1. Overview

1.01 This practice describes the 1B Processor Assembly Language and gives detailed information of instructions used by the processor. Detailed information is provided to enable a wide range of users to obtain an overall exposure to how the 1B Processor Assembly Language operates. This information is a broad, high-level description of the 1B Processor Assembly Language.

1.02 Whenever this practice is reissued, the reason for reissue will be stated in this paragraph.

1.03 This practice does not contain safety labels.

1.04 AT&T welcomes your comments on this practice. Your comments will aid us in improving the quality and usefulness of AT&T documentation. Please use the Feedback Form [REF. 12] [mail or FAX (1-910-727-3043)] or call the AT&T Documentation Comment Hot-Line Service (1-800-334-0404 or 1-910-727-6681 in North Carolina).

1.05 Additional copies of this practice, associated appendixes, and all referenced documentation may be ordered from the AT&T Customer Information Center. To order copies by mail, AT&T employees should mail Form IND 1-80.80, which is available from the AT&T Customer Information Center, to the following address:

AT&T Customer Information Center
Attention: Order Entry Department
2855 N. Franklin Road
P. O. Box 19901
Indianapolis, Indiana 46219-1999

Orders can also be placed by phone Monday through Friday by calling one of the following numbers:

Within the United States: 1-800-432-6600

FAX: 317-322-6484

Bell Operating Companies must process orders through their company documentation coordinator.

Federal Government orders must be processed through CIC.

1.06 This document was developed by the AT&T Customer Information Development and Business Translations Organization.

2. System Organization

2.1 1B Processor

2.1.1 General

2.01 The 1B Processor is a real-time, stored program control system which operates in a real-time environment. The 1B Processor's hardware and software are configured so the processor can function independently from its peripheral networks. The 1B Processor's hardware and software are configured so the processor can function independently from its peripheral switching network. Private access buses connect each community with the CC (Figure -1).

A. 1B Storage Community Two of the communities provide the primary memory for CC. These communities are:

Program Store (PS)
In general, the program store memory contains the program instructions executed by CC and configuration data for both the 1B Processor and the 4ESS(TM) Switch.

Call Store (CS)
In general, the call store memory contains semi-permanent and transient data which may be interrogated and changed during program execution.

B. Auxiliary Unit consists of units which are capable of communicating with the program store and call store communities via CC bus access circuitry. The units control access to bulk memory (file stores and tape units) and data links.

C. Peripheral Unit consists of the switching equipment and associated access control circuitry. It includes units and associated hardware and various types of network, scanner and signal distribution units, depending on the application of the 1B Processor.

2.2 Overview of Instruction Processing

2.2.1 General

2.02 There are three main classes of instructions used by the 1B processor. The following paragraphs provide a brief description of each class and an explanation of its function.

2.03 General Processing -- Uses the program execution facilities (including transfer instructions). The execution time for these instructions varies depending upon the instruction length (one or two 24-bit words). Most general processing instructions require 1 to 5 CC operating cycles (166.67 nanoseconds per CC operating cycle).

2.04 Peripheral -- Used by CC to control 4ESS(TM) Switch peripheral units. These instructions are also used to access input/output (I/O) channels. A peripheral instruction can require from 11 to 136 CC operating cycles (166.67 nanoseconds per CC operating cycle).

2.05 Maintenance -- These instructions provide software control for testing various system functions. They are also used to implement control read and write operations on the 1B Processor memory communities.

2.2.2 Fetching Instructions
2.2.2.1 General

2.06 The CC instruction fetching mechanism presents program instructions to the CC execution facilities in an orderly manner. This function is complicated by factors such as: different classes of instructions, one-word and two-word instructions, various instruction formats, and different instruction fetch routines used to obtain data from program store and call store.

2.07 Instruction addresses are generated in a CC program address register. Instructions from call store are fetched one word at a time, while instructions from program store are fetched two words at a time. Therefore, the instruction address register must have the capability of being incremented in steps of one or two addresses. Instructions are stored in program store in a mixed format to efficiently use all available storage locations. This means that the contents of two adjacent program store data locations, which are fetched by CC, may contain two short instructions, one long instruction, one

short instruction and the first half of a long instruction, two halves of a long instruction, or the second half of a long instruction and a short instruction. An example follows:

PS address n and n+1 (short -- short)

PS address n+2 and n+3 (long 1 -- long 2)

PS address n+4 and n+5 (short -- long 1)

PS address n+6 and n+7 (long 2 -- long 1)

PS address n+8 and n+9 (long 2 -- short).

The CC instruction fetching queue consists of the fetch logic and five data registers. The instruction fetch queue allows the instruction fetching process to get ahead of the indexing/execution process. This queue detects the various formats and appropriately assembles the instructions for presentation to the instruction execution facility.

2.2.2.2 Instruction Decoding and Execution

2.08 The instruction decoding and execution sequence is illustrated in Figure -2. This illustration shows the sequence of events for an instruction which accomplishes the following:

- o Fetches data from a 1B Processor memory community
- o Adds this data to the existing contents of a general processing register (register one)
- o Writes the result in a second general processing register (register two).

Decoding of the instruction begins at the start of the first CC cycle shown in Figure -2. The add instruction with a memory reference is decoded, and the memory data address is generated. After the instruction is decoded, the data currently in register one is moved to the Arithmetic Logic Unit (ALU). As this transfer takes place, the address, generated as a result of instruction decoding, is further decoded to determine whether the address is for call store, program store, or an auxiliary unit (MMIO). When this address is decoded, the appropriate memory community is accessed. The data is received from memory

and applied to the ALU late in the second CC cycle. The ALU performs the required addition and inserts the result into the second register during the early portion of the third CC cycle. Also during the third CC cycle, the CC begins processing the next instruction to be executed.

2.09 This execution sequence will vary greatly if the data is to be obtained from an auxiliary unit; that is, magnetic tape or disk. In this case, an MMIO request is made via the appropriate software mechanism to transfer the required data from the auxiliary unit to call store. The CC continues operation by executing other instructions until the requested data has been transferred to call store. In the previous example, the completion of this particular instruction would be delayed by an indeterminate interval.

2.2.3 Instructions Involving Accessing of Data in Memory

2.10 Many instructions involve the accessing of data in memory. This data is loaded from, or stored into, data memory. The primary source for such data is call store memory, although all access instructions are capable of addressing any memory location within the address spectrum. (The address spectrum includes all valid call store, program store, auxiliary unit, active CC, standby CC, and internal CC addresses).

2.2.4 Internal 1B Processor Interfaces and Communication

2.2.4.1 Memory Community Interfaces

2.11 The duplicated CCs communicate with the Program Stores, Call Stores, and the Auxiliary Unit communities over four bus systems:

- Program Store Bus (PSB)
- Call Store Bus (CSB)
- Interface Bus (IFB)
- Auxiliary Unit Bus (AUB)

2.2.5 Processing Registers

2.12 The CC processing registers are designated as the F, G, J,

K, L, T, X, Y, and Z registers. Each of these registers may contain one 32-bit word. The general purpose index registers are F,G, K, X, Y, and Z which can be specified in any instruction requiring a working register.

2.13 The L and J registers have some unique functions. Therefore, the use of them is subject to certain restrictions. The use of the L register is predefined during masking operations. Therefore, during masking operations the L-register can not be used for any other operation. The register receives data from the peripheral unit reply bus. This restricts the usage

2.2.6 Logical and Arithmetic Functions

2.14 The logical and arithmetic functions carried out in the execution of an instruction are performed in the operation circuitry. These functions may include one or a combination of the following operations:

- Add
- Compare
- Logical product (AND)
- Logical union (OR)
- Exclusive union (Exclusive OR)
- Rotate
- Shift
- Complement
- Rightmost-one-detection
- Insertion masking
- Product masking

These logical and arithmetic operations may have one or two 24-bit arguments (operands) that can come from a memory location or a data field of the instruction and/or a CC register.

On the execution of instructions that send the result of an operation to a CC register by way of the MB, the two control flip-flops (CFs) are set to reflect the sign and homogeneity of the result. The sign flip-flop is set to the state of the sign bit (bit 23) of the result (0 if positive or 1 if negative). The homogeneity flip-flop is set to a 1 if the result is homogeneous (all 24 bits equal to 0 or all 24 bits equal to 1) and to a 0 if the result is not homogeneous. All compare instructions cause the CFs to be set to the result of the comparison even though the result is not sent to a register. The CF states are examined in the execution of some instructions such as conditional transfer instructions.

2.3 Additional Aspects Of Central Control

2.3.1 General

2.15 In addition to the functions of memory address control and instruction decoding and execution, the CC contains circuitry for various other control functions. A brief description of several of these functions are included as general information.

2.3.2 Data or Address Generation

2.16 General--The execution of most 1B instructions requires the generation of a data word or a memory address. Several factors influence this address or data generation. In general a data field in the instruction is added to the contents of a general purpose register used as an index register in the index adder circuitry. The result replaces the contents of the data address register. If this result is to be used as data, it is sent directly to the operation circuitry. If the result is used as an address, it is interpreted by the memory address decoder which determines the memory community to be addressed. The index adder, data address register, and memory address decoder are located in the decode, control, and address formation circuitry.

2.17 Protected Area Check--A protected area check is also included in the memory address interpretation for instructions which store into memory. Two pointers (in CC internal register memory) define the boundaries of this protected area; any memory address which is less than the lower boundary or greater than or equal to the upper boundary lies within the protected area. Separate instructions allow access

to either area, but if either type of instruction is used with an address outside its intended area, a program interrupt (D level) will result and the store operation will not be completed.

2.18 Write Protect Check--Processors equipped with all 1 Meg stores have a write protect capability incorporated in the stores and supported by a write protection program. This capability provides protection against "wild" writes into protected areas of main memory (predefined by the write protect administration program). An attempt to write to a protected area will result in an interrupt (either D or E level).

