TABLE OF CONTENTS

INTRODUCTION

The Burroughs Micro Implementation Language (MIL) is a symbolic coding technique that makes available all the capabilities of the B1700 Processor. The MIL compiler's machine language output is ready for execution directly upon the hardware. The user, however, must be prepared to programmatically control the total environment including bootstrap loading, interrupt servicing, and potential machine malfunctioning (e.g., parity error detection).

To use MIL properly and efficiently, the programmer must have an extensive knowledge of the available registers and their capabilities. This product specification describes the registers, the syntax and the semantics of the MIL language and may be used to write programs without prior knowledge of the system.

RELATED DOCUMENTS

A description of the Input/Output subsystem and the I/O descriptors as well as more detailed information about the registers can be found in the B1700 Systems Reference Manual (form 1057155).

BURROUGHS CORPORATION
SANTA BARBARA PLANT

MIL

COMPANY CONFIDENTIAL
P.S. 2212 5298

MICROPROGRAMMING CONCEPTS

Microprogramming is a method for programming a computer hardware architecture. The microprogrammer is concerned with machine registers which were formerly the domain of the hardware systems designer. Strings of microinstructions manipulate those internal registers to present an outward appearance of system hardware which is more functional for problem-oriented programming. In most machines now in the market place, read only memories (ROM's) contain microprograms which convert the unique internal environment of several different processors into a standard assembly language. Once created, the microprograms are unalterable and may contain compromises in efficiency because of a limited hardware instruction set.

The Burroughs B1700 system makes use of the latest technology to implement a writable control memory and has several microprograms, each optimized for the functions it will perform. The virtual system architectures chosen have been those of the standard (COBOL and FORTRAN), problem-oriented, compiler languages. Other microprogrammers may choose architectures and create languages optimized for other purposes.

MICROINSTRUCTIONS

A microinstruction is the smallest programmable operation within the system. Each microinstruction is fetched from control memory and decoded in the (micro) register to be directly executed by the hardware.

DEFINED FIELD CONCEPTS

Special hardware, called a Field Isolation Unit, has been implemented to achieve bit addressability and variable length fields and to automatically increment addresses. This allows maximum flexibility in defining data fields and resolves the problem of packing and unpacking data fields across hardware container boundaries.

INTERPRETATION OF THE VIRTUAL LANGUAGE

The traditional approach to supporting a higher-level language is to translate the source statements as written by the programmer into another language either directly recognized by the hardware, (e.g., machine object code) or easily translatable into the machine object code (e.g., an assembly language). An alternate technique is the interpretive execution for each source statement with a logically equivalent routine in some lower-level language. A microprogrammed system offers the opportunity to combine the best of both methods. The source statements in the higher-level language are translated into a virtual system code by a compilation process. This system code, also called S-code or S-language, very closely resembles the original source language. Microinstruction routines then interpretively execute each virtual language statement. The results are:

- a faster compilation
- a system architecture, as expressed in the set of microroutines, which is optimized to the source language
- a reduction in the processor time required to perform the logical equivalent of each source statement
- a reduction in the memory space required to encode each source language operation

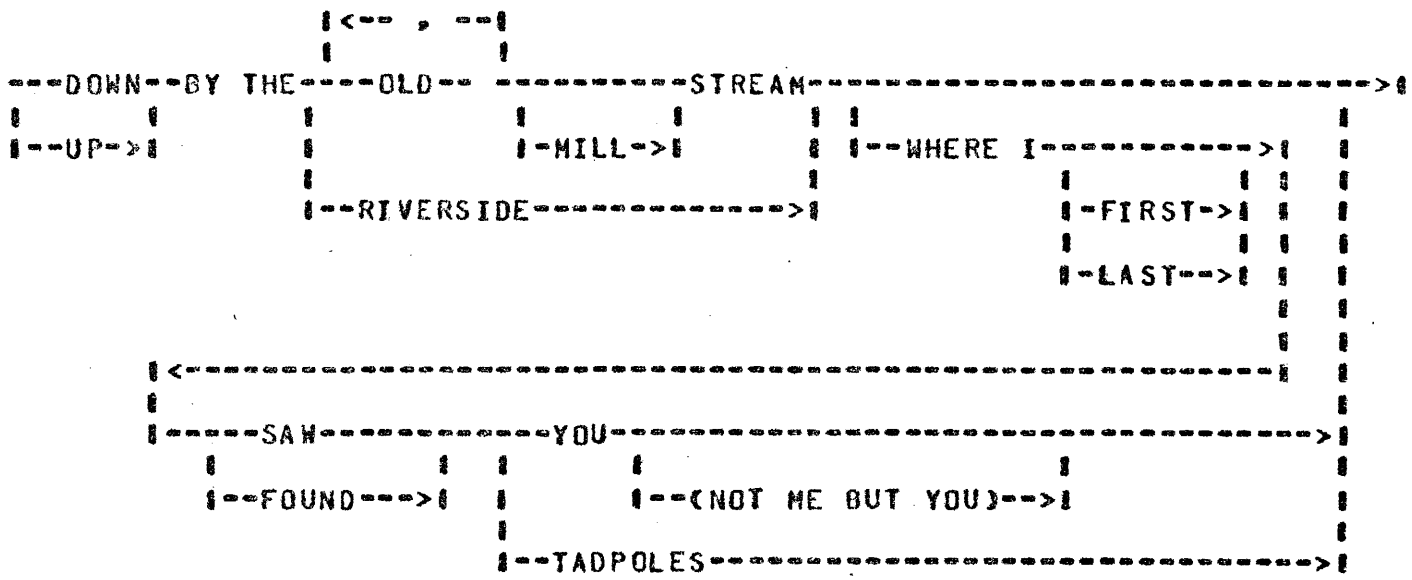
A set of microprogrammed routines is called an interpreter and effectively creates a virtual system architecture for the source language being executed. That is, when the COBOL interpreter is executing, the system is effectively a COBOL machine. When the FORTRAN interpreter is executing, the system is a FORTRAN machine, and so on for any other S-language defined.

SYNTAX DIAGRAMS

The principal means of describing MIL syntax is through the syntax diagram, commonly known as "RAILROAD" notation. The basic conventions are discussed below.

FORWARD ARROWS

Any path traced along the directional flow of the arrows will produce a syntactically valid command. The following example illustrates the technique:



Valid syntax generated from this diagram could be:

DOWN BY THE OLD MILL STREAM
UP BY THE OLD, OLD STREAM
DOWN BY THE RIVERSIDE WHERE I FOUND TADPOLES
DOWN BY THE OLD STREAM WHERE I FIRST SAW YOU (NOT ME BUT YOU)
UP BY THE RIVERSIDE WHERE I LAST FOUND YOU

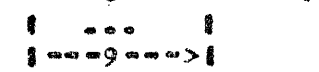
The bridge over OLD, unless otherwise specified, can be crossed any number of times.

END-OF-STATEMENT

The completion of a statement is indicated by the following convention: ----------->!

CONTINUATION

The following convention indicates that any number from 0 THROUGH 9 is syntactically valid:

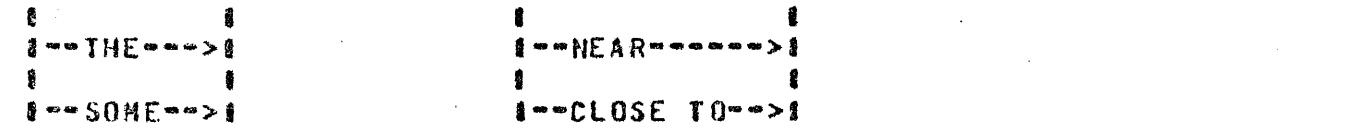


KEYWORDS

Upper-case letters indicate keywords which must literally appear in MIL statements.

VARIABLES

Lower-case letters, words and phrases indicate syntactic variables which require information to be supplied by the programmer. The following example illustrates the technique:



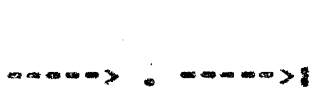
Valid syntax generated from this diagram might be:

THE TADPOLES WERE IN THE STREAM ?
COWS WERE CLOSE TO THE POND ?
SOME BIRDS WERE NEAR THE OCEAN ?

BASIC COMPONENTS OF MIL

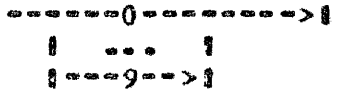
To understand MIL grammar the user should be familiar with the following basic elements of the MIL language.

point:

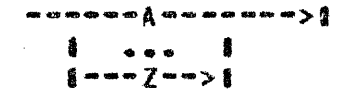


underscore:

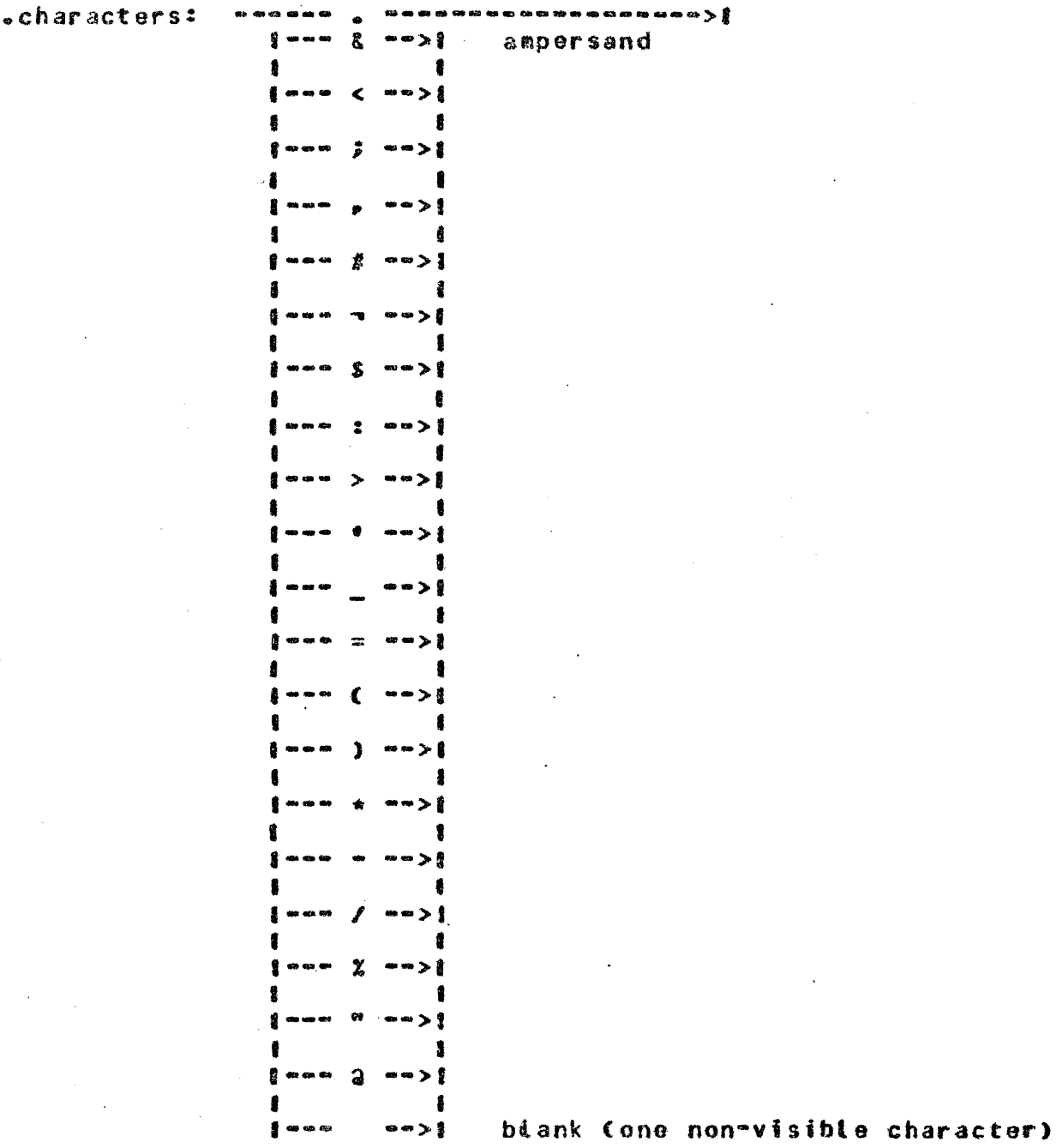
digit:



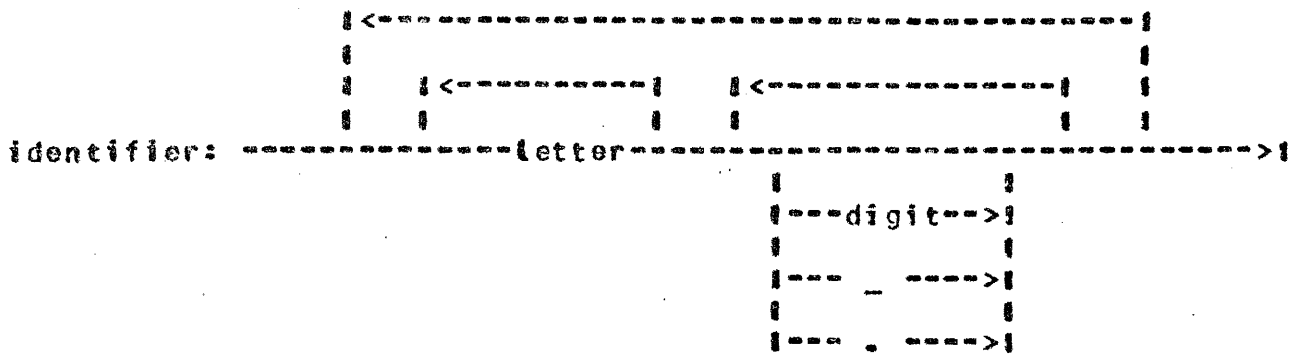
letter:



special.characters:



