiliary storage for the system. Reels containing up to 2400 feet of tape can be processed at 120 inches-per-second and rewound at 320 inches-per-second. Recording density is either 200 or 555.5 alphanumeric frames-per-inch. A dual gap head provides a check on information written by the unit. The unit can read tape forward or backward at information transfer rates of either 24,000 or 66,600 alphanumeric frames-per-second. Up to sixteen tape units can be accommodated on the system.

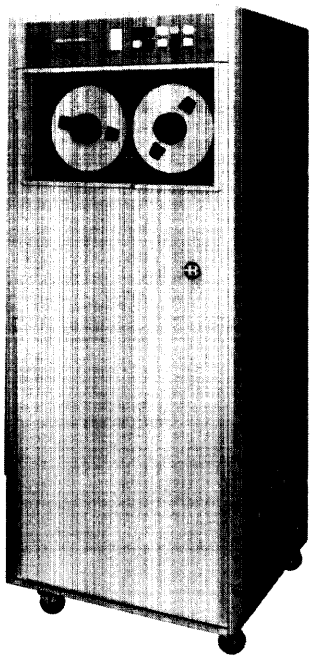


Figure 1-8. B 422 Magnetic Tape Unit

SECTION 2
ORGANIZATION

GENERAL

In order to properly understand the utility of the B 5000 System, a basic working knowledge of the fundamental organization within the B 5280 and B 5281 Processors is necessary. This Section presents a general description of the logical operation within the processors.

OPERATION

The fundamental operation of the processors is based on the flow chart pictured in Figure 2-1. There are three different conditions under which the processors must be operating at all times. These three conditions are:

1) State
2) Level
3) Mode

States

There are two states; the Normal State and the Control State. The Normal State is the predominant state of operation. However, when the Interrupt Register signals that a special condition has arisen within the system, Processor A automatically switches to the Control State and causes a branch to a specific location. While in the Control State, certain operations can be performed, which would be ignored by the processor if they were encountered in the Normal State. These operations are defined in Section 5.

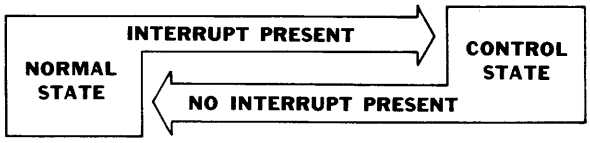


Figure 2-2. State Operation

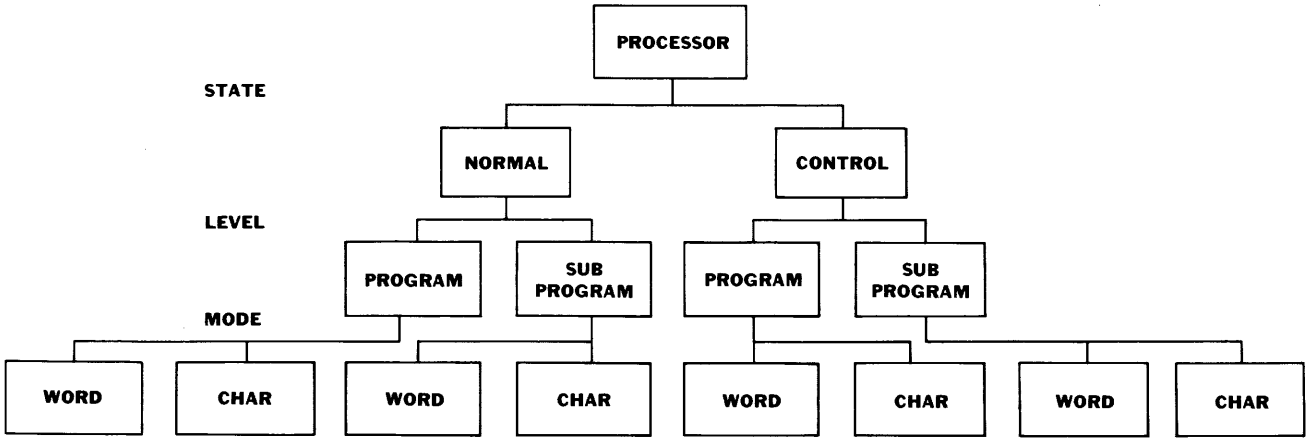


Figure 2-1. Processor Organization

A return to the Normal State is initiated when an examination of the Interrupt Register discloses that no interrupt bits are on. The processor is returned to the Normal State and an automatic branch is normally made to the program which was interrupted, unless a program of higher priority is ready for processing or the interrupted program is flagged as not ready.

Levels

Within either state, there are two levels of operation. These are the Program Level and the Sub-Program Level. The Program Level may be thought of as the level at which a main program is operating. This main program may call upon subroutines in the course of processing. Whenever this occurs, the processor switches automatically to the Sub-Program Level of operation. An automatic exit is provided in a manner specified in detail in Section 3. While in the Sub-Program Level, direct reference can be made to a wider variety of special memory areas than are available in the Program Level. For example, in the Sub-Program Level, convenient reference can be made to a range of locations in the Stack, Program Reference Table, Temporary Working Storage, and the Program Segment. This provides flexible operation for independent subroutines. In the Program Level, only the top of the Stack and the Program Reference Table are required. Both levels, of course, permit access to all other nonrestricted areas of memory through the use of Descriptors in the Program Reference Table.

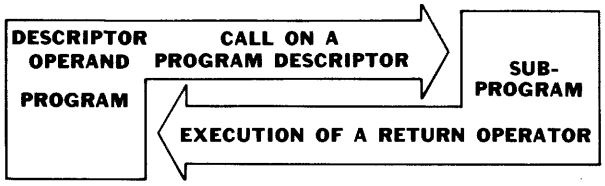


Figure 2-3. Single Level Operation

In addition, one subroutine may call for another subroutine or for itself in a recursive operation. This type of nesting can be practically infinite. To return from one subroutine to another or to the Program Level, a specific Return or Exit operator is executed. This operator automatically resets the processor to the level which existed prior to entry into the subroutine, and resets pertinent registers.

Modes

There are two basic modes of operation within the processor. These are called the Word Mode and the Character Mode.

In the Word Mode, information is normally treated as words of 48 bits in length. Arithmetic and comparison operations are performed through the use of a parallel binary adder. Operands are formated as 13 octal digit mantissas plus sign, with an exponent of two octal digits plus sign. Program syllables have the format of a two-bit identifier and a 10-bit literal operator code or relative address.

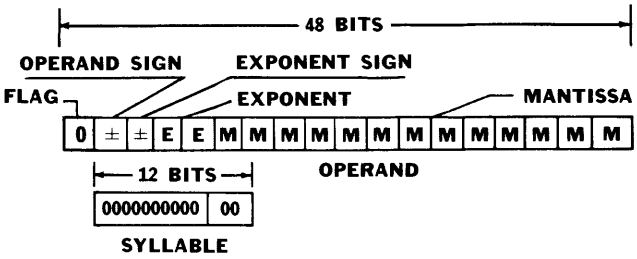


Figure 2-4. Word Mode Syllable and Operand Format

In the Character Mode, information is normally handled as six-bit alphanumeric characters. Fields can start at any character position in a word and a single operation can operate on fields of any length up to 63 characters long. Arithmetic is performed on binary-coded decimal numbers in a serial, character-at-a-time manner. As a result, memory is treated as strings of alphanumeric characters. Program syllables in this mode have a normal format of a six-bit repeat field and a six-bit operator code.

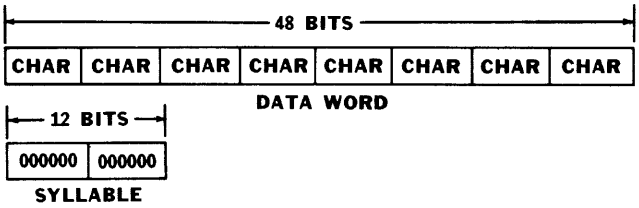


Figure 2-5. Character Mode Syllable and Data Word

Both the Word Mode and the Character Mode are available in the Sub-Program Level.

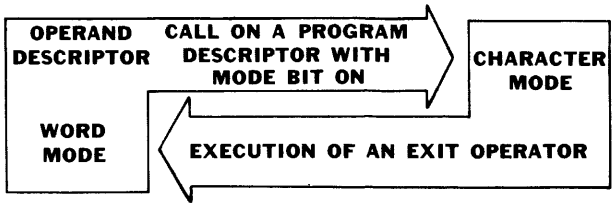


Figure 2-6. Mode Operation

Whenever a program descriptor is called for which has the mode bit set to one, the processor is automatically switched from the Word Mode to the Character Mode. The Exit Character Mode operator

is used to return to the previous Program or Sub-Program Level from which the Character Mode was entered.

PROGRAMING

Programing for the B 5000 is performed at the level of problem statements. These statements are converted into machine language through the use of powerful compilers which derive their power from the unique machine language employed by the B 5000 Processors.

This language basically involves the separation of instruction from control information. The instructions are called syllables and are contained in areas apart from control information. Control information is retained in the form of words called descriptors and stored in another area of memory called the Program Reference Table (PRT).

Syllables

An instruction is contained in a 12 bit group called a syllable. There is one set of syllables for the Word Mode and another set for the Character Mode. However, in either mode there are four syllables contained within a 48 bit word. A definition and the format of the syllables used in both modes is found in Section 5.

Word Mode Syllables

In the Word Mode, there are four types of syllables. These are:

Operators—arithmetic and logical control
Literals—program constants and indexes
Operand Calls—storage references
Descriptor Calls—storage references

Each type of syllable has specific functions which are defined in more detail in Section 5. An example of Word Mode operation is given in Figure 2-7.

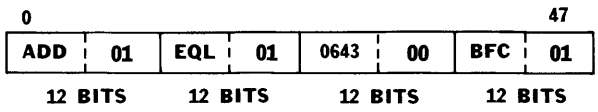


Figure 2-7. Program Word—Word Mode

The first syllable adds two 13 octal-digit fixed or floating point operands together. The second syllable compares an operand from the stack to the result of the addition. The third syllable is a literal syllable which, in this case, supplies a relative address within the program to which a branch may occur. The fourth syllable checks to see if the previous comparison was true, and if so, a branch operation takes place; otherwise, control continues in sequence.

Syllables are provided in the Word Mode for performing operation such as:

Stack Manipulation
Parallel Arithmetic
Storing
Logical Operations
Relational Operations
Branching
Operand/Descriptor Manipulation
Bit Manipulation

Character Mode Syllables

In the Character Mode, there is a single type of syllable called an operator. Its format differs from that of operators in the Word Mode. A detailed definition is found in Section 5. However, an example of Character Mode operation is given in Figure 2-8.

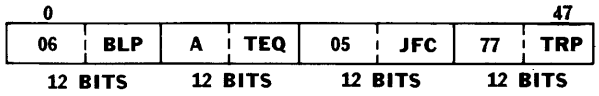


Figure 2-8. Program Word—Character Mode

The first syllable initiates a program loop which is to be repeated six times. The second syllable tests the first character of data to ascertain if it is an "A". The third syllable causes a relatively-indexed forward jump over five syllables if the previous test was true. If the test result was false, then the fourth syllable causes 63 characters to be transferred from one area in memory to another area. (Note: 77 in octal notation is equal to 63 in decimal notation.)

Syllables are provided in the Character Mode for the following operations:

Data String Addressing
Comparing
Data String Skipping
Testing
Transferring
Bit Operations
Counting
Converting
Addition and Subtraction
Branching
Looping

Descriptors

The previous examples demonstrate the function of syllables. Their purpose is to provide control over the internal functions of the processors. For providing indirect addressing and supplementary control when necessary, a single 48 bit word, called a descriptor, is used.

There are two types of descriptors: Program Descriptors and Data Descriptors. In addition to the description given here, a detailed definition of these descriptors is contained in Section 4.

Descriptors always have a "flag" bit of one. They normally contain a base address which can be indexed to locate a specific word in memory. Beyond this, they also contain supplementary control bits and addresses which are necessary to a specific operation.

Program Descriptors

The general layout of a Program Descriptor is shown in Figure 2-9.

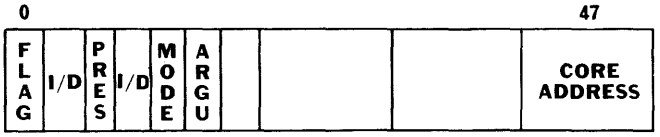


Figure 2-9. Program Descriptor

It contains an identification field, a "presence" bit for determining whether the program segment is on the magnetic drum or in core memory, and a "mode" bit to identify whether the segment is composed of Word or Character Mode syllables. The "arguement" bit is used to indicate that a segment requires parameters. A Program Descriptor specifies core memory location of the segment.

Data Descriptors

A Data Descriptor is illustrated in Figure 2-10.

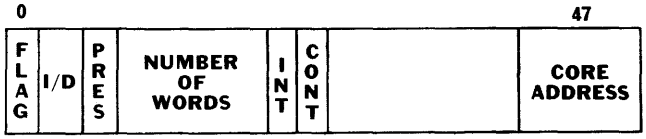


Figure 2-10. Data Descriptor

This descriptor contains an identification field, a presence bit for determining whether or not the data is in the core memory, and an integer bit used in conjunction with storing operations. The number of words or size of the field addressable by the descriptor and core memory location is also specified.

A by-product of specifying the number of words in the area addressed by a Data Descriptor, is the ability to prevent a program from accidentally storing information beyond the specified area and interfering with other areas or programs. If the final indexed address of a Data Descriptor exceeds the specified area, an automatic interrupt occurs before the operation is executed to notify the system of this condition. This feature and others combine to make Data Descriptors a highly efficient means of controlling working storage areas.

Data Descriptors are also used to initiate input/output operations. When used for this purpose, the Data Descriptor also contains the input/output unit number and control bits for controlling the operation.

Syllable-Descriptor Operation

In the course of operating a program, frequent reference is made to new information, working storage, output areas, and other program segments or subroutines. In order to reference this information, syllables utilize the descriptors. A feature of this concept is that several syllables may reference a single descriptor to obtain different words from one area.

Descriptors for a single program are stored in consecutive words relative to an address contained in the R register which is described later in this section. This series of consecutive words is called the Program Reference Table. A specific descriptor is obtained by referencing the PRT through the means of the R register and an Operand or Descriptor Call syllable. See Figure 2-11. The R register contains the 3 high order octal digits of a memory address. For purposes of exposition the example is shown with decimal digit representations.

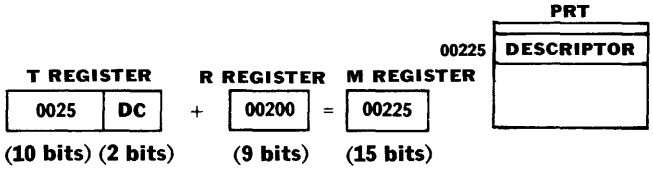


Figure 2-11. Referencing a Descriptor

The final result of referencing the PRT depends on the specific syllable and descriptor involved.

A table of initial conditions and ultimate results is shown in Figure 2-12. This table is intended as a summary of interactions between different syllables and descriptors. Section 3 contains a more detailed explanation.

SYLLABLE	PROGRAM REFERENCE TABLE	RESULT
Operand Call	Program Descriptor	Enter Program Seg. to Obtain Operand
Operand Call	Data Descriptor	Operand
Operand Call	Operand	PRT Operand
Descriptor Call	Program Descriptor	Enter Program Seg. to Obtain Descriptor
Descriptor Call	Data Descriptor	Data Descriptor
Descriptor Call	Operand	Descriptor for the Operand

Figure 2-12. Syllable—Descriptor Table

Operands

The processors can operate either with fixed length words or with variable length fields. The former is used in the Word Mode while the latter is used in the Character Mode. Since both modes are available in each processor, the system can operate in the mode most desirable for the operation at hand.

When a processor is operating in the Word Mode, the standard format for data words is illustrated in Figure 2-13.

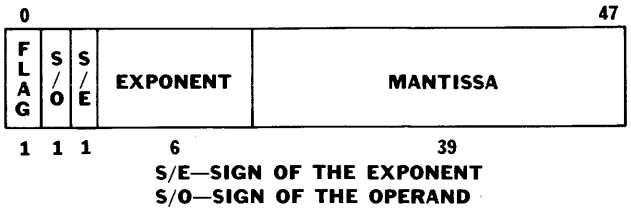


Figure 2-13. Data Word—Word Mode

Note that the standard word is an octal floating point word. However, the mantissa is treated as an integer with the decimal point to the right rather than as a fraction. This provides two features. First, an integer has the same internal representation as its un-normalized floating point correspondent. Second, the range of the numbers that can be expressed is from 8^{+76} to 8^{-63} rather than being 8^{+64} to 8^{-63}. The first feature eliminates the need for fixed-to-floating point conversion and a separate set of instructions. The second feature expands the range where difficulty with range is most often encountered, namely, in numbers of extremely large magnitude. Thus, this data word provides as much resolution for floating point arithmetic as many methods offer in fixed point arithmetic.

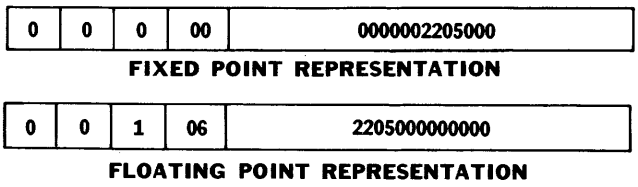


Figure 2-14. Fixed and Floating Point Representation

The "flag" serves a dual purpose. The function of the "flag" depends on how the program references the data word. If the data word is a single variable or an element of an array, a "flag" bit of zero identifies the word as being an operand. If the word is an element of an array, a "flag" bit of one may be used to identify this particular element as an array boundary which is not to be processed by the normal program. This latter case causes an interrupt which may be used to notify the program that a boundary point has been reached in an array.

When operating in the Character Mode, each data word consists of eight 6-bit alphanumeric characters as illustrated in Figure 2-15.

Programs in the Character Mode can address any character or any bit in a word. Fields may start at any position in a word and single operations may process fields up to 63 characters in length. A more detailed explanation of the Character Mode is found in Section 3.

Control Words

Control words are automatically created by the processor when certain operators are executed, or when an interrupt occurs. They contain the contents of various registers and the settings of control flip-flops, and are used for restoring the registers and flip-flops when required.

1. Return Control Word
   It is placed in the stack at the time of subroutine entry and contains the contents of the C, F, K, G, V, L, and H registers and the setting of the Descriptor/Operand Call Indicator. It provides the information required for restoration of registers when leaving a subroutine and the location of the associated mark stack control word.

2. Mark Stack Control Word
   It is placed in the stack as a result of executing a Mark Stack operator or entering a subroutine which does not require arguments and contains the contents of the F and R registers and the settings of the Mark Stack and Program Level flip-flops.

3. Loop Control Word (1)
   It is used in conjunction with Character Mode syllables and contains the repeat field of its associated Begin Loop operator, the address of the next syllable following the Begin Loop operator and F register setting.

   Loop Control Word (2)
   It is formed when an interrupt occurs when the

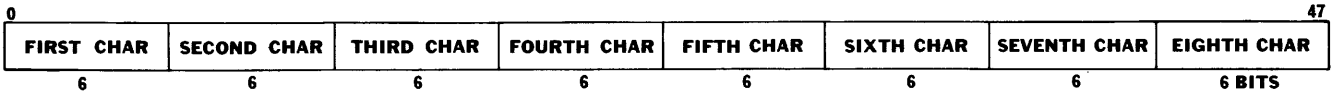


Figure 2-15. Data Word—Character Mode

processor is in the Character Mode. The organization of Loop Control Word (2) is identical with that of Loop Control Word (1), except that the field containing the contents of the F register will contain the contents of the S register.

4. Interrupt Control Word
   It is placed in the stack when a processor is interrupted or when Processor B is idled as a result of a Halt Processor B operator and contains the contents of the M and R registers and the setting of the Mark Stack flip-flop.

   It is formed when an interrupt occurs and contains the contents of the C, F, K, G, V, L, and H registers and an indicator specifying whether the B register was full or empty at the time the interrupt occurred.

6. Initiate Control Word
   It is used to identify the top of the stack when performing the Initiate operator and contains the contents of the S register and the setting of the mode bit.