2.19 Addressing Techniques--Three addressing techniques are provided in the 1B CC to simplify the administration of program loading and to reduce the number of bits required to encode an address within a program instruction. The techniques are relative addressing, vector table addressing, and absolute addressing.

Memory addresses are referred to by symbolic names when they are coded in instructions; they are rarely referred to by numeric representation of the memory location. These symbolic names are defined in Datapool.

a. Relative Addressing--Used for addressing within a program identification (PIDENT). The address is formed as the sum of the data field of the instruction, the address of the instruction itself, and the contents of an index register (if indexing is specified). Thus, the data field of the instruction has the value of the displacement of the desired address relative to the address of the instruction itself.

b. Vector Table Addressing-- Used for addressing memory locations external to a PIDENT. This technique uses a table of vector addresses that is created by the 1B loader program and located at a fixed location in program store. The location of the external address is formed as the sum of the data field of the instruction and the address of the beginning of the vector table; thus, the data field of the instruction is interpreted as an index on the vector table. The resultant vector table address is read as the final step in the generation of the desired external address. The vector table contains only addresses.

c. Absolute Addressing--Allows certain memory addresses to be

referenced absolutely by the CC in the execution of specific control functions. These are fixed locations within the address spectrum.

2.3.3 Stack

2.20 The stack is sixty-four 32-bit words located in call store. Stack access can be requested, using the PUSH or POP instructions. Stack accesses are also used for address storing and restoring during subroutine linkage. In these cases, the transfer references to the T register for destination index or return address purposes. The stack address is provided by a 6-bit Stack Counter (SC) register which is expanded to 30 bits by a hard-wired constant. Word usage in the stack address advances from low addresses to the high addresses. Therefore the stack counter is automatically incremented after each usage in a PUSH operation. Attached to the stack counter is a combinational decrementer to improve the time efficiency in POP operations. The output of the decrementer is used for addressing in POP, making the appropriate address immediately available. After the POP access, the stack counter is automatically loaded from the decrementer.

2.3.4 Interject Facility

2.21 Closely related to the use of the pushdown stack is a facility for interjecting in the passing of control from one program location to another. It is used to interject priority processing in the normal cycle of program jobs. This interject facility is implemented as an option only on transfers to addresses in the stack. If the option is selected, a check of the interject status flip-flops is made to determine if control is to be interjected to the fixed interject destination address. The interjecting program is then responsible for returning control to the address in the top of the pushdown stack.

2.3.5 Interrupt Facilities

2.22 In order to provide rapid response to system faults and to provide a source for precise timing, circuitry in the CC controls a hierarchy of INTERRUPT LEVELS. Various SOURCES or system conditions are considered as fixed-priority inputs to this control circuitry. Programs are executed on a given level

and the execution of a program on one level is subject to an interruption from a source on a higher level.

2.23 When an interrupt on a given level occurs, the contents of the data buffer register, the control flip-flops, and the address of the interrupted instruction are saved in the Interrupt Bins (two 24-bit registers) of the interrupting level and control is passed to the fixed Interrupt Address for that level. At the completion of processing, the contents of the interrupt bins are restored and control is returned to the interrupted program by use of the Go-Back-To-Normal order. If no interrupt sources are active, processing is said to be back to L Level.

2.3.6 Processor Configuration Facilities

2.24 Processor Configuration (PC) circuitry, in conjunction with associated programs, monitors the processing ability, referred to as System Sanity, of the 1B System and takes action to correct a loss of this ability. This monitoring is a check on fixed cyclic events in the processing control. The corrective action is the selection of one of a series of possible System Configurations (combinations of basic units and connecting buses) and possible Memory Reinitialization. The status of System Recovery attempts is recorded in the circuitry.

2.3.7 Matching Facility

2.25 In order to detect faults, circuitry is provided in the duplicated CCs to Match selected processing results. Match Registers are set by program control to determine the data to be matched. Facilities also allow the Active CC to write into internal registers of the Stand-By in preparation for setting up the Matched Mode of operation.

2.3.8 Master Control Center

2.26 The Master Control Center (MCC) is the primary link between the 1B System and the office personnel. The MCC provides the means for monitoring and controlling certain aspects of the system's operation. It allows for direct monitoring and controlling of the processor configuration as well as indirect (program controlled) display and status control.

3. Programming Terms And Definitions

3.1 Arithmetic Expression

3.01 An arithmetic expression may be a number, symbol, function, attribute, indirect symbol, or a string of these items separated with arithmetic operators. The resulting expression yields a value of a 24-bit binary integer (including insignificant bits).

The allowable arithmetic operators (in order of descending precedence) are:

**	exponentiation
+,-,-	unary plus, unary minus, complement
*,/	multiplication, division
+,-	addition, subtraction

3.2 Memory Instruction

3.02 A memory instruction is a central processor order which must access program store or call store in order to carry out the execution of the instruction.

3.3 Nonmemory Instruction

3.03 A nonmemory instruction is a central processor order which does not access memory. The processor internally generates data necessary to carry out execution of the instruction.

3.4 Single-Word Instructions

3.04 One-word instructions, also called SHORT instructions, have their operation field and variable field encoded into a 24-bit string. Each instruction of this type occupies one program store word.

3.5 Double-Word Instructions

3.05 Double-word instructions, also called LONG instructions have their operation field and variable field encoded into

a 48-bit string. Each instruction of this type occupies two consecutive program store words.

3.6 Item Attributes

3.06 Memory locations are 32 bits, numbered 0,1,2,...,31 from right to left. There are attributes automatically defined on items within a memory location once the item has been defined (for example, displacement and size).

The Displacement of an item in a memory location is equal to the number of the bit position occupied by the rightmost bit in the item. It is symbolically represented as H(ITEM) or DISP(ITEM) where ITEM is a symbolic name of an item in memory.

The Size of an item in a memory location is equal to the number of bits occupied by the item. It is symbolically represented as S(ITEM) or SIZE(ITEM) where ITEM is the symbolic name of an item in memory.

3.7 Direct Transfer

3.07 A direct transfer is an instruction passing program execution to the location specified by the operand field of the instruction.

3.8 Indirect Transfer

3.08 An indirect transfer is an instruction passing program execution to the location given by the contents of the memory location specified by the operand field of the instruction.

3.9 Relative Addressing

3.09 When a location, AA, in a program is referenced by an instruction in the same program at location BB, AA is assembled as a relative address (that is, relative to BB). Therefore, the assembler interprets AA as BB+-n where n is the address of AA minus the address of BB. (This is true if the relative addressing feature is available on the instruction.)

At execution time, the processor will add this relative amount

to the Current Address Register (CAR) to obtain the absolute address. This addition process occurs for each relative address once for each time it is encountered.

3.10 Condition Codes

3.10 The C control flip-flops consist of a sign flip-flop, S, and a homogeneity flip-flop, H. If S=1, it represents a negative number (minus zero included); if S=0, it represents a positive number. If H=1, it represents a binary integer with all bits 1 or all bits 0; if H=0, it represents a binary integer with not all bits 1 nor all bits 0.

The following table contains the possible combinations of the values of the C control flip-flops for the eight condition codes (AZ, arithmetic zero; AU, arithmetic unzero; LZ, logical zero; LU, logical unzero; GT, greater than zero; GE, greater than or equal to zero; LT, less than zero; LE, less than or equal to zero).

	AZ	AU	LZ	LU	GT	GE	LT	LE
S	0	0	0	0	0	0	1	0
H	1	0	1	0	0	0	0	1
S	1	1		1		0		1
H	1	0		0		1		0
S				1		1		1
H				1		1		1

3.11 Arithmetic Zero

3.11 Arithmetic zero in the 1B Processor includes the representation of plus zero (that is, all 24 bits are zeros) and minus zero (that is, all 24 bits are ones). The condition code of the C control flip-flops for arithmetic zero is a homogeneity bit of 1 and a sign bit of 0 or 1. If a homogeneity bit equals 1, it implies that all bits have the same value; if it equals 0, it implies that all bits do not have the same value.

3.12 Logical Zero

3.12 Logical zero in the 1B Processor is the plus zero representation only (that is, all 24 bits are zeros). The condition code of the C control flip-flops for logical zero is a homogeneity bit of 1 and a sign bit of 0.

3.13 Cycle Time

3.13 It is often convenient to refer to the base cycle time of a particular instruction; that is the time required by the processor to carry out the functions specified by the instruction. The 1B Processor is designed to execute instructions in fixed units of time referred to as processor cycles. Thus the actions defined for a given instruction are carried out in a fixed number of processor cycles, and the number of processor cycles is referred to as the instruction's base cycle time.

This processor cycle is 166.67 nanoseconds, which is the time necessary to perform an operation in which a memory access is not required. When memory access is requested, the instruction execution time and memory access time are overlapped so that a base-cycle time of 2 cycles is required for completion of the instruction. The selection of an instruction option or the conditional nature of an instruction may affect its base cycle time.

These base cycle times are included with the description of each instruction and are tabulated in Part 8 for quick reference.

It should be noted that the fetching of an instruction from memory and the execution of the instruction occur during different processor cycles and are overlapped with the executing and fetching of adjacent instructions. Occasionally the 1B Processor will complete the execution of an instruction before the next instruction arrives from memory. In such cases a 1- or 2-cycle delay is inserted into the execution sequence. For this reason the sum of the base cycle times for a sequence of instructions may differ slightly from the time required to execute the sequence.

3.14 Indexing

3.14 A quantity is said to be indexed with a register when the

contents of that register is added to the quantity. The register used is an index register.