IDENTIFIERS

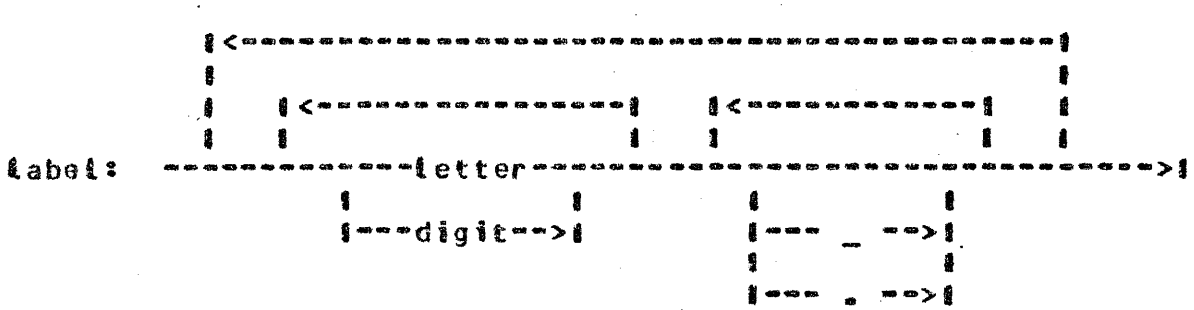


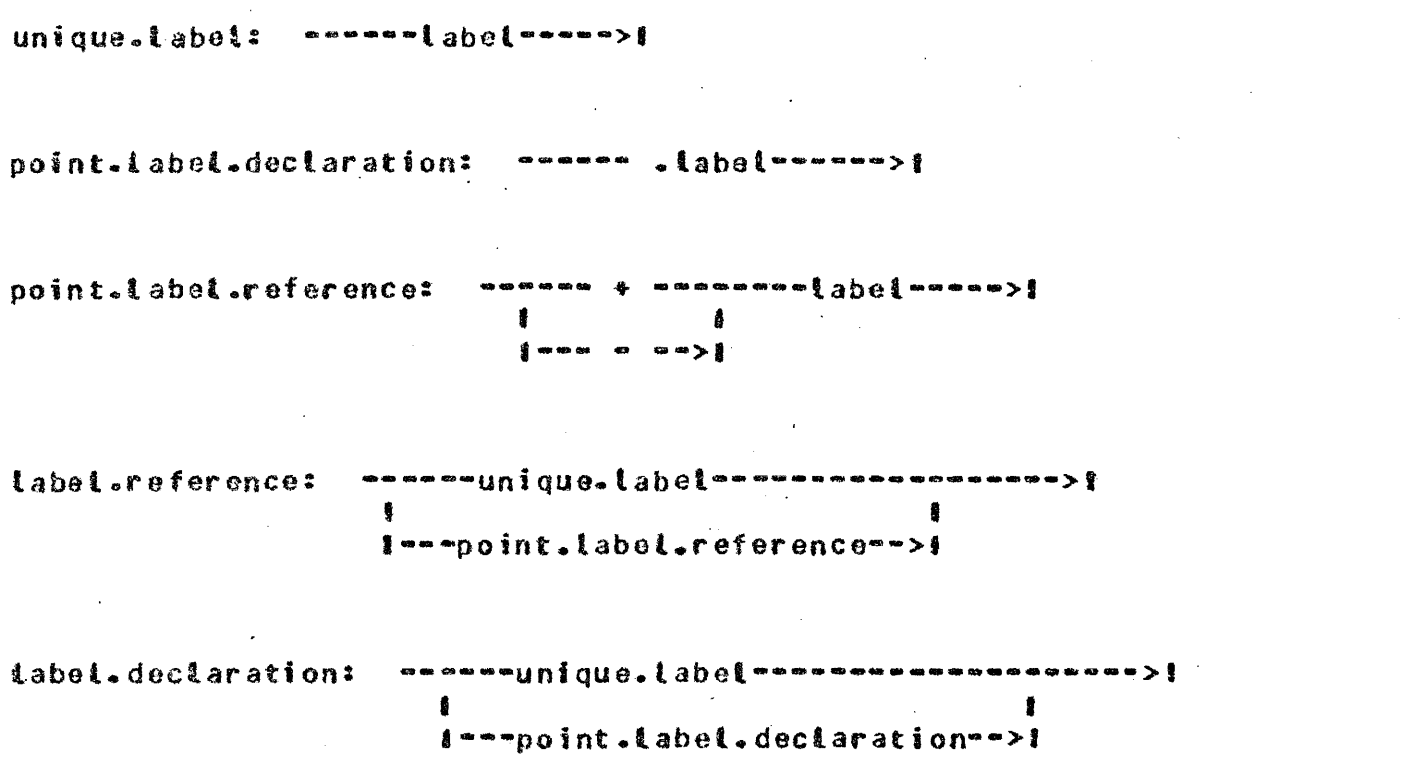
RESTRICTIONS:

1. An identifier must begin with a letter.
2. An identifier may not contain blanks.
3. Reserved words may not be used as identifiers. (See Appendix C: Reserved Words and Symbols.)
4. An identifier is limited to a maximum of 63 characters: only the the first 25 characters are used in uniqueness detection.

EXAMPLES: TEST.NAME.1 T.123.0 ABC LOOP_12

LABELS





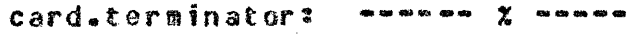
Labels may be declared by: (1) starting the label anywhere in columns 1 through 5 of a source image, or (2) starting the label immediately after the reserved words TABLE, SEGMENT, or CODE.SEGMENT. (See also Segmentation: Label Addresses.)

RESTRICTIONS:

1. A label must begin with a letter or a digit.
2. A label may not contain blanks.
3. A label is limited to a maximum of 63 characters: only the first 25 charcters are used in uniqueness detection.
4. Unique labels may be declared only once.
5. Point labels may or may not be unique.

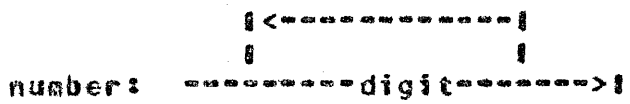
EXAMPLES: .A.POINT.LABEL REGULAR.LABEL LOOP BEGINNING.OF.TEST.1

CARD TERMINATORS

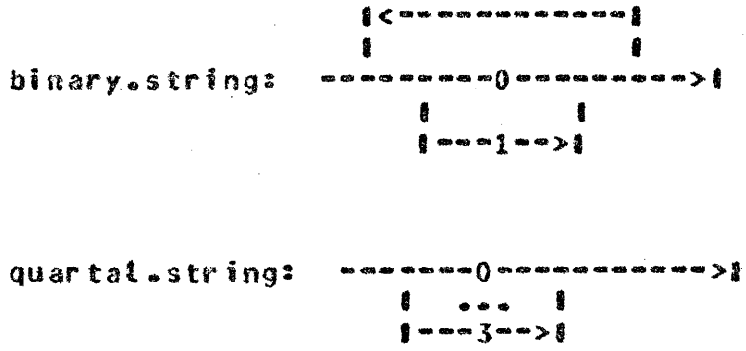


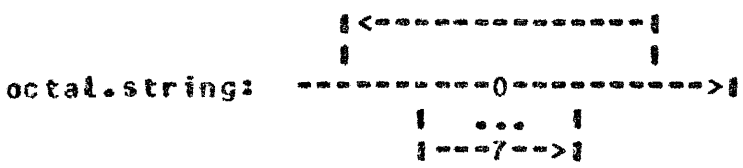
RESTRICTION: A percent sign (%) is treated as any other string character if it is contained within a character.string. However, in all other cases, a % will cause the scanning of the current source image to terminate.

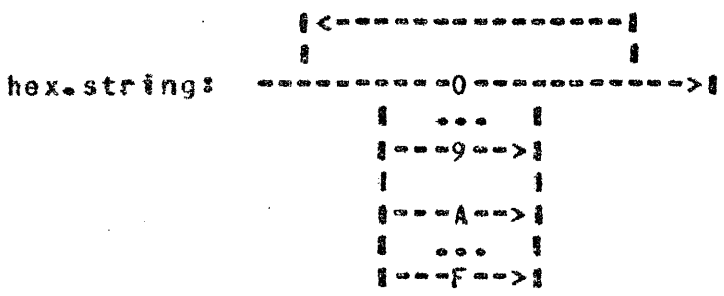
NUMBERS

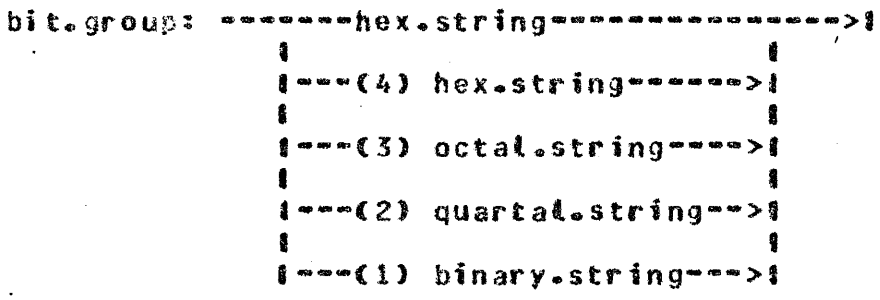


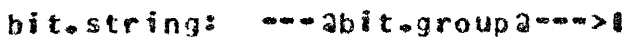
BIT STRINGS



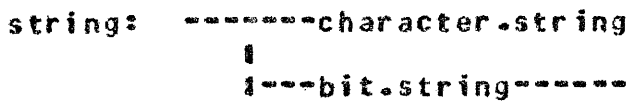

octal.string:
0
...
7


hex.string:
0
...
9
A
...
F

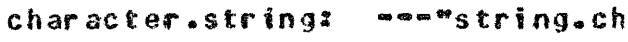

bit.group:
hex.string
(4) hex.string
(3) octal.string
(2) quartal.string
(1) binary.string

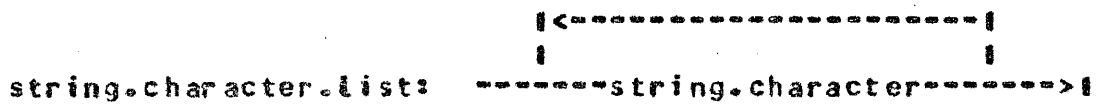


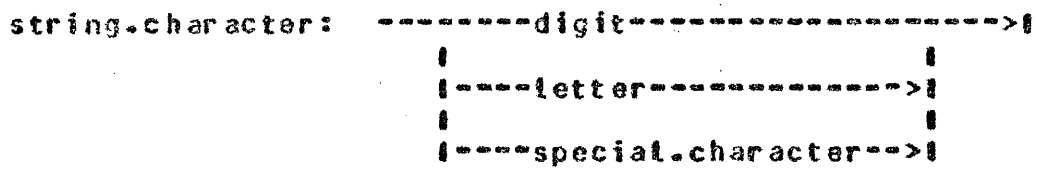
RESTRICTION: If no bit mode is specified (i.e., the indicator digit in parentheses is omitted), then "hex" is assumed.



CHARACTER STRINGS

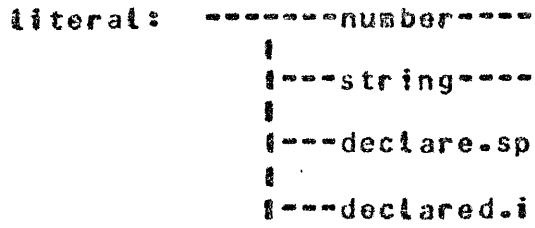


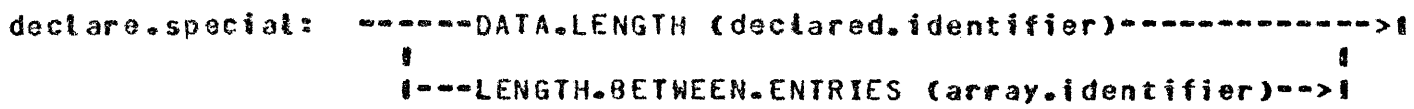


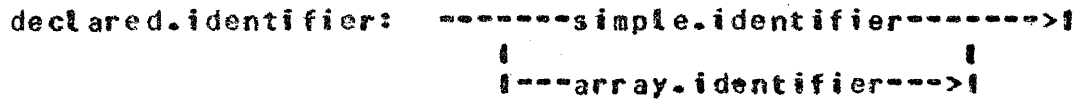


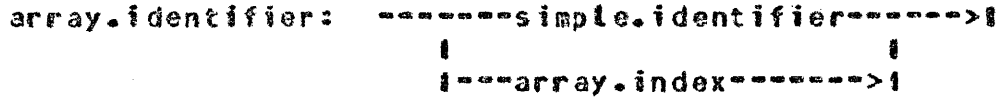
EXAMPLES: "**THIS IS AN EXAMPLE OF A CHARACTER STRING"
"### ROW THE BOAT GENTLY ... "

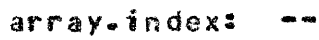
LITERALS











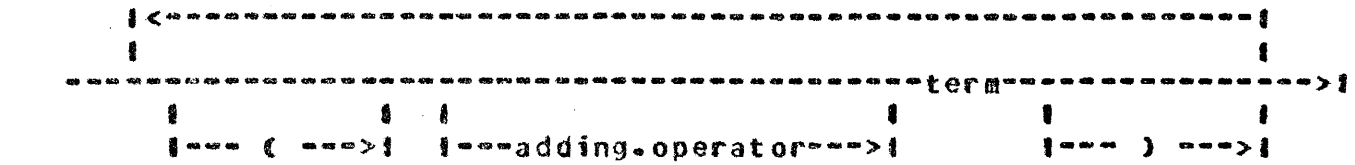
DATA.LENGTH (declared.identifier) will supply the specified or computed length in bits of the indicated declared.identifier. For an array.identifier, the length will be the length of one of the items in the array, not the length of the entire array.

LENGTH.BETWEEN.ENTRIES (array.identifier) will supply the bit difference between the beginning of one item in the specified array and the next item in the array. Note that in the case of structured arrays (See Structured Declarations) this will not always be the same as DATA.LENGTH (array.identifier).

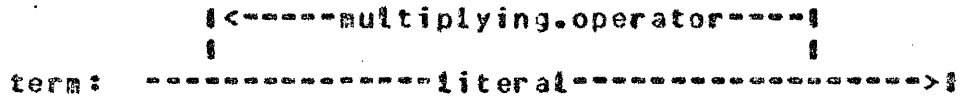
EXAMPLE: 1587
"STRING"
DATA.LENGTH (AN.ITEM)
ARRAY.ELEMENT (7)

ARITHMETIC EXPRESSIONS

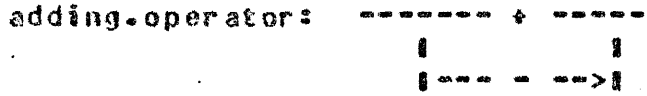
arithmetic.expression:



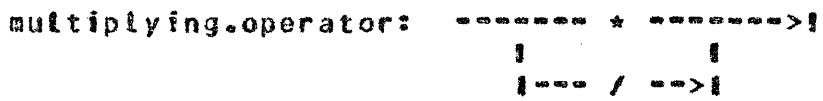
term:



adding.operator:



multiplying.operator:



Arithmetic expressions yield numerical values by combining literals in accordance with specified operations. The operators +, -, *, and / have the conventional mathematical meanings of addtion, subtraction, multiplication, and division, respectively.

The sequence in which operations are performed is determined by the precedence of the operators involved. The order of precedence is:

First: * /

Second: + -

When operators have the same order of precedence, the sequence of operation is determined by the order of their appearance, from left to right. Parentheses can be used in normal mathematical fashion to override the usual order of precedence.