REGISTERS

The B 5280 and B 5281 Processors *each* contain a complete set of 15 control registers. They are generally grouped in four classifications:

Program Registers
Primary Stack and Source Registers
Secondary Stack and Destination Registers
Utility Registers

Program Registers

The program registers are used to control the direction of program segments which are in operation. These registers are called the P register, C register, L register, and T register. The P register is a 48-bit register used to hold the current word from the program segment being executed. As such, it can access and control four syllables at a time. See Figure 2-16.

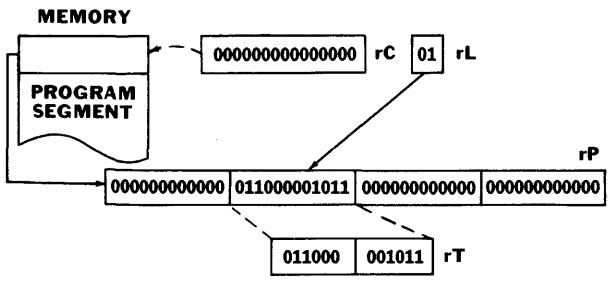


Figure 2-16. Program Registers

The C register is a 15-bit register used to specify the memory location where the program word in the P register was accessed. The syllable which is being executed at any one time is contained in the 12-bit T register, and its position within the P register is indicated by the L register. The latter is a two-bit register which counts up as each syllable is executed. When this register overflows, it carries into the low order position of the C register to provide the address of the location from which the next program word will be fetched.

Primary Stack and Source Registers

To implement the Word and Character Modes of operation, two concepts called the stack and source string are implemented by the logical operation of the B 5000. These concepts are discussed in more detail in Section 3. A group of registers called the primary stack and source registers are used to facilitate the stack and the source string operations.

In the Word Mode, these registers are used to implement the operation of the stack by controlling the information in the A register when the stack is fully "pushed up." In the Character Mode, they are used to provide control over the operation of the source string word contained in the register.

The information register in this grouping is called the A register. It is 48 bits in length and associated with it is a one-bit flip-flop which indicates the presence or absence of information. The A register is normally used to hold an operand prior to Word Mode arithmetic operations, although it may contain a descriptor during certain data transfer operations. In the Character Mode, one data word from the source string is contained in the A register.

Three address registers are used to control information moving between the A register and memory. These are the M register, G register, and H register as shown in Figure 2-17.

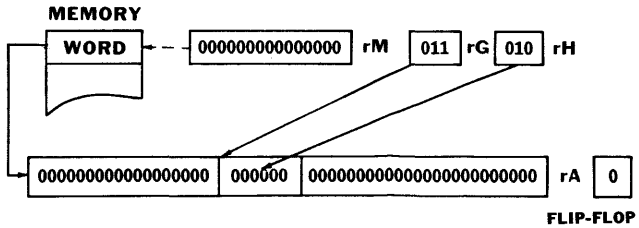


Figure 2-17. Primary Stack and Source Registers

The M register is 15-bits in length and specifies the location of the word in memory associated with the transfer of data to and from the A register both in the Word Mode and Character Mode.

The G register is used to locate an individual character or group of six-bits within the A register. The G register is three bits in length. During character operations, it automatically overflows to count up the M register.

The H register is used to locate specific bits within the character position addressed by the G register. This register is also three bits in length; but it recycles after six counts, since there are only six bits in a character position.

Secondary Stack and Destination Registers

This grouping of registers is similar in function to that of the primary stack and source registers. However, the functions differ in some respects. In the Word Mode, these registers are used again to implement the operation of the stack; but in this case, they control the next to the top word when the stack is completely "pushed up." In the Character Mode, the registers are used to provide control over the operation of the destination string.

The information register in this grouping is called the B register. This register is 48 bits in length, and associated with it is a one-bit flip-flop which indicates the presence or absence of information. It is normally used to hold an operand prior to Word Mode arithmetic operations, and the results of these operations are generally located here. In the Character Mode, one word from the destination string is contained in the B register.

Three registers are used to control information moving between the B register and memory. These registers are the S register, K register, and V register as illustrated in Figure 2-18.

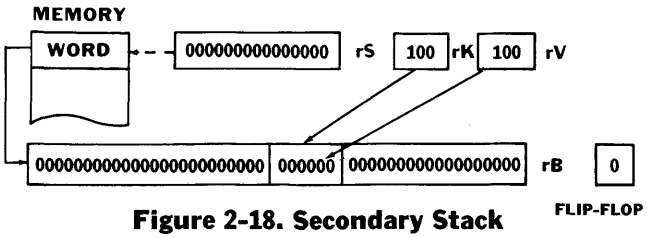


Figure 2-18. Secondary Stack and Destination Registers

The S register is 15-bits in length and specifies the location in memory of the word associated with the transfer of data to and from the B register, both in the Word Mode and Character Mode.

The K register is used to locate an individual character or group of six bits within the B register. This register is three bits in length. While character operations are proceeding, it automatically overflows to count up the S register.

The V register is used to locate specific bits within the character position addressed by the K register. This register is three bits in length; but it recycles after six counts, since there are only six bits in a character position.

Utility Registers

Certain utility registers are also provided within each processor to complement fully the previously described array of registers. These registers have specific purposes, but they do not come under any of the foregoing headings.

The F register is a 15-bit register used to hold an address when program control is transferred from one program level to another. Normally, the address in the F register is the address that was contained in the S register when control information was transferred to the stack, as in Figure 2-19. This control information contains, among other control information, the previous setting of the F register so that sub-program levels may be indefinitely nested.

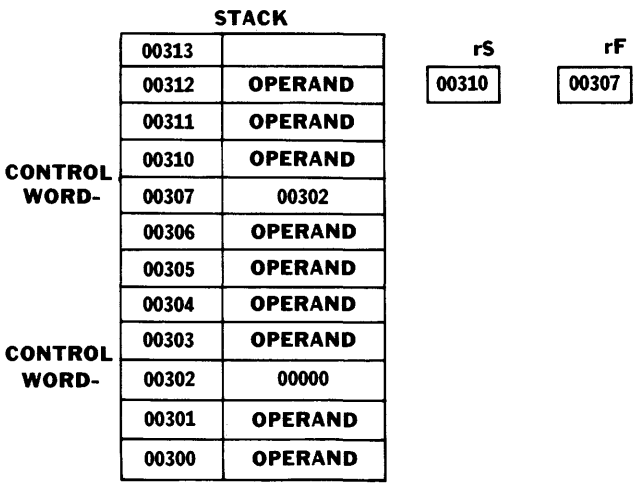


Figure 2-19. Subroutine Nesting Control

Figure 2-19 shows the condition of the stack after transfer of control to sub-program levels. The F register always contains the location where the control information is stored. This greatly facilitates automatic exits from subroutines.

The Program Reference Table was discussed in an earlier section. The base location for the Program Reference Table of any one program is maintained in the R register while in the Word Mode. This is a nine-bit register which contains the high-order bits of a 15-bit address in memory. The low-order bits of the address, for a specific descriptor in the Program Reference Table, are supplied by the particular Operand Call or Descriptor Call syllable referencing

that descriptor. Note that there is an overlap since the call syllables have ten-bit address fields. The examples in this manual, however, show it as having 15-bits for making incrementation more readily understood.

In the Character Mode, the R register is used as a counting device and can be manipulated by a program.

The X register is used as an extension of either the A or the B registers during certain arithmetic operations in the Word Mode. This register is 39 bits in length so as to accommodate an extension of the mantissa of an operand. For certain internal control operations, it is used as a holding register for the G, H, K, and V registers.

INTERRUPT SYSTEM

A high performance computer system requires an extensive interrupt system in order to provide optimum operation. An interrupt system furnishes a means for continuous automatic recognition of exception conditions which, otherwise, would have to be checked programatically at intervals.

For the comprehensive interrupt system within the B 5000, there are two types of interrupts. These are the processor independent type, and the processor dependent type.

Independent Interrupts

Processor independent interrupts are those which are not initiated or generated by any program code, but are received from an external source. These interrupts are:

Time Interval
Processor B Busy
Input/Output Channel Busy
Keyboard Request
Input/Output Channel Finished

Any syllable in process will always be completed when one of these interrupts occurs. That is, the actual interrupt will occur after execution of a syllable is completed, but it may occur between syllables of a program word.

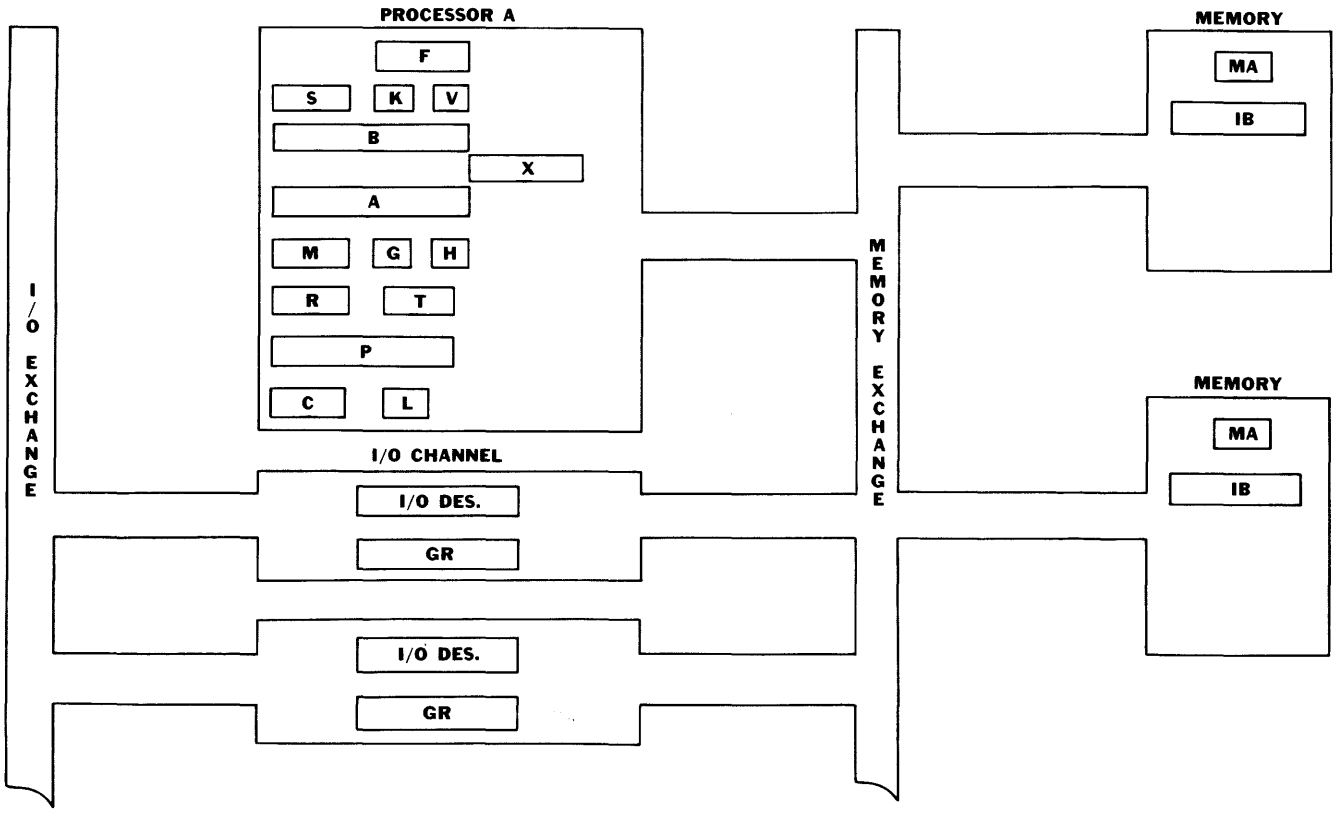


Figure 2-20. Register Configuration

Dependent Interrupts

Processor dependent interrupts are initiated or generated by a program code operating within a processor. These interrupts are:

Memory Parity Error.
Invalid Address
Communication Operator
Flag Bit
Continuity Bit
Program Release
Stack Overflow
Presence Bit
Invalid Index
Exponent Underflow
Exponent Overflow
Integer Overflow
Divide By Zero

In this type of interrupt, the syllable in process is immediately terminated as soon as the condition is detected with the exception of Memory Parity Error and Stack Overflow.

Interrupt Register

The Interrupt Register performs the function of co-ordinating the recognition of exceptional conditions within the system. Upon recognition of such a condition, an automatic transfer of control is made to a specific memory location. A partial list of the interrupts contained in the Interrupt Register is shown in Figure 2-21. Scanning precedence is also shown in this figure.

A brief explanation of several types of interrupt is presented below.

Time Interval— an internal clock turns this bit on every second for checking program running time.

Memory Parity— indicates a parity error in a word read from memory.

Processor B Busy—used to determine the presence of or to indicate a malfunction of Processor B.

I/O Channel Busy— used to determine the system configuration available or to indicate a malfunction of an I/O Channel.

Invalid Address—used to determine the presence of or to indicate a malfunction of a memory module or program error, and to protect MCP memory.

MEMORY PARITY ERROR—PROCESSOR A
INVALID ADDRESS—PROCESSOR A
TIME INTERVAL
I/O BUSY
KEYBOARD REQUEST
PRINTER 1 FINISHED
PRINTER 2 FINISHED
I/O CHANNEL 1 FINISHED
I/O CHANNEL 2 FINISHED
I/O CHANNEL 3 FINISHED
I/O CHANNEL 4 FINISHED
PROCESSOR B BUSY
INQUIRY REQUEST
SPECIAL INTERRUPT 1
SPECIAL INTERRUPT 2
SPECIAL INTERRUPT 3
MEMORY PARITY ERROR—PROCESSOR B
INVALID ADDRESS—PROCESSOR B
STACK OVERFLOW—PROCESSOR B
COMMUNICATION OPERATOR—PROCESSOR B
PROGRAM RELEASE OPERATOR—PROCESSOR B
CONTINUITY BIT—PROCESSOR B
PRESENCE BIT (I/O STATUS BIT)—PROCESSOR B
FLAG BIT—PROCESSOR B
INVALID INDEX—PROCESSOR B
EXPONENT UNDERFLOW—PROCESSOR B
EXPONENT OVERFLOW—PROCESSOR B
INTEGER OVERFLOW—PROCESSOR B
DIVIDE BY ZERO—PROCESSOR B
STACK OVERFLOW—PROCESSOR A
COMMUNICATION OPERATOR—PROCESSOR A
PROGRAM RELEASE—PROCESSOR A
CONTINUITY BIT—PROCESSOR A
PRESENCE BIT (I/O STATUS BIT)—PROCESSOR A
FLAG BIT—PROCESSOR A
INVALID INDEX—PROCESSOR A
EXPONENT UNDERFLOW—PROCESSOR A
EXPONENT OVERFLOW—PROCESSOR A
INTEGER OVERFLOW—PROCESSOR A
DIVIDE BY ZERO—PROCESSOR A

Figure 2-21. Interrupt Conditions

Communication Operator—Communicate information to MCP.

Flag Bit—used to indicate the end of a data array.

Keyboard Request—indicates that the system operator has a request to enter via the keyboard.

Continuity Bit—indicates multiple input/output areas with linked descriptors.

Invalid Index—indicates that a program index value exceeds the size of a descriptor area.

Exponent Underflow—indicates that an arithmetic operation has resulted in an exponent value less than—63

(operand less than 8^{-51}).

Exponent Overflow— indicates that an arithmetic operation has resulted in an exponent value greater than +63 (operand greater than or equal to 8^{+76}.

Integer Overflow— indicates that an operand exceeds 8^{13} when a floating point number is being converted to an integer.

Divide by Zero— indicates that the divisor is zero when a divide operation is executed.

I/O Channel Finished—indicates that an External Result descriptor has been returned to memory.

Program Release— indicates an input/output area is freed to receive or transfer information.

Stack Overflow— indicates that the S register is equal to the R register and the stack is about to exceed its area.

Presence Bit—indicates that a program has referred either to information that is not present in memory or to input/output information that is not available.

Interrupt Detection

When operating in the Normal State and an interrupt occurs, all necessary registers and flip-flops are stored in the stack to allow the program to be continued after the interrupt has been processed. Following the interrupt, Processor A is placed in the Control State and the address of the cell assigned to the interrupt is transferred to the C register.

All interrupts are processed on a priority basis. All possible interrupts are sampled continuously and simultaneously. There is no queuing of interrupts.

Processor B cannot be placed in the Control State. When an interrupt occurs that is associated with Processor B, the processor stores its registers, forms and stores the appropriate control words and then idles. When Processor A is operating in the Control State, all interrupts remain set until an Interrogate Interrupt operator is executed.

WORD MODE INTERRUPT

The presence of an interrupt results in the following action: If the A and/or B registers are full, they are pushed into the stack. An Interrupt Control Word followed by a Return Control Word (2) is placed into the stack.

The resulting stack in memory is shown in Figure 2-22.

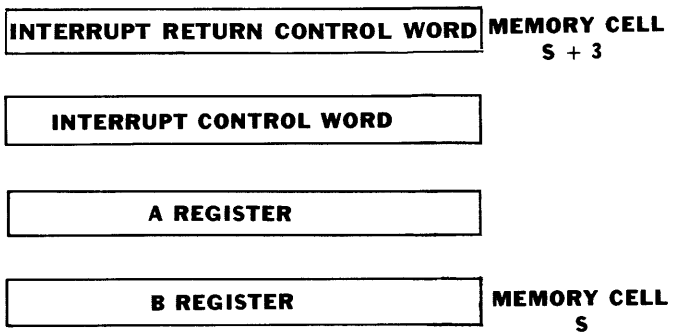


Figure 2-22. Resulting Stack In Memory—Interrupt Control

An Initiate Control Word is stored in memory location R + 8.

CHARACTER MODE INTERRUPT

All Character Mode operators, with the exception of the Call Repeat Field operator, allow interrupt only at the completion of the operator. In the case of the Call Repeat Field operator, the operator following the Call Repeat Field operator is executed and completed before interrupt is allowed.

The presence of an interrupt results in the following action: If the B register is occupied it is placed in the stack followed by a Loop Control Word (2), Interrupt Control Word and a Return Control Word (2). The resulting stack in memory is shown in Figure 2-23.

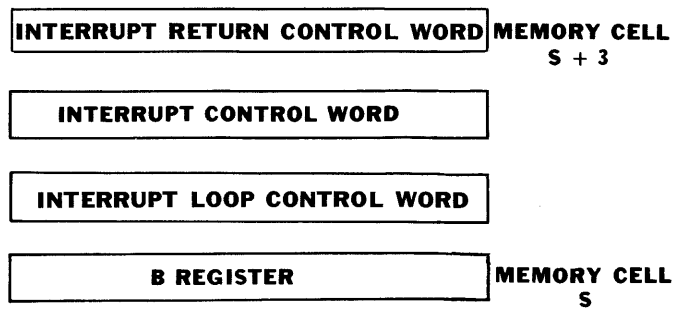


Figure 2-23. Resulting Stack in Memory—Loop Control

An Initiate Control Word is stored in memory location R + 8.

RETURN TO NORMAL

When an Initiate Processor P1 operator is executed, the 15 low-order bits of the A register are placed in the S register and the Mode flip-flop is set. The Control Words are then automatically taken from the stack and the registers restored. Control is returned to the next syllable in sequence in the interrupted program.

PARALLEL AND SERIAL ARITHMETIC

The B 5000 Processors each contain both a parallel word adder and a serial character adder. The parallel adder operates on an octal number base while the serial character adder operates on a binary-coded decimal number base. The system also has the ability to automatically convert from one number base to another by hardware means.

Parallel

The Word Mode uses the octal number system, and information is handled one word at a time. A parallel binary adder is used for performing all arithmetic operations in this mode. Fixed and floating point information can be intermixed and operated on by all arithmetic commands, see Figure 2-24.