3.15 Concatenation

3.15 If X and Y are two strings of symbols, so that X=x(1)x(2)x(3)...x(j) and Y=y(1)y(2)y(3)...y(k), then X concatenated with Y yields:
XY=x(1)x(2)x(3)...x(j)y(1)y(2)y(3)...y(k)

3.16 Logical Product

3.16 Given two binary integers X and Y where

X=x(k)x(k-1)...x(1)x(0)

Y=y(k)y(k-1)...y(1)y(0)

Where x(i),y(i) are bits, then the logical product of X and Y yields

Example:

(Using a 4-bit register for brevity)

```
    X  ___  1010
    Y  ___  1000
X & Y  ___  1000
```

3.17 Logical Union

3.17 Given two binary integers, X and Y, defined as above, the logical union of X and Y yields.

Example:

```
    X  ___  1010
    Y  ___  1100
X | Y  ___  1110
```

3.18 Exclusive Or

3.18 Given two binary integers, X and Y, defined as before, the exclusive OR of X and Y yields

Example:

```
          X  ___  1010
          Y  ___  1100
X ! Y  ___  0110
```

3.19 Complement

3.19 Given a binary integer defined as before, the complement of X yields
-X=-x(k)-x(k-1)...-x(1)-x(0)

Example:

```
 X  ___  1010
-X  ___  0101
```

3.20 Product Masking

3.20 Product masking is the operation of taking some constant, the contents of a register (other than the L register), or the contents of a memory location and ANDing it with the contents of the L (logical) register before it reaches its destination.

Example:

(Using an 8-bit register)

```
Contents of location AA  ___  00111100
 Contents of L register  ___  11110000
Contents of destination  ___  00110000
(after product masking)
```

3.21 Insertion Masking Memory Location as Destination

3.21 To insertion mask an item from a central control register into a memory location, the bit positions in the data buffer register (B), corresponding to the bit positions in the logic register (L) which contain 1s, are replaced by the contents of the corresponding bit positions of the central control register. The other bits in the buffer register remain unchanged, and the new contents of B then replace the contents of the memory location.

Algebraically (Boolean), insertion masking can be represented thus:

B<-(R&L) | (B&-L)

where &, |, -, are the AND, OR, and NOT functions defined previously.

Example:

(Using an 8-bit register for brevity)

Contents of CC register	___	10101010
Contents of L register	___	00001111
Original Contents of B	___	01010101
After insertion masking	___	01011010

The above result then replaces the contents of the memory location in which the insertion masking takes place.

3.22 Insertion Masking Register as Destination

3.22 This operation is possible as an option on many load operations described later. To insertion mask an item into a central control register, the operation is similar to that just described. The contents of the logic register (L) and the contents of the destination register are gated through insertion-mask circuitry, where the item is insertion masked into the contents of the destination register. This result then replaces the contents of the destination register. The C control flip-flops are set prior to loading the register.

3.23 Elementary Register Operations

3.23 Given a central control register, R, with k+1 bit positions (in the 1B Processor, k=23), then R=r(k)r(k-1)...r(1)r(0), where r(i) are bits.

A. Register Shift:

1. Shift R left 1 position
   Rule: r(i) <- r (i-1)
   r(0) <- 0, where i=1,..., k-1

2. Shift R right 1 position
   Rule: r(i) <- r (i+1)
         r(k) <- 0, where i=0,1,..., k-1

B. Register Rotate:

1. Shift R left 1 position
   Rule: r(i) <- r (i-1)
         r(0) <- 0, where i=1,..., k-1

2. Shift R right 1 position
   Rule: r(i) <- r (i+1)
         r(k) <- 0, where i=0,1,..., k-1

4. Arithmetic Expressions

4.1 Definition of an Arithmetic Expression

4.01 An arithmetic expression is made up of a string of operands separated by operators and/or parentheses and followed by a termination character. The expression may be algebraic and/or logical and yield a 24-bit equivalence or value. Negative values are represented in one's complement form.

(a) operand--An operand in an expression may be a number, symbol, function, attribute, indirect symbol, array element, or string.

(b) symbol--A symbol is any string of no more than 255 alphanumerics with at least one nonnumeric, from the following set: [A-Z], [0-9], [_,%]. The percent(%) should not be used in symbol names that are to be used with ESS(TM) simulators, since the simulators will interpret them as simulator key words.

(c) equivalence value--The equivalence or value of a symbol is a 24-bit integer which may be used to identify a program store location, a call store address, or a program parameter. The exception is a symbol defined by the TEXT pseudo-operation: such a symbol has as its value a string of characters.

(d) location counter reference--A relocatable location counter reference is the special reserved symbol asterisk (*) whose value is the current value of the location counter. The location counter value is the address of the next

available location in the program store.

(e) array element--An array element is an array name followed by subscript expressions enclosed in parentheses (see ARRAY pseudo-operation). The value of an array element is a 24-bit integer.

(f) number--A number is a string of digits that represent a 24-bit integer.

(g) function--A function may be either predefined or user defined (see section on FUNCTIONS).

(h) attributes--Attributes may be associated with any symbol. The ATT attribute of symbol SYMB is expressed as ATT(SYMB). Attribute names are formed according to the symbol name conventions, except that attribute names need not contain at least one nonnumeric. A maximum of 250 attribute names may be defined in one assembly. The attribute of a symbol has a 24-bit equivalence.

(i) indirect symbol--An indirect symbol is one that was defined with the TEXT pseudo-operation. The string value of the symbol must represent a valid arithmetic expression, if it is used in an arithmetic expression. For purpose of parsing it is as though there were parentheses around all indirects.

(j) string--A string is a sequence of EBCDIC characters delimited by single quotes. The string may not contain a quote. A string may also be denoted by the C function (see section on FUNCTIONS). When a string is longer than 24/8 characters, the rightmost 24 bits are used in all operations except the relational ones which operate on the entire string. Leading binary zeros are assumed on strings containing less than 24 bits. The null string is considered the same as binary zero.

(k) operator--An operator is any character or characters that indicate an arithmetic or logical operation to be performed (see section on Operators) [REF. 4.2].

(l) termination character--A termination character indicates the end of an expression or subexpression. The termination characters are blank, comma, colon, unbalanced right parenthesis, and the sharp sign.

(m) boolean expression--A boolean expression is formed according to the rules for forming arithmetic expressions; its value is the value of the corresponding arithmetic expression. A boolean expression is interpreted to have value TRUE/FALSE if its value is nonzero/zero. Boolean expressions treat both 0 and -0 as zero.

4.2 Operators

4.02 The evaluation of an expression is performed according to the precedence of the operators from left to right within each group. Operations of higher precedence are performed before operators with a lower precedence. Parentheses have their usual mathematical bracketing function. In the following text operators will be listed in order of descending precedence.

(a) Arithmetic operators--All arithmetic operators use 24 bit one's complement arithmetic.

**	exponentiation
+,-,-	unary plus,unary minus, unary complement
*,/	multiplication, division
+,-	addition,subtraction

(b) Relational operators--The following are all the same presidence and yield a result of one if the relation is true, zero otherwise.

=	Equal
-=	Not equal
<	Less than
=< or <=	Less than or equal
>	Greater than
=> or >=	Greater than or equal

A R1 B R2 C is equivalent to (A R1 B) & (B R2 C) where R1 and R2 are relational operators. In string comparisons, the EBCDIC collating sequence is used.

4.3 Logical Operators

The following operators have the save presidence:

&	and (intersection)
-	and of complement; that is, A-B = A & (-B)

The following operators have the save presidence:

```
            !   exclusive or: A!B = (A-B) | (B-A)
            |   or (union)
```

4.03 The logical operators may be applied to either 24-bit arguments or 1-bit Boolean arguments. If the arguments are both 24-bit amounts, the result is formed by bitwise (that is, bit by bit) operation. If the arguments are both Boolean, the result is a 1-bit truth value. Otherwise, the result has the type of the right-hand argument.

4.4 Examples of Arithmetic Expressions

```
    A*B**2+1          is equivalent to   (A*(B**2))+1
    A-B/C*D           is equivalent to   A-((B/C)*D)
    A&B|C             is equivalent to   (A&B) | C
    A-B|C&D           is equivalent to   (A&(-B))|(C&D)
    A>B>C             is equivalent to   (A>B)&(B>C)

    7/3               equals   2
    X/0               equals   X
    -3**2             equals   -9
    (-3)**2           equals   9
    10/3&4|1          equals   ((10/3)&4)|1                equals   1
    -1&6              equals   (-1)&6                      equals   6
    8>5               equals   1
    4<5=4             equals   (4<5)&(5=4)                 equals   0
    2+3>4&16>1|16<0   equals   (((2+3)>4)&(16>1))|(16<0)   equals   1
    'ABC'='ABC'       equals   1
    '*'='/'           equals   0
    'AB'<'BC'         equals   1
    'AB'<'Z'          equals   1
    'AB'/256='A"      equals   1
    ''=0              equals   1
    '1'>'Z'           equals   1
    -0=               equals   1
    -0<0              equals   0
    -0>0              equals   0
    1|-0              equals   1|-0                        equals   -0
```

Note: The EBCDIC collating sequence is used when comparing strings with the < and > relations. This type of comparison (left to right) if used in a sort routine would result in the strings sorted in

the same sequence as they would appear in a dictionary.

```
X  EQU  'B'+256*'A'  is equivalent to  X  EQU  'AB'

X  EQU  'C'+1  is equivalent to  X  EQU  'D'

BUT

Y  EQU  'I'+1  is NOT equivalant to  Y  EQU  'J'

SINCE

 'I'  =  C9(hex)  and  'J'  =  D1(hex)
```

The following demonstrates the use of indirect symbols in an arithmetic expression.

```
A  TEXT  'B+C'
B  TEXT  'D*E'

Then

X  EQU  2*A

Is equivalent to

X  EQU  2*(D*E+C)
```

4.5 Relocation Rules for Arithmetic Expressions

4.04 The final result of an expression must be either relocatable or absolute. An expression is absolute if its value is unaffected by the address at which the program will be loaded; otherwise, it is relocatable. Thus, if there are K positive relocatable symbols in the expression there must be either K or K-1 balancing relocatable symbols preceded by a minus sign.

If R is relocatable and A is absolute, then:

```
A  *  =  A
R  *  A  and  A  *  R  are illegal unless A  = 1 or 0
R  *R  illegal
A  /  A  =  A
A  /  R  and  R  /  R  are illegal
```

```
  R  /  A  is illegal unless A is 1 or 0
  If A is 0 the division is done as if A  =  1.
```

The operands of a logical operator may be relocatable.