Parenthesized expressions are treated as terms, i.e., they are evaluated by themselves and the resulting value is subsequently combined with the other elements of the arithmetic.expression. Thus the normal precedence of operators may be overridden by careful placement of parentheses.

STRUCTURE OF A MIL PROGRAM

There are two parts or sections to a MIL program: the declarations and the body. The declarations should contain:

1. A comment description of the function of the MIL program.

2. Any global data structures (DECLARES). Note that "global" refers to use throughout the program; local refers to use restricted to a part of the program.

3. Any global DEFINEs.

4. Any MACRO definitions.

The body follows the declarations and will contain all code-producing statements. The statements should be logically grouped in labeled BEGIN...END blocks. Each BEGIN...END block may contain its own local data structures, LOCAL.DEFINEs or labels. The last statement of the body should be FINI.

The following is a basic outline of a MIL program using the above general rules. For specific details on assembly coding forms and program examples refer to: Programming Techniques.

```
                 |--->
                 |          % descriptive comment
Declarations     |          DECLAREs
                 |          DEFINEs
                 |          MACROs
                 |--->

                 |--->
                 |
                 | LABEL.A
                 |          BEGIN A
                 |            (code for A)
                 |          END A
   Body          |          BEGIN B
                 |            (code for B)
                 |          END B
                 |          FINI
                 |--->
```

SEGMENTATION

Segmentation in MIL is a multi-faceted and somewhat complicated subject. Because MIL is the language of the 1700's, and because it is used for many different purposes (Diagnostics, Emulators, Interpreters, I/O Drivers, MCP Kernels, etc.), it must attempt to satisfy the needs of a wide range of users. Segmentation plays a particularly important role on the B1700's because of the READ/WRITE access capability of the hierarchical memory structure (M-Memory, S-Memory, Disk).

LABEL ADDRESSES

To begin the discussion on Segmentation, we must first identify the label types pertaining to address assignment. They are: regular.label and physical.label. (These should not be confused with the two types of label representation: unique.label and point.label. See Basic Components Of MIL: Labels.) The types are based on how the labels are declared which in turn determines how the address of the label is to be assigned.

A label which is declared by starting it in columns 1-5 of a source image is always a regular.label.

A label which is declared by starting it immediately after the reserved words TABLE, SEGMENT, or CODE.SEGMENT is always a physical.label.

A regular.label is always given the current segment.code.address when the label is declared.

A physical.label is always given the current physical.code.address when the label is declared.

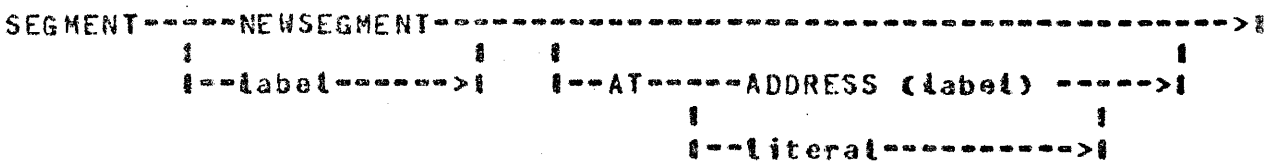
The segment.code.address is updated by 16 as each microinstruction is generated and can be changed to a new value by the appearance of a SEGMENT or CODE.SEGMENT statement.

The physical.code.address is also updated by 16 as each microinstruction is generated and can be changed to a new value by the appearance of an ADJUST LOCATION statement. (See MIL Statements: ADJUST)

Both the physical.code.address and the segment.code.address are initialized to 0 (zero) when a compilation begins.

SEGMENT STATEMENT

SYNTAX:



Note: The literal must be MOD 16.

SEMANTICS:

Through the use of the SEGMENT statement, the user has the means to divide his/her MIL program into several parts such as a single primary.code.block and one or more segment.block(s). The primary.code.block should provide one or more areas suitable for containing the individual segment.block(s). These areas are designated by declaring one or more regular.label(s) somewhere within the primary.code.block. Quite often there will be only one designated area for segment.block(s), and it will begin at the end of the primary.code.block.

The purpose of the SEGMENT statement is to inform the compiler exactly where the segment.block will be (relative to the primary.code.block) when its code is executed. In this way the compiler can generate the correct branch/call displacements whenever a statement in the primary.code.block branches to or calls a routine in one of the segment.block(s). In the same way, a statement in one of the segment.block(s) may branch to or call a routine in either the primary.code.block or in any of the segment.block(s). (See MIL Statements: EMIT.RETURN.TO.EXTERNAL, CALL.EXTERNAL, BRANCH.EXTERNAL.)

All code is assumed to be in the primary.code.block until the first SEGMENT statement is encountered. From this point on, all code is assumed to be in that segment until the next SEGMENT statement is encountered, and so on.

The SEGMENT statement may also be used to specify logical breaks within a continuous stream of code. In this case, only the name or the segment needs to be specified since the code addresses are to continue linearly. The entire program and all of the segment.block(s) are given segment dictionary entries as part of the parameter blocks associated with a MIL code file. From these dictionary entries, the addresses and lengths for each segment are available and can be used to do sophisticated static binding prior to execution of the code.

CODE.SEGMENT STATEMENT

SYNTAX:

SEMANTICS:

Another form of Segmentation in MIL is used when a microprogram is running with the MCP, or under MCP control. All of the interpreters as well as GISMO are examples of this situation. With this mechanism, a microprogrammer is able to specify which portions of the program are to reside on disk until they are actually needed for execution. This provides the programmer with the same facility normally only found in higher level languages.

In order to use this facility, the programmer must follow certain rules and remember some restrictions. First, some definitions:

main.code.block: all code generated until the first CODE.SEGMENT statement is encountered: this may encompass the primary.code.block and one or more segment.block(s).

external.code.block: all code generated between a given CODE.SEGMENT statement and the next CODE.SEGMENT statement, or the end of the program, whichever comes first.

main.code.base: the M-Memory bit address of the first microinstruction in the main.code.block. If no part of the main.code.block resides in M-Memory, then the main.code.base should be 0.

If the processor is an S-Memory processor, then the main.code.base should be the memory address of the first microinstruction in the program. (See MIL Statements: MAIN.CODE.BASE.)

mbr.topm: a 24-bit bucket containing the MBR value for the main.code.block. In addition, since the MBR value is always a MOD 16 number, the low order 4 bits mbr.topm should be the TOPM value of the main.code.block.

The microprogrammer must provide the following items in a program:

1. A define for MAIN.CODE.BASE to indicate the Scratchpad register containing main.code.base. Example:

```
DEFINE MAIN.CODE.BASE = S148#
```

2. A define for MBR.TOPM to indicate the Scratchpad register containing mbr.topm. Example:

```
DEFINE MBR.TOPM = S15A#
```

The above defines must be included in the main.code.block and must not be defined within some LOCAL.DEFINE scope. In addition, the two Scratchpad locations must be initialized by the interpreter when it is given control from GISMO.

3. A routine labeled GO.TO.EXTERNAL.SEGMENT to interrogate the interpreter dictionary and generate a communicate (if necessary) to guarantee that the requested external.code segment is present in S-Memory. In addition, it must perform the initial transfer to the external.code.segment. Example:

```
GO.TO.EXTERNAL.SEGMENT
  % T CONTAINS SEGMENT NUMBER
  % L CONTAINS BIT DISPLACEMENT WITHIN SEGMENT
SHIFT T LEFT BY 6 BITS TO X      % T * 64
SHIFT T LEFT BY 4 BITS TO Y      % T * 16
MOVE SUM TO FA                   % T * 80
ADD ADDR.INTERP.SEG.DICT TO FA
READ 2 BITS TO X
IF LSUX THEN                     % THE SEGMENT IS PRESENT
  BEGIN PRESENT
       COUNT FA UP BY 32
       READ 24 BITS TO X         % SEGMENT BASE ADDRESS
       IF SUBSET THEN INCLUDE    % FOR S-MEMORY PROCESSORS
     BEGIN
          MOVE L TO Y
          MOVE SUM TO A
     END ELSE
     BEGIN
          MOVE 0 TO TAS          % NECESSARY FOR
                                 % M-MEMORY SYSTEM
          MOVE L TO T            % NEW A AND TOPM VALUE
          MOVE X TO L            % NEW MBR VALUE
          TRANSFER.CONTROL
     END
  END PRESENT
MOVE T TO L
MOVE 58 TO T                     % COMMUNICATE NO.FOR
                                 % NON PRESENT SEGMENT
SHIFT T LEFT BY 16 BITS
SET L(0)                         % ONE LEVEL SEG DICT.
GO TO GIVE.UP.CONTROL.           % SAVE STATE AND XFER TO
                                 % MCP VIA GISMO
```

POINTS TO NOTE:

A. The initial "T" and "L" values are supplied by the compiler prior to entering the above routine.

B. Other registers may be destroyed depending on how the routine is written.

C. The routine must push a 0 (zero) onto the A stack for the M-Memory Processor. This is necessary so that an exit within an external.code.block can be trapped into a routine that will transfer control back to the main.code.block. This also implies that parameters may not be passed via the A stack when initially transferring to an external.code.block.

The compiler will provide all other routines necessary to effect the transfer to and from external.code.block(s).

The only kind of transfers allowed are calls and branches from the main.code.block to an external.code.block, and from an external.code.block to the main.code.block. Transfers between external.code.block(s) are not allowed. In addition, such calls and branches must be syntactically separated from calls and branches within the the same code block. Instead of CALL, the command CALL.EXTERNAL must be used. Instead of GO, the command BRANCH.EXTERNAL must be used. (See MIL Statements: EMIT.RETURN.TO.EXTERNAL, CALL.EXTERNAL and BRANCH.EXTERNAL.)

Following is the code the compiler generates when CODE.SEGMENTs are used. (All labels used in the examples are shown for clarity only: the compiler has its own internal representation for the labels.)

MAIN CODE BLOCK

A. For each different label occurring after a CALL.EXTERNAL or BRANCH.EXTERNAL statement in the main.code.block, the compiler will divert the call or branch to the following code which is generated at the end of, and part of, the main.code.block:

```
MOVE ADDRESS (label) TO L
MOVE label.segment.number TO T
GO TO GO.TO.EXTERNAL.SEGMENT
```

B. If the program executes on an M-Memory Processor (B1726), the following code will be emitted in the main.code.block:

```
EXIT.TO.EXTERNAL
     MOVE TAS TO L
     MOVE TAS TO T
     MOVE LF TO TF
     MOVE 0 TO LF
     TRANSFER.CONTROL
```

EXTERNAL CODE BLOCK

A. If the program executes on an M-Memory Processor (B1726), the following code will be emitted at the beginning of every external.code.block:

```
        MOVE TAS TO T
    LEAVE.EXTERNAL.SEGMENT
        MOVE MBR.TOPM TO L
        MOVE LF TO T
        SET LF TO 0
        TRANSFER.CONTROL
```

B. For each different label occurring after a CALL.EXTERNAL or BRANCH.EXTERNAL statement in the external.code.block, the compiler will divert the call or branch to the following code which is generated at the end of, and part of, the external.code.block.

(1) If the program executes on an S-Memory Processor (B1712-B1714) the following code is generated:

```
MOVE ADDRESS (label) TO X
GO TO SUBSET.BRANCH.TO.MAIN
```

(2) If the program executes on an M-Memory Processor (B1726) the following code is generated for each different label in a BRANCH.EXTERNAL statement:

```
MOVE ADDRESS (label) TO X
GO TO BRANCH.TO.MAIN
```

(3) If the program executes an an M-Memory Processor (B1726) the following code is generated for each different label in a CALL.EXTERNAL statement:

```
MOVE ADDRESS (label) TO X
GO TO CALL.TO.MAIN
```

C. At the end of every external.code.block the following code is emitted.

(1) For S-Memory Processors (B1712-B1714):

```
SUBSET.BRANCH.TO.MAIN
        MOVE MAIN.CODE.BASE TO Y
        MOVE SUM TO A
```

(2) For M-Memory Processors (B1726):

```
BRANCH.TO.MAIN
     MOVE TAS TO NULL
     MOVE MAIN.CODE.BASE TO Y
     MOVE SUM TO T
     GO TO LEAVE.EXTERNAL.SEGMENT

CALL.TO.MAIN
     MOVE MAIN.CODE.BASE TO Y
     MOVE SUM TO T
     MOVE MBR TO L
     MOVE TOPM TO LF
     MOVE L TO TAS
     MOVE ADDRESS (EXIT.TO.EXTERNAL) TO X
     MOVE SUM TO TAS
     GO TO LEAVE.EXTERNAL.SEGMENT
```

NOTES:

1. When branching from the main.code.block to an external.code.block T and L registers are used, plus whatever registers the GO.TO.EXTERNAL.SEGMENT routine uses.

2. When calling or branching to a routine in the main.code.block, the X and Y registers are used: This means that they cannot also be used for passing parameters. In addition CP should be equal to 24, otherwise the transfer may not take place correctly.

Also, on an M-Memory Processor, the T and L registers, as well as the A stack are used. So a good rule of thumb is to avoid X, Y, T, L, and TAS when passing parameters to/from the main.code.block and external.code.block(s).

3. The code for S-Memory Processors is different than the code for M-Memory Processors. Thus, if a programmer wishes to run a program on either processor without recompilation, CODE.SEGMENTs cannot be used. (See Appendix A: $ NO EXTERNAL).

DECLARATIONS

DATA TYPES

Three main types of data fields may be declared in MIL:

1) BIT
2) CHARACTER
3) FIXED

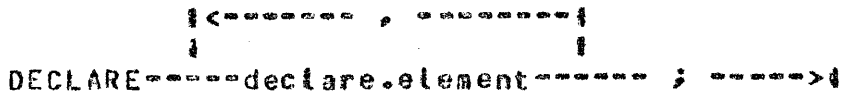
A bit field consists of a number of bits specified by a number in parentheses following the reserved word BIT.

A character field consists of a number of characters, 8 bits each, specified by a number in parenthesis following the reserved word CHARACTER.

A fixed data field is the same as a BIT(24) field but is allowed in order to keep declare syntax consistent with SDL.

DECLARE STATEMENT

SYNTAX:



SEMANTICS:

The DECLARE statement specifies the addresses and characteristics of contents of memory storage areas.