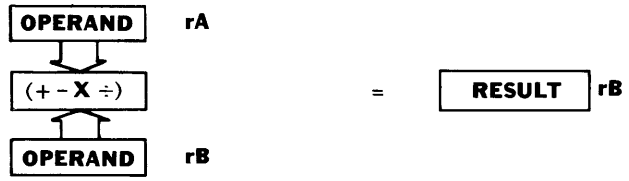


Figure 2-24. Parallel Binary Add Operation

Serial

The Character Mode operates on decimal information when performing arithmetic operations. Each decimal or alphanumeric character is handled individually. As a result, the arithmetic operations of add and subtract operate on successive pairs of characters in a serial fashion. For this reason, a serial decimal adder is provided for this mode. See Figure 2-25 for an example of its operation.

With two processors, one processor may be computing with its parallel adder while the other processor performs its computations with a serial adder, or both may be computing with parallel adders or both with serial adders. The inclusion of two distinct types of adders provides a high degree of flexibility in handling a wide range of arithmetic operations.

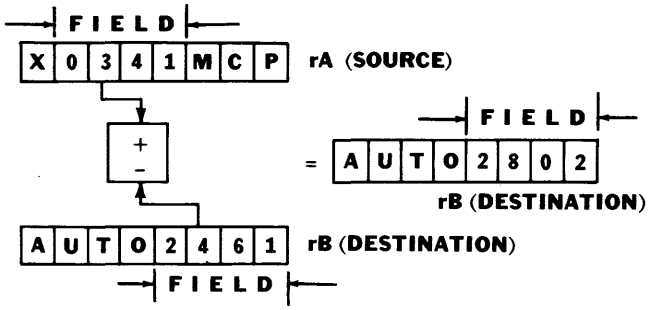


Figure 2-25. Serial Decimal Adder

COMMUNICATION

The processors on the B 5000 can communicate directly with any memory module through a special Memory Exchange. This exchange also permits simultaneous parallel access into multiple memory modules.

The Memory Exchange is the focus of data flow within the system. It provides automatic parallel routing and control of communications and information. This exchange resolves communication conflicts by scheduling, rather than by merely buffering. It accomplishes its function with a comparatively small amount of circuitry and without delay, either for the no-conflict case or for the priority case of conflicting operations. If two or more units simultaneously address the same memory module, the exchange automatically resolves the conflict, using a priority technique, and queues the lower priority request. One unit gains immediate access while the next unit is delayed only until completion of the first memory transfer. Resolution of multiple conflicts is performed in parallel with no lost time to any memory modules involved in the conflicts. The priority system is pre-emptive in that a new request with a high priority will precede a low priority request already in the queue.

To store or access data in a memory module, the processor sets the desired address in the Memory Address register for the particular module. If the operation is a store, the processor transfers the data from one of its registers to the Information Buffer register in the module. The processor is then released while the core storage operation takes place. If the operation is an access, the information is transferred from core storage to the Information Buffer. The processor then transfers the data from the Information Buffer to the required register.

In order to transfer data between memory and peripheral input/output units, an input/output descriptor (Data Descriptor) is transmitted to an

I/O Control Unit. An Initiate I/O operator causes an I/O data descriptor address to be transmitted to one of the available I/O units. The I/O unit then continues independently of the Processor by fetching the input/output descriptor from the specified address to the I/O control register. Upon termination of an I/O operation, the original input/output descriptor is modified and sent to a specific memory address as a result descriptor.

The registers in the processors that communicate with the Memory Address registers are the S, M, and C registers.

Information is transferred between the Information Buffer registers in individual modules and the A, B, and P registers of the processors.

DUAL PROCESSORS

The B 5280 Processor is a high speed computation and control unit. The capabilities of this unit are more than sufficient for most applications. However, in certain cases a higher computational workload will require more computer facilities. To provide this, a second processor, the B 5281, can be added to the system.

Dual processors provide completely independent parallel control and computational abilities. The B 5281 Processor has its own logic and control as well as registers which allow it to control a separate set of programs. In addition, this processor has an independent pair of adders to permit parallel computation.

The efficiency of parallel dual processors is dependent on the availability of multiple memory modules. The multiple memory modules permit each processor to utilize separate modules of memory and eliminate time-sharing of storage facilities. When there is a large quantity of input or output, multiple I/O Channels will allow the processors optimum access to the peripheral units.

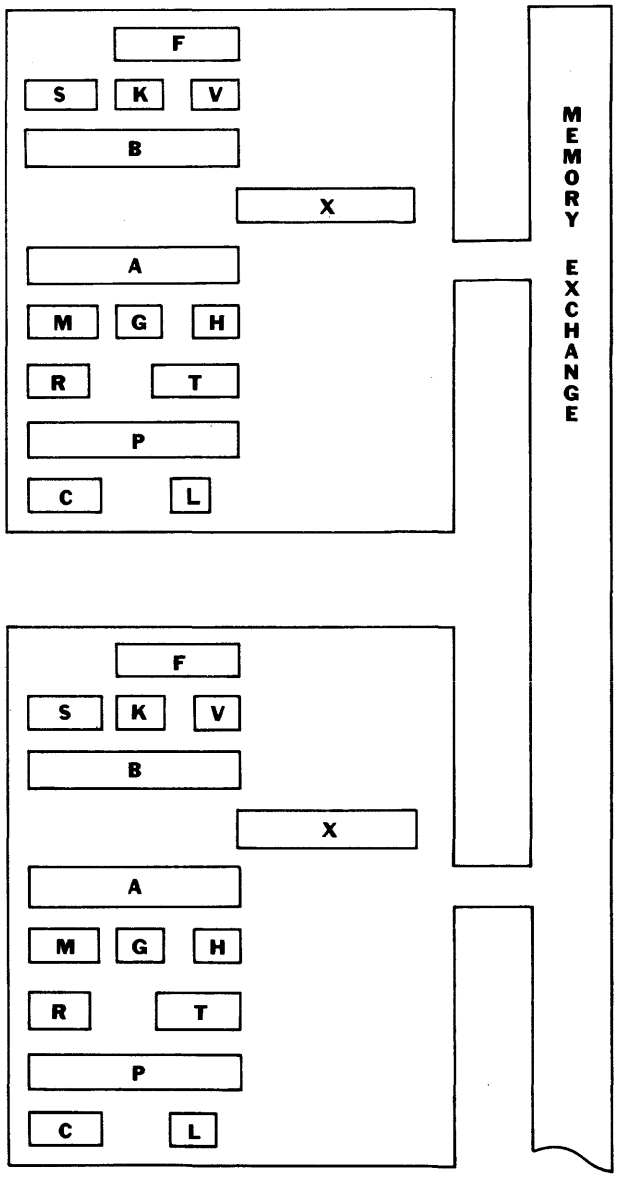


Figure 2-26. Dual Processor Registers

SECTION 3
OPERATION

GENERAL

The design of the processors for the B 5000 System is directed toward the implementation of several new concepts. One important purpose of the implementation of these concepts is to permit highly efficient machine code programs to be generated automatically by advanced compilers. These compilers derive their power, to a large extent, from the improved internal logical organization of the processors. This is accomplished through the translation of the problem-oriented language statements into a machine language which is operationally efficient. The instructions and operands are combined in a manner compatible with the source language expressions, and at the same time, they are in a form which is immediately useful for computer operation.

STORAGE

The allocation of internal storage areas is performed in a manner which provides a high degree of standardization and control. Storage areas within the internal memory of the B 5000 are divided into five general types. These are stacks, program reference tables, program segments, data storage, and input/output areas as illustrated in Figure 3-1. The syllables used for communication between the areas are also shown, as well as the resultant information.

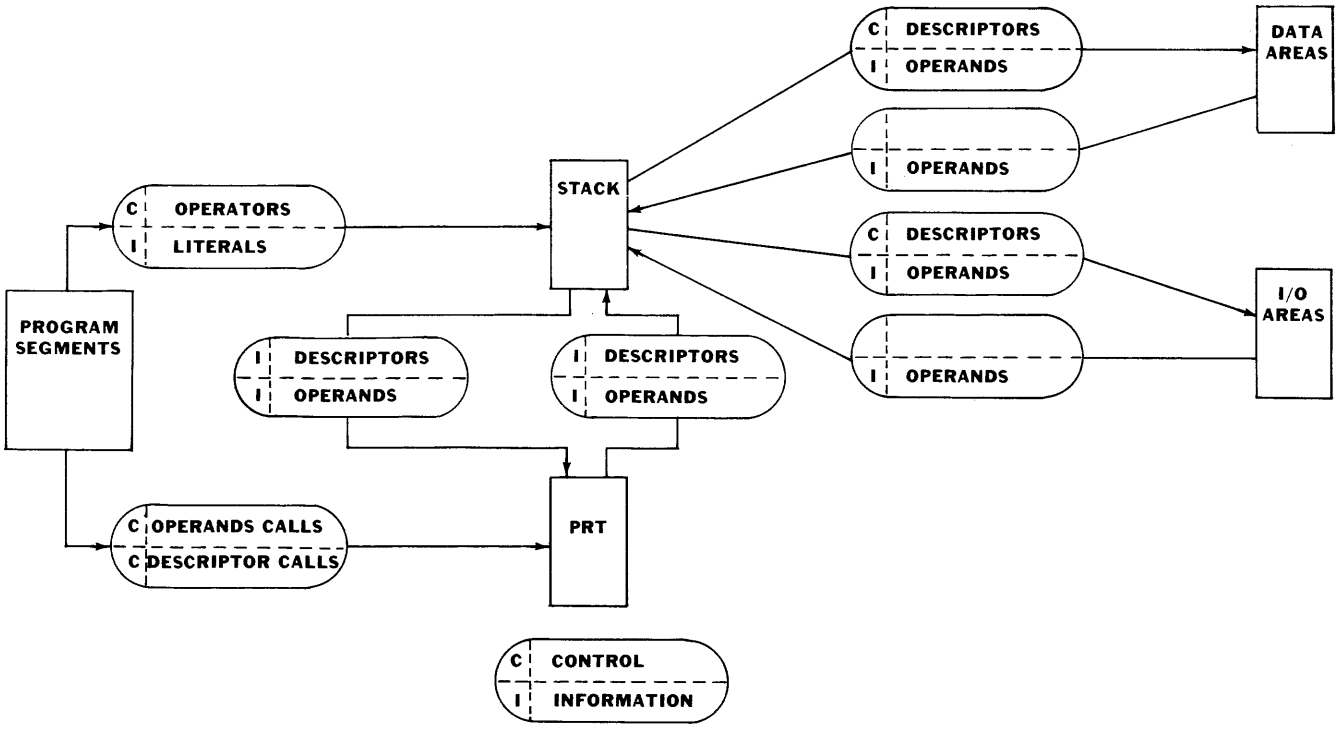


Figure 3-1. Storage Co-ordination

Stacks

Stacks are an efficient form of automatic temporary storage. A stack is essentially a list of ordered items of information. In the B 5000, it is a list of operands and control information stored sequentially in the order of processing. The physical stack is composed of the A and B registers and the memory area addressed by the S register.

A new word coming into the stack pushes down the information previously held in the registers. The information contained in the registers is the last information entered into the stack; the stack operates on a last in, first out principle.

As operands are fetched by a program, they are placed in the A register. If the A register already contained a word, that word is transferred to the B register prior to loading the operand into the A register. If the B register is also occupied by information, then the S register is automatically increased by one, and the word in the B register is stored in a cell addressed by the S register. Then the word in the A register can be transferred to the B register and the operand brought into the A register, see Figure 3-2.
As information is operated on in the stack, operands are eliminated from the stack and results of operations are returned to the stack. The need for information contained in the stack may require an automatic "push-up" to occur. This operation causes a word to be brought to the A or B register from the memory area addressed by the S register. The S register is then counted down by one.

The flip-flops associated with the A and B registers are used to eliminate unnecessary stack operations. When an operand is to be placed in the stack, and either of the registers is empty, no push-down into memory occurs. No push-up is executed either, when an operation leaves one or both of the registers empty.

Note that the use of the stack, combined with the internal logic of the processor, eliminates the need for programing the storage or recall of intermediate results.

In the case of multiprocessing, each program has its own stack. When an interrupt occurs, all required registers and control flip-flops are automatically pushed into the appropriate stack and the last S register setting is stored in a fixed location. To return to a program, this location is programatically fetched by the operating system and the S register is reset from the contents of the word. The other registers are then automatically reset and control continues in sequence.

Program Reference Table

Programs for the B 5000 System are independent of machine locations. This is achieved by the use of a Program Reference Table. A separate reference table is used for each program.

The PRT is a relocatable area in memory that can be up to 1024 words in length. The R register contains the address of the base location of the reference table for the program being executed. The PRT is used primarily for storing words that locate data areas, program segments, or describe input-output operations. These words are called descriptors and are discussed in more detail in Section 4. They contain the base address and size of data areas, program segments and input-output areas as well as other control information.

Operands may also be stored in the PRT, providing direct access to single values such as indexes, counts and other control information.

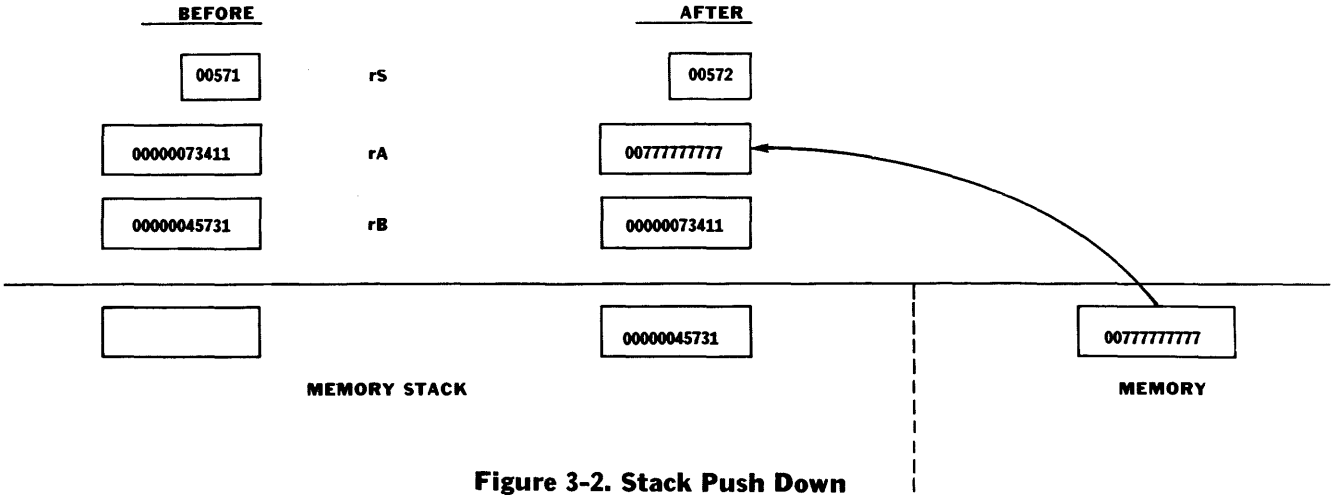


Figure 3-2. Stack Push Down

As a result of keeping all base addresses in the PRT, the program itself does not contain any actual addresses, but only references to the PRT. To specify one of the possible 1024 positions in the PRT requires only 10 bits. This is an important factor in providing a high program density in the B 5000.

Since the PRT is relocatable, program references to it are to locations which are relative to the R register. The program is, therefore, completely freed from any dependence on actual memory location, see Figure 3-3.

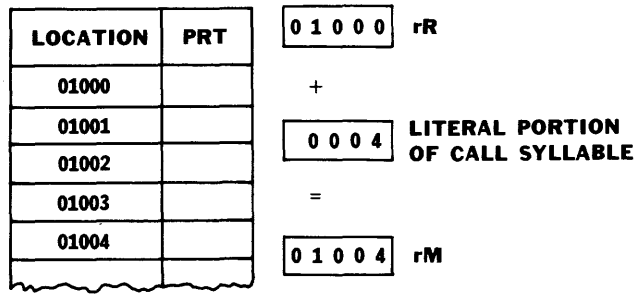


Figure 3-3. Indexing the PRT

Program Segments

Program segments are logical portions of a program. There is a program descriptor for each segment. One of the features of the B 5000 is that a program is independent of the actual memory locations for both itself and the data it is processing. Through automatic program segmentation, the program size is independent of the size of the core memory.

Program segments are composed of strings of syllables. Each program word contains four syllables and they are executed sequentially in a left-to-right order within the program word. Each word is executed sequentially in an ascending manner. Branching is allowed to any syllable. Branching within a program segment is self-relative since the distance to jump either forward or backward is specified, rather than an actual address.

Program segments are linked together by the Communication Operator, which causes an interrupt to permit entry to the next segment.
Entry is made to a subroutine via its program descriptor in the PRT. The program descriptor contains a core address, drum address and an indication if the subroutine is currently in core memory.

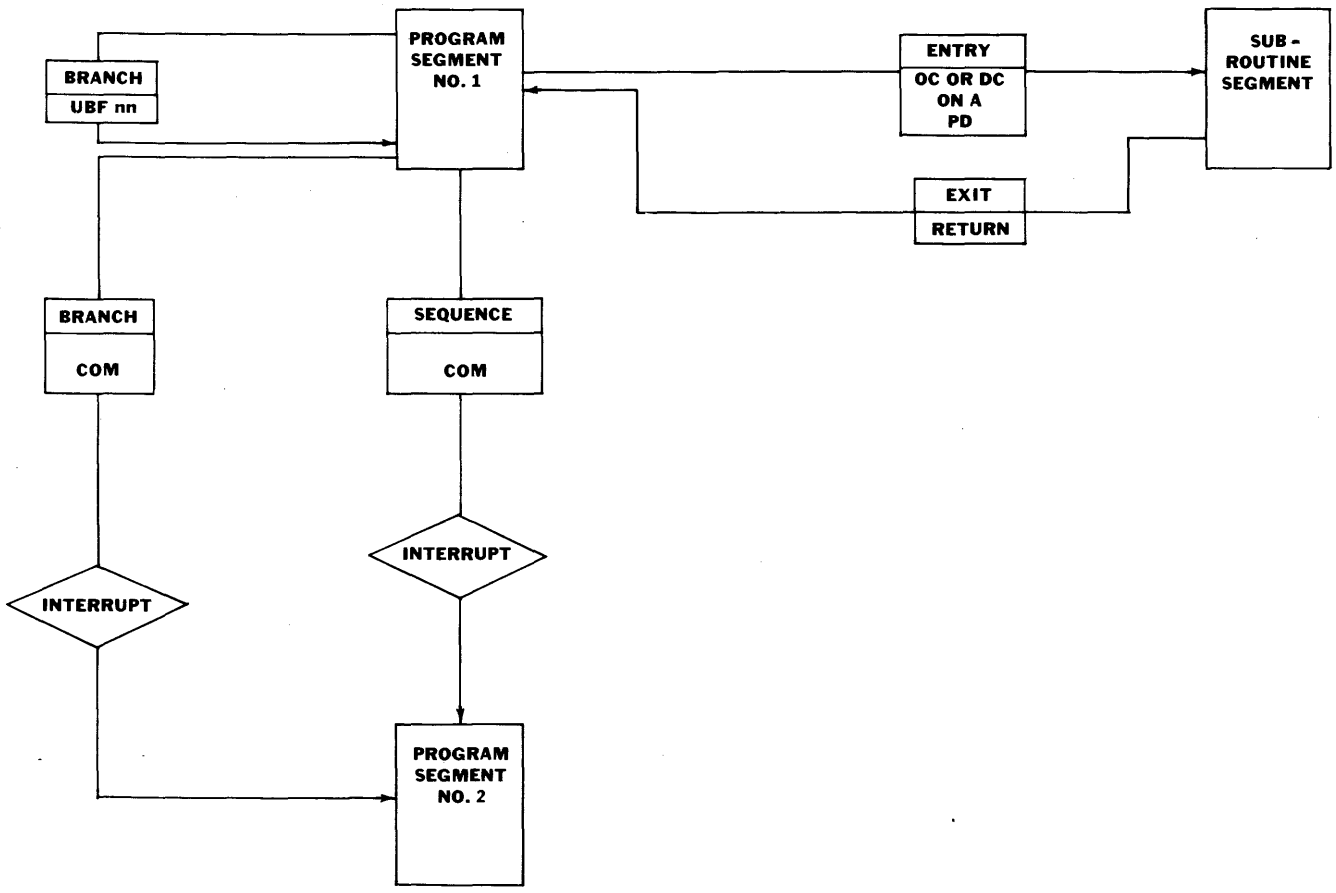


Figure 3-4. Program Segment Operation

If the segment is not in core memory when it is called for, an interrupt occurs to provide notification of this fact. These cases are illustrated in Figure 3-4.