If P is a truth value operator, then:

```
 R  P  A  and  A  P  R   are illegal
 A  P  A  and  R  P  R   are illegal
```

If X is a relocatable program store symbol (type L) and Y is a call store symbol (type C), then the expression X-Y would be given a warning flag. The programmer may circumvent the warning flag by writing X-STYP(Y,A). Here STYP tells the arithmetic scanner to forget about the real type of Y (C) and to use type A (absolute) instead. Thus the scanner considers Y to be an absolute symbol instead of a call store symbol.

4.6 Scanner Functions

4.6.1 Introduction to Scanner Functions

4.05 A scanner function is expressed as fcn(arg1,...argm) and may be used an an operand of an arithmetic expression just as a number or symbol might be. The name of the function is fcn and the argi are its arguments. The argi may in turn be arithmetic expressions of any degree of complexity. Each function has a 24-bit one's complement value and a type (that is, relocatable value, absolute value, etc). The value is obtained by first evaluating the argi and then applying the function to the argument values. The type of the function is determined either by the function itself or by the function arguments.

4.6.2 Predefined Functions

4.06 Some functions are merely modal qualifiers. They set the prevailing arithmetic scanning mode to, for example, octal for the duration of the evaluation of the function. This means that any unqualified numbers in the function argument will be interpreted as octal, decimal, etc, depending upon the prevailing scan mode.

B(ae) Causes any unqualified numbers present in the arithmetic expression "ae" to be interpreted as binary.

C(string)
Causes "string" to be interpreted as a string of characters. It is equivalent to 'string'. The function "string" may be a maximum of 255 characters long and may not contain an odd number of quotes or an imbalanced parenthesis.

D(ae) Causes any unqualified numbers present in the arithmetic expression "ae" to be interpreted as decimal.

O(ae) Causes any unqualified numbers present in the arithmetic expression "ae" to be interpreted as octal.

X(ae) Causes any unqualified numbers present in the arithmetic expression "ae" to be interpreted as hexadecimal.

Some scanner functions are Boolean or truth value functions.

EXIST(par) Equals one if macro parameters "par" exists (that is, is not missing), zero otherwise. The function is valid only in statements inside macro definitions. A parameter whose value is explicitly (or occasionally implicitly) null is considered to EXIST. Since "missing" has an unusual meaning, this function should be used with caution.

NOT(ae) Equals one (or zero) if the value of the arithmetic expression (ae) is zero (or nonzero).

TV(ae) Equals one (or zero) if the value of the arithmetic expression "ae" is nonzero (or zero); that is, the "truth value" of "ae". TV treats 0 and -0 as zero.

EQV(ae1,ae2) Has as its value the logical equivalence of "ae1" and "ae2" computed bit by bit: that is, a bit in the result will be one if the corresponding bits in "ae1" and "ae2" are logically equivalent (both ones or both zeros) and zero otherwise.

Other scanner functions calculate a 24-bit value from the value of their arguments.

ABS(ae) Equals the absolute value of the arithmetic

	expression "ae".
E(ae)	Equals a 24-bit pattern of all zeros with a 1 in the bit position determined by the value of "ae". "ae" may be arithmetic expression such that 0 <= ae <= 24. E(ae1,....,aen) equals E(ae1) | ... | E(aen).
M(it)	Equals MSK(SIZE(it),DISP(it)). "it" must be a symbol with SIZE and DISP attributes. Items defined with the ITEM pseudo-operation always have SIZE and DISP attributes. M(it1,....,itn) equals M(it1) | .... | M(itn)
MAX(ae,...)	Equals the algebraically larger value in the list. Each "ae" may be any arithmetic expression.
MIN(ae,...)	Equals the algebraically smallest value in the list. Each "ae" may be any arithmetic expression.
MOD(ae1,ae2)	Equals the remainder of "ae1" divided by "ae2" are "ae1" modulo "ae2". "ae1" and ae2" may be any arithmetic expression with a positive value.
MSK(ae1[,ae2])	Equals E(ae1) - 1 or an ae1-bit mask. If "ae2" is specified, the mask is displaced to the left by ae2-bit positions. "ae1" and "ae2" may be any arithmetic expression such that ae1 > 0 and ae2 >= 0.
PAR(ae)	Has as its value the even parity of "ae"; that is, it equals one (zero) if "ae" has an odd (even) number of bits equal to one.
"SET(sym,ae)"	Equals the value and type of the arithmetic expression "ae". In addition, the symbol named "sym" is defined to be the value and type of "ae". "sym" may be redefined if desired. Since this function has a side-effect, its use in an actual parameter in a macro call may be dangerous as sym will be set at (possibly several) points in time in the middle of the macro expansion.

STRIP(ae) If "ae" is a relocatable expression, this function has as its value the strip number of the strip in which "ae" was defined and zero otherwise.

Note: If "ae" is relocatable and defined in the main strip, STRIP(ae) will return a value of zero and be indistinguishable from nonrelocatable expression's result.

STYP(ae,char) Equals the value of the arithmetic expression "ae" and the type of "char". The argument "char" is a letter corresponding to one of the valid SWAP types.

V(ae) Equals the value of the arithmetic expression "ae"; that is, V is the identity function.

XTYP(ae) The character corresponding to the type of "ae" can be obtained by using the macro function BCD inside macros.

EXAMPLES

A. O(14) = D(12) = D(O(10) + 4) = 12

B. E(3) = MAX(1,8,5) = MIN(12,8,10) = 8

C. E(1,2,3) = B(1110) = 14

D. O(D11) + 3) -= O(14) since
   O(D(11) + 3) = O(D(11)) + O(3) = D(11) + O(3) = 14

E. O(D(10) + 10) = O(12 + 10) = O(22)

F. MSK(3) = B(111) = 7

G. C(A) = 'A' = X(C1) = O(301) = B(11000001) = 193

H. C() = '' = 0

I. NOT(2 + 3 = 4) equals 1
   NOT(17) equals 0

J. Assume that E is a symbol with a value of 3.
   B(10 + E) = 5
   X(10 + E) = X(1E) = 30
   (E is interpreted as a hexadecimal number.)

K. MSK(1) * E(4) = MSK(1,4) = 16

L. SET(Y,3) + SET(y,Y + 1) + Y = 11

M. EQV(0,-0) = 0
   EQV(0,0) = O(77777777)
   EQV(1,-0) = 1

N. Suppose we have the following symbol definitions:

```
          SORG   0,4   #ADDRESS 0 OF STRIP 4
   SYM1   EQU    *
   SYM2   EQU    4
          SORG   4,0   #ADDRESS 4 OF MAIN STRIP
   SYM3   EQU    *
   SYM4   EQU    5
```

   Then:

   STRIP(SYM1) = 4 that is, a relocatable expression in strip
   4
   STRIP(SYM2) = 0 that is, an absolute expression
   STRIP(SYM3) = 0 that is, a relocatable expression in the
   main strip.
   STRIP(SYM4) = 0 that is, an absolute expression

O. PAR(7) = 1
   PAR(102) = PAR(0(146)) = 0
   PAR(-0) = PAR(O(77777777)) = 0

P. Suppose we have the following macro and symbol definitions:

```
           MACRO
           TESTXTYP   AA,BB,CC,DD
           PMC        ON
   PRINT   EDIT       .LABI
```

   #/XTYP(AA) = DEC(XTYP(AA)) = BCD(XTYP(AA))/
   #/XTYP(BB) = DEC(XTYP(BB)) = BCD(XTYP(BB))/
   #/XTYP(CC) = DEC(XTYP(CC)) = BCD(XTYP(CC))/
   #/XTYP(DD) = DEC(XTYP(DD)) = BCD(XTYP(DD))/

```
.LABI   NULL
        MEND
S4      EQU     *
S3      PBLOCK  1
S2      BLOCK   1
S1      EQU     3
```

Then the macro call TESTXTYP S4,S3,S2,S1 would result in:

```
#/XTYP(S4) = 211 = L/
#/XTYP(S3) = 198 = F/
#/XTYP(S2) = 195 = C/
#/XTYP(S1) = 193 = A/
```

Q. This example shows that -0 and 0 are both treated as zero in logical expressions

```
TV(0) = 0
TV(-0) = 0
```

However when - operates on a truth value of 0, the result is a truth value of 1.

```
-TV(0) = 1
```

4.7 DFN--User Defined Functions

4.07 The DFN pseudo-operation allows the user to define new arithmetic functions in addition to those predefined by the assembler.

```
**********************************************************************
          [loc]    DFN    f(p1,....pn)=a1:b1,....,an:bn
**********************************************************************
```

loc If specified may be used to set the prevailing scan mode.

The name of the function being defined.

"pi" The dummy parameters.

ai/bi Any legal arithmetic expression, which may contain the "pi" and other variables, and which is used to determine

the definition of "f".

The expression "ai" will be used as the defining expression if the value of "bi" is nonzero or true. If the value of "bi" is zero (that is, false) scanning will continue with the next set of expressions to the right in an attempt to find a defining expression. If "bi" is not present, "ai" will unconditionally be used as the defining expression. If all the "bi" are zero, the function has the value zero.

If the location field "loc" is not null and begins with one of the letters B, D, O, or X then the prevailing scan mode will be set to binary, decimal, octal, or hexadecimal when the arguments are evaluated in the function call. This means that the arguments would not have to be qualified with the scanner function B, D, O, or X in order to be interpreted properly. If the location field is empty or does not begin with B, D, O, or X arguments, evaluation will be performed in the current scan mode. In either case the interpretation of the "ai" and "bi" is unaffected.

When the function "f" is called the arithmetic expressions corresponding to the "pi" are evaluated and the resulting values (and not the expression itself) are used whenever "pi" appears in the "ai" and "bi". This means that the "pi" should be used in the defining expression as a number would be used but not as an attribute name, function name, or contents of a string. The following statements demonstrate the way the "pi" should not be used in a definition.

```
          DFN    F(X)=X(Y)
          DFN    F(X)=1:'X'='Y',0
```

The actual arguments present in the call of function F may be symbol names, strings, attributes of symbols, etc.