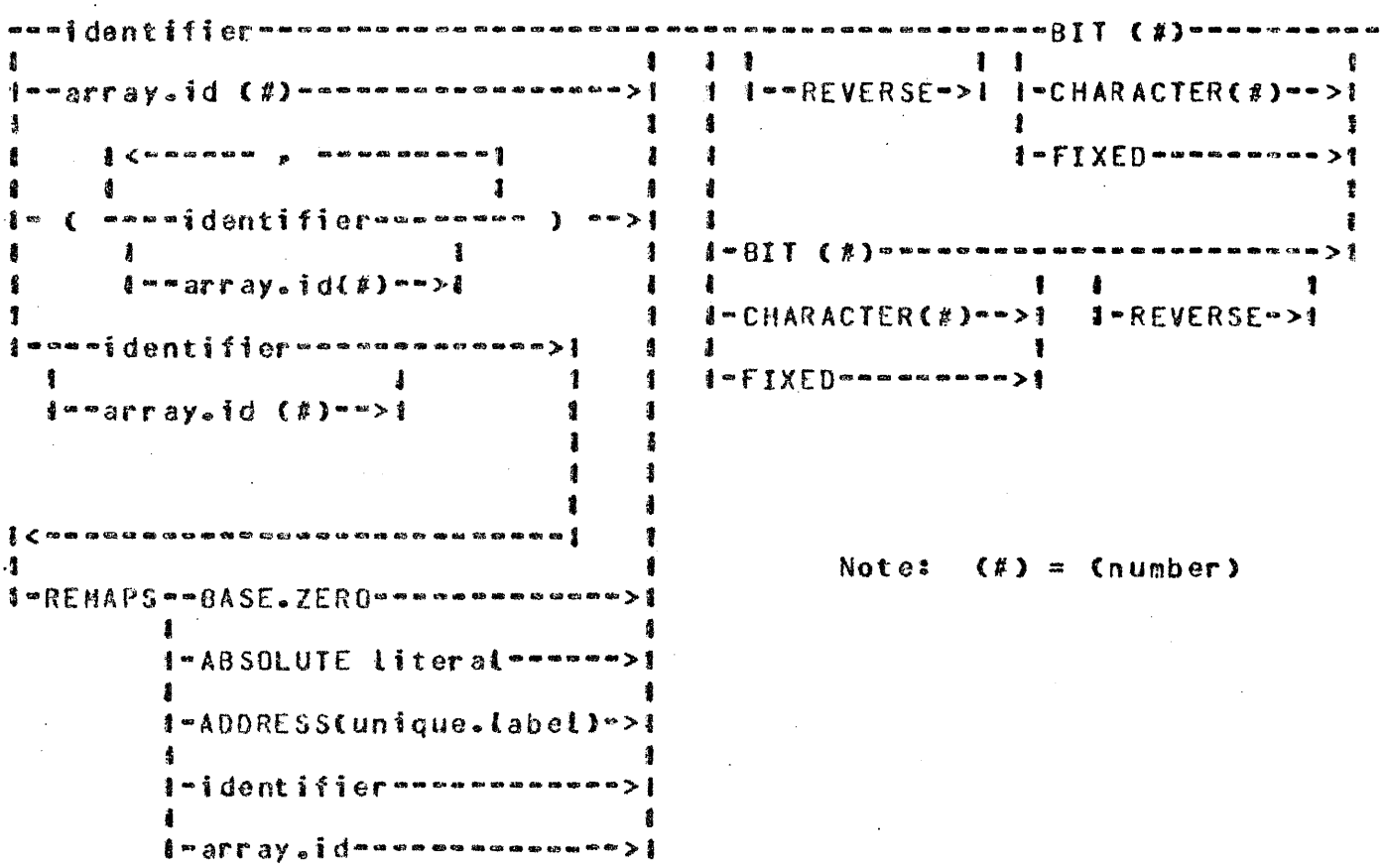
The maximum number of data elements (including fillers, dummys, and implicit fillers) contained in one structure is 50. Any attempt to declare more will cause a table overflow error to be detected at compile time.

An array may have a maximum of 65535 elements, each being a maximum of 65535 bits (8191 characters).

The two types of declare.elements are each discussed below.

NON-STRUCTURED DECLARATIONS

declare.element:



Data may be declared as simple, having one occurrence; or as subscripted, having as many occurrences as specified by the array bound (number).

BIT, CHARACTER or FIXED specifies the type of data in the field and the field size.

REVERSE specifies that an item or a structure is to be accessed in a reverse manner or in a reverse direction from some base. The easiest way to remember what is happening is to realize that the compiler will simply compute the address of a declared identifier normally, and then, if reverse is specified, add the identifier's length to the address to get the ending address of the identifier.

As the syntax indicates, different data fields having the same format may be declared collectively inside parenthesis ( ).

The following examples illustrate the various options available in this type of declaration statement.

```
DECLARE B FIXED,
        C CHARACTER (10),
        D BIT (40),
        (E, F, G (5)) BIT (10),
        H (20) FIXED,
        I (5) CHARACTER (6);
```

where

B is a 24 bit numeric field

C is a 10 byte character field.

D is a 40 bit field.

E and F are 10-bit fields each.

G is also a 10-bit field and occurs 5 times.

H occurs 20 times and is a 24-bit numeric field.

I is a 6-byte character field occurring 5 times.

Data fields may be re-formatted by the use of the REMAPS option. Remapping is subject to the same general rules discussed above. The following example best illustrated its use:

```
B FIXED, C BIT (50),
BB REMAPS B CHARACTER (3),
CC(2) REMAPS C FIXED;
```

Note that CC specifies 48-bits (or 2 elements, 24-bits each). The last two bits will be considered as an implied filler by the compiler. A field may not be remapped larger than its original size.

There is no limit on the number of times a field may be remapped. A field which has remapped another may itself be remapped. The remap option specifies that the identifier on the left side of the reserved word REMAPS will have the same starting address as the identifier on the right side.

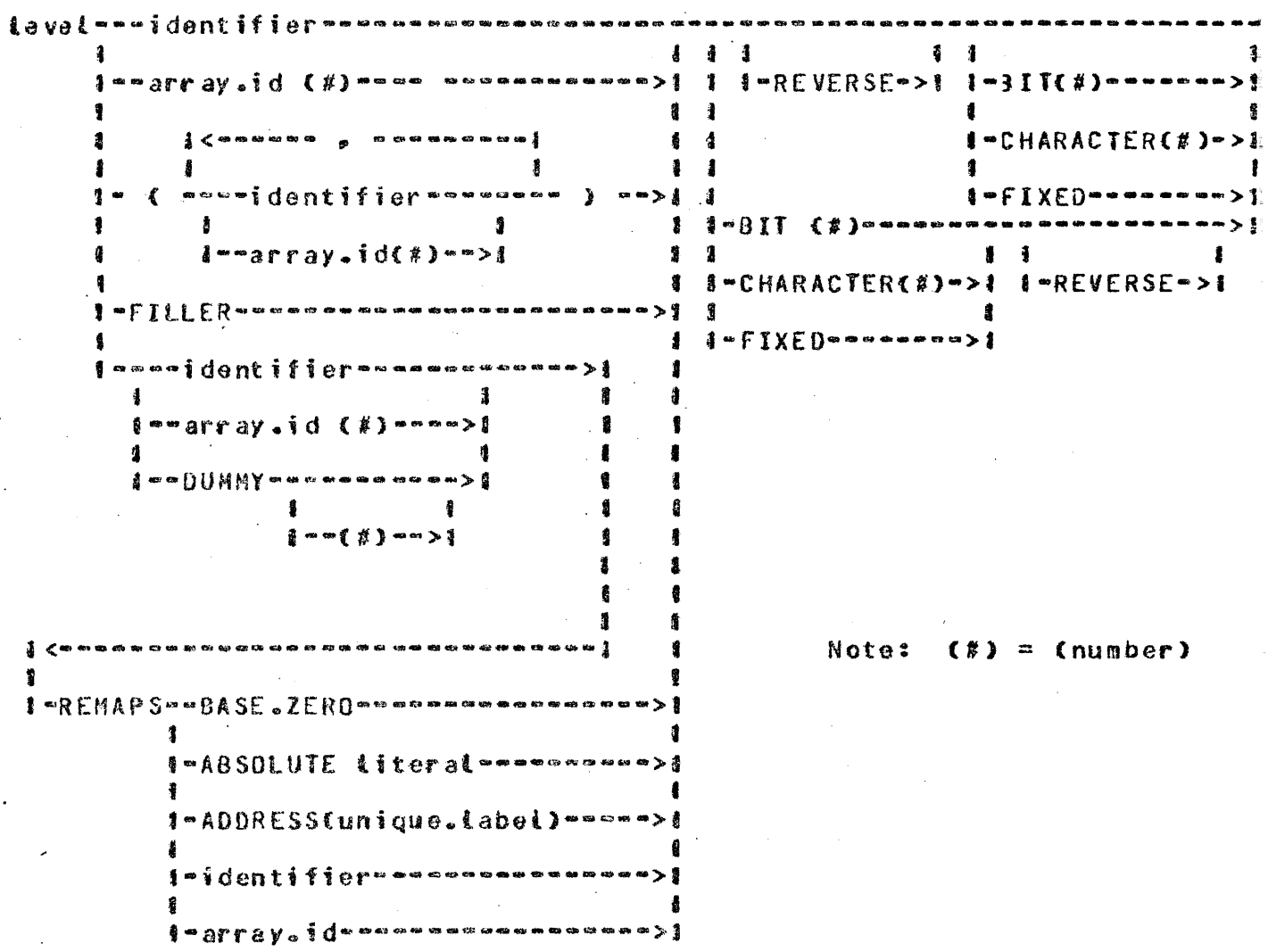
A data field may be remapped to BASE.ZERO which will give the field a relative address of zero. For example:

```
DECLARE X REMAPS BASE.ZERO BIT(7);
```

This device is used as a free-standing declaration since it does not remap a previously declared data item.

STRUCTURED DECLARATIONS

declare.element:



MIL allows the structuring of data where a field may be subdivided into a number of sub-fields, each of which has its own identifier. The whole structure is organized in a hierarchical form, where the most general declaration is a level 01(or 1). No declaration may be on a level greater than 99. A subdivided field is called a Group Item, and a field not subdivided is known as an Elementary Item.

The type and length of data need not be specified on the group level. All Elementary Items must indicate type and length; the compiler will assume type bit and add the lengths of the components to determine the length of the Group Item. Note that the length of the Group Item is the sum of its Elementary Items.

In the following example, both B and C are considered Group Items: B has a total length of 90 bits; C is 50 bits long.

```
DECLARE 01 B,
           02 C,
              03 D BIT(20),
              03 E BIT(30),
           02 D CHARACTER(5);
```

Fillers may be used to designate certain Elementary Items which the program does not reference. If the filler is the last item in a structure, it may be omitted: the compiler will consider the item to be an implied filler. A filler may never by used as a Group Item.

If the 01 level Group Item is an array, it is mapped as a contiguous area in memory. However, subdivisions of this array are not contiguous as shown in the example structure below:

```
01 B(5) BIT(48),          01 B(5),
   02 C FIXED,      or        02 C FIXED,
   02 D FIXED;                02 D FIXED;


    !--> (each item of B) 48 bits
    !
    !
-------------------------------------
   B0      B1      B2      B3      B4
 C0  D0  C1  D1  C2  D2  C3  D3  C4  D4
-------------------------------------
  !
  !
  !--> (each element of C and D) 24 bits
```

If a Group Item is an array, an array specification may not appear in any subordinate item; that is, only one-dimensional arrays are allowed. An array specification is implied for all subordinate items.

If a Group Item is declared with the REVERSE option, then REVERSE is also implied for all subordinate items in that group. Specification of the REVERSE option for subordinate items would be redundant.

Structured data may be remapped in the same manner as non-structured data. In addition, structured data may be remapped with a dummy group identifier. The purpose of this construct is to allow the user to remap data items without having to declare another Group Item which describes the same area in memory. Thus in the following example:

```
01 B BIT(100),
   02 C BIT(20),
   02 D BIT(80);
```

"B" might be remapped as

```
01 BB REMAPS B BIT(100),
   02 CC BIT(30),
   02 DD BIT(70);
```

or

```
01 DUMMY REMAPS B BIT(100),
   02 CC BIT(30),
   02 DD BIT(70);
```

Both B and BB refer to the same area in memory: hence BB is redundant.

If a remapped item contains the REVERSE option, then REVERSE is also implied for the remapping item.

The user should note the distinction between DUMMY and FILLER. DUMMY is used in conjunction with REMAPS to eliminate the necessity of declaring a redundant Group Item. FILLER is used if one desires to skip over a part of the structure.

The following restrictions apply to the use of DUMMY REMAPS:

1. DUMMY may only be used with remap declarations.
2. All restrictions applying to REMAPS apply to DUMMY REMAPS.
3. DUMMY must not remap another DUMMY.
4. DUMMY Group Items must have at least one non-filler component.