Data Storage

For the storage of data arrays such as tables, working areas, and other information of this type, areas in memory can be allocated and referenced by Data Descriptors. These descriptors contain array size, core location of the first element and, if required, a drum location, as well as an indication of whether the information is in core memory or on the drum.

Any element within an array can be accessed by incrementing the core address of the first element of the array by an index value which is not greater than the size field in the descriptor. This method provides the features of completely generalized indexing and complete storage protection, both within and outside a program. Another result is that the PRT can be considered as a series of index registers up to a total of 1024 for any single program.

Input/Output Areas

Input/Output areas are sections of memory that contain information read from or information to be written to a peripheral input/output unit.

The input/output Data Descriptors contain the beginning core address, size field, unit number, and special control information for the specific unit when necessary. These descriptors can be used to either reference data within an input/output area or to execute the input/output operation when it is called for.

SUBROUTINES

Subroutines in the B 5000 System are normally handled by entering the Sub-Program Level which was discussed in a prior section. The subroutine control provided in the processors allows for nesting subroutines to an indefinite level. It also allows complete freedom for using recursive procedures. Dynamic allocation of storage for parameter lists and temporary working storage simplify the use of subroutines. Storage is automatically allocated and released as required.

To enter a subroutine, control in the processor which is performing the operation is set to the Sub-Program Level. This has the following effects on the program being executed:

Operand and Descriptor Call syllables are formated in a slightly different manner. This is explained in Section 5.

The call syllables are allowed to directly reference limited areas in the stack and subroutine segment, as well as in the PRT.

Entry to the subroutine, exit from the subroutine, and housekeeping for temporary storage areas and registers is automatically provided.

In the Sub-Program Level, the F register plays a vital part. It is used in conjunction with the S register and Mark Stack Flip-Flop (MSFF) to provide efficient control of subroutine entries, nesting, parameter, and temporary storage separation and exits.

Mark Stack Flip Flop

The Mark Stack Flip-Flop is controlled by a special operator called Mark Stack. This operator is used to do as its name implies, that is, mark the stack. The purpose of marking the stack is to provide a defined area for the storage of parameters before actually entering a subroutine. The action of a

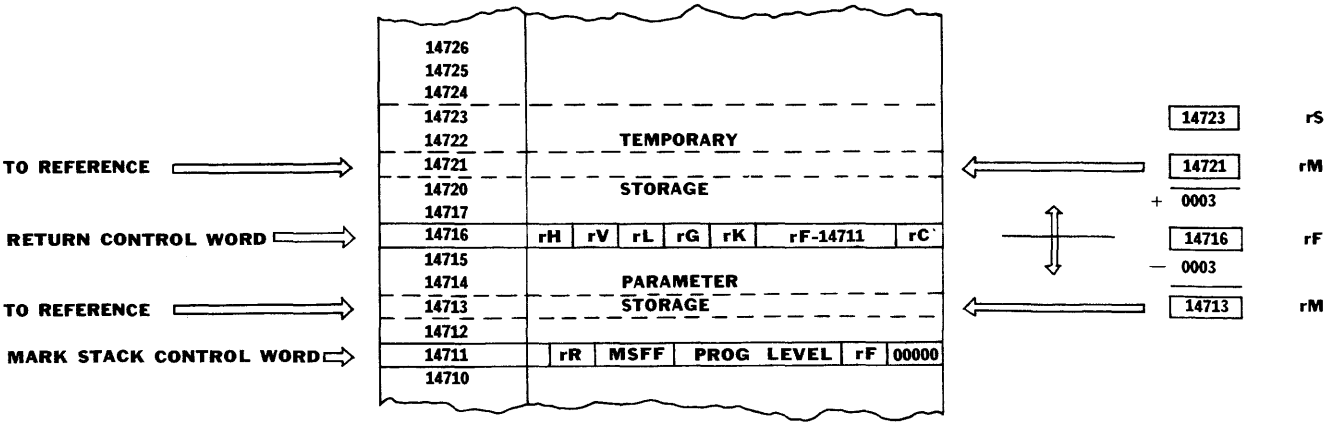


Figure 3-5. Parameter and Temporary Storage for Subroutines

Mark Stack operator is as follows:

The contents of the A and B registers are pushed into the memory stack.

The F and R register contents and settings of the MSFF and Program Level Flip-Flops are stored in the next location in the stack.

If the MSFF is off, the Mark Stack control word is stored in the PRT (R+7).

The contents of the S register are copied into the F register.

The MSFF is turned on.

Once the Mark Stack operation is executed, the program may then store parameters in ascending locations in the stack. When this is complete, a call syllable is used to enter the actual subroutine as described in Case 1. This operation also turns the Mark Stack Flip-Flop off. The subroutine may then obtain parameters by referring to locations the addresses of which are lower than the address specified in the F register, see Figure 3-5.

Locations above the F register are considered as temporary working storage for the subroutine, and these words are also referenced directly by use of the contents of the F register as a base address. Note that the S register continues to control the extent of the temporary storage area.

Subroutine Entry

When a call syllable references a Program Descriptor in the PRT, the Program Descriptor is brought to the A register and checked for its presence in core memory. If the segment is not present, an interrupt occurs to provide notification. When the segment is in core memory, one of the following cases will occur:

Case 1: Operand Call or Descriptor Call with the Mark Stack Flip-Flop turned on. Mark Stack control word has been stored.

A Return control word is stored in the stack.

The contents of the S register are copied into the F register.

The 15 low-order bits of the A register are copied into the C register and the L register is set to zero.

The Sub-Program Level is entered (if it has not been previously entered by another call syllable).

The A and B registers are marked empty.

Control is transferred to the program word specified by the C register.

Case 2: Operand Call or Descriptor Call with the Mark Stack Flip-Flop turned off.

A Return control word is stored in the stack and a MKS control word is stored in the stack. Set F from register A.

The 15 low-order bits of the A register are copied into the C register and the L register is set to zero.

The Sub-Program Level is entered (if it had not been previously entered by another call syllable).

The A and B registers are marked empty.

Control is transferred to the program word specified by the C register.

In general, Operand Call syllables are used to enter a subroutine and obtain an operand, while Descriptor Call syllables are used to enter a subroutine to obtain an address.

Subroutine Exit

When a subroutine has completed its operation and an exit to the calling program is required, a Return or Exit operator is executed depending on the mode. The following operations then take place to provide an automatic exit:

The Return control word addressed by the F register is accessed and placed in B register.

The S, C, G, H, K, V and L registers are restored from this location. If the Operand/Descriptor bit in this location is zero, the flag bit in the A register is set to zero, otherwise it is set to one.

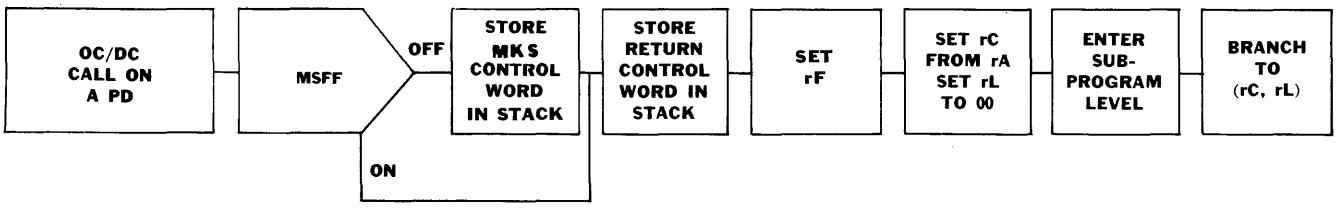


Figure 3-6. Subroutine Entry Options

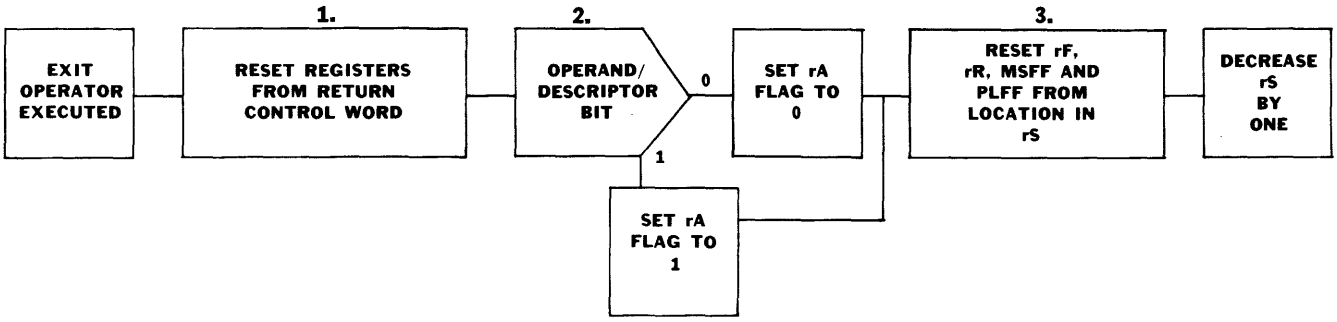


Figure 3-7. Subroutine Exit Options

The Mark Stack control word addressed by the S register is accessed and the F and R registers and the Mark Stack and Program Level Flip-Flops are restored from this word. If the Mark Stack bit is one and the Program Level bit is one, steps 1, 2 and 3 are repeated until a Mark Stack bit of zero is found. A and B registers are marked empty.

The contents of the S register are decreased by one.

The functions thus described for subroutine handling provide a highly efficient and automatic method of operation.

ADDRESSING

Addressing techniques used in the B 5000 System make programs completely independent of actual memory locations. This concept allows programs to be loaded into different and non-contiguous areas of memory to suit operating conditions present at the time of a run. It also permits very large programs to be segmented and run on any machine, regardless of memory size. Finally, this allows multiprocessing techniques to be efficiently implemented on a B 5000 System.

Programs

Programs are divided into segments. Within a segment, syllables are executed in a sequential manner. Transfers of control within a segment are self-relative and only specify the number of syllables that a branch will span. References to other program segments in the Program Level are made indirectly through the use of the Communication Operator. References to subroutines in the Sub-Program Level are made indirectly through the use of Operand and Descriptor Calls on Program Descriptors.

Data Addressing

Data areas are referenced indirectly through Data Descriptors in the PRT; each descriptor references a unique area in memory. If an area consists of a single word, the Data Descriptor contains the address of that word. However, if an area consists of a data array, the Data Descriptor contains the base address of the array. To obtain a word from an array, this base address is indexed to obtain the required address.

The base core address contained in a descriptor can be indexed in any of several ways. Multilevel indexing is also provided so that indexes of arrays can themselves be elements of arrays. For example, when an Operand Call syllable references a Data Descriptor with a size field greater than zero, an automatic index operation occurs. The index value is then checked to determine that it is within the area defined by the descriptor. The address of the descriptor is then incremented by the 10 low-order bits of the B register. The operand at the constructed address is then brought to the A register for subsequent operation. This operation as described, assumes that the index value was inserted in the B register as the result of a prior fetch or computation operation. Literal syllables from the program segment are also used for this operation.

Literals are normally used for *constant* indexing. This is shown in Figure 3-8. In this illustration a literal of 10 is placed in the A register from the program word. An Operand Call then fetches a descriptor whose address field (14711) is indexed by this value. Following the index operation, an operand is automatically fetched from the indexed location (14721).

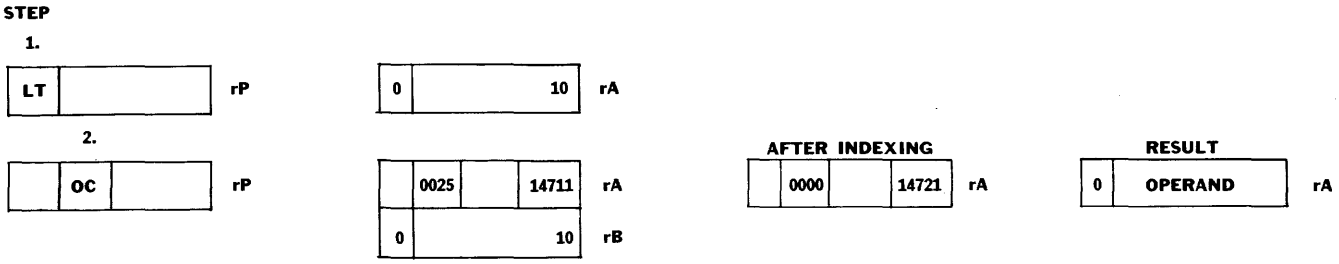


Figure 3-8. Constant Indexing

For *variable* indexing, any algebraic computation may be performed with the result left in the B register for subsequent indexing. An illustration of this is given in Figure 3-9. Two successive Operand Calls are used to fetch two operands which are then added together to form an index value. The remainder of the operation is similar to the previous example.

If multilevel indexing is desired, it is only a normal extension of these operations in the system. A brief example of multilevel indexing is presented in Figure 3-10.

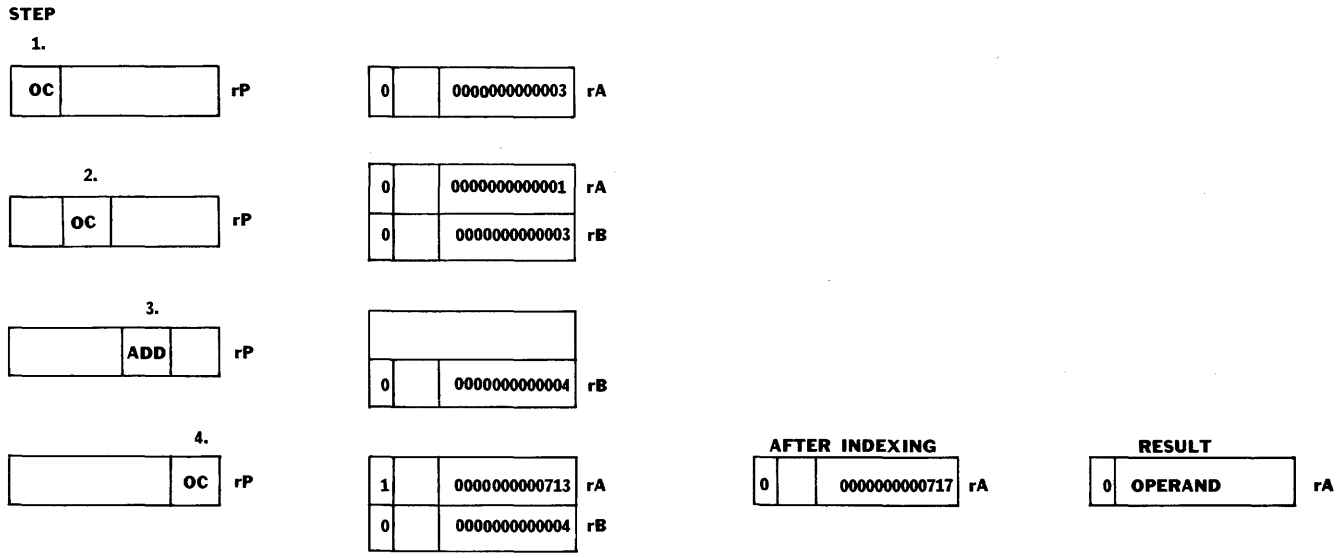


Figure 3-9. Variable Indexing

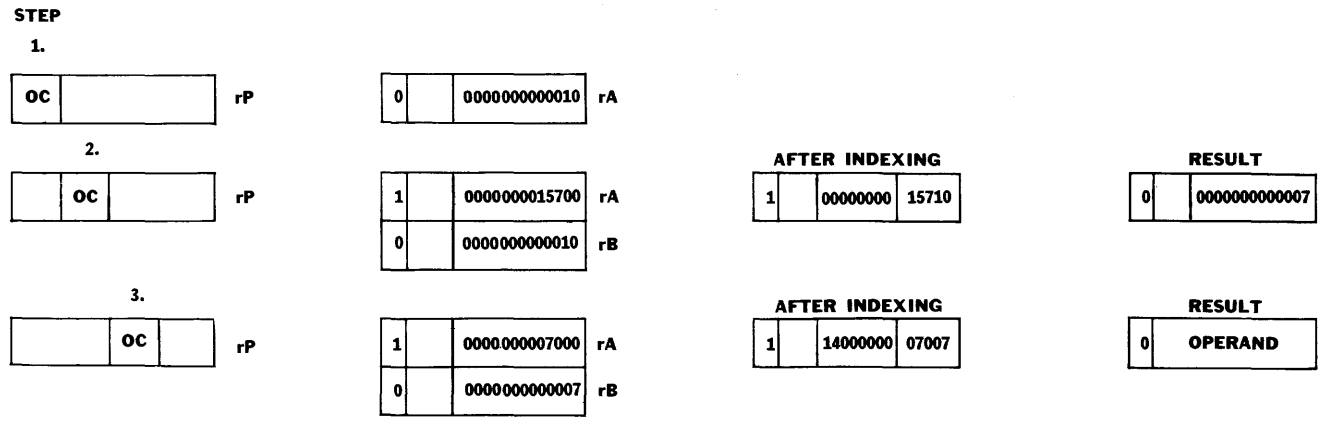


Figure 3-10. Multilevel Indexing

Note that this example used an indexed descriptor to obtain an index value. This latter index was then used to obtain the required operand for some problem.

Multilevel indexing can be expanded indefinitely in many combinations to take into account complex addressing requirements. It should be noted that if the presence bit in a descriptor indicates that the information is not in core memory, an automatic interrupt occurs to provide notification of this fact.

Input/Output Addressing

Input/Output areas are addressed in the same manner as data areas. These areas are also addressed indirectly through the PRT, and can thus be relocated to any area of memory without affecting the program segments. If an input/output area is in the process of receiving new information from a peripheral unit or transmitting information to such a unit, an interrupt immediately occurs if the area is referenced by a program segment during this operation.

Subroutine Addressing

Subroutines are provided with a generalized method of addressing in order to make them virtually independent of the Program Reference Table which is specially composed for each program. To achieve this, the abilities of the Operand Call and Descriptor Call syllables are extended in the Sub-Program Level so that they can address the stack directly, and so that they can use it for parameters and temporary storage. Constants may be stored in the subroutine string, and these syllables can be used to address them also.

Normally the parameters required by each subroutine are stored in, or referenced by, the Program Reference Table. These parameters are transferred to the stack just prior to the execution of the call syllable which references the Program Descriptor. Once control is transferred, the subroutine can address the stack for required parameters, as described in Section 3, as well as constants in its own program string. The subroutine may, under certain circumstances, reference items in the Program Reference Table, and through it reference data in general storage.

The format of Operand Call and Descriptor Call syllables in the Sub-Program Level is illustrated in Figure 3-11. Indicator bits in high order positions of the address field indicate whether reference is to be made to parameters and temporary storage in the stack, to constants located within the subroutine segment, or to the Program Reference Table.

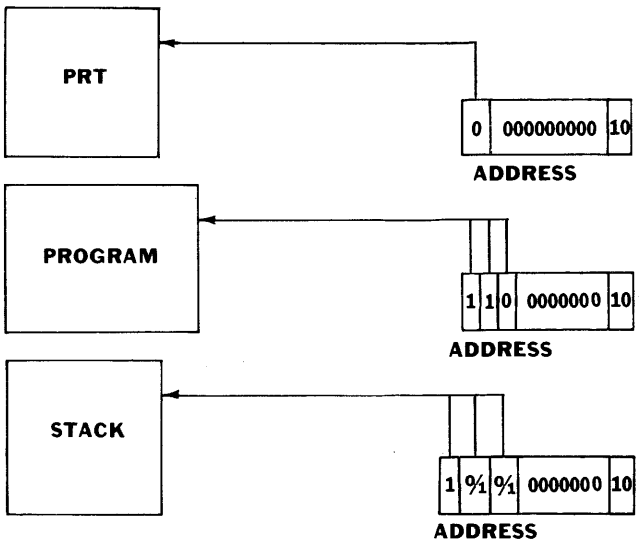


Figure 3-11. Sub-Program Level Call Syllable Formats

DATA EDITING

The B 5280 and B 5281 Processors can operate with fixed length words or variable length fields. These two modes of operation are called the Word Mode and the Character Mode. For certain operations, a processor operating on words is most useful; for other operations, a variable field length mode of operation is most desirable. By combining both abilities in one processor, the system can operate in the mode most desirable for the operation at hand. In the B 5000 System, it is even possible for one processor to be operating in the Word Mode and the other in the Character Mode.