Two other features are provided to allow an arbitrary number of arguments in the call of a function. The first is the ability to ask if an argument was implicitly null (explicitly null arguments are treated as zero). This is done by immediately following the dummy argument by a question mark. If the argument was supplied in the function call the value of the argument-question mark is one (true), otherwise the value is zero (false). For example, consider the following statement:

```
          DFN     F(X,Y)=X+Y:Y?,X+1
```

F(2,5) yields a result of 7 where as the value of F(3) is 4.

The second facility is the ability to loop over a part of the expression, substituting successive argument values wherever the last dummy argument appears in the definition. The facility is invoked by the appearance of an ellipsis (...) in the defining expression. The range of the loop is from the operator immediately preceding the ellipsis backward to the first occurrence of the same operator at the same level of parentheses. For example, consider the following statement:

```
          DFN     SUM(A,B,X,Y) = A + Y**(X + 1) + ...B/2
```

The range of the loop is from the plus immediately preceding Y to the plus immediately preceding the ellipsis. The call SUM (2,18,1,1,2,3) has as its value:

```
          2+1**(1+1)+2**(1+1)+18/2 = 25
```

The loop may also extend over the expression between two commas as shown in the example.

5. Coding Format

5.1 General

5.01 A typical symbolic instruction contains four major fields: the location field, operation field, variable (or operand) field, and the comment field.

The location field may contain a symbol which may be referenced by transfer instructions. The operation field contains the mnemonic of the instruction. The variable field normally contains the parameters related to the operation. The comment field exists only for the programmer and has no function in machine execution.

Example:

```
Location   Operation   Variable     Comment

AAA        L           X,CSLOC(Y)
           IF:T        C=LZ:AAA     #T ON LZ
```

The location field must start in column 1. The operation field follows the location field. If there is no location field, column 1 is blank (or contains a comma) and the operation field

may start in column 2. The variable field follows the operation field. If the operation field is absent, a comma is required. The variable field extends to the comment field or the logical end of the parameter string. The comment field must begin with a # sign. All characters to the right of a # sign are regarded as a comment. A # sign in column 1 implies that the entire line is a comment. Trailing commas in the variable field may be omitted. See Series 2B, Volume 1 for detailed coding procedures necessary for assembly.

5.2 Variable Subfield Identification

5.02 Most variable fields (operands) of the 1B Processor machine instructions follow one of two general formats, with a few slight variations.

The basic operand for a nontransfer instruction has three major subfields (separated by commas):

DESTINATION OR SOURCE REGISTERS,DATA(INDEX),OPTIONS	MASKING/ COMPLEMENT

The basic operand for a transfer instruction has two major subfields (separated by commas):

DATA OR ADDRESS(INDEX),RETURN OPTIONS

Many instructions perform combined functions. These instructions will be denoted with a colon in both the operation field and the operand field. For example:

```
IF:T     C=LZ:ADDR
```

The colon separates the operation field into two suboperations corresponding respectively to the suboperands created by the colon in the operand field.

6. Index Register Modifications

6.1 General

6.01 In general, index register modifications are allowed only on long nontransfer orders.

A When "A" appears concatenated to the mnemonic of any index register, Ri, in the indexing field, 1 is added to the contents of the specified index register immediately after the indexing operation. (Also refer to add-one-to-memory instruction couplet.)

S When "S" appears concatenated to the mnemonic of any index register, Ri, in the indexing field, indexing does not occur and the contents of Ri is set equal to the contents of the data/address field. However, if relative addressing is used in the data/address field, the absolute program store address, referenced by the data/address field, replaces Ri.

W When "W" appears concatenated to the mnemonic of any index register, Ri, in the indexing field, Ri is set equal to the contents of the data/address field plus the contents of Ri. "W" occurs immediately after the normal indexing operation. However, if relative addressing is used in the data/address field, the absolute program store address referenced by the data/address field, added to the contents of Ri, replaces Ri.

If a register is selected as both the index register and an argument register, any index register modification will occur before the register is used as an argument.

6.2 Stack Indexing Options

6.02 The indexing subfield of the data/address field is also used for specifying options on transfer instructions whose destination is derived from the pushdown stack. These options are only available on short transfer instructions. On conditional transfers, the options apply only if the transfer condition is satisfied.

T If "T" appears in the indexing field, the address in the stack register T is used with the contents of the data/address field for the indexing process. As a result, the pushdown stack is automatically "popped."

I If "T" appears concatenated with "I", a check is made to

see if the interject flip-flop is set. If so, a transfer is made to a hard-wired address to begin processing interject work. If not, the instruction functions as a simple transfer, with "T" in the indexing field.

N If "T" appears concatenated with "N", the option is the same as "T", but the pushdown stack is not "popped".

NI If "T" appears concatenated with "NI", a check is made to see if the interject work flip-flop is set. If so, a transfer is made to a hard-wired address to begin processing interject work, and the stack is not popped. If not, the instruction functions as a simple transfer with "TN" in the indexing field.

6.3 Return Options

J When "J" appears in the return field of a transfer order, the contents of the return address register J is set to the address of the instruction following the transfer.

T When "T" appears in the return field of a transfer order:

1. The present contents of the stack register T replaces the contents of the data buffer register B, and enters the pushdown stack at the location specified by the stack pointer.

2. The address of the instruction following the transfer order replaces the contents of the stack register.

3. The value of the stack pointer is incremented by 1 and is tested for overflow.

6.4 Mask Complement Options

C When "C" appears in the mask/complement field of an instruction, the contents of the indexed data field (if it is a nonmemory instruction), or the contents of a memory location specified by the contents of the indexed data field (if it is a memory read instruction), or the contents of a CC register (if it is a memory write instruction), is complemented immediately after possible product masking and immediately before possible insertion masking. (Rule: PCE)

PL When "PL" appears in the mask/complement field of an instruction, product masking is to be done with the contents of the logic register L, always preceding possible complementing.

EL When "EL" appears in the mask/complement field of an instruction, insertion masking is to be done with the contents of the logic register L, always following possible complementing.

P(ae) When this option appears in the mask/complement field of an instruction:

1. The logic register L, is set equal to the value of the arithmetic expression, and

2. Product masking occurs as if PL were the selected option.

E(ae) When this option appears in the mask/complement field of an instruction:

1. The logic register L, is set equal to the value of the arithmetic expression, and

2. Insertion masking occurs as if EL were the selected option.

6.5 General Procedures

6.03 The L register cannot be both the source of a masking operation and the destination register in the same instruction.

L can be both the source for masking and the argument or source register in the same instruction. In such cases, any setting of L for masking will occur prior to its use as an argument.

7. Parameter Mnemonics

7.1 General

7.01 The following mnemonics can be used to reference that particular piece of hardware:

B -- Data Buffer Register

C -- C Control Flip-Flops

F -- F Index Register

G -- G Index Register

J -- J Index Register

K -- K Index Register

L -- Logic Register

X -- X Index Register

Y -- Y Index Register

Z -- Z Index Register

T -- Top-of-Stack Register

7.2 Dummy Variables

7.02 The following symbols are used in instruction descriptions only (for brevity) and cannot be used in the coding of an instruction. In coding, an actual mnemonic is substituted for the variable.

- R,R1,R2,etc, represent these central control registers: L, F, G, K, X, Y, Z, and J.
- Ri represents these allowable central control registers which may be used for indexing: F, G, K, X, Y, Z, and J.
- Rt represent these central control registers: T, F, G, K, X, Y, Z, and J.
- PAR represents the program address register which contains the address of the next instruction to be executed.
- AE,AE1,AE2,etc, represent valid arithmetic expressions (see Programming Terms and Definitions) [REF. 3].
- COND represents the eight condition codes of the C control flip-flops: AZ, AU, LZ, LU, GT, GE, LT, and LE.

- o Rp represents the window pointer register: XAWP0, ..., XAWP7.

7.3 Symbols

7.03 The following symbols are used as descriptors:

<> This notation, when used around a symbol, arithmetic expression, etc., specifies the "contents of" the location given by that symbol, arithmetic expression, etc. It is used for explanatory purposes only (that is, it may not be used as part of the coding of an instruction).

[ ] A pair of brackets is used to display a collection of choices, including the null choice. Either one choice or no choice should be made from the collection. After the selection, delete all other choices within the bracket pair, including the brackets.

___ The overscore is used many times with a collection of choices. It is defined to mean the null choice.

{ } The braces are used to display a set of a particular type of arithmetic expressions representing binary integers (see following description of set identifiers).

7.4 Set Identifiers

7.04 The set identifiers are as follows:

(a) {nnADDR} are sets containing all the AEs representing unsigned binary integers interpreted as addresses, with the maximum number of bits necessary to encode the address given by the numeral prefix nn.

(b) {nnDATA} are sets containing all the AEs representing signed binary integers interpreted as either data or an address (depending on the operation of the instruction) with the maximum number of bits necessary to encode the data or address (including the sign bit) given by the numeral prefix nn.

(c) {24MSK} is the set containing all the AEs representing unsigned binary integers interpreted as (maximum) 24-bit

masks of noncontiguous 1s or contiguous 1s.

(d) {9SD} is the set containing all AEs representing unsigned binary integers interpreted as (maximum) 24-bit masks of contiguous 1s. The 24-bit contiguous mask is actually encoded by the assembler as a 9-bit size and displacement value.

(e) {24REL},{14REL} is the set containing all AEs representing signed binary integers interpreted as (maximum) 24- or 14-bit increments (including the sign bit) added to the current value of the CAR prior to being used in the instruction (that is, it is used as a signed amount for relative addressing).

(f) {13VEC} is the set containing all AEs representing binary integers interpreted as (maximum) 13-bit addresses in the Vector Table. The 13-bit integer is actually added to the starting address of the Vector Table to obtain the desired location.