DECLARE EXAMPLES

The following will illustrate by example exactly how declarations might be used in a MIL program. (See also Programming Techniques.)

```
%       THE DECLARE STATEMENT IN MIL IS ONE WHICH ALLOWS THE USER TO
%   LOGICALLY ASSIGN NAMES TO PHYSICAL OR RELATIVE MEMORY ADDRESSES
%   IN A STRUCTURED MANNER.  THIS FACILITY ALLOWS ONE TO CONSTRUCT
%   DATA STRUCTURES IN A FORMAT THAT IS SIMPLE TO UNDERSTAND AND EASY
%   TO CHANGE WHEN THE OCCASION ARISES.
%
%                          "NON.REMAP ITEMS"
%
%       THE MIL COMPILER MAINTAINS A VARIABLE WHICH IS INITIALIZED TO
%   0. WHEN AN ITEM IS DECLARED, IT IS ASSIGNED THE CURRENT VALUE OF
%   THE VARIABLE AND THE VARIABLE IS INCREMENTED BY THE BIT LENGTH
%   OF THE DECLARED ITEM.  EXAMPLE:
%
        DECLARE
            DISPATCH.REGISTER       BIT(24),
            GLOP1                   BIT(48),
            ADDR.GISMO              BIT(24),
            LOCN.MAKE.NCP.BE.HERE   BIT(33),
            GLOP2                   BIT(29),
            ADDR.MCP.LIMIT          FIXED;
%
%       NOTE THAT THE DECLARE STATEMENT IS COMPLETELY FREE FORM,
%   MUST BEGIN WITH THE WORD "DECLARE", MUST END IN A ";", AND
%   EACH ELEMENT MUST BE SEPARATED BY A ",".
%       EACH ELEMENT THUS DECLARED IS USED EXACTLY LIKE A LITERAL
%   AND MOST OFTEN REPRESENTS A MEMORY ADDRESS.
%   EXAMPLE:
%
        MOVE ADDR.GISMO TO FA
        READ 24 BITS TO X
%
%       WOULD ASSIGN THE LITERAL 72(=24+48=ADDR.GISMO) TO REGISTER
%   FA, AND WOULD CAUSE THE CONTENTS OF MEMORY AT ADDRESS 72 TO BE
%   READ INTO REGISTER X.
%       NOW, ANOTHER DECLARE LIKE THE ONE ABOVE WILL SIMPLY START
%   ASSIGNING ADDRESSES WHERE THE LAST ONE LEFT OFF.     EXAMPLE:
%
        DECLARE
            GLOP3                   BIT(10),
            CHAR.SAVE.AREA          CHARACTER(8);
```

```
%
%       DECLARE ELEMENTS MAY ALSO BE STRUCTURED SUCH THAT SOME
%   NAMES OVERLAP PIECES OF MEMORY DESCRIBED BY OTHER NAMES.
%
        DECLARE
             01 TRACE.BITS              BIT(27),
                05 FILLER               BIT(15),
                05 NS.HALT.ONLY         BIT(1),
                05 TB.FLAGS             BIT(4),
                05 TB.TYPE              BIT(4),
                   07 FILLER            BIT(1),
                   07 TB.STORES.ONLY    BIT(1),
                   07 TB.BRANCHES       BIT(1),
                   07 TB.REST           BIT(1),
                05 TB.GET.SPACE.TYPE,
                   99 ( TBGS.STORES.ONLY,
                        TBGS.BRANCHES,
                        TBGS.REST)      BIT(1);
%
%       THE ABOVE EXAMPLE ILLUSTRATES SEVERAL POINTS.
%
%            1.  THE ADDRESS PICKS UP WHERE THE PREVIOUS
%                DECLARE LEAVES OFF.
%                NOTE HOWEVER THAT THIS IS NOT TRUE
%                IF THE PREVIOUS ITEM OR STRUCTURE IS A
%                "REMAP ITEM". THE COMPILER'S INTERNAL
%                VARIABLE USED FOR DEFAULT ADDRESS
%                ASSIGNMENT IS MAINTAINED AND
%                INCREMENTED ONLY FOR NON.REMAP ITEMS
%                OR STRUCTURES.
%
%            2.  DECLARES MAY BE STRUCTURED SO THAT SOME  FIELDS ARE
%                DENOTED AS BEING CONTAINED WITHIN OTHER FIELDS.
%
%            3.  "FILLER" CAN BE USED IN STRUCTURES AS OFTEN AS
%                NECESSARY AS A PLACE HOLDER.
%
%            4.  ITEMS WITH THE SAME TYPE AND LENGTH CAN BE PUT
%                INTO A LIST, SURROUNDED BY PARENTHESIS, WITH
%                THE TYPE AND LENGTH SPECIFIED AT THE END.
%
%            5.  THE LENGTH OF AN ITEM NEED NOT BE SPECIFIED IF
%                IT HAS SUB ITEMS WHOSE LENGTHS CAN BE DETERMINED.
%
%       NOTE:  STRUCTURES MUST BEGIN WITH AN "01" LEVEL IDENTIFIER.
%              SUBSTRUCTURES MAY THEN HAVE ANY LEVEL FROM 02-99,
%              WITH THE SUBSTRUCTURE ALWAYS HAVING HIGHER LEVEL
%              NUMBERS THAN ITS SUPERSTRUCTURE.
%
```

```
%                          "REMAP ITEMS"
%
%       IT IS POSSIBLE TO TEMPORARILY SUSPEND THE MECHANISM WHICH
%   CAUSES ADDRESSES TO BE ASSIGNED BASED ON WHERE THE LAST DECLARE
%   LEFT OFF BY USING REMAPS STRUCTURES.
%       FOR EXAMPLE, IF WE WISH TO DECLARE A "TEMPLATE", WHERE THE
%   DECLARED ADDRESSES  ARE ADDED TO SOME BASE PRIOR TO THEIR USE,
%   THEN WE WOULD DO THE FOLLOWING.
%
        DECLARE
            01 SYSTEM.DESCRIPTOR REMAPS BASE.ZERO,
               02 SY.MEDIA              BIT(2),
               02 SY.LOCK               BIT(1),
               02 SY.IN.PROCESS         BIT(1),
               02 SY.INITIAL            BIT(1),
               02 SY.FILE               BIT(1),
               02 FILLER                BIT(10),
               02 SY.TYPE               BIT(4),
               02 SY.ADDRESS            BIT(36),
                  03 FILLER             BIT(12),   % PORT,CHAN,UNIT
                  03 SY.CORE            BIT(24),
               02 SY.LENGTH             BIT(24);
%
%       ONE MIGHT USE THE ABOVE STRUCTURE AS FOLLOWS:
%
        DEFINE SYS.DESC.BASE = S14A#
    %
        MOVE SY.TYPE TO FA
        ADD SYS.DESC.BASE TO FA
        READ DATA.LENGTH(SY.TYPE) BITS TO X
    %
%
%       NOTE THE USE OF A NEW RESERVED WORD "DATA.LENGTH".
%   THIS CONSTRUCT ALLOWS ONE TO USE THE LENGTH OF A DECLARED ITEM
%   WITHOUT HAVING TO DEFINE IT ELSEWHERE.
%
%
%       THE REMAP STRUCTURES THAT MAY BE USED ARE:
%            1. REMAPS BASE.ZERO
%            2. REMAPS ABSOLUTE literal
%            3. REMAPS ADDRESS(some.label)
%            4. REMAPS BASE.ZERO REVERSE
%
%       IF ONE KNEW THE ABSOLUTE ADDRESS OF SOME DATA STRUCTURE
%   IN MEMORY, THE FOLLOWING COULD BE DONE:
```

```
%
          DECLARE
               01 SAVE.AREA REMAPS ABSOLUTE 1024,
                  02 SA.FIRST.ITEM     FIXED,
                  02 SA.SECOND.ITEM    CHARACTER(200),
                  02 SA.THIRD.ITEM     BIT(256);
%
%         OR, IF A LABEL THAT WAS THE START OF A TABLE OF CONSTANTS
%    WAS PRESENT IN THE PROGRAM, THE FOLLOWING WOULD BE SUITABLE.
%
          DECLARE
               01 TRACE.TABLE(10) REMAPS ADDRESS(TRACE.MNEMONICS),
                  02 ADDR.TRACE.NAME    CHARACTER(4);
     %
TRACE.MNEMONICS
          TABLE
          BEGIN
               "LA  "
               "ALA "
               "STN "
               "STO "
               "LIT "
               "ILA "
               "STO "
               "CASE"
               "IFTH"
               "IFEL"
          END
     %
          MOVE ADDR.TRACE.NAME(2) TO FA
          READ 24 BITS TO X INC FA
          READ 8 BITS TO Y
     %
%
%         NOTE THE USE OF ARRAYS IN THE ABOVE EXAMPLE.  IF THE
%    PROGRAMMER DOES NOT KNOW THE INDEX TO USE AT COMPILE
%    TIME, THE FOLLOWING COULD BE DONE:
%
          DEFINE TRACE.INDEX = SOB#
     %
          MOVE TRACE.INDEX TO X
          MOVE LENGTH.BETWEEN.ENTRIES(TRACE.TABLE) TO Y
          CALL MULTIPLY.X.Y
          % ETC
     %
MULTIPLY.X.Y
          % MULTIPLY CODE
          EXIT
```

```
%       THE ABOVE EXAMPLES HAVE SHOWN, AMONG OTHER THINGS, TWO OF
%   THE "SPECIALS" THAT ARE INCLUDED IN MIL SYNTAX TO GO ALONG WITH
%   DECLARES.  THEY ARE:
%            DATA.LENGTH(declared.identifier)
%            LENGTH.BETWEEN.ENTRIES(array.identifier)
%
%   NOTE THAT WHEN ARRAY NAMES ARE USED WITH THE SPECIALS, THE
%   SUBSCRIPT IS NOT PRESENT, AND IS A SYNTAX ERROR WHEN IT IS
%   PRESENT.
%
%       ANOTHER TYPE OF REMAPS IS ONE THAT REMAPS A PREVIOUSLY
%   DECLARED STRUCTURE.  IN THIS CASE, THE ADDRESSES OF THE REMAP
%   STRUCTURE WILL BEGIN AT THE ADDRESS OF THE REMAPPED STRUCTURE.
%   EXAMPLE:
%
        DECLARE
            01 SAVE.AREA.CHARS REMAPS SA.SECOND.ITEM,
               02 SA.NAME                 CHARACTER(30),
                  03 SA.PACK.ID          CHARACTER(10),
                  03 SA.FAMILY.NAME      CHARACTER(10),
                  03 SA.OFFSPRING.NAME   CHARACTER(10);
%
%                          "REVERSE"
%
%       ANOTHER ATTRIBUTE THAT MAY BE APPLIED TO A SIMPLE ITEM OR
%   A STRUCTURE IS THE "REVERSE" ATTRIBUTE.  THIS ATTRIBUTE CAUSES
%   THE FINAL ADDRESS ASSOCIATED WITH A DECLARED IDENTIFIER TO BE
%   ITS NORMALLY CALCULATED ADDRESS PLUS ITS DECLARED LENGTH.
%
%       FOR INSTANCE, SUPPOSE A PROGRAMMER WISHES TO SPECIFY A
%       STRUCTURE THAT DESCRIBES THE TOP OF MEMORY AND WANTS TO LIST
%       THE IDENTIFIERS FROM THE TOP OF MEMORY DOWNWARD. THE FOLLOWING
%       COULD THEN BE DONE:
%
        DECLARE
            01 TOP.OF.MEMORY REMAPS BASE.ZERO REVERSE,
               02 SLOP                      BIT(32),
               02 ADDR.INTERRUPT.QUEUE      BIT(553),
               02 ADDR.SAVED.ASTACK         BIT(240),
               02 ADDR.GISMO.WORK.SPACE     BIT(384),
               02 ADDR.TEMP.FIB             BIT(920),
               02 ADDR.TRACE.SPACE          BIT(2232),
                  03 ADDR.TRACE.CODE        BIT(24);
%
%       THESE IDENTIFIERS COULD THEN BE USED IN MIL STATEMENTS
%       AS FOLLOWS:
%
        MOVE ADDR.INTERRUPT.QUEUE TO Y
        EXTRACT ADDR.TRACE.CODE FROM T TO X
```

REGISTERS AND SCRATCHPAD

This section is intended only as a brief overview of the registers within the processor. It is assumed that the reader is familiar with the contents of the B1700 Systems Reference Manual (form 1057155). (See also Appendix B in this manual).

NOTE: The most-significant (left-most) bit in any register is identified in the MIL syntax as bit 0 (zero), the next most-significant as bit 1, etc. This is particularly advantageous in a bit-addressable machine since, for software purposes, it is often desirable to think of a register as being an extension of main memory. It should be noted that this convention is at variance with the hardware bit numbering convention where, generally, all bits are numbered right to left, 0 through 23. This difference has particular significance when any bit data is to be ORed into the M register at run time.

REGISTER GROUPS

The registers briefly described in this section are divided into the following logical groups:

Active
Result
Scratchpad
Constant
Input/Output
Condition

ACTIVE REGISTERS

Source & Sink	4-bit Source & Sink
X	* TOPM
Y	
T	
L	
A	
M	
BR	
LR	
FA	
FB	
TAS	
CP	
*MSMA	
*MBR	

T sub register

4-bit source & sink	
TA TB TC TD TE TF	

FB sub register

source & sink	
FU FT FL	

L sub register

4-bit source & sink	
LA LB LC LD LE LF	

C sub register

source & sink	
CA CB CC *CD	**CP

* MSMA, TOPM, MBR and the low order 3 bits of CD are not physically present in the S-Memory Processor. When addressed as a source they will yield a binary value of zero. When addressed as a sink (destination) the data is lost.

** CPU, a 2-bit sub register of CP, is not addressable as a source or a sink.

RESULT REGISTERS

Source
SUM
CMPX
CMPY
XANY
XEQY
MSKX
MSKY
XORY
DIFF

SCRATCHPAD

Single Scratchpad

Source & Sink
S0A
...
S15A
S0B
...
S15B

Double Scratchpad

Source & Sink
S0
...
S15

CONSTANT REGISTERS

Source
MAXS
MAXM

INPUT/OUTPUT REGISTERS

Source	Sink	Source & Sink
U	CMND	DATA

CONDITION REGISTERS

4-bit Source
BICN
FLCN
*INCN
XYCN
XYST

* INCN is not physically present on the S-Memory Processor. When adressed as a source it yields a binary value of 0. When addressed as a sink (destination) the data is lost.

ALPHABETICAL LISTING OF REGISTERS AND KEY CONCEPTS

Name	Length In Bits	Source Sink	Note
A	*	so & sk	Control Memory Microinstruction Address * 14 (1726), 19 (S-1), 20 (S-2)
BICN	4	source	boolean conditions
BR	24	so & sk	Base Register or low address S-Memory protection
C	24	---	Control; not addressable as a unit
CA	4	so & sk	subfield of C; general purpose
CB	4	so & sk	subfield of C; general purpose
CC	4	so & sk	subfield of C; interrupts and flags
CD	4	so & sk	subfield of C; interrupts and flags
CMND	24	sink	I/O Command Register
CMPX	24	source	Result: complement of X; masked by CPL
CMPY	24	source	Result: complement of Y; masked by CPL
Console Switches	24	source	the 24 toggle switches located on the Console front panel
Control Memory	16-BIT words	so & sk	location of microinstructions on M-Memory Processor
CP	8	so & sk	Control Parallel; subfield of C
CPL	5	so & sk	Control Parallel Length; subfield of CP
CPU	2	---	Control Parallel Unit; subfield of CP
CYD	1	---	Carry Difference or carry of borrow
CYF	1	---	Carry Flip-Flop; subfield of CP
CYL	1	---	Carry Level or carry of sum; masked by CPL

Name	Length In Bits	Source Sink	Note
DATA	24	so & sk	I/O Data Register
DIFF	24	source	result of X-(Y + CYF); masked by CPL
F	48	---	Field in S-Memory;FA and FB concatenated
FA	24	so & sk	Field Address in S-Memory
FB	24	so & sk	S-Memory Field Unit(FU), Field Type(FT), and Field Length (FL)
FL	16	so & sk	Field Length in S-Memory
FT	4	so & sk	subfield of FB
FLC	4	so & sk	subfield of FL
FLD	4	so & sk	subfield of FL
FLE	4	so & sk	subfield of FL
FLF	4	so & sk	subfield of FL
FLCN	4	source	boolean Field Length Conditions
FU	4	so & sk	S-Memory Field Unit size; subfield of FB
INCN	4	source	boolean dispatch Interrupt Conditions M-Memory Processor
L	24	so & sk	Local register also used in DISPATCH, OVERLAY, TRANSFER.CONTROL, READ/WRITE M5ML AND S-MEMORY ACCESS
LA	4	so & sk	subfield of L
LB	4	so & sk	subfield of L
LC	4	so & sk	subfield of L
LD	4	so & sk	subfield of L
LE	4	so & sk	subfield of L
LF	4	so & sk	subfield of L

Name	Length In Bits	Source Sink	Note
LR	24	so & sk	Limit Register or high address S-Memory protection
M	16	so & sk	current Microinstruction register
MBR	24	so & sk	Main Memory Microinstruction Base Register; not on S-Memory Processor
MAXM	24	source	hardwired Constant; number of 16-bit words of M-Memory
MAXS	24	source	Constant; size in bits of available S-Memory
MSKX	24	source	Result; mask of X; length by CPL
MSKY	24	source	Result; mask of Y; length by CPL
MSMA	16	so & sk	Control Memory addressed by the A register; M-Memory Processor only
Main Memory	--	---	S-Memory
NULL	24	so & sk	always zero
PERR	4	source	Parity Error Register; reflects error conditions from S & M-Memory, & cassette
READ	24	source	Console switch position; reads S-Memory addressed by FA to Console lights (A on 1714)
SFL	16	---	subfield of SOB corresponding to FL in FB
S0-S15	48	so & sk	Double Scratchpad Words
S15A-S15B	48	so & sk	Single Scratchpad Words of S15
S-Memory	--	---	Main Memory
SU	4	---	subfield of SOB corresponding to FU in FB
SUM	24	source	Result (X + Y + CYF) length by CPL

Name	Length In Bits	Source Sink	Note
T	24	so & sk	Transform - will ROTATE, SHIFT or EXTRACT bits; used also in S-MEMORY ACCESS and TRANSFER.CONTROL
TAS	24	so & sk	Top of A Register-Stack
TA	4	so & sk	subfield of T
TB	4	so & sk	subfield of T
TC	4	so & sk	subfield of T
TD	4	so & sk	subfield of T
TE	4	so & sk	subfield of T
TF	4	so & sk	subfield of T
TOPM	4	so & sk	Top of Control Memory; not on S-Memory Processor
U	16	source	cassette input only
WRIT	24	---	Console position switch; writes Console switches to address of memory contained in FA (A on 1714)
X	24	so & sk	input to Function Box
XANY	24	source	Result; X AND Y; length by CPL
XEOY	24	source	Result; X EOR Y; length by CPL
XORY	24	source	Result; X OR Y; length by CPL
XY	48	source	X AND Y concatenated
XYCN	4	source	boolean XY Conditions
XYST	4	source	boolean XY States
Y	24	so & sk	input to Function Box

ACTIVE REGISTERS

The following is a description of the Active registers.