The purpose of the Word Mode is to provide the advantages of high speed parallel operations, floating point abilities and the inherent information density possible in a binary machine. The purpose of the Character Mode is to provide editing, scanning, comparison, and data manipulative abilities, although addition and subtraction are also provided. This latter mode is also particularly well suited to list structures.

The Character Mode is entered by an Operand or Descriptor Call or Initiate operator against a Program Descriptor in the PRT. The Program Descriptor must have the mode bit set to one. The processor is placed in the Sub-Program Level as a result of the call against the Program Descriptor. An Exit operator is used to exit from the Character Mode subroutine to the calling routine, which is normally in the Word Mode.

When operating in the Character Mode, each data word consists of eight alphanumeric characters as illustrated in Figure 3-12.

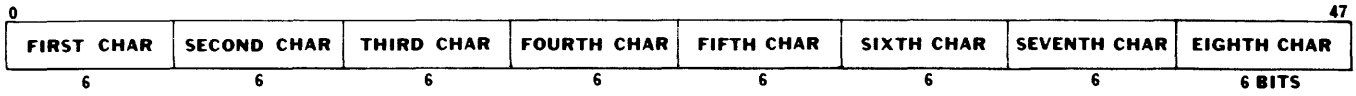


Figure 3-12. Data Word—Character Mode

Programs in the Character Mode can address any character or any bit within a character. Fields can start at any position in a word. A processor in a single operation can operate on fields of any length up to 63 characters long. For example, two 63 character fields can be compared in a single operation. Operations on fields of greater length can easily be programed.

There are three instances when the Character Mode operates with words of the type used in the Word Mode. Operations are provided in the Character Mode for converting numeric information in the alphanumeric representation to the octal notation used in the Word Mode, as well as converting octal information to alphanumeric representation. In both instances, the length of the alphanumeric fields being converted to or from the Word Mode representation can be no greater than eight characters long. Again, conversion of fields of greater length can easily be programed. Transfer Words specify the number of words to be moved.

In the Character Mode of the B 5280 and B 5281 Processors, a special type of syllable is used. The Character Mode syllable is divided into two 6-bit parts: The last part specifies the operation to be performed and, when applicable, the first part specifies the number of times the operation is to be performed, see Figure 3-13.

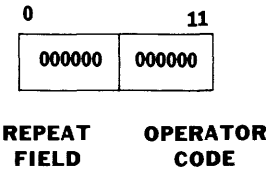


Figure 3-13. Character Mode—Syllable Format

Syllable operators are provided for transferring, deletion, comparison, and insertion of characters or bits. There are also operations which allow the repetition of syllable strings. This latter feature is very useful for complex table look-up operations, and for editing information which contains repeated patterns.

Program segments in the Character Mode are constructed of strings of these syllables. The Character Mode is designed to provide editing, formating, comparison, and other forms of data manipulation. In so doing, a processor uses two general areas of memory—the source area and the destination area. Before a program switches from Word Mode to Character Mode, two descriptors containing the base addresses of these areas are supplied to the stack. When the call syllable references the appropriate Program Descriptor with the mode bit on, the source and destination address registers are set from the descriptors previously supplied to the stack as parameters. The source area or destination area may be changed at any time during Character Mode operation, so that one program may act on several areas.

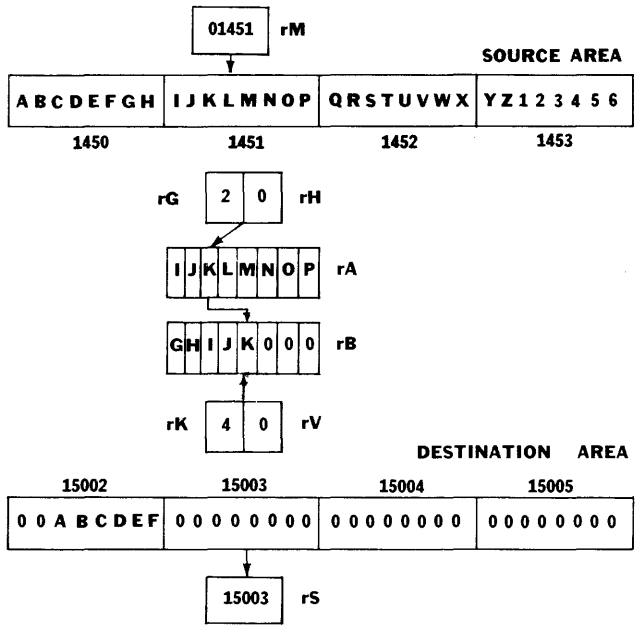


Figure 3-14. Source—Destination Operation

An example of Character Mode operation is illustrated in Figure 3-14. The source string address registers (M, G, and H) indicate that the third character of the word in location 01451 is being processed and has already been transferred from the

source register (A register) to the destination register (B register). The exact location to which it has been sent is identified by the destination string address registers (S, K, and V) which specify the fifth character of the word in location 15003. Note that in this case, the character "A" was transferred from the first position in a word to the third position in another word, demonstrating that fields may begin at any character position (or bit position) and end at any character position (or bit position). Furthermore, insertions, deletions, comparisons, introduction of different source strings, retention of previous destination string information, and arithmetic computations may take place during a single editing operation.

All of these facilities combine to make the Character Mode a very flexible tool for data manipulation.

SECTION 4
DESCRIPTORS

GENERAL

Descriptors are used to make programs independent of a fixed data location and program location. This provides a flexible means of indirect addressing. Each descriptor contains an actual base memory location and a size field, along with special control information where necessary. The Program Reference Table is used as a storage area for these descriptors.

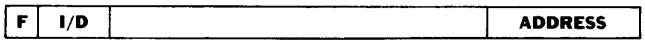
Since the base addresses for any one program are located in a single PRT, they are easily modified whenever necessary. For instance, core memory address fields are easily modified when segment overlays are made during the operation of a large program in a small memory. They are also easily modified in the case of multiprocessing several programs in a single system.

Several syllables may reference a single descriptor in the PRT. This feature permits a highly efficient utilization of memory, because addresses and control information do not have to be repetitively stored for each instruction. Control information may consist of unit number, binary/alpha representation, status, presence, continuity, mode, and other similar types of control fields.

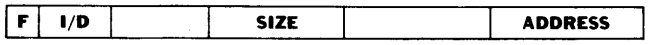
Program protection is provided through use of a size field. This field contains the maximum number of words allocated to the area defined by the descriptor. All index values are automatically checked against the size field before the actual index operations take place. This insures that references are always made to words within the particular area.

Descriptors are identified by the high-order bit known as the "flag" bit. When this bit is on, and the word has come from the PRT, the associated word is a descriptor. There are two general types of descriptors. These are identified by the I/D (Identification) field.

1. Program Descriptors—used to identify and locate segments of a program.



2. Data Descriptors—used to identify and locate working storage areas, tables, input/output areas, and other similar information for a program.

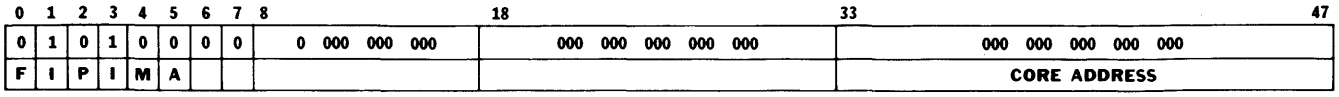


A detailed description of each descriptor is presented in the following section.

PROGRAM SEGMENTS

Program Descriptor

This type of descriptor is used to identify the location of a program segment in memory and on the drum. If the presence bit is off, an interrupt will occur when attempting to enter the segment. The program segment must then be read into an available memory area. The processor will enter either the Word Mode or Character Mode as specified by the Mode bit before transferring control to the first syllable of the new program segment. The parameter bit indicates whether parameters are needed for the execution of the program segment. The segment will not be entered if they are required and have not been stored in the stack.

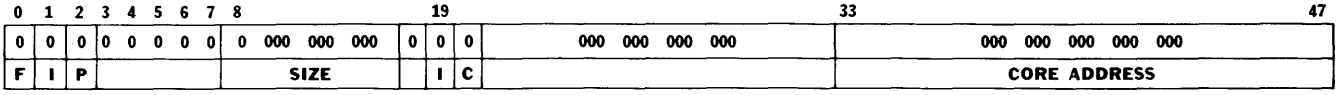


BIT POSITION	USE
0	FLAG—type of word 1-descriptor
1-3	IDENTIFICATION—type of descriptor 1P1—Program Descriptor
2	PRESENCE—availability of segment for execution 0—not in core memory 1—in core memory
4	MODE—type of syllables in this segment 0—word 1—character
5	ARGUMENT BIT If the M bit is one and the A bit is zero, the A register is marked full and the operation ended. If the M bit is zero and the A bit is zero, a Mark Stack control word is formed and placed in the stack. If the A bit is one and the Mark Stack Flip-Flop is zero, the A register is set to full and the operation is ended. 0—argument not required 1—argument required
6-32	Reserved for use by Programing Systems
33-47	CORE ADDRESS—of the first program word. Succeeding program words are stored in consecutively ascending locations.

DATA AND INPUT/OUTPUT

Data Descriptor

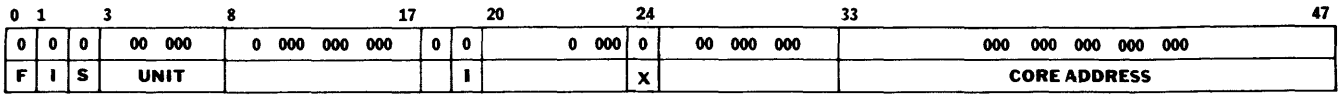
This type of descriptor is used to indicate the core address of the base of a data array. The size field indicates the length. If the presence bit is off, an interrupt will occur. The integer bit can be used to specify whether a word is to be stored in fixed or floating point form.



BIT POSITION	USE
0	FLAG—type of word 1—descriptor
1	IDENTIFICATION—type of descriptor 0—Data Descriptor
2	PRESENCE—availability of data for processing. 0—not in core memory 1—in core memory
3-7	Reserved for use by Programing Systems
8-17	SIZE—number of words in data array. Zero indicates a one word area.
18	Reserved for use by Programing Systems
19	INTEGER—for Conditional Integer Store operators use 0—data is to remain in floating point notation . 1—data is to be converted into an integer.
20	CONTINUITY BIT—for controlling the type of interrupt caused by a Program Release operator. 0—set the Program Release Interrupt —I/O areas not tanked. 1—set the Continuity Interrupt—I/O areas are tanked.
21-32	Reserved for use by Programing Systems
33-47	CORE ADDRESS—of the first data word. Succeeding words will be located in consecutively ascending locations.

Supervisory Printer Descriptor

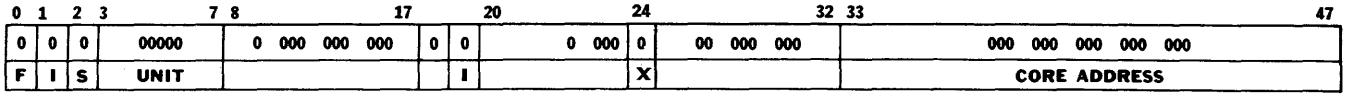
This descriptor is used to transfer alphanumeric data from memory to the Supervisory Printer. The transfer is terminated when a group mark is encountered in the message.



BIT POSITION	USE
0	FLAG—type of word 1—descriptor
1	IDENTIFICATION—type of descriptor 0—Data Descriptor
2	STATUS—availability of output area 0—area available to output unit only 1—area available to program only
3-7	UNIT DESIGNATION 11110—Supervisory Printer Unit 30
8-23	Reserved for use by Programing Systems
24	EXTERNAL 0—printout
25-32	Reserved for use by Programing Systems
33-47	CORE ADDRESS—the address in memory from which the first character will be printed.

Keyboard Descriptor

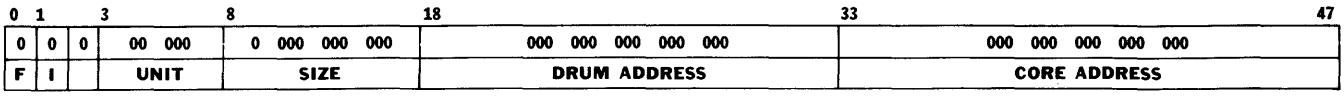
This descriptor is used to transfer alphanumeric data from the supervisory keyboard to memory. The transfer is terminated when the End-Of-Message (EOM) key is depressed, causing a group mark to be stored as the last character of the message.



BIT POSITION	USE
0	FLAG—type of word 1—descriptor
1	IDENTIFICATION—type of descriptor 0—Data Descriptor
2	STATUS—availability of input area 0—area available to input unit only 1—area available to program only
3-7	UNIT DESIGNATION 11110—Supervisory Printer Unit 30
8-23	Reserved for use by Programing Systems
24	EXTERNAL 1—keyboard input
25-32	Reserved for use by Programing Systems
33-47	CORE ADDRESS—the address in memory into which the first character will be stored.

Drum Read Descriptor

This descriptor is used to transfer program segments or data from a drum storage unit to a memory module. Up to 1023 words can be transferred from the drum by use of a single descriptor.



BIT POSITION — USE

0 FLAG—type of word
1—descriptor

1 IDENTIFICATION—type of descriptor
0—Data Descriptor

2 EXTERNAL—type of operation
1—read

3-7 UNIT DESIGNATION
00100—Drum No. 1 Unit 4
01000—Drum No. 2 Unit 8

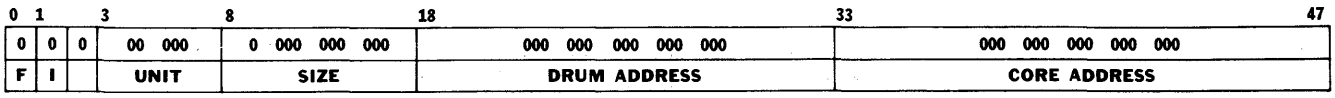
8-17 SIZE—number of words to be read
From 0 to 1023 words

18-32 DRUM ADDRESS—the address on the drum from which the first word is to be read.

33-47 CORE ADDRESS—the address in memory into which the first word is to be read. Following words are read into consecutively ascending locations.

Drum Write Descriptor

This descriptor is used to transfer data from a memory module to a drum band. Up to 1023 words may be transferred to a drum by a single descriptor.



BIT POSITION — USE

0 FLAG—type of word
1—descriptor

1 IDENTIFICATION—type of descriptor
0—Data Descriptor

2 EXTERNAL—type of operation
0—write

3-7 UNIT DESIGNATION
00100—Drum No. 1 Unit 4
01000—Drum No. 2 Unit 8

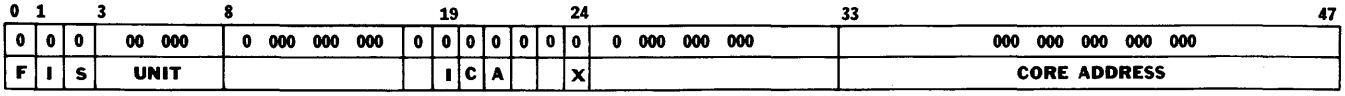
8-17 SIZE—number of words to be written
From 0 to 1023 words

18-32 DRUM ADDRESS—the address on the drum to which the first word is to be written.

33-47 CORE ADDRESS—the address in memory which contains the first word to be written. Following words are written from consecutively ascending locations.

Card Read Descriptor

This descriptor is used to cause a card to be read from a designated card reader into a set of contiguous memory locations. The core address of the descriptor gives the location into which the first word will be read.



BIT POSITION USE

0 FLAG—type of word
1—descriptor

1 IDENTIFICATION—type of descriptor
0—Data Descriptor

2 STATUS—availability of input area
0—area available to input unit only
1—area available to program only

3-7 UNIT DESIGNATION
01010—Card Reader No. 1 Unit 10
01110—Card Reader No. 2 Unit 14

8-18 Reserved for use by Programing Systems

19 INTEGER—for Conditional Integer Store operators use
0—data is to remain in floating point notation.
1—data is to be converted into an integer.

20 CONTINUITY—used for tanking
1—one of two or more descriptors for a tank, but not the last.
0—last descriptor for a tank, or the only descriptor if no tank.

21 FORMAT—information representation
1—information is in column binary format
0—information is in alphanumeric format

22-23 Reserved for use by Programing Systems

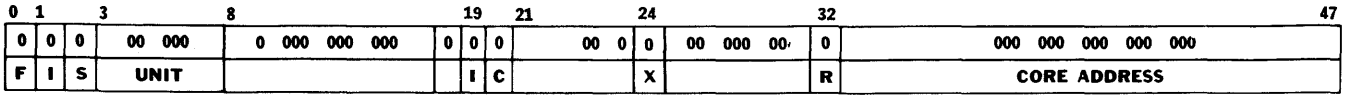
24 EXTERNAL
1—input

25-32 Reserved for use by Programing Systems

33-47 CORE ADDRESS—the address into which the first information from the card will be read. Following information will be read into consecutively ascending locations.

Card Punch Descriptor

This descriptor is used to cause a card to be punched from memory. The core address of the descriptor gives the base address of the area from which the card will be punched. Following words will be punched from consecutively ascending locations.



BIT POSITION USE

0 FLAG—type of word
1—descriptor

1 IDENTIFICATION—type of descriptor
0—Data Descriptor

2 STATUS—availability of output area
0—area available to output only
1—area available to program only

3-7 UNIT DESIGNATION

01010—Card Punch Unit 10

8-18 Reserved for use by Programing Systems

19 INTEGER—for Conditional Integer Store operators use
0—data is to remain in floating point notation.
1—data is to be converted into an integer.

20 CONTINUITY—used for tanking
1—one of two or more descriptors for a tank, but not the last one.
0—last descriptor for a tank or the only descriptor if no tank.

21-23 Reserved for use by Programing Systems

24 EXTERNAL
0—output

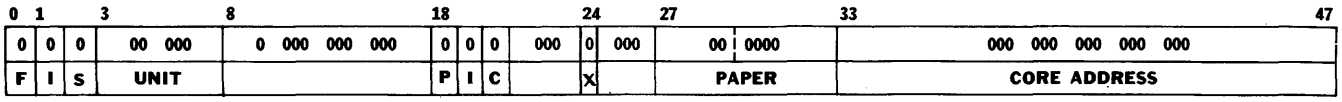
25-31 Reserved for use by Programing Systems

32 STACKER (300 CPM only)
1—select auxiliary stacker
0—select primary stacker

33-47 CORE ADDRESS—the address from which the first word will be punched. Following words are punched from consecutively ascending locations.

Line Printer Descriptor

This descriptor is used to cause information from memory to be printed on a designated line printer. Each word printed from memory is assumed to be alphanumeric format. Line spacing or skipping occur after a line has been printed.



BIT POSITION USE

0 FLAG—type of word
1—descriptor

1 IDENTIFICATION—type of descriptor

0—Data Descriptor

2 STATUS—availability of output area
0—area available to output unit only
1—area available to program only

3-7 UNIT DESIGNATION
10110—Printer No. 1 Unit 22
11010—Printer No. 2 Unit 26

8-17 Reserved for use by Programing Systems

18 PRINT—
0—print
1—inhibit print, space paper as specified in 27-32.

19 INTEGER—for Conditional Integer Store operators use
0—data is to remain in floating point notation.
1—data is to be converted into an integer.

20 CONTINUITY—used for tanking
1—one of two or more descriptors for a tank, but not the last one.
0—last descriptor for a tank, or the only descriptor if no tank.

21-23 Reserved for use by Programing Systems

24 EXTERNAL
0—output

25-26 Reserved for use by Programing Systems

27-32 PAPER—used to control spacing and skipping.