(g) {7SGNDTA} {11SGNDTA} is the set containing all AEs representing signed binary integers interpreted as (maximum) 7- or 11-bit data (including the sign bit).

(h) {9INCR},{6INCR} are the sets containing all AEs representing signed binary integers used as increments. The maximum number of bits necessary to encode the increment is the numeral prefix.

(i) {5SH} is the set containing all AEs representing unsigned binary integers interpreted as (maximum) 5-bit amounts of a register left shift of left rotation.

(j) {6SH} is the same as {5SH} except a sign is added to the encoding of the binary integer allowing for left and right shifts and rotations.

(k) {1BIT} is the set containing all AEs representing unsigned binary integers interpreted as a 1-bit mask (that is, a single 1 in a field of 0s).

8.1 Add (A)

****

```
    __
A R1R2,MEMLOC,MASKOPT

R1 or R2 = R1 + <MEMLOC>
or
R1 or R2 = R1 + (<MEMLOC> & MASKOPT)
*********************************************************************

     Function            The contents of the memory location replaces the
                         contents of the B register, and after possible
product
                         masking and/or complementing, is added to the
contents
                         of R1.  If R2 is specified, R2 receives the sum and
                         R1 is unchanged.

     Control             The CFs are set immediately after the addition
     Flip-Flops          operation.

     L Register          If a mask is specified for product masking, the L
                         register is set to the value of the mask.
Otherwise,
                         the L register value is not affected.

     Cycle Time          Base cycle time is 2 cycles.
     Restrictions        If masking is selected, the destination register
                         (R1,R2) cannot be L.  See encodings for other
                         restrictions, including option combinations
                         and field sizes.


     Encoding            There are three short forms and two long forms.

     Binary encodings of the instruction:

     SHORT forms:

*********************************************************************
A R1,14ADDR

A X,SYMADDR
*********************************************************************

     ___________________________________________________________

     23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
     ________,________,________,________,_______,___________,______
```

```
    | 0 0  0  0  1  0  1 |  R1    |            14ADDR              |
    |____________________|________|________________________________|
```

****

A R1,+-10ADDR(Ri)

A X,SYMADDR(Z)

****

```
     ____________________________________________________________
     23 22 21 20 19 18 17 16 15 14 13  12 11 10  9 8 7 6 5 4 3 2 1 0
     ________,________,________,________,_______,___________,_______
    | 0 0  0  1  1  0  1 |  R1    | +-| Ri     |      10ADDR        |
    |____________________|________|___|________|____________________|
```

****

```
          ______
          PL
```

A R1,(Ri),P(9SD)

A X,(Z),P(SYMSD)

****

```
     ____________________________________________________________
     23 22 21 20 19 18 17 16 15 14 13  12 11 10  9 8 7 6 5 4 3 2 1 0
     ________,________,________,________,_______,___________,_______
    | 0 0  1  0  1  0  1 |  R1    |         9SD           |* *| Ri  |
    |____________________|________|_______________________|___|_____|
```

LONG forms:

The following bits are encoded to indicate the corresponding conditions:

Bit 3 (word 1)-- Relative addressing: 0 = no relative address
1 = relative address

****

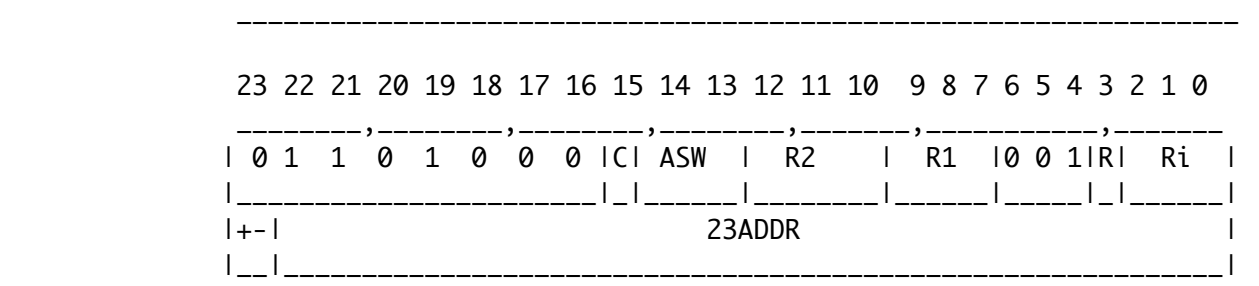
```
                  _
    __            A  _
```

A R1R2,+-23ADDR(RiS),C

```
                  W
```

AXY,SYMADDR(ZA),C

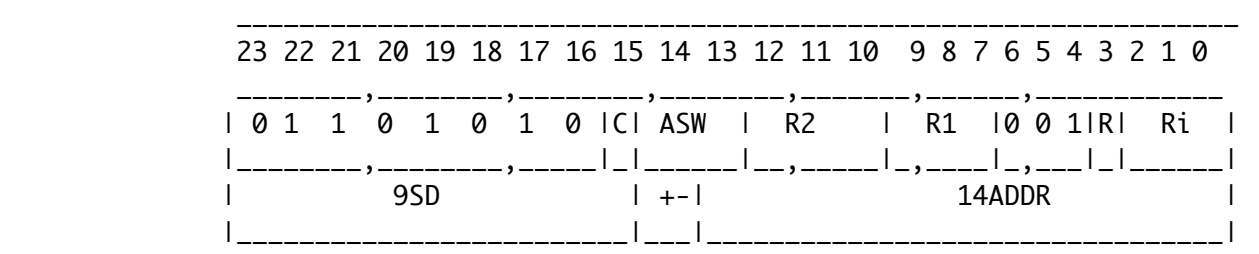
****



****

```
                  ___
                _ PCL
    __          A PL
A R1R2,+-14ADDR(RiS),C
                W (9SD)
                  PC(9SD)

A X,SYMADDR(ZW),PC(SYMSD)
```

****



8.2 Add Word (AW)

****

```
    __
AW R1R2,ADDRDATA,MASKOPT

R1 or R2 = R1 + ADDRDATA
or
R1 or R2 = R1 + (ADDRDATA & MASKOPT)
```

****

Function The address or data, after possible product masking and/or complementing, is added to the contents of R1. If R2 is specified, R2 receives the sum and R1 is unchanged.

Control Flip-Flops	The CFs are set immediately after the addition.
L Register	If a mask is specified for product masking, the L register is set to the value of mask. Otherwise, the L register value is not affected.
Cycle Time	Base cycle time is 1 cycle. If both relative addressing and indexing are specified, base cycle time is 2 cycles.
Restrictions	If masking is selected, the destination register (R1,R2) cannot be L. See encodings for other restrictions, including option combinations and field sizes.
Encoding	There are three short forms and two long forms.

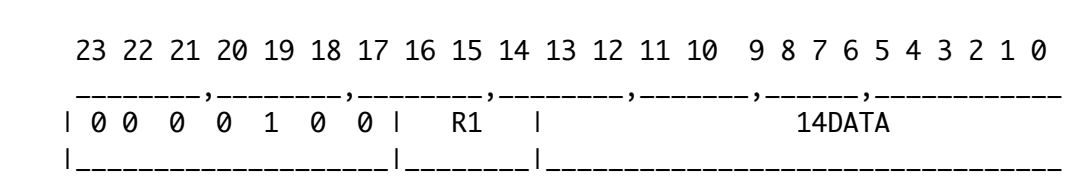
Binary encodings of the instruction:

SHORT forms:

****

```
AW R1,14DATA

AW X,SYMDTA
```

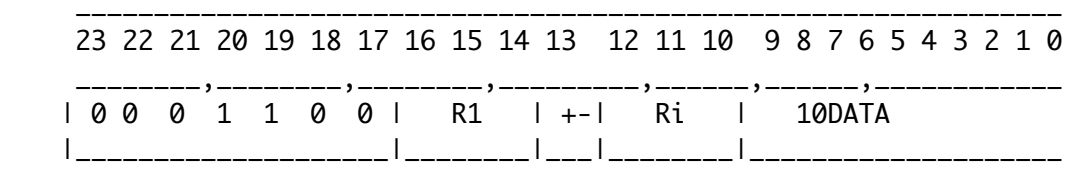
****



****

```
AW R1,+-10DATA(Ri)

AW X,SYMDTA(Z)
```

****



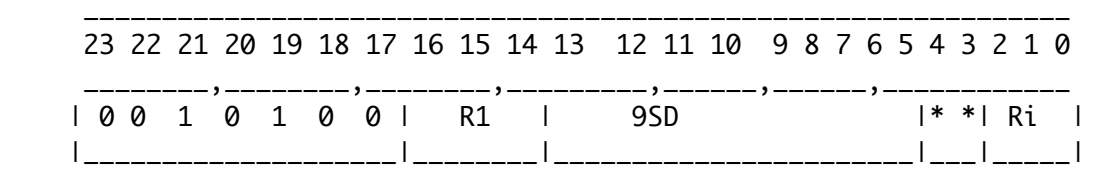
****

PL

AW R1,(Ri),P(9SD)

AW X,(Z),P(SYMSD)

****



LONG forms:

The following bits are encoded to indicate the corresponding conditions:

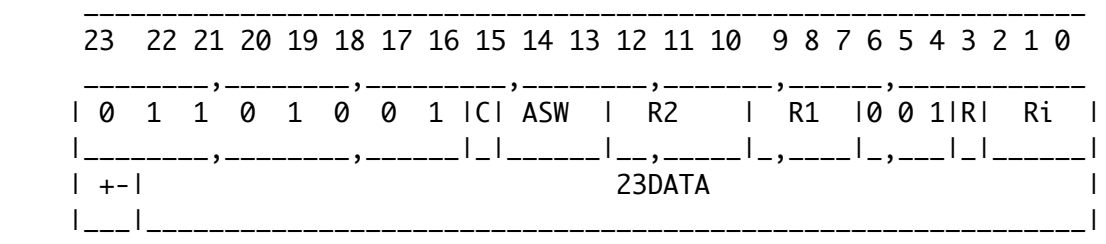
Bit 3 word 1-- Relative addressing: 0 = no relative address 1 = relative address