X AND Y REGISTERS

The X and Y registers (both of which are 24 bits wide) are used as inputs into the 24-bit Function Box (see below). All functions are performed under control of the C (Control) register, which regulates the length of the operation, class of arithmetics, and least-significant carry input. The X and Y registers are capable of being shifted or rotated individually or as a unit and may receive or transmit data from or to main memory.

FIELD (F) REGISTER

The F register is divided into FA and FB, each sub-register being 24 bits wide. The FA (Field Address) portion is used to address main memory. FB is divided into FU (Field Unit, consisting of four bits used to indicate arithmetic unit size; FT (Field Type), a general-purpose 4-bit field; and FL (Field Length), consisting of 16 bits used to indicate the length of fields in main memory. FL is subdivided into FLC, FLD, FLE and FLF, each four bits in length.

LOCAL (L) REGISTER

The L register is 24 bits wide and is subdivided into LA, LB, LC, LD, LE and LF, each four bits in length. L and its subdivisions are generally used to temporarily hold the contents of other processor registers. It is also used as a source and destination for main memory access and has implicit use in the DISPATCH, OVERLAY, READ/WRITE MSML and TRANSFER CONTROL microinstructions.

TRANSFORM (T) REGISTER

The T register is a 24-bit transformation register used extensively for interpretation of virtual-language operators. It is subdivided into TA, TB, TC, TD, TE and TF, each four bits in length. T has strong SHIFT and EXTRACT logics associated with it and is the principal formatting register of the processor. This register also has the capability of receiving or transmitting data from and to main memory.

MICROINSTRUCTION (M) REGISTER

The M register is a 16-bit register which holds the micro-operator for decoding and subsequent execution by the hardware. It is addressable as a source and sink register; when used as a sink register the source is bit-ORed with the upcoming M-op, except in TAPE mode.

BASE (BR) AND LIMIT (LR) REGISTERS

The BR and LR registers are each 24 bits wide and are used to hold the main memory base and limit addresses for the currently active main memory process. The M-Memory processor hardware uses these registers to determine if addresses in the Field Address (FA) register are within the base/limit boundaries.

ADDRESS (A) REGISTER

The A register is the microprogram address register which contains the bit address of the next microinstruction. Values in the A register are always MOD 16; i.e., the low-order four bits are always zero. It is capable of addressing 16,384 microinstructions located in either control memory or main memory or both. The A register is automatically incremented to the next microinstruction before the current microinstruction is executed. It is also capable of having any value from 0 to 4,095 added to or subtracted from it to facilitate microcode branching.

A STACK (TAS)

The A stack is a 32-element-deep, 24-bit wide, push-down, pop-up memory, i.e., a last-in-first-out (LIFO) storage structure. The A stack is used to nest microroutine linkages and allows highly shared routines, thus reducing control memory requirements. Although the A stack was intended for microcode addresses, it has been made 24-bits wide to allow for any operand storage.

NOTE: The S-Memory Processor A stack has only 16 storage elements.

TOP OF CONTROL MEMORY (TOPM) REGISTER
(M-Memory Processor only)

The TOPM register is four bits wide and is used to determine which memory (control or main) contains the next microinstruction. If the A register is equal to or greater than (TOPM*512*16), the next microinstruction will be fetched from main memory rather than control memory. The TOPM register is addressable as a source or as a sink (destination). The fetch from S-Memory takes place at address A+(TOPM*512*16)+MBR.

MEMORY BASE REGISTER (MBR)
(M-Memory Processor only)

The MBR register is used with the A and TOPM registers to obtain the main memory address of the next microinstruction. (See above formula). The MBR register is addressable as both a source and as a sink

CONTROL (C) REGISTER

The C register is a 24-bit control register for the microprocessor. It contains the 24-bit Function Box controls and carry input plus some of the processor interrupts and flags. It is subdivided into CA, CB, CC, CD, each four bits wide; and CP, eight bits wide. CA and CB may be used as general-purpose registers. CC and CD represent processor interrupts and flags (see discussion under Condition Registers below). CP contains Function Box controls: CYF (0 bit of CP), CPU (1 and 2 bits of C^2), and CPL (3,4,5,6, and 7 bits of CP). CYF (Carry Flip Flop) notifies the Function Box that a previous unit carry must be added to its summary results. CPU (Control Parallel Unit) notifies the Function Box of the type of unit contained in X and Y: 00 = binary, 01 = 4-bit decimal. CPL (Control Parallel Length) specifies the width, in bits, of the Function Box and Read/Write microinstructions.

COMBINATORIAL LOGIC OR FUNCTION BOX.

The Combinatorial Logic, often called the Function Box, produces the Result Registers. Inputs are the X register, the Y register and the Carry Flip-Flop (CYF). The inputs are combined under control of the Control Parallel Unit (CPU) register and the Control Parallel Length (CPL) register. When values are loaded into the X and Y registers, a large collection of output values and comparisons (called Result Registers) is made available to all subsequent microinstructions.

RESULT REGISTERS

The Result registers are outputs from the 24-bit Function Box. Their contents are produced immediately and automatically from the inputs to the Function Box (X, Y and CYF) and cannot be changed except by changing inputs or by changing CPU (Control Parallel Unit) or CPL (Control Parallel Length). If the value of CPL is less than 24, then the 24 minus CPL most-significant bits of all Result registers will be zero. These registers are source registers only and therefore cannot be used as the sink (destination) register in a MOVE or in any other instruction.

XORY RESULT REGISTER

This register contains the INCLUSIVE OR of the X register combined with the Y register. This is a bit by bit operation with corresponding pairs of bits treated independantly.

XANY RESULT REGISTER

This register contains the AND of the X register combined with the Y register. This is the logical product of the X register and the Y register. Corresponding pairs of bits are treated independently.

XEOY RESULT REGISTER

This register contains the EXCLUSIVE OR of the X register combined with Y register.

CMPX RESULT REGISTER

This register contains the 1's complement of the X register.

CMPY RESULT REGISTER

This register contains the 1's complement of the Y register.

MSKX RESULT REGISTER

Masked X contains the low-order bits of the X register. The value of CPL determines the number of bits placed in MSKX. All other high-order bits are zero. If CPL is equal to 24, then MSKX is identical to the X register.

MSKY RESULT REGISTER

Masked Y contains the low-order bits of the Y register. The value of CPL determines the number of bits placed in MSKY. All other high-order bits are zero. If CPL is equal to 24, MSKY is identical to the Y register.

SUM RESULT REGISTER

Sum is the decimal or binary value (determined by CPU) of the X register plus the Y register plus the CYF register. Corresponding pairs of bits are grouped by CPU control, and grouping may be binary or 4-bit decimal. If the sum of (X+Y+CYF) is larger than the size specified by CPL, then the CYL (Carry Level) will be true (one). CYL may be gated into CYF through use of the CARRY instruction.

DIFFERENCE RESULT REGISTER (DIFF)

DIFF stores the amount resulting from the subtraction of the sum of the contents of the Y and CYF registers from the contents of the X register. The contents of the CPU register determine whether the subtraction is decimal or binary. Corresponding pairs of bits are grouped by CPU. If the difference is negative, X-(Y+CYF)<0, then Diff Result will be in 2's complement form or 10's complement form depending upon the mode, either binary or decimal respectively; and CYD (Carry Difference) will be true (one).

NOTE: The CYD register is not conditioned by CPL; it is always based on a 24-bit comparison. The programmer, therefore, must know what is in the high-order positions of the X register and the Y register if CPL is less than 24.

SCRATCHPAD

The processor makes use of the Scratchpad for temporary storage of Active registers. The Scratchpad may be addressed as sixteen, 48-bit double words or thirty-two, 24-bit words.

SCRATCHPAD WORDS - 24 BITS EACH

S0A	S4A	S8A	S12A
S0B	S4B	S8B	S12B
S1A	S5A	S9A	S13A
S1B	S5B	S9B	S13B
S2A	S6A	S10A	S14A
S2B	S6B	S10B	S14B
S3A	S7A	S11A	S15A
S3B	S7B	S11B	S15B

DOUBLE SCRATCHPAD WORDS - 48 BITS EACH

S0	S4	S8	S12
S1	S5	S9	S13
S2	S6	S10	S14
S3	S7	S11	S15

(Sn = SnA AND SnB concatenated, where n = 0 through 15)

CONSTANT REGISTERS

The following is a description of the Constant registers.

MAXIMUM MAIN MEMORY REGISTER

The 24-bit MAXS register is set by the field engineer and contains the value of the maximum installed number of main memory bits. It is addressable as a source only. Main memory addresses begin at zero. The lower 15 bits are always zero, i.e., MAXS has a 4096 byte (32K bit) resolution.

MAXIMUM CONTROL MEMORY REGISTER

The 24-bit MAXM register is set by the field engineer and contains the value of the maximum installed number of control memory words, each word comprising 16 bits. It is addressable as a source only. The lower 10 bits are always zero, i.e., MAXM has a 1024 word resolution. On the B1712/B1714 MAXM will always contain zero.

NULL REGISTER

The NULL register is a 24-bit, addressable field of zeros.

INPUT/OUTPUT REGISTERS

The following is a description of the Input/Output registers.

CONSOLE.SWITCHES
(M-Memory Processor only)

This 24-bit register reflects the current state of the 24 Console switches on the processor.

CONSOLE CASSETTE TAPE INPUT (U) REGISTER

The U register accumulates the data read from the tape cassette on the Console control panel. It is addressable as a source in the RUN mode with the MOVE REGISTER microinstruction and in the TAPE mode with the MOVE 24-BIT LITERAL microinstruction. (See MIL Statements: LOAD.MSMA.) It is not addressable as a sink (destination).

COMMAND REGISTER

The CMND register is used to transfer commands to the I/O controls. It is 24 bits wide and is addressable as a sink only.

DATA REGISTER

The DATA register is used to transfer data to and from the I/O controls and their peripherals. It is 24 bits wide and is addressable as a source or as a sink.

CONDITION REGISTERS

There are five Condition registers:

Binary Conditions (BICN)
Field Length Conditions(FLCN)
Interrupt Conditions (INCN)
X AND/OR Y registers(s) Conditions (XYCN)
X AND/OR Y registers(s) States Conditions (XYST)

Each Condition register consists of four bits. The bits are identified from left to right and are assigned the position numbers 0 thru 3, with 0 being the most-significant bit.

All Condition registers are source registers only. They may be moved to another register or tested, using the IF and SKIP instructions, for their current contents. They may not be the sink (destination) register of any instruction.

BIT	BICN	XYCN	XYST	FLCN	INCN
0	LSUY	MSBX	LSUX	FL=SFL	NO-DEVICE
1	CYF	X=Y	ANY.INTERRUPT	FL>SFL	HI-PRIORITY
2	CYD	X<Y	Y NEQ 0	FL<SFL	INTERRUPT
3	CYL	X>Y	X NEQ 0	FL NEQ 0	LOCKOUT

BINARY CONDITIONS (BICN) REGISTER

LSUY is true if the least-significant unit of the Y register is 1 and the Control Parallel Unit (CPU) register specifies binary (CPU = 0); or 9 and the CPU register specifies decimal (CPU = 1).

The Carry Flip-flop (CYF) register indicates the value of the carry-in bit in the Control Parallel (CP) register. The CYF register may be manipulated as part of the CP register and by the CARRY instruction.

The Carry Difference (CYD) register is true if X-(CYF+Y)<0. This condition is not affected by CPL, i.e., a 24-bit compare is always made.

The Carry Level (CYL) register is true if (X+Y+CYF), limited by the Control Parallel Length (CPL) register, overflows.

XY CONDITIONS (XYCN) REGISTER.

MSBX is true if the most-significant bit of the X register, as determined by the Control Parallel Length (CPL) register, is a 1.

NOTE: The comparisons of the X register to the Y register are not affected by CPL; they are always 24-bit compares.

XY STATES (XYST) REGISTER.

LSUX is true if the least-significant unit of the X register is 1 and the Control Parallel Unit (CPU) register specifies binary (CPU = 0); or 9 and the Control Parallel Unit (CPU) register specifies decimal (CPU = 1). The comparisons of the X register or the Y register to zero are not affected by CPL; all 24 bits of the X register and/or the Y register are used in the comparisons.