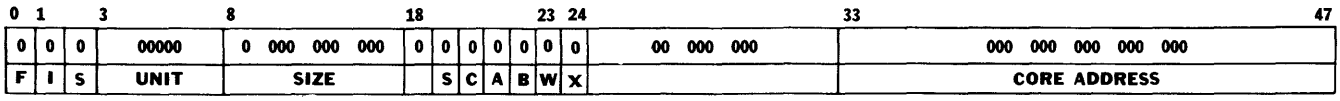
Bits 27-28
00—no space
01—double space
10—single space
11—double space

Bits 29-32
f=0—space paper as indicated in bits 27 and 28
f≠0—skip to stop specified by selected channel (f) in Carriage Control Tape

33-47 CORE ADDRESS—the address from which the first eight characters are printed. Following characters are printed from consecutively ascending locations.

Magnetic Tape Read Descriptor

This descriptor is used to cause the next record to be read from a designated tape storage unit. Tapes may be read forward or backward as specified by the descriptor. Data is read as octal or alphanumeric characters depending on the Format bit of the descriptor.



BIT POSITION USE

0 FLAG—type of word
1—descriptor

1 IDENTIFICATION-type of descriptor

0—Data Descriptor

2 STATUS—availability of input area
0—area available to input unit only
1—area available to program only

3-7 UNIT DESIGNATION
XXXX1—all odd unit numbers from 1 to 31, inclusive

8-17 SIZE—number of words to be read. Any additional words are lost.

18 0—unit control bit

19 INTEGER—for Conditional Integer Store operators use
0—data is to remain in floating point notation.
1—data is to be converted into an integer.

20 CONTINUITY—used for tanking
1—one of two or more descriptors for a tank, but not the last one.
0—last descriptor for a tank, or the only descriptor if no tank.

21 FORMAT—information representation
1—information is in binary format
0—information is in alphanumeric format

22 DIRECTION—tape movement
0—forward
1—backward

23 WORD CONTROL
1—use size field to control read
0—ignore size field

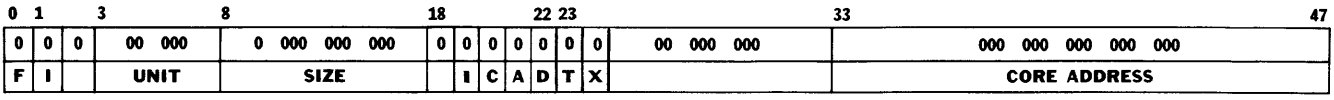
24 EXTERNAL
1—input

25-32 Reserved for use by Programing Systems

33-47 CORE ADDRESS—the address into which the first character will be read from tape. The memory address steps up when reading in the forward direction or steps down when reading in the backward direction.

Magnetic Tape Write Descriptor

This descriptor is used to write data from memory to a designated tape storage unit. When writing in binary format,size field is used to specify the record length. When writing in alphanumeric format, writing continues until a group mark is sensed. The group mark is not written on tape.



BIT POSITION USE

0 FLAG—type of word
1—descriptor

1 IDENTIFICATION—type of descriptor

0—Data Descriptor

2 STATUS—availability of output area
0—area available to output unit only
1—area available to program only

3-7 UNIT DESIGNATION
XXXX1—all odd unit numbers from 1 to 31, inclusive.

8-17 SIZE—number of words to be written (binary format only—group mark is encountered in the alphanumeric format)

18 1—erase

19 INTEGER—for Conditional Integer Store operators use
0—data is to remain in floating point notation.
1—data is to be converted into an integer.

20 CONTINUITY—used for tanking
1—one of two or more descriptors for a tank, but not the last one.
0—last descriptor for a tank, or the only descriptor if no tank.

21 FORMAT—information representation
1—data written in binary code by words.
0—data written in alphanumeric code by characters.

22 DIRECTION
0—forward

23 TERMINATION
0—alphanumeric (terminated by group mark)
1—binary or binary erase (terminated by N words)

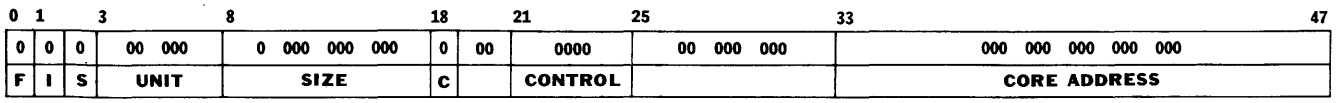
24 EXTERNAL
0—output

25-32 Reserved for use by Programing Systems

33-47 CORE ADDRESS—the address from which the first word or characters will be written. Following characters or words are written from consecutively ascending locations.

Paper Tape Read Descriptor

This descriptor is used to cause the next record to be read from a designated unit. Paper tape can be read in the forward direction only. Data is read as octal or alphanumeric characters depending on the format bit of the descriptor.



BIT POSITION — USE

0 FLAG—type of word
1—descriptor

1 IDENTIFICATION—type of descriptor
0—Data Descriptor

2 STATUS—availability of input area
0—area available to input unit only
1—area available to program only

3-7 UNIT DESIGNATION
10010—Reader No. 1 Unit 18
10100—Reader No. 2 Unit 20

8-17 SIZE—number of words to be read

19-20 Reserved for use by Programing Systems

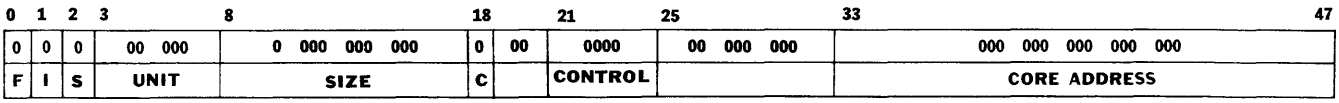
18, 21-24 CONTROL—
00011—Read alpha
01011—Read binary
10011—Space—Stop on control code
11011—Space—Stop on word counter
XX111—Rewind

25-32 Reserved for use by Programing Systems

33-47 CORE ADDRESS—starting memory address

Paper Tape Write Descriptor

This descriptor is used to write data from memory to a designated punch unit. When writing in binary format, size field is used to specify the record length. When writing in alphanumeric format, control code or size field is used to specify record length.



BIT POSITION USE

0 FLAG—type of word
1—descriptor

1 IDENTIFICATION—type of descriptor
0—Data Descriptor

2 STATUS—availability of output area
0—area available to output unit only
1—area available to program only

3-7 UNIT DESIGNATION
10010—Punch No. 1 Unit 18
10100—Punch No. 2 Unit 20

8-17 SIZE—number of words to be punched in binary mode. Maximum number of words to be punched in alpha mode.

19-20 Reserved for use by Programing Systems

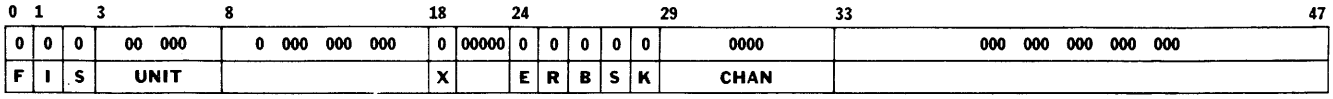
18, 21-24 CONTROL—
00 X 10—Punch alpha—Stop control code
01 X 10—Punch binary—Stop on word counter
10 X 10—Punch all channels—Stop on control code
11 X 10—Punch all channels—Stop on word counter

25-32 Reserved for use by Programing Systems

33-47 CORE ADDRESS—Starting memory address

External Control Descriptor

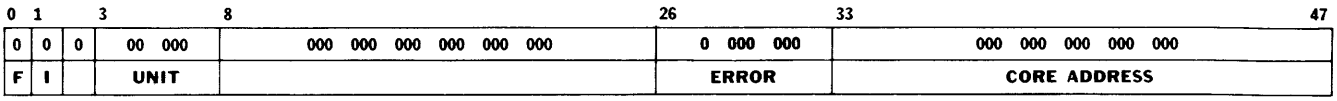
This descriptor is used to control operations of the magnetic tape units and line printers which do not require information to be read from or written into core memory.



BIT POSITION	USE
0	FLAG—type of word 1—descriptor
1	IDENTIFICATION—type of original I/O Descriptor 0—Data Descriptor
2	STATUS—availability of area 0—area available to unit only 1—area available to program only 1—Data Descriptor
3-7	UNIT DESIGNATION—magnetic tape unit or line printer descriptors.
8-17	Reserved for use by Programing Systems
18	EXTERNAL—type of operation 1—perform functions indicated by bits 24 through 32
19-23	Reserved for use by Programing Systems
24	ERASE 1—Erase tape to next record 0—no operation
25	REWIND 1—rewind tape 0—no operation
26	Reserved for use by Programing Systems
27-32	PAPER—used to control spacing and skipping. *Bits 27-28* 00—no space 01—double space 10—single space 11—double space *Bits 29-32* f=0—space paper as indicated in bits 27 and 28 f≠0—skip to stop specified by selected channel (f) in paper tape loop control
33-47	Reserved for use by Programing Systems

External Result Descriptors

These descriptors are the original Input/Output Descriptors sent to the I/O Channels, but with certain control information inserted in them to describe the results of the I/O operation. This control information is presented in Table 4-1, Indicated Error Conditions.



BIT POSITION USE

0 FLAG—type of word
1—descriptor

1 IDENTIFICATION—type of descriptor
0—Data Descriptor

2 Irrelevant

3-7 UNIT DESIGNATION

8-25 Irrelevant

26-32 ERROR CONDITIONS
See Table 4-1

33-47 CORE ADDRESS—the address of the last location referenced in memory.

Table 4-1 Indicated Error Conditions

	BIT POSITIONS						
UNIT	26	27	28	29	30	31	32
Message Printer	Memory Overflow			Parity Error Memory To I/O	Not-Ready Not-Present	Descriptor Parity	Busy
Keyboard	Memory Overflow		Character Input Error		Malfunction Power-Off	Descriptor Parity	Busy
Drum Read	Memory Overflow			Parity Error Drum To I/O	Not-Ready Not-Present Malfunction	Descriptor Parity	Busy
Drum Write	Memory Overflow	Lockout		Parity Error Memory To I/O		Descriptor Parity	Busy
Card Read	Memory Overflow	End of File	Read Error	Invalid Character	Not-Ready Hopper Empty Stacker Full	Descriptor Parity	Busy
Card Punch	Memory Overflow		Punch Error	Parity Error Memory To I/O	Card Jam Not-Present	Descriptor Parity	Busy
Line Printer	Memory Overflow	End of Page	Print Check Previous Line	Parity Error Memory To I/O	Not-Ready No Paper Power-Off Malfunction Not-Present	Descriptor Parity	Busy
Magnetic Tape Read	Memory Overflow	End of File	Character Parity Error Tape to I/O		Not-Ready Local Tape Break	Descriptor Parity	Busy
Magnetic Tape Write	Lockout (26 & 28 bits)	End of Tape	Parity Error	Parity Error Memory To I/O	Power-Off Not-Present	Descriptor Parity	Busy
External Control		End of Page		Unit Not-Present	Not-Ready See Mag Tape and Printer	Descriptor Parity	Busy
Paper Tape Punch	Memory Overflow	Low Tape		Parity Error Memory To I/O	Not-Ready Local Tape Break Power-Off	Descriptor Parity	Busy
Paper Tape Read	Memory Overflow	End of Tape	Beginning of Tape	Parity Error	Not-Ready Local Tape Break Power-Off	Descriptor Parity	Busy

SECTION 5
SYLLABLES

GENERAL

The coding structure for the B 5000 System is composed of a set of instructions called syllables. Each syllable is twelve bits in length and contains a code which directs the processor to execute a defined function. These functions are described in a general manner in this section.

There are two modes of operation as described in Section 2, Word Mode and Character Mode. Each mode has its own specific set of syllables. As a result, the syllables for each mode are described separately in the following sections. They are presented in functional order by mnemonic code within each section. An alphabetic listing of these operators is contained in the Appendix, as well as a listing by function.

WORD MODE

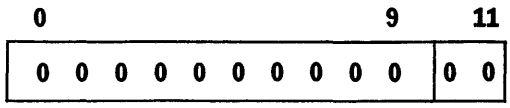
When operating in this mode, data is usually manipulated as a 48-bit word. For all operations requiring operands in the A and/or B registers, the stack control tests for the required conditions and adjusts the contents of these registers as necessary. The description of the syllables assumes that this stack control operation has occurred.

Unless otherwise stated, after the execution of any syllable, the A and/or B registers are set to empty if they contained an operand required by the syllable and do not contain a result developed by the operation.

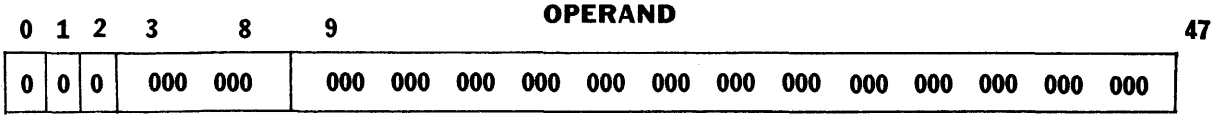
For arithmetic operations, a word is considered as having the following format:

BIT	USE
0	Flag 0—Operand
1	Sign of the operand 0—Positive 1—Negative
2	Sign of the exponent 0—Positive 1—Negative
3-8	Exponent Range 00 through 77 (octal). Each bit represents one octal digit of the integer
9-47	Integer Consists of 13 octal digits. Normalization or shifting is accomplished by moving the integer left or right in an octal manner and adjusting the exponent.

A program string consists of a series of 12-bit syllables. There are four types of syllables. The format and a description of each type is as follows:



BITS	USE
0-9	Operation code and/or data depending on syllable type. 01—Operator 00—Literal



10—Operand Call
11—Descriptor Call

10-11 Syllable type

Syllable Description

LITERAL

The ten high-order bits of the syllable are placed in the A register as a positive integer.

OPERAND CALL

The ten high-order bits of the syllable are added to the contents of the R register and the resulting address is stored in the M register. The word found at this address is brought into the A register and examined:

If it is an operand, no further action occurs.

If it is a Data Descriptor, the size field of the descriptor is examined. If it is non-zero, the ten low-order bits of the B register are first checked against the size field and then added to the fifteen low-order bits of the descriptor. If it is zero, no incrementation takes place. In either case, the word found at the address specified by the descriptor is brought into the A register.

If it is a Program Descriptor and the Mark Stack Flip-Flop is on, the contents of the C, L and F registers and the syllable-type indicator are stored in the stack at an address specified by the S register. If the MSFF is off, see Section 3 for a complete description. The fifteen low-order bits of the descriptor are stored in the C register and the L register is set to zero. The mode bit of the descriptor is examined. If it is zero, the processor remains in the Word Mode. If it is one, the processor enters the Character Mode.

If it is a control word, it is treated as an operand.

DESCRIPTOR CALL

The ten high-order bits of the syllable are added to the contents of the R register and the resulting address is stored in the M register. The word found at this address is brought into the A register and examined.

If it is an operand, it is replaced by the contents of the M register which was the address of the operand. The flag bit is set to one making it a descriptor and the size field is set to zero.

If it is a Data Descriptor, the size field of the descriptor is examined. If it is non-zero, the ten low-order bits of the B register are first checked against the size field, added to the fifteen low-order bits of the descriptor and the size field is set to zero. If it is zero, no incrementation occurs.

If it is a Program Descriptor and the Mark Stack Flip-Flop is on, the contents of the C, L and F registers and the syllable-type indicator are stored in the stack at an address specified by the S register. If the MSFF is off, see Section 3 for a complete description. The fifteen low-order bits of the descriptor are stored in the C register and the L register is set to zero. The mode bit of the descriptor is examined. If it is zero, the processor remains in the Word Mode; if it is one, the processor enters the Character Mode.

If it is a control word, it is made a descriptor of size zero.

OPERATORS

This type of syllable designates the manner in which the data in the A and/or B register is to be operated on. Note that for some operators, the six high-order bits of the syllable are used as a counter or modifier. A description of each operator is given in the following paragraphs. Special conditions which would cause interrupts to occur are not covered in this manual.

Arithmetic Operators

ADD	01

ADD. Add algebraically the operands in the A and B registers. If the exponent of either or both operands was non-zero, round and normalize the sum. Mark the A register empty.

SUB	01

SUBTRACT. Subtract algebraically the operand in the A register from the operand in the B register. If the exponent of either or both operands was non-zero, round and normalize the difference. Store the difference in the B register. Mark the A register empty.

MUL	01

MULTIPLY. Multiply algebraically the operand in the B register by the operand in the A register and store the product in the B register. Mark the A register empty.

DIV	01

DIVIDE. Divide the operand in the B register by

the operand in the A register. Store the normalized and rounded quotient in the B register. Mark the A register empty.

IDV	01

INTEGER DIVIDE. Normalize the operands in the A and B registers. Divide the operand in the B register by the operand in the A register. Store the quotient in the B register. Mark the A register empty.

RDV	01

REMAINDER DIVIDE. Normalize the operands in the A and B registers. Divide the operand in the B register by the operand in the A register. Store the remainder in the B register. Mark the A register empty.

ADL	01

ADD DOUBLE LENGTH. Add the double length operand in the A and B registers to the double length operand located in the next two words of the stack. Normalize the sum, and store it in the A and B registers. The A register contains the most significant part of the result.

SDL	01

SUBTRACT DOUBLE LENGTH. Subtract the double length operand in the A and B registers from the double length operand located in the next two words of the stack. Store the normalized difference in the A and B registers. The A register contains the most significant part of the result.

MDL	01

MULTIPLY DOUBLE LENGTH. Multiply the double length operand in the A and B registers by the double length operand located in the next two words of the stack. Store the normalized product in the A and B registers. The A register contains the most significant part of the result.

DDL	01

DIVIDE DOUBLE LENGTH. Divide the double length operand in the A and B registers into the double length operand located in the next two words of the stack. Store the normalized quotient in the A and B registers. The A register contains the most significant part of the result.

Stack Operators

XCH	01

EXCHANGE. Replace the contents of the A register by the contents of the B register. Simultaneously replace the contents of the B register by the contents of the A register.

DUP	01

DUPLICATE. Adjust the stack until the A register is empty and the B register is full. Duplicate the contents of the B register in the A register and mark the A register full.

Logical Operators

LND	01

LOGICAL AND. Examine corresponding bits of the A and B registers. If a one appears in both registers, retain the one in the B register; otherwise, place a zero in that position in the B register. The flag bit remains unaltered. Mark the A register empty.

LOR	01

LOGICAL OR. Examine corresponding bits of the A and B registers. If a one appears in either register, place a one in the corresponding bit position of the B register. The flag bit remains unaltered. Mark the A register empty.

LQV	01

LOGICAL EQUIVALENCE. Examine corresponding bits of the A and B registers. If they are equal, place a one in the corresponding bit position of the B register; otherwise, place a zero in that position of the B register. The flag bit remains unaltered. Mark the A register empty.

LNG	01

LOGICAL NEGATE. Replace every one bit in the A register with a zero and each zero bit with a one. The flag bit remains unaltered.

Relational Operators

These operators perform comparisons on the two top operands in the stack. The operands are removed from the stack and the result of the comparison is placed in the B register. The operands may be in an un-normalized form with the required scaling taking place in the comparison operation. Operands of zero, minus zero and a zero mantissa with a non-zero exponent are considered equal.

GTR	01

B GREATER THAN A. Compare the operand in the B register to the operand in the A register. If the value of the operand in the B register is greater than the value of the operand in the A register, set the B register to one; otherwise, set it to zero. Mark the A register empty.

LSS	01

B LESS THAN A. Compare the operand in the B register to the operand in the A register. If the value of the operand in the B register is less than the value of the operand in the A register, set the B register to one; otherwise, set it to zero. Mark the A register empty.

LEQ	01

B LESS THAN OR EQUAL TO A. Compare the operand in the B register to the operand in the A register. If the value of the operand in the B register is less than or equal to the value of the operand in the A register, set the B register to one; otherwise, set it to zero. Mark the A register empty.