****

```
                  _
     __           A  _
AW R1R2,+-23DATA(RiS),C
                  W
```

AW XY,SYMDTA(ZA),C

****

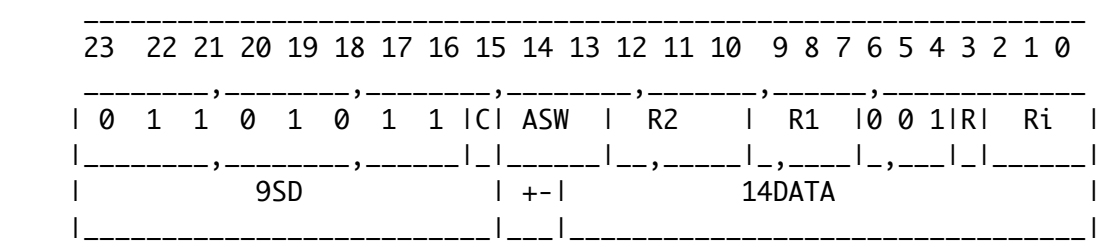


****

```
                  _  _______
     __           A  PL
AW R1R2,+-14DATA(RiS),PCL
                  W  C
                     P(9SD)
                     PC(9SD)
```

AW X,SYMDTA(ZS),P(SYMSD)

****



8.3 Compare (C)

****

C R1,MEMLOC,MASKOPT

CFs = R1 - <MEMLOC>
or
CFs = R1 - (<MEMLOC> & MASKOPT)

****

FUNCTION	The contents of the memory location replaces the contents of the B register, and after possible product masking and/or complementing, is internally subtracted from the contents of R1 and the result sets the CFs. The contents of R1 is unchanged.
Control Flip-Flops	The CFs are set as a result of the compare.
L Register	If a mask is specified for product masking, the L register is set to value of the mask. Otherwise, the L register value is not affected.
Cycle Time	Base cycle time is 2 cycles.
Restrictions	See encodings for restrictions, including option combinations and field sizes.
Encoding	There are three short forms and two long forms.

Binary encodings of the instruction:

SHORT forms:

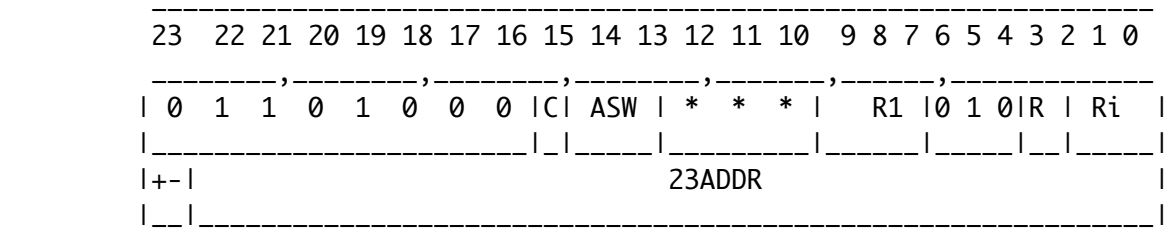
****

C R1,14ADDR

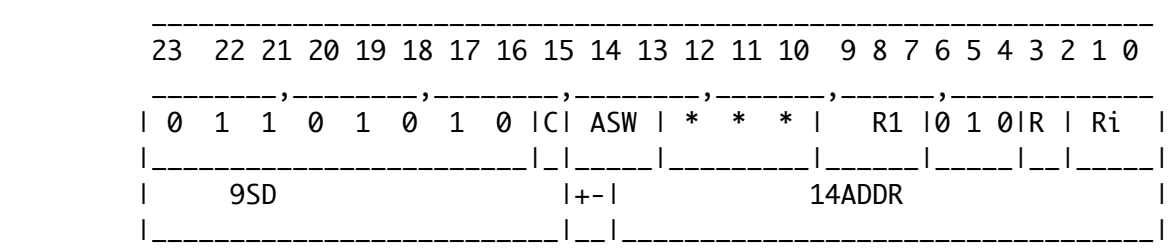
C X,SYMADDR

```
**********************************************************************

     ____________________________________________________________

     23  22 21 20 19 18 17 16 15 14 13 12 11 10  9 8 7 6 5 4 3 2 1 0
     _______,_______,_______,_______,_______,______,_____________
    | 0  0  0  0  0  1  1 |  R1    |            14ADDR               |
    |_____________________|________|_________________________________|


**********************************************************************
C R1,+-10ADDR(Ri)

C X,SYMADDR(Z)
**********************************************************************

     ____________________________________________________________
     23  22 21 20 19 18 17 16 15 14 13 12 11 10  9 8 7 6 5 4 3 2 1 0
     _______,_______,_______,_______,_______,______,_____________
    | 0  0  0  1  0  1  1 |  R1    |+-|  Ri    |    10ADDR           |
    |_____________________|________|__|________|_____________________|


**********************************************************************

          ______
          PL
C R1,(Ri),P(9SD)

C X,(Z),P(SYMSD)
**********************************************************************

     ____________________________________________________________
     23  22 21 20 19 18 17 16 15 14 13 12 11 10  9 8 7 6 5 4 3 2 1 0
     _______,_______,_______,_______,_______,______,_____________
    | 0  0  1  0  0  1  1 |  R1    |    9SD    |          |* * | Ri  |
    |_____________________|________|___________|__________|____|_____|

     LONG forms:

     The following bits are encoded to indicate the corresponding
     conditions:

     Bits 12-10 word 1--These bits are not used.

     Bit 3 word 1-- Relative addressing: 0 = no relative addressing
      1 = relative addressing
```

```
**************************************************************************

                _
                A  _
C R1,+-23ADDR(RiS),C
                W

C X,SYMADDR(ZA),C
**************************************************************************
```




```
**************************************************************************

                   ___
                _  PCL
                A  PL
C R1,+-14ADDR(RiS),C
                W  P(9SD)
                   PC(9SD)

C X,SYMADDR(ZW),PC(SYMSD)
**************************************************************************
```



8.4 Compare Word (CW)

```
**************************************************************************
CW R1,ADDRDATA,MASKOPT

CFs = R1 - ADDRDATA
or
```

CFs = R1 - (ADDRDATA & MASKOPT)

****

Function	The data, after possible product masking and/or complementing, is internally subtracted from the contents of R1 and the results set the CFs. The contents of R1 is unchanged.
Control Flip-Flops	The CFs are set as a result of the compare.
L Register	If a mask is specified for product masking, the L register is set to the value of the mask. Otherwise, the L register value is not affected.
Cycle Time	Base cycle time is 1 cycle. If both relative addressing and indexing are specified, base cycle time is 2 cycles.
Restrictions	See encodings for restrictions, including option combinations and field sizes.
Encoding	There are three short forms and two long forms.

Binary encodings of the instruction:

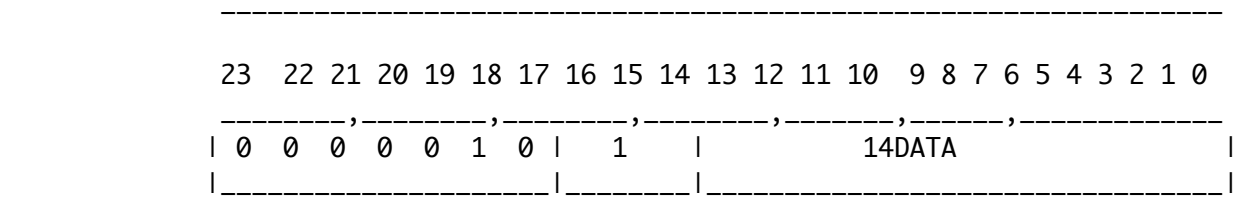
SHORT forms:

****

CW R1,14DATA

CW X,SYMDTA

****



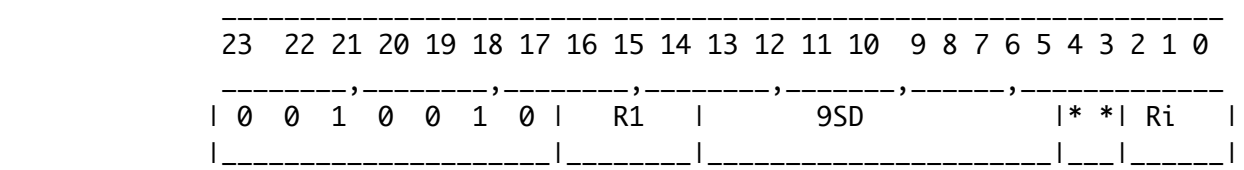
****

PL

CW R1,(Ri),P(9SD)

CW X,(Z),P(SYMSD)

****



LONG forms:

The following bits are encoded to indicate the corresponding conditions:

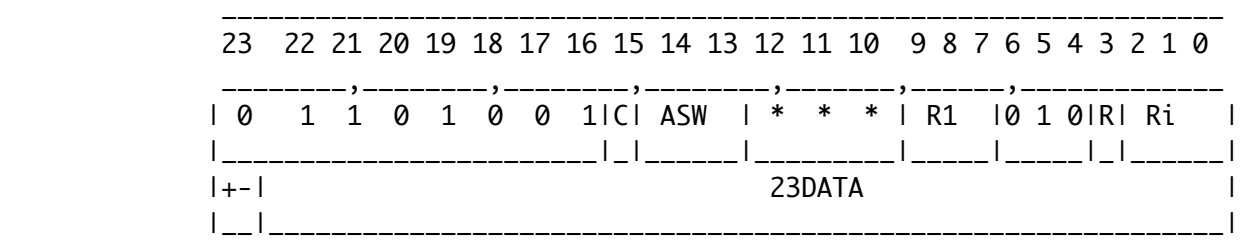
Bit 12-10 word 1--These bits are not used.