ANY.INTERRUPT

This bit is true if any of the following conditions in registers CC, CD, or INCN (M-Memory Processor) are true:

Event	Register (Bit Position)
MISSING DEVICE	INCN(0)
PORT INTERRUPT	INCN(2)
I/O SERVICE REQUEST INTERRUPT	CC(2)
CONSOLE INTERRUPT	CC(3)
MAIN MEMORY READ PARITY ERROR INTERRUPT	CD(0)
MEMORY WRITE/SWAP ADDRESS OUT OF BOUNDS INTERRUPT	CD(3)

The CC and CD registers are both 4-bit source and sink (destination) registers within the C register. The bits in each are numbered 0 through 3, with bit 0 being the most significant. They have been assigned the following uses and meanings:

CC(0) STATE LIGHT
CC(1) TIMER INTERRUPT
CC(2) I/O SERVICE REQUEST INTERRUPT
CC(3) CONSOLE INTERRUPT
CD(0) MAIN MEMORY PARITY ERROR
CD(1) MAIN MEMORY WRITE/SWAP ERROR OVERRIDE
CD(2) MAIN MEMORY READ OUT OF BOUNDS ERROR
CD(3) MAIN MEMORY WRITE/SWAP OUT OF BOUNDS ERROR

All bits in the CC and CD portions of the C register, once set, remain set even though the conditions that caused them to be set may no longer exist. Therefore, if it is desired to clear any of these bits to zero, this must be done explicitly. CD(1), CD(2), and CD(3) of the C register are always zero in the S-Memory Processor but still may be addressed and tested.

CONSOLE INTERRUPT
(CC(3))

This bit is set when the interrupt toggle switch on the Console control panel is turned on. It remains set as long as the switch is on. It can be reset programmatically but not by turning the Console toggle switch off. This bit is also reported in ANY.INTERRUPT when it is on.

MAIN MEMORY READ PARITY ERROR INTERRUPT
(CD(0))

This bit is set when a main memory parity error is detected during a READ or a READ portion of a SWAP operation or when an attempt is made to access non-existent main memory.

MAIN MEMORY ADDRESS OUT OF BOUNDS OVERRIDE
(CD(1)) (A-Memory Processor only)

This bit is tested if the Field Address (FA) register setting is less than the Base Register (BR) setting or greater than or equal to the Limit Register (LR) setting; then WRITE or SWAP operations will be inhibited unless this bit is set (one). The state of this bit does not affect the setting of CD(2) or CD(3).

READ ADDRESS OUT OF BOUNDS INTERRUPT
(CD(2)) (M-Memory Processor only)

This bit is set when a READ operation is attempted and the Field Address (FA) register setting is either less than the Base Register (BR) setting or greater than or equal to the Limit Register (LR) setting. The READ operation is not inhibited.

WRITE/SWAP ADDRESS OUT OF BOUNDS INTERRUPT
(CD(3)) (M-Memory Processor only)

This bit is set when a WRITE or SWAP operation is attempted and the Field Address (FA) register setting is either less than the Base Register (BR) setting or greater than or equal to the Limit Register (LR) setting. This bit, when on, is also reported in ANY.INTERRUPT.

FIELD LENGTH CONDITIONS (FLCN) REGISTER

All conditions are based upon comparisons between the 16 bits of the FL register and either zero or the corresponding low-order 16 bits of the first word in the Scratchpad (SOB).

INTERRUPT CONDITIONS (INCN) REGISTER
(M-Memory Processor only)

NO DEVICE is true if an interrupt message is present in the dispatch buffer for a port or channel which does not have a device attached to it. This condition is normally cleared by the processor with a DISPATCH READ AND CLEAR instruction.

HI PRIORITY is true if there is a high-priority message present in the dispatch buffer.

INTERRUPT is true if there is a message present in the dispatch buffer for the processor. This condition is normally cleared by a DISPATCH READ AND CLEAR instruction. It is also reported in ANY.INTERRUPT.

LOCKOUT is true if the interrupt system is locked (marked as "in use").

REGISTER DESIGNATIONS AND AREAS OF APPLICATION

The following is a list, arranged by areas of application, of registers and their associated designations.

MICROINSTRUCTION CONTROLS

A (Microinstruction Address)
M (Current Microinstruction)
TAS (Top of Address Stack)
TOPM (Logical Top of M-Memory)
MBR (Microinstruction Base Register)

S-MEMORY CONTROLS

BR (Base Register)
LR (Limit Register)
FA (Field Address)
FL (Field Length)
CP (Control Parallel)

INTERRUPT CONTROLS

CC
CD
INCN

PARALLEL WIDTH CONTROLS

C
CP
CPL
CPU

ORGANIZATION OF FIELDS AND SUBFIELDS

The following is a description of the organization of register fields and subfields, expressed in the notation of MIL structured data declarations.

```
01 C BIT(24),
   02 CA BIT(4),
   02 CB BIT(4),
   02 CC BIT(4),
   02 CD BIT(4),
   02 CP BIT(8),
      03 CYF  BIT(1),
      03 CPU  BIT(2),
      03 CPL  BIT(5);
```

NOTE: C does not exist as a composite, only as subfields.

```
01 F BIT(48),
   02 FA BIT(24),
   02 FB BIT(24),
      03 FU BIT(4),
      03 FT BIT(4),
      03 FL BIT(16),
         04 FLC BIT(4),
         04 FLD BIT(4),
         04 FLE BIT(4),
         04 FLF BIT(4);
```

```
01 L BIT(24),
   02 LA BIT(4),
   02 LB BIT(4),
   02 LC BIT(4),
   02 LD BIT(4),
   02 LE BIT(4),
   02 LF BIT(4);
```

```
01 T BIT(24),
   02 TA BIT(4),
   02 TB BIT(4),
   02 TC BIT(4),
   02 TD BIT(4),
   02 TE BIT(4),
   02 TF BIT(4);
```

MIL STATEMENTS

Following is an alphabetical list of MIL statements found in this section.

* Available on B1720 systems only

ADD SCRATCHPAD

ADD SCRATCHPAD

SYNTAX:

ADD-----scratchpad.word-----TO FA------>1

SEMANTICS:

This instruction adds the left half of any scratchpad word (S0A...S15A) to the Field Address (FA) register. The result is placed in FA; the contents of scratchpad.word remain unchanged. (See also: SUBTRACT SCRATCHPAD.)

EXAMPLE:

ADD S9A TO FA

Table B-3: Microinstructions

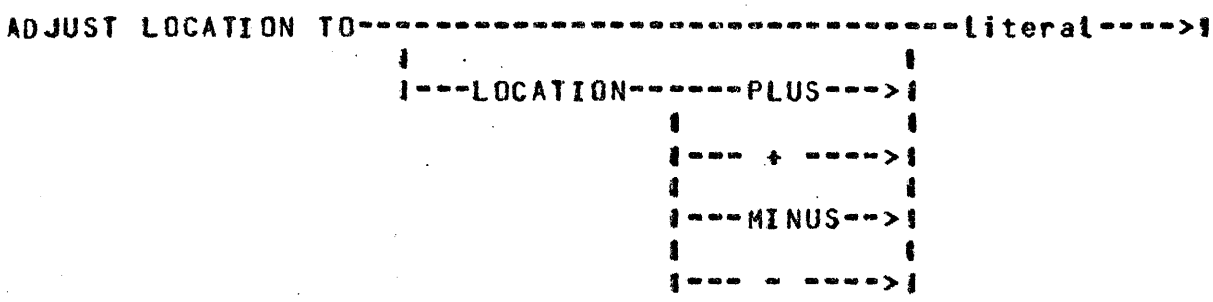
MICRO NAME	MC 15	14	13	12	MD 11	10	9	8	ME 7	6	5	4	MF 3	2	1	0
REGISTER MOVE	0	0	0	1	REG 1 GROUP SOURCE REGISTER				REG 1 SELECT		REG 2 SELECT		REG 2 GROUP SINK REGISTER			
SCRATCHPAD MOVE	0	0	1	0	REGISTER GROUP SOURCE OR SINK				REG SELECT		MOV DIR	DPW 1/2	DOUBLE PAD WORD ADDRESS			
4 BIT MANIPULATE	0	0	1	1	REGISTER GROUP, 4 BIT SOR & SNK				REG SEL	MANIPULATE VARIANTS			4 BIT MANIP. LITERAL			
BIT TEST REL BRANCH FALSE	0	1	0	0	REGISTER GROUP, 4 BIT SOURCE				REG SEL	TESTBIT NUMBER		DSP SGN	RELATIVE BRANCH DISPLACEMENT MAG			
BIT TEST REL BRANCH TRUE	0	1	0	1	REGISTER GROUP 4 BIT SOURCE				REG SEL	TESTBIT NUMBER		DSP SGN	RELATIVE BRANCH DISPLACEMENT MAG			
SKIP WHEN	0	1	1	0	REGISTER GROUP 4 BIT SOR & SNK				REG SEL	SKIP TEST VARIANTS			4 BIT TEST MASK			
READ/WRITE MEMORY	0	1	1	1	R/W VAR	COUNT FA/FL VARIANTS			DATAREG CODE		TW SGN		DATA TRANSFER WIDTH MAGNITUDE			
MOVE 8 BIT LITERAL	1	0	0	0	REGISTER GROUP; REG SEL IS 2.				ENTIRE 8 BITS OF 8 BIT LITERAL							
MOVE 24 BIT LITERAL	1	0	0	1	REGISTER GROUP; REL SEL IS 2.				8 MOST SIGNIFICANT BITS OF FULL 24 BIT LITERAL							
SHIFT/ROTATE T REG	1	0	1	0	SINK REGISTER GROUP				SNK REG SELECT		S/R VAR		LEFT SHIFT/ROTATE COUNT			
EXTRACT FROM T REG	1	0	1	1	RIGHT BIT POINTER FOR EXTRACTION FLD				SNK REG CODE				EXTRACTION FIELD WIDTH			
BRANCH RELATIVE	1	1	0	DSP SGN	RELATIVE DISPLACEMENT MAGNITUDE											
CALL RELATIVE	1	1	1	DSP SGN	RELATIVE CALLED ADDRESS MAGNITUDE											
SWAP MEMORY	0	0	0	0	0	0	1	0	DATAREG CODE		TW SGN		DATA TRANSFER WIDTH MAGNITUDE			
CLEAR REGISTERS	0	0	0	0	0	0	1	1	L REG	T REG	Y REG	X REG	FA REG	FL REG	FU REG	CP REG
SHIFT/ROTATE X OR Y	0	0	0	0	0	1	0	0	S/R DIR VARIANT		X/Y VAR		LEFT OR RIGHT, X OR Y SHIFT/ROTATE COUNT			
SHIFT/ROTATE X AND Y	0	0	0	0	0	1	0	1	S/R DIR VARIANT				LEFT OR RIGHT X AND Y SHIFT/ROTATE COUNT			
COUNT FA/FL	0	0	0	0	0	1	1	0	COUNT FA/FL VARIANTS				COUNT SCALAR MAGNITUDE			
EXCHANGE DPW	0	0	0	0	0	1	1	1	SINK DPW ADDRESS				SOURCE DPW ADDRESS			
SCRATCHPAD RELATE FA	0	0	0	0	1	0	0	0				DSP SGN	LEFT HALF PAD WORD ADDRESS			
MONITOR	0	0	0	0	1	0	0	1	LITERAL OCCURRENCE IDENTIFIER							
DISPATCH	0	0	0	0	0	0	0	0	0	0	0	1	DISPATCH VARIANTS			SKP FLG
CASSETTE CONTROL	0	0	0	0	0	0	0	0	0	0	1	0	CASSETTE MANIP VARIANTS			
BIAS	0	0	0	0	0	0	0	0	0	0	1	1	BIAS VARIANTS			TST FLG
STORE F INTO D PW	0	0	0	0	0	0	0	0	0	1	0	0	SINK DPW ADDRESS			
LOAD F FROM D PW	0	0	0	0	0	0	0	0	0	1	0	1	SOURCE DPW ADDRESS			
CARRY FF MANIPULATE	0	0	0	0	0	0	0	0	0	1	1	0	CYF ← CYD	CYF ← CYL	CYF ← 1	CYF ← 0
EXERCISE MSM(L)	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	R/W VAR
HALT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
OVERLAY M-STRING	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
NORMALIZE X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
BIND ALTERM MBR	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
NO OPERATION	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

VARIANTS	0 00 000	1 01 001	10 010	11 011	100	101	110	111
MOV DIR: 1/2 DPW:	F←R LEFT	R←F RIGHT						
MANIP. VARIANTS:	SET	AND	OR	EOR	INC	INC TEST	DEC	DEC TEST
DSP SIGN:	+	−						
DSP SIGN:	+	−						
SKIP TEST VARIANTS:	ANY CLR/	ALL CLR/	EQL CLR/	ALL CLR	ANY/ CLR/	ALL / CLR /	EQL / CLR /	ALL / CLR
R/W VAR:	READ	WRT						
CNT VAR:	NØP	FA↑	FL↑	FA↑ FL↑ L	FA↑ FL↑	FA↑	FL↑	FA↑ FL↑
REG SEL: TW SIGN:	X +	Y −	T					
S/R VAR:	SHFT	ROT						
SINK REG CODE:	X	Y	T	L				
DSP SIGN:	+	−						
DSP SIGN:	+	−						
TW SIGN: REG CODE:	+ X	− Y	T	L				
X/Y VAR: S/R, DIR:	X SFT←	Y SFT→	ROT←	ROT→				
S/R, DIR VARIANTS:	SFT←	SFT→	ROT←	ROT→				
COUNT FA/FL VAR:	NØP	FA↑	FL↑	FA↑ FL↑	FA↑ FL↑	FA↑	FL↑	FA↑ FL↑
DSP SIGN:	+	−						
SKP FLAG: DISP VAR:	FAIL LOCK	SUCC WRTLO	READ	R&C	WRTHI	ABSNT	UNDEF	UNDEF
CASSETTE MANIP:	START TAPE	STOP@ CAP	STOP ON X≠Y	UNDEF	UNDEF	UNDEF	STOP ON X=Y	UNDEF
TEST FLG: BIAS VAR:	TST/ UNIT	TEST F	S	FS	NØP	FCP	NØP	NØP
R/W VAR	READ	WRIT						

ADJUST

ADJUST

SYNTAX:



SEMANTICS:

This pseudo-operation adjusts the physical.code.address of the compiler. The value of the physical.code.address specifies the location (control memory address) into which the next generated microinstruction is to be placed, generally by a user-developed loader. (See also Segmentation: Label Addresses.)

LOCATION PLUS(+) or MINUS(-) increments/decrements the physical.code.address by the value of the literal. If this option is not used, the the physical.code.address is set to the value of the literal.

The literal must have a value of 0 MOD 16.

NOTE: This instruction is generally used to compensate for disposable loader routines.

EXAMPLE:

```
ADJUST LOCATION TO @1000
ADJUST LOCATION TO LOCATION + 32
ADJUST LOCATION TO LOCATION MINUS 128
```

AND

AND

SYNTAX:

```
AND---source.sink.register---WITH-----source.register------>|
                                    |                  |
                                    |--literal-------->|
```

SEMANTICS:

This instruction logically ANDs the contents of a 4-bit source and sink (destination) register with the bit configuration of the literal or the contents of a 4-bit source register. The result is placed in source.sink.register; the contents of source.register remain unchanged. (See also: OR and EOR.)