EQL	01

B EQUAL TO A. Compare the operand in the B register to the operand in the A register. If the value of the operand in the B register is equal to the value of the operand in the A register, set the B register to one; otherwise, set it to zero. Mark the A register empty.

NEQ	01

B NOT EQUAL TO A. Compare the operand in the B register to the operand in the A register. If the value of the operand in the B register is not equal to the value of the operand in the A register, set the B register to one; otherwise, set it to zero. Mark the A register empty.

GEQ	01

B GREATER THAN OR EQUAL TO A. Compare the operand in the B register to the operand in the A register. If the value of the operand in the B register is greater than or equal to the value of the operand in the A register, set the B register to one; otherwise, set it to zero. Mark the A register empty.

Subroutine Operators

MKS	01

MARK STACK. Push down the contents of the A and B registers into the stack if they are full. Construct a Mark Stack control word containing the contents of the F and R registers and the settings of the Mark Stack and the Program Level Flip-Flops and store it in the stack. Copy the address of the cell containing the Mark Stack control word into the F register. Examine the setting of the Mark Stack Flip-Flop. If it is zero, store the Mark Stack control word in a specific location. Set the Mark Stack Flip-Flop to one.

XIT	01

EXIT. Access the word addressed by the F register, the Return control word, and place it in the B register. Reset the contents of the C, L, G, H, K, and V registers from the control word. Set the S register to the contents of the F register field of the Return control word.

Access the word addressed by the S register, the Mark Stack control word. Reset the contents of the R and F registers and the settings of the Mark Stack and the Program Level Flip-Flops from the Mark Stack control word.

Decrease the setting of the S register by one and mark the A and B registers empty.

Examine the Mark Stack bit in the Mark Stack

control word. If it is a zero, store the Mark Stack control word in a specific location.
If the Mark Stack bit is a one, access the previous Mark Stack control word addressed by the F register. Repeat the process until a Mark Stack control word is obtained with a Mark Stack bit of zero. Store the control word in a specific location.

RNO	01

RETURN NORMAL. Adjust the stack until the A register is full and the B register is empty.

Access the word addressed by the F register, the Return control word, and place it in the B register. Reset the C, L, G, H, K, and V registers from their respective fields of the control word. Set the S register to the contents of the F register field of the Return control word.

Access the word addressed by the S register, the Mark Stack control word. Reset the contents of the R and F register and the settings of the Mark Stack and the Program Level Flip-Flops from the Mark Stack control word. Decrease the setting of the S register by one.

Examine the Mark Stack bit in the Mark Stack control word. If it is a zero, store the Mark Stack control word in a specific location.

If the Mark Stack bit is a one, access the previous Mark Stack control word addressed by the F register. Repeat the process until a Mark Stack control word is obtained with a Mark Stack bit of zero. Store the Mark Stack control word in a specific location.

Examine the Syllable Indicator bit in the Return control word. If it is a zero, the word in the A register is treated as though it was obtained by an Operand Call syllable; if it is a one, the word in the A register is treated as though it was obtained by a Descriptor Call syllable.

RSP	01

RETURN SPECIAL. Adjust the stack until the A register is full and the B register is empty.

Access the word addressed by the S register, the Return control word, and place it in the B register. Reset the C, L, G, H, K, and V registers from their respective fields of the Return control word. Replace the contents of the S register with the contents of the F register field of the Return control word.

Access the word addressed by the S register, the Mark Stack control word. Reset the contents of the R and F registers and the settings of the Mark Stack and the Program Level Flip-Flops from the Mark Stack control word. Decrease the setting of the S register by one.

Examine the Mark Stack bit in the Mark Stack control word. If it is a zero, store the Mark Stack control word in a specific location. If the Mark Stack bit is a one, access the previous Mark Stack control word addressed by the F register. Repeat the process until a Mark Stack control word is obtained with a Mark Stack bit of zero. Store the Mark Stack control word in a specific location.

Examine the Syllable Indicator bit in the Return control word. If it is a zero, the word in the A register is treated as though it was obtained by an Operand Call syllable; if it is a one, the word in the A register is treated as though it was obtained by a Descriptor Call syllable.

Branching Operators

The word in the top of the stack is used to specify the cell or syllable to which branching will occur. If it is an operand, it will specify the number of syllables to be jumped, forward or backward, relative to the location of the branch operator. If it is a descriptor, it will specify the address to which the branch will be made.

Conditional branches test the low-order bit of the B register. Branches will occur on a false condition.

BFU	01

BRANCH FORWARD UNCONDITIONAL. Examine the word in the A register. If it is an operand, increase the contents of the C and L registers by the 12 low-order bits of the operand. If it is a descriptor, replace the contents of the C register by the 15 low-order bits of the descriptor and set the L register to zero. In either case, mark the A register empty.

BBU	01

BRANCH BACKWARD UNCONDITIONAL. Examine the word in the A register. If it is an operand, decrease the contents of the C and L registers by the 12 low-order bits of the operand. If it is a descriptor, replace the contents of the C register by the 15 low-order bits of the descriptor and set the L register to zero. In either case, mark the A register empty.

BFC	01

BRANCH FORWARD CONDITIONAL. If the low-order bit of the B register is a one, mark the A and B registers empty and terminate the operation. If the low-order bit of the B register is a zero, examine the word in the A register. If it is an operand, increase the contents of the C and L registers by the 12 low-order bits of the operand. If it is a descriptor, replace the contents of the C register by the 15 low-order bits of the descriptor and set the L register to zero. In either case, mark the A and B registers empty.

BBC	01

BRANCH BACKWARD CONDITIONAL. If the low-order bit of the B register is a one, mark the A and B registers empty and terminate the operation. If the low-order bit of the B register is a zero, examine the word in the A register. If it is an operand, decrease the contents of the C and L registers by the 12 low-order bits of the operand. If it is a descriptor, replace the contents of the C register by the 15 low-order bits of the descriptor and set the L register to zero. In either case, mark the A and B registers empty.

BRT	01

BRANCH RETURN. Examine the presence bit of the word in the A register. If it is zero, set the presence bit in the Interrupt Register and terminate the operation. If it is one, reset the S and C registers from the contents of the word in the A register. Set the L register to zero.

Access the word referred to by the S register and restore the R and F registers and the settings of the Mark Stack and the Program Level Flip-Flops from the contents of this word. Decrease the setting of the S register by one and mark the A and B registers empty.

Bit Operators

RFB	01

RESET FLAG BIT. Set the flag bit of the word in the A register to zero, regardless of its previous setting, making it an operand.

SFB	01

SET FLAG BIT. Set the flag bit of the word in the A register to one, regardless of its previous setting, making it a descriptor.

TFB	01

TEST FLAG BIT. Examine the flag bit of the word in the B register. If it is zero, clear the A register and then set the low-order bit to one. If it is a one, clear the A register. In either case, mark the A register full.

nn	DIA	01

DIAL A. Examine the contents of nn, the six high-order bits of the operator. If they are not zero, replace the contents of the G and H registers by nn; otherwise no operation takes place.

nn	DIB	01

DIAL B. Examine the contents of nn, the six high-order bits of the operator. If they are not zero, replace the contents of the K and V registers by nn; otherwise no operation takes place.

RSB	01

RESET SIGN BIT. Set the sign of the operand in the A register to zero, regardless of its previous setting.

SSB	01

SET SIGN BIT. Set the sign of the operand in the A register to one, regardless of its previous setting.

CSB	01

CHANGE SIGN BIT. If the sign of the operand in the A register is positive, change it to negative. If it is negative, change it to positive.

nn	TFR	01

TRANSFER BITS. Transfer bits from the A register to the B register. The number of bits to be transferred is indicated by the six high-order bits of the operator. The G, H, K, and V registers

determine the initial bit positions in the A and B registers. Reset the G, H, K, and V registers to their original setting. Transfer of bits will terminate if either the A or B registers have been exhausted before nn reaches zero. In either case, mark the A register empty.

nn	CFE	01

COMPARE FIELD EQUAL. Compare a field of the A register whose high-order bit is indicated by the G and H registers to a field of the B register whose high-order bit is indicated by the K and V registers. The number of bits to be compared is specified by nn, the six high-order bits of the operator. Comparison of the low-order bit of either register also may terminate the operation. If the result of the comparison is equal, set the A register to one; otherwise, set it to zero. Reset the contents of the B, G, H, K, and V registers to what they contained prior to the operation.

nn	CFL	01

COMPARE FIELD LOW. Compare a field of the A register whose high-order bit is indicated by the G and H registers to a field of the B register whose high-order bit is indicated by the K and V registers. The number of bits to be compared is specified by nn, the six high-order bits of the operator. Comparison of the low-order bit of either register may also terminate the operation. If the value of the specified B register bit is lower, set the A register to one; otherwise, set it to zero. Reset the contents of the B, G, H, K, and V registers to what they contained prior to the operation.

Store Operators

SND	01

STORE NONDESTRUCTIVE. If the A register contains a descriptor, store the contents of the B register at the address specified by the descriptor. If the A register contains an operand, store the contents of the B register at the address specified by modifying the R or F register with portions of the 10 low-order bits of the operand depending on the operating level. In either case, the contents of the B register are retained. Mark the A register empty.

STD	01

STORE DESTRUCTIVE. If the A register contains a descriptor, store the contents of the B register at the address specified by the descriptor. If the A register contains an operand, store the contents of the B register at the address specified by modifying the R or F register contents with portions of the 10 low-order bit of the operand depending on the operating level. In either case, mark the A and B registers empty.

ISN	01

INTEGER STORE NONDESTRUCTIVE. Convert the contents of the B register to an integer. If the A register contains a descriptor, store the contents of the B register at the address specified by the descriptor. If the A register contains an operand, store the contents of the B register at the address specified by modifying the R or F register contents with portions of the 10 low-order bits of the operand depending on the operating level. In either case, the contents of the B register are retained. Mark the A register empty.

ISD	01

INTEGER STORE DESTRUCTIVE. Convert the contents of the B register to an integer. If the A register contains a descriptor, store the contents of the B register at the address specified by the descriptor. If the A register contains an operand, store the contents of the B register at the address specified by modifying the R or F register with portions of the 10 low-order bits of the operand depending on the operating level. In either case, mark the A and B registers empty.

CIN	01

CONDITIONAL INTEGER STORE NONDESTRUCTIVE. If the integer bit position of the word in the A register is on, convert the contents of the B register to an integer. If the A register contains a descriptor, store the contents of the B register at the address specified by the descriptor. If the A register contains an operand, store the contents of the B register at the address specified by modifying the R or F register with the 10 low-order bits of the operand depending on the operating level. In either case, the contents of the B register are retained. Mark the A register empty.

CID	01

CONDITIONAL INTEGER STORE DESTRUCTIVE. If the integer bit position of the word in the

A register is on, convert the contents of the B register to an iteger. If the A register contains a descriptor, store the contents of the B register at the address specified by the descriptor. If the A register contains an operand, store the contents of the B register at the address specified by modifying the R or F register contents with the 10 low-order bits of the operand depending on the operating level. In either case, mark the A and B registers empty.

Miscellaneous Operators

PRL	01

PROGRAM RELEASE. Examine the flag bit of the word in the A register.

If it is a one and

the presence bit is zero, set the presence bit in the Interrupt Register and terminate the operation.

the presence bit is a one, obtain the word addressed by the 15 low-order bits of the A register and place it in the A register. Set the presence bit of the word in the A register to zero and store it back in its original location.

If it is a zero, obtain the word at the address specified by modifying the R or F register with portions of the 10 low-order bits of the word in the A register, depending on the operating level. Set the presence bit of the word in the A register to zero and return the word to its original location.

Examine the continuity bit of the word obtained from memory. If it is a one, set the continuity bit in the Interrupt Register; if it is a zero, set the program release bit in the Interrupt Register. In either case, mark the A register empty.

If the processor is in the Normal State the address of the word obtained from memory is stored in a specific location.

COC	01

CONSTRUCT OPERAND CALL. Exchange the contents of A and B registers. Turn on the flag bit and the identification bits of the word in the A register making it a Data Descriptor. If its size field is non-zero, make the contents of the B register an integer. Compare it against the size field of the word in the A register and add the 10 low-order bits of the integer to the descriptor address in the A register. Mark the B register empty. Subsequent action of this syllable is identical to that of an Operand Call syllable after having caused a word to be read from memory.

CDC	01

CONSTRUCT DESCRIPTOR CALL. Exchange the contents of the A and B registers. Turn on the flag bit and the identification bits of the word in the A register making it a Data Descriptor. If its size field is non-zero, make the contents of the B register an integer. Compare the integer against the size field in the A register and add the 10 low-order bits of the integer to the descriptor address in the A register. Mark the B register empty. Subsequent action of this syllable is identical to that of a Descriptor Call syllable after having caused a word to be read from memory.

COM	01

COMMUNICATION OPERATOR. If operating in Processor A, store the contents of the word at the top of the stack in a specific location. Delete the word from the stack. Set the communication bit in the interrupt register.

LOD	01

LOAD OPERATOR. If a descriptor is in the A register, replace it with the word located by the address in the descriptor. If an operand is in the A register, obtain the word at the address specified by modifying the contents of the R register with portions of the 10 low-order bits of the operand depending on the operating level.

INX	01

INDEX. Add the 15 low-order bits of the B register to the 15 low-order bits of the A register and suppress overflow. Mark the B register empty.

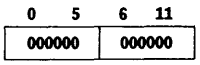
CHARACTER MODE

In this mode, data is normally treated as 6-bit alphanumeric characters. However, word and bit operations can also be performed. The basic format for data is as follows:

BA 8421	BA 8421	BA 8421	BA 8421	BA 8421	BA 8421	BA 8421	BA 8421
00 0000	00 0000	00 0000	00 0000	00 0000	00 0000	00 0000	00 0000
CHAR 1	CHAR 2	CHAR 3	CHAR 4	CHAR 5	CHAR 6	CHAR 7	CHAR 8

The concept for manipulation of information between source and destination strings was explained in Section 3. For all operations in this mode, transfer of information from memory to the source and destination registers, and back to memory from the destination register is performed automatically. The description of the operators given here assumes that these memory transfers are taking place on a continuing basis.

The program string in the Character Mode consists of a series of 12-bit syllables. The format for these syllables is as follows:



BITS	USE
0-5	REPEAT FIELD
6-11	OPERATOR CODE

Syllable Description

The description of the individual operator syllables in the Character Mode is presented in the following paragraphs.

OPERATORS

Transfer Operators

These operators are used to transfer information from one area in memory to another. In general, transfer operations proceed from high-order to low-order within a word and to successively higher addresses by word. Transfers may occur from any character position within a word to any character position in another word.

000000	TWD

TRANSFER WORDS. Align the source string and destination string to the beginning of the next word unless they are already aligned. Transfer the next successive words from the source string to the destination string. The repeat field specifies the number of words to be moved. If this field is zero, only alignment occurs. At the completion of this operation the M and S registers specify the next word in sequence.

000000	TRP

TRANSFER PROGRAM CHARACTERS. Transfer successive characters from the program segment to the successive positions in the destination string, beginning with the next group of six bits adjacent to this syllable. The repeat field specifies the number of characters to be moved. If the number of characters is odd, the first group of six bits is ignored and the next group is transferred.

000000	TRS

TRANSFER SOURCE CHARACTERS. Transfer the next successive characters from the source string to successive positions in the destination string. The repeat field specifies the number of characters to be moved.

000000	TNU

TRANSFER NUMERIC. Transfer the numeric bits of successive characters in the source string to successive character positions in the destination string. If the source string field is negative, set the True/False toggle to true. The repeat field specifies the number of characters whose numeric bits are to be transferred. Set the destination zone bits of these characters to zero.

000000	TZN

TRANSFER ZONE. Transfer the zone bits of successive characters in the source string to successive character positions in the destination string. The repeat field specifies the number of characters whose zone bits are to be transferred. The numeric bits of the characters in the destination string are unaltered.

Test Operators

These operators provide the ability to test a character or bit in the source area against a predetermined character or bit in the program string. These operations do not cause an advancement in the source area, thus enabling repeated tests of a character.

000000	TFA

TEST FOR ALPHANUMERIC. Compare the character in the repeat field with the next character

in the source string. If the source string character is greater than or equal to the character in the repeat field, and it is other than a multiply or not equal character, set the True/False toggle to one; otherwise, set it to zero. Do not advance the M and G registers.

000000	TEQ

TEST FOR EQUAL. Compare the character in the repeat field with the character in the source string. If the source character is equal to the repeat field character, set the True/False toggle to one; otherwise, set it to zero. Do not advance the G and M registers.

000000	TNE

TEST FOR NOT EQUAL. Compare the character in the repeat field with the character in the source string. If the source character is not equal to the repeat field character, set the True/False toggle to one; otherwise, set this toggle to zero. Do not advance the G or M registers.

000000	TGR

TEST FOR GREATER. Compare the character in the repeat field with the character in the source string. If the source character is greater than the repeat field character, set the True/False toggle to one; otherwise, set this toggle to zero. Do not advance the G or M registers.

000000	TEL

TEST FOR EQUAL OR LESS. Compare the character in the repeat field with the character in the source string. If the source character is equal to or less than the repeat field character, set the True/False toggle to one; otherwise, set this toggle to zero. Do not advance the G or M registers.

000000	TLS

TEST FOR LESS. Compare the character in the repeat field with the character in the source string. If the source character is less than the repeat field character, set the True/False toggle to one; otherwise, set this toggle to zero. Do not advance the G or M registers.

000000	TGE

TEST FOR GREATER OR EQUAL. Compare the character in the repeat field with the character in the source string. If the source character is equal to or greater than the repeat field character, set the True/False toggle to one; otherwise, set this toggle to zero. Do not advance the G or M registers.

000000	TBT

TEST BIT. Compare the addressed bit in the source string with the low-order bit of the repeat field. If they are equal, set the True/False toggle to one; if they are unequal set it to zero. Do not advance the M, G and H registers.

Comparison Operators

These operators are used for comparing two identical length fields of alphanumeric characters. Fields may start at any position within a word; word boundaries are ignored. Although the result of the comparison may be known before all characters of the fields have been compared, the address registers are advanced the full amount.

000000	CNE

COMPARE FOR NOT EQUAL. Compare the next successive characters in the source string with the next successive characters in the destination string. The number of characters to compare is specified by the repeat field. If the source string is not equal to the destination string, set the True/False toggle to one; otherwise, set this toggle to zero.

000000	CMG

COMPARE FOR GREATER. Compare the next successive characters in the source string with the next successive characters in the destination string. The number of characters to compare is specified by the repeat field. If the source string is greater than the destination string, set the True/False toggle to one; otherwise, set this toggle to zero.

000000	CLS

COMPARE FOR LESS. Compare the next successive characters in the source string with the next successive characters in the destination string. The number of characters to compare is specified by

the repeat field. If the source string is less than the destination string, set the True/False toggle to one; otherwise, set this toggle to zero.

000000	CGE

COMPARE FOR GREATER OR EQUAL. Compare the next successive characters in the source string with the next successive characters in the destination string. The number of characters to compare is specified by the repeat field. If the source string is equal to or greater than the destination string, set the True/False toggle to one; otherwise, set this toggle to zero.

000000	CEQ

COMPARE FOR EQUAL. Compare the next successive characters in the source string with the next successive characters in the destination string. The number of characters to compare is specified by the repeat field. If the source string is equal to the destination string, set the True/False toggle to one; otherwise, set this toggle to zero.

000000	CEL

COMPARE FOR EQUAL OR LESS. Compare the next successive characters in the source string with the next successive characters in the destination string. The number of characters to compare is specified by the repeat field. If the source string is equal to or less than the destination string, set the True/False toggle to one; otherwise, set this toggle to zero.

Jump Operators

These operators are used to adjust the C and L registers to provide branching in the program string or for executing repeated program strings.

000000	JFU

JUMP FORWARD UNCONDITIONAL. Increase the contents of the C and L registers by the contents of the repeat field. Prior to this operation the C and L registers contain the address of the next syllable in sequence.

000000	JRU

JUMP REVERSE UNCONDITIONAL. Decrease the contents of the C and L registers by the contents of the repeat field. Prior to this operation the C and L registers contain the address of the next syllable in sequence.