Bit 3 word 1-- Relative addressing: 0 = no relative address 1 = relative address

****

```
                _
                A  _
CW R1,+-23DATA(RiS),C
                W

CW Y,SYMDTA(ZA),C
```

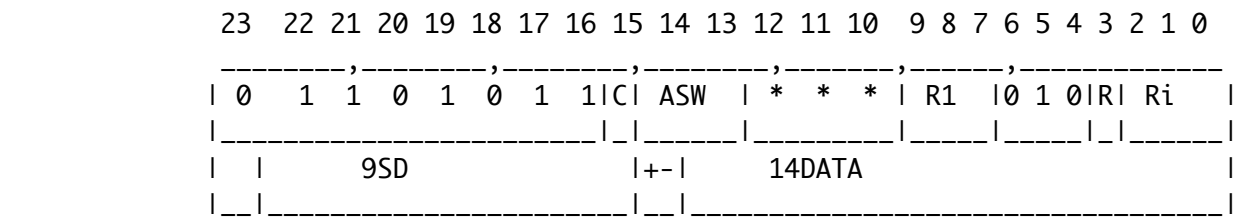
****



****

```
                _  _______
                A  PL
CW R1,+-14DATA(RiS),PCL
                W  C
                   P(9SD)
                   PC(9SD)

CW X,SYMDTA(ZS),P(SYMSD)
```

****



8.5 Execute (EXC)

****
EXC MEMLOC
****

Function	Program control is transferred to the memory location specified. This instruction is executed and program control then returns to the location immediately following the EXC instruction. All input/output interrupts are inhibited for EXC until the completion of the execution of next sequential instruction.
Control Flip-Flops	The CFs are set if the executed instruction does so. (See the particular instruction executed.)
L Register	See the particular instruction executed.
Cycle Time	Base cycle time is 2 cycles plus time at destination. If both relative addressing and indexing are specified, base cycle time is 3 cycles plus time at destination.
Restrictions	If the executed order is an executed transfer, program control does not return to the location following the EXC instruction. It does, however, proceed to the location specified by the transfer. EXC instructions may be nested: that is, the executed order may be an EXC which executes an EXC, etc. If any EXCed instruction processed is a nontransfer or a mismatched conditional transfer, control is returned, after processing, to the instruction immediately following the initial EXC instruction. If the executed order is an executed transfer and specifies a J or T return option, the J or T register receives the address of the location of the instruction immediately

following the EXC order.

If an EXC order uses an index register modification option,
the order which is being executed may not change the contents of the index register. This restriction is not because of hardware design, but it is required to enable the system program to go back to normal after an interrupt.

If the executed order is a SEARCH instruction, a G-level interrupt will occur.

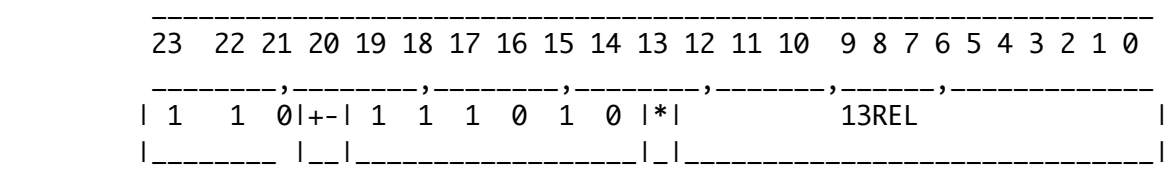
Encoding There are two short forms and one long form.

Binary encodings of the instruction:

SHORT forms:

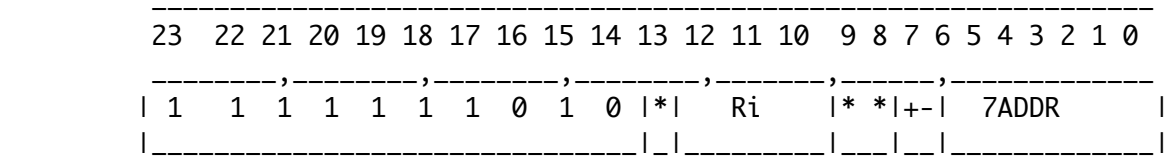
****
EXC +-REL

EXC SYMREL
****



****
EXC +-ADDR(Ri)

EXC SYMADDR(Z)
****



LONG form:

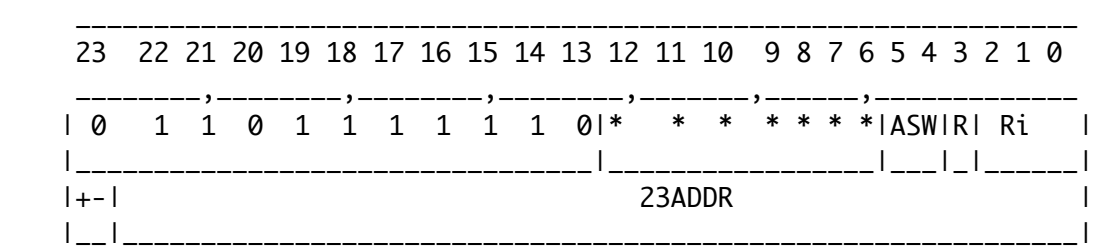
The following bits are encoded to indicate the corresponding conditions:

Bit 3 word 1-- Relative addressing: 0 = no relative address 1 = relative address

****

```
              _
              A
EXC +-23ADDR(RiS)
              W

EXC SYMADDR(ZA)
```

****



8.6 Find Rightmost One (F:T{I})

****

```
F:T  R1R2:DESTINATION

R2 = Position of Rightmost One in R1
IF(R2 == 0) GOTO DESTINATION
or
F:TI  R1R2:DESTINATION

R2 = Position of Rightmost One in R1
IF(R2 == 0) GOTO <DESTINATION>
```

****

Function The bit position of R1 containing the rightmost 1 replaces the contents of R2 and program control proceeds to the next sequential instruction, leaving R1 unchanged. If R1 = R2, R1 receives the bit position of the rightmost 1 in R1. If there is no rightmost 1 (that is, if the contents of R1 is logical zero), R2 is unchanged and program control passes to the instruction

at DESTINATION, or if TI, then the location that is the contents of the DESTINATION.

Control The CFs are set according to the contents of the original R1.

Flip-Flops

L Register The L register value is not affected.

Cycle Time If the contents of R1 is logical zero, base cycle time is 5 cycles. If the contents of R1 is not logical zero, base cycle time is 2 cycles.

Encoding There is one long form.

Binary encodings of the instruction:

LONG form:

The following bits are encoded to indicate the corresponding conditions:

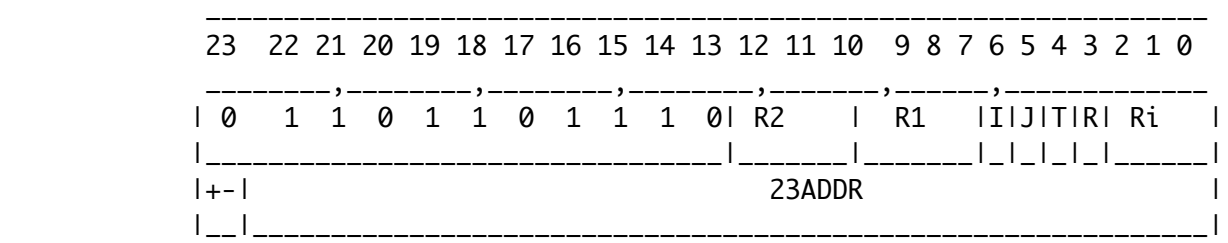
Bit 6 word 1-- Transfer direct or indirect: 0 = direct transfer 1 = indirect transfer Bit 3 word 1-- Relative addressing: 0 = no relative address 1 = relative address

****

F:T R1R2:+-23ADDR(Ri),J
T

F:T XY:SYMADDR(Z),T

****



8.7 Fill (FILL)

****

FILL VALUE
****

Function FILL is a no operation instruction.
Its primary purpose is to provide flexibility in arrangement
of instructions. For example, a FILL can be used to force a long instruction, which would normally begin at an odd address, to begin at an even address, resulting in a possible savings in execution time. If VALUE is specified, it is loaded in the low-order 16 bits
(FILL does not use VALUE).

Control Flip-Flops The CFs are not set.

L Register The L register value is not affected.
If FILL appears at an odd store address and follows a short instruction, the base cycle time is no cycles.
Otherwise, base cycle time is 1 cycle.

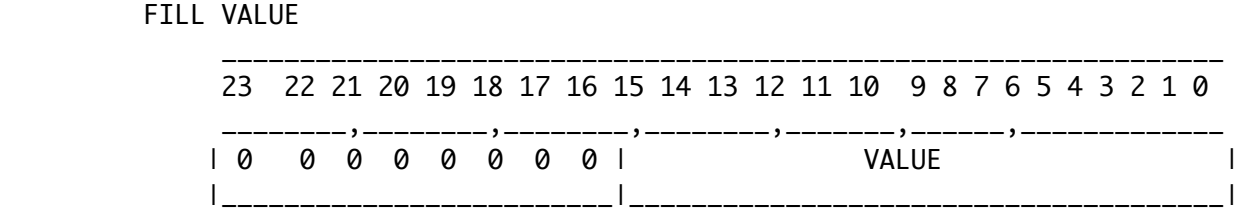
Encoding The only encoding is short.

Binary encodings of the instruction:

SHORT Form:

.Sc "0" 10

FILL VALUE



8.8 Go Back To Normal (GBN)

****
GBN
****

Function The highest level flip-flop which is a 1 is reset to a 0 in the interrupt level activity register.
The contents of the B register and the state of the CFs are restored to their original values before the interrupt occurred. Program control returns to the interrupted

program.

Control	The CFs are restored to their state just prior to the interrupt.
Flip-Flops	
L Register	The L register value is not affected.
Cycle Time	Base cycle time is 5 cycles.
Encoding	The encoding is always short.

Binary encodings of the instruction:

SHORT Form:

****
GBN
****