000000	JFC

JUMP FORWARD CONDITIONAL. If the True/False toggle is false, increase the contents of the C and L registers by the contents of the repeat field. Prior to this operation the C and L registers contain the address of the next syllable in sequence. If the toggle is true, the next syllable in sequence is fetched. In either case the True/False toggle remains unchanged.

000000	JRC

JUMP REVERSE CONDITIONAL. If the True/False toggle is false, decrease the contents of the C and L registers by the contents of the repeat field. Prior to this operation the C and L registers contain the address of the next syllable in sequence. If the toggle is true, the next syllable in sequence is fetched. In either case, the True/False toggle remains unchanged.

000000	BLP

BEGIN LOOP. Execute the following series of syllables which is terminated by an END LOOP syllable. The repeat field specifies the number of times the loop will be executed. If the repeat field is zero or one, execute the loop one time.

000000	ELP

END LOOP. Identifies the end of a program loop. The repeat field is irrelevant. If the loop has been executed the specified number of times, execute the next syllable following the END LOOP syllable. Otherwise return to the syllable beginning the loop.

000000	JLP

JUMP OUT LOOP. Jump forward over the number of syllables specified by the repeat field and delete the count of the repetitions associated with the program string.

000000	JLC

JUMP OUT LOOP CONDITIONAL. If the True/False toggle is false, jump forward over the number of syllables specified by the repeat field and delete count of repetitions associated with loop; otherwise, continue in sequence.

Skip Operators

These operators are used to set the address registers associated with the source and destination character strings.

000000	SFS

SKIP FORWARD SOURCE. Skip forward the number of successive source string characters specified by the repeat field. The characters skipped over remain in the source string.

000000	SRS

SKIP REVERSE SOURCE. Skip backward the number of successive source string characters specified by the repeat field. The characters skipped over remain in the source string.

000000	SRD

SKIP REVERSE DESTINATION. Skip backward the number of successive destination string characters specified by the repeat field. The characters skipped over remain in the destination string.

000000	SFD

SKIP FORWARD DESTINATION. Skip forward the number of successive destination string characters specified by the repeat field. The characters skipped over remain in the destination string.

000000	SBS

SKIP BIT SOURCE. Skip successive bits in the source string. The repeat field specifies the number of bits to skip.

000000	SBD

SKIP BIT DESTINATION. Skip successive bits in the destination string. The repeat field specifies the number of bits to skip.

Address Operators

These operators are used for storing addresses in the stack, calling addresses from the stack, and addressing locations in the stack. In addition, it is possible to obtain source and destination addresses from the source and destination character strings.

000000	TSA

TRANSFER SOURCE ADDRESS. Set the source string address by loading the M and G registers from the next three characters in the source string. Place the three most significant bits in the G register. Set the H register to zero.

000000	TDA

TRANSFER DESTINATION ADDRESS. Set the destination string address by loading the S and K registers from the next three characters in the destination string. Place the three most significant bits in the K register. Set the V register to zero.

000000	SES

SET SOURCE ADDRESS. Obtain an address by decreasing the contents of the F register by the contents of the repeat field. Set the M register with the address. Set the G and H registers to zero. Retain the original contents of the F register.

000000	SED

SET DESTINATION ADDRESS. Obtain an address by decreasing the contents of the F register by the contents of the repeat field. Set the S register with this address. Set the K and V registers to zero. Retain the original contents of the F register.

000000	SSA

STORE SOURCE ADDRESS. Store the source string address, as contained in the M and G registers, in the stack at the address formed by decreasing the contents of the F register by the contents of the repeat field. Retain the original contents of the F register. Set the flag bit of the word at that address to zero.

000000	SDA

STORE DESTINATION ADDRESS. Store the destination string address, as contained in the S and K registers, in the stack at the address formed by decreasing the contents of the F register by the contents of the repeat field. Retain the original contents of the F register. Set the flag bit of the word at that address to zero.

000000	SCA

STORE CONTROL ADDRESS. Store the contents of the C and L registers in the stack at the address formed by decreasing the contents of the F register by the contents of the repeat field. Set the flag bit of the word at that address to zero. Retain the original contents of the F register.

000000	RCA

RECALL CONTROL ADDRESS. Obtain a word from the stack at the address formed by decreasing the contents of the F register by the contents of the repeat field. Retain the original contents of the F register. If the word is a descriptor, reload the C register from the contents of the word and set the L register to zero. If the word is an operand, reload the C and L registers from the contents of this word. Advance the L register by one to specify the next syllable in sequence.

000000	RSA

RECALL SOURCE ADDRESS. Obtain a word from the stack at the address formed by decreasing the contents of the F register by the contents of the repeat field. Retain the original contents of the F register. If the word is a descriptor reload the M register from the contents of this word and set the G and H registers to zero. If the word is an operand reload the M and G registers from the contents of this word and set the H register to zero.

000000	RDA

RECALL DESTINATION ADDRESS. Obtain a word from the stack at the address formed by decreasing the contents of the F register by the contents of the repeat field. Retain the original contents of the F register. If the word is a descriptor, reload the S register from the contents of this word and clear the K and V registers. If the word is an operand, reload the S and K registers from the contents of this word and set the V register to zero.

Arithmetic Operators

000000	FAD

FIELD ADD. Add algebraically a field of successive source string characters to a field of successive destination string characters. The repeat field specifies the field length, in characters, for both strings. Ignore zone bits except those of the low-order character in each field which contain the sign. If there is field overflow, set the True/False toggle to true; otherwise, set it to false.

000000	FSU

FIELD SUBTRACT. Subtract algebraically a field of successive source string characters from a field of successive destination string characters. The repeat field specifies the field length, in characters, for both strings. Ignore zone bits except those of the low-order character in each field which contain the sign. If there is field overflow, set the True/False toggle to true; otherwise, set it to false.

Conversion Operators

000000	OCV

OUTPUT CONVERT. Align the source string address registers to the beginning of the next word unless already aligned. Convert the mantissa of the octal word in the source string to a field of successive decimal digits in the destination string. The field length of the destination string is specified by the repeat field up to a maximum of eight decimal digits. Translate the sign of the octal operand and store it in the zone bits over the low-order digit in the decimal field. If the number of digits in the result is greater than the field length specified in the repeat field, set the True/False toggle to false; otherwise, set it to true.

000000	ICV

INPUT CONVERT. Align the destination string address registers to the beginning of the next word if alignment is necessary. Convert the decimal value of a field of successive characters in the source string to a one word octal integer and store it in the destination string. The field length of the source string is specified by the repeat field, up to a maxi-

mum of eight decimal characters. Translate the zone bits of the low-order decimal character and store it as the sign of the octal word.

Miscellaneous Operators

000000	SET

SET TALLY. Set the R register to the value specified in the repeat field.

000000	INT

INCREASE TALLY. Increase the setting of the R register by the increment specified in the repeat field. Ignore any overflow of the R register.

000000	STT

STORE TALLY. Store the contents of the R register in the stack at the location specified by the repeat field, relative to the address in the F register.

000000	SEB

SET BIT. Set successive bits in the destination string to one. The repeat field specifies the number of bits to be set.

000000	REB

RESET BIT. Set successive bits in the destination string to zero. The repeat field specifies the number of bits to be reset.

000000	CRF

CALL REPEAT FIELD. Obtain a word from the stack at the address formed by decreasing the contents of the F register by the contents of the repeat field. Examine the six low-order bits of the word. If they are not zero, use them as the repeat field for the next syllable. If they are zero, use the repeat field of the next syllable and treat it as a Jump Forward Unconditional operator.

000000	ECM

EXIT CHARACTER MODE. Obtain the Return control word from the stack and set the C, L, G, H, K, and V registers from their respective fields of the control word. Set the S register to the contents of the F register field of the Return control word. Obtain the Mark Stack control word from the location specified by the S register. Set the R and F registers and the Mark Stack and Program Level Flip-Flops from their respective fields in the Mark Stack control word. Decrease the S register contents by one. Place the processor in the Word Mode.

CONTROL STATE

Entry to this state from the Normal State is made when any bit in the interrupt register is turned on as explained in Section 3. At entry time all pertinent registers and flip-flops are automatically stored in contiguous cells of the stack.

When a processor is in the Control State it may use all of the operators used in the Normal State plus the following operators. These operators would function as no-ops if encountered when in the Normal State. Note that these operators have the same format as operators in the Word Mode.

SYLLABLE DESCRIPTION

Halt P2

HLB	01

Cause Processor B to store its registers in its stack and idle after the completion of processing the current syllable.

Initiate I/O

IIO	01

Access the word in the A register and store it in a specific location. Mark the A register empty. Send an Initiate I/O signal to Central Control for selection of an I/O channel.

Initiate P1

INA	01

Place the 15 low-order bits of the A register in the S register. Set mode. Set all pertinent registers of Processor A from stack and exit from the Control State.

Initiate P2

INB	01

Access the word in the A register and store it in a specific location. Mark the A register empty. Send an Initiate P2 signal to Central Control to activate Processor B.

I/O Release

IOR	01

If the word in register A is a descriptor and the presence bit is one, the contents of the location addressed by the 15 lower bits of the A register are placed in the A register. The presence bit of the word in the A register is set to one and the word stored back at the location initially addressed. A register is set to empty.

If the word in register A is an operand, the ten lower bits of the word in the A register are used as a relative address. Indexing takes place with R or F registers. The word addressed is placed in A register, presence bit set to zero and stored back in original location.

Interrogate Interrupt

INI	01

Interrogate the Interrupt Register for interrupt bits. If an interrupt bit is on, transfer control to the memory location corresponding to that bit, reset the bit in the Interrupt Register, clear the L register and set the S register to a specific setting. If no interrupt bits are on, control continues in sequence.

Read Timer

RTM	01

Place the six bits of the timer setting into the A register as an integer.

APPENDIX A

PROGRAM OPERATORS—FUNCTIONAL LISTING WORD MODE

	Arithmetic Operators	Page
ADD	Add	5-2
SUB	Subtract	5-2
MUL	Multiply	5-2
DIV	Divide	5-2
IDV	Integer Divide	5-3
RDV	Remainder Divide	5-3
ADL	Add Double Length	5-3
SDL	Subtract Double Length	5-3
MDL	Multiply Double Length	5-3
DDL	Divide Double Length	5-3
	Logical Operators	
LND	Logical And	5-3
LOR	Logical Or	5-3
LQV	Logical Equivalence	5-3
LNG	Logical Negate	5-4
	Relational Operators	
GTR	B Greater Than A	5-4
GEQ	B Greater Than or Equal to A	5-4
EQL	B Equal to A	5-4
LEQ	B Less Than or Equal to A	5-4
LSS	B Less Than A	5-4
NEQ	B Not Equal to A	5-4
	Branch Operators	
BFU	Branch Forward Unconditional	5-5
BBU	Branch Backward Unconditional	5-5
BFC	Branch Forward Conditional	5-6
BBC	Branch Backward Conditional	5-6
BRT	Branch Return	5-6
	Store Operators	
STD	Store Destructive	5-7
SND	Store Nondestructive	5-7
ISD	Integer Store Destructive	5-7
ISN	Integer Store Nondestructive	5-7
CID	Conditional Integer Store Destructive	5-7
CIN	Conditional Integer Store Nondestructive	5-7

APPENDIX A (Continued)

	Bit Operators	Page
DIA	Dial A	5-6
DIB	Dial B	5-6
TFR	Transfer Bits	5-6
CFE	Compare Field Equal	5-7
CFL	Compare Field Low	5-7
RFB	Reset Flag Bit	5-6
SFB	Set Flag Bit	5-6
TFB	Test Flag Bit	5-6
RSB	Reset Sign Bit	5-6
SSB	Set Sign Bit	5-6
CSB	Change Sign Bit	5-6
	Subroutine Operators	
MKS	Mark Stack	5-4
XIT	Exit	5-4
RNO	Return Normal	5-5
RSP	Return Special	5-5
	Stack Operators	
XCH	Exchange	5-3
DUP	Duplicate	5-3
	Miscellaneous Operators	
LOD	Load Operator	5-8
INX	Index	5-8
COC	Construct Operand Call	5-8
CDC	Construct Descriptor Call	5-8
COM	Communication Operator	5-8
PRL	Program Release	5-8

CHARACTER MODE

	Transfer Operators	
TRS	Transfer Source Characters	5-9
TRP	Transfer Program Characters	5-9
TZN	Transfer Zone	5-9
TNU	Transfer Numeric	5-9
TWD	Transfer Words	5-9
	Test Operators	
TGR	Test for Greater	5-10
TGE	Test for Greater or Equal	5-10
TEQ	Test for Equal	5-10
TEL	Test for Equal or Less	5-10
TLS	Test for Less	5-10
TNE	Test for Not Equal	5-10
TFA	Test for Alphanumeric	5-9
TBT	Test Bit	5-10

APPENDIX A (Continued)

	Comparison Operators	Page
CMG	Compare for Greater	5-10
CGE	Compare for Greater or Equal	5-11
CEQ	Compare for Equal	5-11
CEL	Compare for Equal or Less	5-11
CLS	Compare for Less	5-10
CNE	Compare for Not Equal	5-10
	Jump Operators	
JFU	Jump Forward Unconditional	5-11
JRU	Jump Reverse Unconditional	5-11
JFC	Jump Forward Conditional	5-11
JRC	Jump Reverse Conditional	5-11
BLP	Begin Loop	5-11
ELP	End Loop	5-11
JLP	Jump Out Loop	5-11
JLC	Jump Out Loop Conditional	5-12
	Skip Operators	
SFS	Skip Forward Source	5-12
SRS	Skip Reverse Source	5-12
SFD	Skip Forward Destination	5-12
SRD	Skip Reverse Destination	5-12
SBS	Skip Bit Source	5-12
SBD	Skip Bit Destination	5-12
	Address Operators	
SSA	Store Source Address	5-12
SDA	Store Destination Address	5-13
SCA	Store Control Address	5-13
RSA	Recall Source Address	5-13
RDA	Recall Destination Address	5-13
RCA	Recall Control Address	5-13
SES	Set Source Address	5-12
SED	Set Destination Address	5-12
TSA	Transfer Source Address	5-12
TDA	Transfer Destination Address	5-12
	Arithmetic Operators	
FAD	Field Add	5-13
FSU	Field Subtract	5-13
	Conversion Operators	
ICV	Input Convert	5-13
OCV	Output Convert	5-13

APPENDIX A (Continued)

	Miscellaneous Operators	Page
SET	Set Tally	5-14
INT	Increase Tally	5-14
STT	Store Tally	5-14
REB	Reset Bit	5-14
SEB	Set Bit	5-14
CRF	Call Repeat Field	5-14
ECM	Exit Character Mode	5-14

CONTROL STATE

INI	Interrogate Interrupt	5-15
IOR	I/O Release	5-15
IIO	Initiate I/O	5-14
INA	Initiate P1	5-14
INB	Initiate P2	5-15
HLB	Halt P2	5-14
RTM	Read Timer	5-15

APPENDIX B

Program Operators—Alphabetical Listing by Mnemonic Code

MNEMONIC	MODE	OPERATOR	PAGE
ADD	W	Add	5-2
ADL	W	Add Double Length	5-3
BBC	W	Branch Backward Conditional	5-6
BBU	W	Branch Backward Unconditional	5-5
BFC	W	Branch Forward Conditional	5-6
BFU	W	Branch Forward Unconditional	5-5
BLP	C	Begin Loop	5-11
BRT	W	Branch Return	5-6
CDC	W	Construct Descriptor Call	5-8
CEL	C	Compare for Equal or Less	5-11
CEQ	C	Compare for Equal	5-4
CFE	W	Compare Field Equal	5-7
CFL	W	Compare Field Low	5-7
CGE	C	Compare for Greater or Equal	5-11
CID	W	Conditional Integer Store Destructive	5-7
CIN	W	Conditional Integer Store Nondestructive	5-7
CLS	C	Compare for Less	5-10
CMG	C	Compare for Greater	5-10
CNE	C	Compare for Not Equal	5-10
COC	W	Construct Operand Call	5-8
COM	W	Communication Operator	5-8
CRF	C	Call Repeat Field	5-14
CSB	W	Change Sign Bit	5-6
DDL	W	Divide Double Length	5-3
DIA	W	Dial A	5-6
DIB	W	Dial B	5-6
DIV	W	Divide	5-2
DUP	W	Duplicate	5-3
ECM	C	Exit Character Mode	5-14
ELP	C	End Loop	5-11
EQL	W	B Equal to A	5-4
FAD	C	Field Add	5-13
FSU	C	Field Subtract	5-13
GEQ	W	B Greater Than or Equal to A	5-4
GTR	W	B Greater Than A	5-4
ICV	C	Input Convert	5-13
IDV	W	Integer Divide	5-3
INT	C	Increase Tally	5-14
INX	W	Index	5-8
ISD	W	Integer Store Destructive	5-7
ISN	W	Integer Store Nondestructive	5-7
JFC	C	Jump Forward Conditional	5-11
JFU	C	Jump Forward Unconditional	5-11
JLC	C	Jump Out Loop Conditional	5-12
JLP	C	Jump Out Loop	5-11
JRC	C	Jump Reverse Conditional	5-11
JRU	C	Jump Reverse Unconditional	5-11

APPENDIX B (Continued)

MNEMONIC	MODE	OPERATOR	PAGE
LEQ	W	B Less Than or Equal to A	5-4
LND	W	Logical And	5-3
LNG	W	Logical Negate	5-4
LOD	W	Load Operator	5-8
LOR	W	Logical Or	5-3
LQV	W	Logical Equivalence	5-3
LSS	W	B Less Than A	5-4
MDL	W	Multiply Double Length	5-3
MKS	W	Mark Stack	5-4
MUL	W	Multiply	5-2
NEQ	W	B Not Equal to A	5-4
OCV	C	Output Convert	5-13
PRL	W	Program Release	5-8
RCA	C	Recall Control Address	5-13
RDA	C	Recall Destination Address	5-13
RDV	W	Remainder Divide	5-3
REB	C	Reset Bit	5-14
RFB	W	Reset Flag Bit	5-6
RNO	W	Return Normal	5-5
RSA	C	Recall Source Address	5-13
RSB	W	Reset Sign Bit	5-6
RSP	W	Return Special	5-5
SBD	C	Skip Bit Destination	5-12
SBS	C	Skip Bit Source	5-12
SCA	C	Store Control Address	5-13
SDA	C	Store Destination Address	5-13
SDL	W	Subtract Double Length	5-3
SEB	C	Set Bit	5-14
SED	C	Set Destination Address	5-12
SES	C	Set Source Address	5-12
SET	C	Set Tally	5-14
SFB	W	Set Flag Bit	5-6
SFD	C	Skip Forward Destination	5-12
SFS	C	Skip Forward Source	5-12
SND	W	Store Nondestructive	5-7
SRD	C	Skip Reverse Destination	5-12
SRS	C	Skip Reverse Source	5-12
SSA	C	Store Source Address	5-12
SSB	W	Set Sign Bit	5-6
STD	W	Store Destructive	5-7
STT	C	Store Tally	5-14
SUB	W	Subtract	5-2
TBT	C	Test Bit	5-10
TDA	C	Transfer Destination Address	5-12
TEL	C	Test For Equal or Less	5-10
TEQ	C	Test for Equal	5-10
TFA	C	Test for Alphanumeric	5-9
TFB	W	Test Flag Bit	5-6

APPENDIX B (Continued)

MNEMONIC	MODE	OPERATOR	PAGE
TFR	W	Transfer Bits	5-6
TGE	C	Test for Greater or Equal	5-10
TGR	C	Test for Greater	5-10
TLS	C	Test for Less	5-10
TNE	C	Test for Not Equal	5-10
TNU	C	Transfer Numeric	5-9
TRP	C	Transfer Program Characters	5-9
TRS	C	Transfer Source Characters	5-9
TSA	C	Transfer Source Address	5-12
TWD	C	Transfer Words	5-9
TZN	C	Transfer Zone	5-9
XCH	W	Exchange	5-3
XIT	W	Exit	5-4

Burroughs Corporation

DETROIT 32, MICHIGAN

Offices in Principal Cities

In Canada: Burroughs Business Machines Ltd., Toronto, Ontario

LITHO IN U. S. A.

3-5-6

5000—21005