The register may be any of the following:

source.sink.register	source.register
CA CB *CC *CD	source.sink.register
FT FU	BICN
FLC FLD FLE FLF	FLCN
LA LB LC LD LE LF	INCN (available on B1720 only)
TA TB TC TD TE TF	PERR (available on B1720 only)
TOPM (avaiable on B1720 only)	XYCN
	XYST

* CC and CD represent processor interrupts and flags

The literal has a decimal range from 0 to 15.

AND
cont

AND
cont

SEMANTICS cont:

TABLE 8-1: AND Truth Table

Source & Sink Register	Literal Source Register			Source & Sink Register
0	AND	0	Yields	0
0	AND	1	Yields	0
1	AND	0	Yields	0
1	AND	1	Yields	1

EXAMPLE:

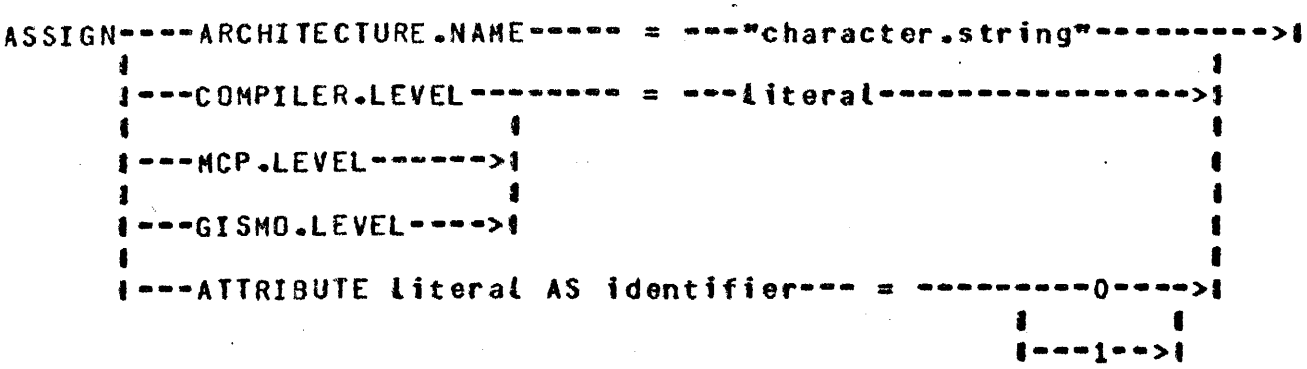
AND TB WITH 3

	TA	TB	TC	TD	TE	TF	
T	0000	1010	1111	0011	0001	0010	before (0AF312)
	--	0011	--	--	--	--	literal (3)
T	0000	0010	1111	0011	0001	0010	after (02F312)

ASSIGN

ASSIGN

SYNTAX:



SEMANTICS:

This statement assigns values to the various interpreter verification attributes. These attributes occupy fields in the IPB (Interpreter Parameter Block) of all MARK IV.1 and later interpreters. They are accessed at BOJ (Beginning of Job) time by the MCP and are used to verify that the proper interpreter has been chosen.

The character.string for ARCHITECTURE.NAME must be a string of 10 or fewer characters, and must be enclosed within quotation marks.

Literal has a decimal range from 0 to 255 for COMPILER.LEVEL, MCP.LEVEL, and GISMO.LEVEL; and from 0 to 79 for ATTRIBUTE.

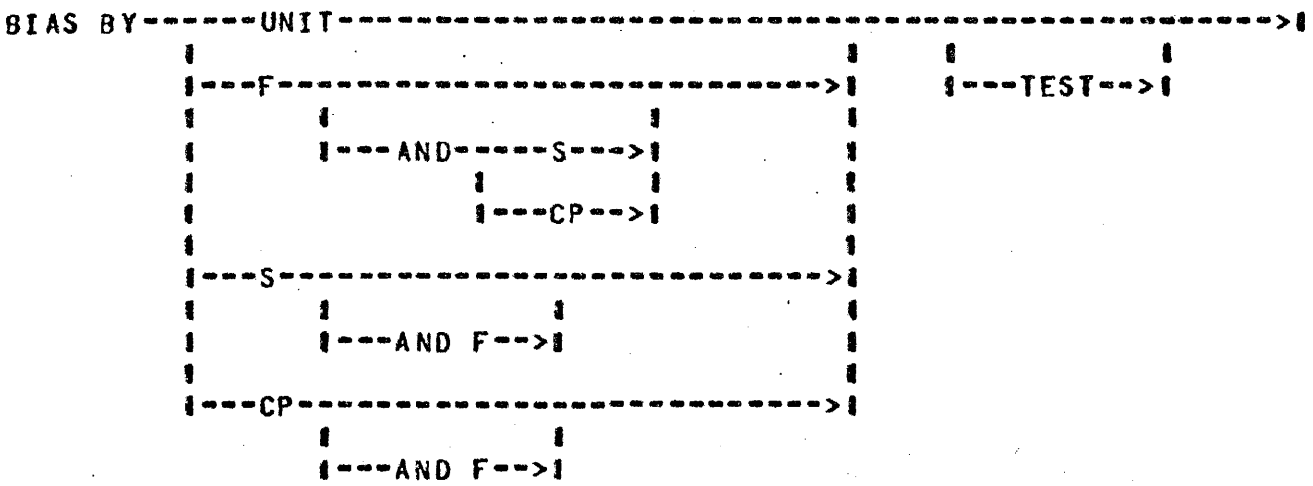
EXAMPLE:

```
ASSIGN ARCHITECTURE.NAME = "GISMO.26"
ASSIGN MCP.LEVEL = 197
ASSIGN ATTRIBUTE 64 AS ITEM.01=1
```

BIAS

BIAS

SYNTAX:



SEMANTICS:

This instruction sets the Control Parallel Length (CPL) register and the Control Parallel Unit (CPU) register to values calculated from the given operands.

NOTE: All references to register S refer to the SFL or SFU registers in the second half of the first scratchpad word, e.g., the SLF (low order 16 bits) part of the SOB register.

The CPU register will be set to 1 if the value of the the Field Unit (FU) register is set to 4 or 8; otherwise CPU is set to 0. This is done for all variations of BIAS except BIAS BY S, which sets the CPU register from SFU rather than from the FU register.

BIAS BY ... sets the CPL register equal to 24 or to the value in the specified register if it is less than 24. BIAS BY UNIT sets the CPL register equal to the FU register (4 for 4-bit decimal, 8 for 8-bit decimal, or any other value less than 16 for binary).

If the TEST option is used the above actions are performed, and the next microinstruction is skipped if CPL has not been set to zero.

BIAS
cont

EXAMPLE:

BIAS BY F — This instruction sets the CPL register to 24 or to the value of the Field Length (FL) register, if it is less than 24. It also sets the CPU register equal to the unit in the FU register.

BIAS BY F AND CP — This instruction sets the CPL register to 24, to the value in the FL register, or to the value in the CPL register, whichever is the smallest. It also sets the CPU register to the unit in the FU register.

BIAS BY UNIT — This instruction sets the CPL register equal to the length of the unit of the type specified by the FU register. It also sets the CPU register equal to one unit of the type specified in the FU register, i.e., 4-bit decimal, 8-bit decimal, or binary.

NOTE

In all cases except UNIT, CPU is set to 1 if FU (or SFU) is 4 or 8; otherwise CPU is set to 0. If UNIT is specified, CPL is set directly to the value in FU.

BRANCH.EXTERNAL

SYNTAX:

BRANCH.EXTERNAL----TO----label----->1

SEMANTICS:

This instruction transfers control to the external segment location specified by label. (See: Segmentation.)

Label must be associated with a run time address that has a displacement from the BRANCH.EXTERNAL instruction of less than 4096 microinstructions.

NOTE: If an external segment does not exist because $NO EXTERNAL has been specified, BRANCH.EXTERNAL is equivalent to GO TO.

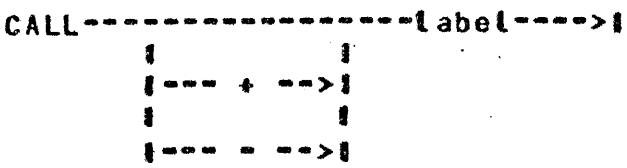
EXAMPLE:

BRANCH.EXTERNAL TO EXTERNAL.SEGMENT.LABEL

CALL

CALL

SYNTAX:



SEMANTICS:

This instruction stores the address of the next microinstruction in the A stack, then branches to the location specified by label.

The location specified by the label may be a maximum of 4095 microinstructions away from the CALL instruction.

EXAMPLE:

```
CALL M.IN.OUT
CALL +ABC
```

CALL.EXTERNAL

CALL.EXTERNAL

SYNTAX:

CALL.EXTERNAL-----label----->!

SEMANTICS:

This instruction stores the address of the next microinstruction in the A stack, then branches to the external segment location specified by label. (See: Segmentation.)

Label must be associated with a run time address that has a displacement from the CALL.EXTERNAL instruction of less than 4096 microinstructions.

NOTE: If an external segment does not exist, because $NO EXTERNAL has been specified, CALL.EXTERNAL is equal to CALL.

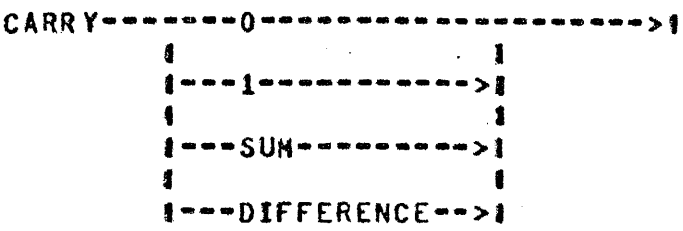
EXAMPLE:

```
CALL.EXTERNAL BEGINNING.OF.LOOP.1
```

CARRY

CARRY

SYNTAX:



SEMANTICS:

This instruction sets the Carry (CYF) register to either 0 or 1.

CARRY 0 or CARRY 1 sets the CYF register to 0 or 1 respectively.

CARRY SUM sets the CYF register to the value of CYL single bit.

CARRY DIFFERENCE sets the CYF register to the value of the Carry Difference (CYD) register:

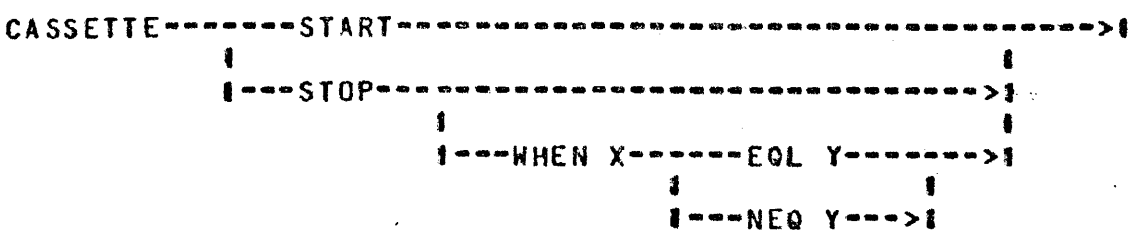
	X>Y	X=Y AND CYF=0	X=Y AND CYF=1	X<Y
CYD	0	0	1	1

The CYD register, unlike the CYL register is not conditioned by the CPL register. That is, all 24 bits of the X and Y registers are compared when setting CYF by the CYD register. The programmer should, therfore, know what is in the high-order position of the X and Y registers when using the CYD register if the CPL register is set to less than 24.

CASSETTE

CASSETTE

SYNTAX:



SEMANTICS:

This instruction causes the system cassette tape to start or stop a READ operation at the next inter-record gap.

The information read from the cassette is loaded into the U register and remains there for a maximum of two clock cycles before the U register is cleared.

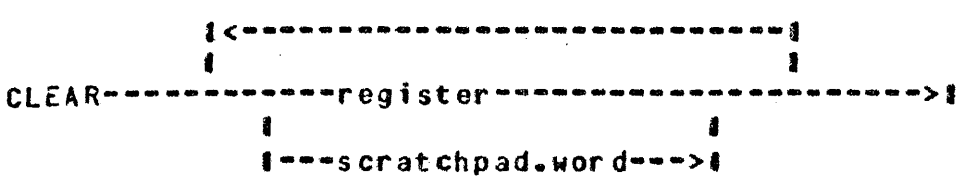
EXAMPLE:

```
CASSETTE STOP
CASSETTE STOP WHEN X EQL Y
```

CLEAR

CLEAR

SYNTAX:



SEMANTICS:

This instruction sets the specified register(s) or 24-bit scratchpad word(s) to zero.

The following may be cleared:

register	scratchpad.word
A	S0A
BR	...
CA CB *CC *CD CP CPU	S15A
FA FB FL FT FU	
FLC FLD FLE FLF	S0B
LA LB LC LD LE LF	...
TA TB TC TD TE TF TAS	S15B
TOPM (available on B1720 only)	

* CC and CD represent processor interrupts and flags

Each register clear takes one clock cycle; each scratchpad word takes two clock cycles.

NOTE: MOVE NULL TO register will be generated for each register specified on B1710 systems.

EXAMPLE:

```
CLEAR S10A
CLEAR BR L CB S4B TOPM FU
```

CODE.SEGMENT

CODE.SEGMENT

SYNTAX:

CODE.SEGMENT-----label----->|

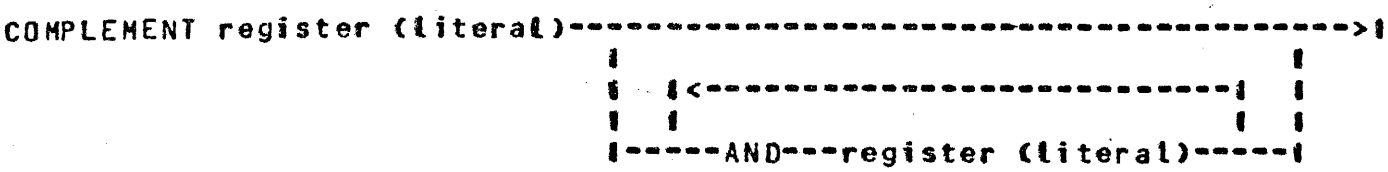
SEMANTICS:

See Segmentation: CODE.SEGMENT

COMPLEMENT

COMPLEMENT

SYNTAX:



SEMANTICS:

This instruction COMPLEMENTs (switches the state of) the specified bit. By using the options, more than one bit in any one register can be complemented with the same instruction if ALL BITS are in the SAME 4-BIT REGISTER. (See also: SET and RESET.)

The register may be any 4-bit source and sink (destination) register below:

CA CB CC CD (CC and CD represent processor interrupts and flags)
FT FU
FLC FLD FLE FLF
LA LB LC LD LE LF
TA TB TC TD TE TF
TOPM (avaiable on B1720 only)

It may also be the FL, FB, L, or T register: all bits must then be in the same 4-bit subfield.

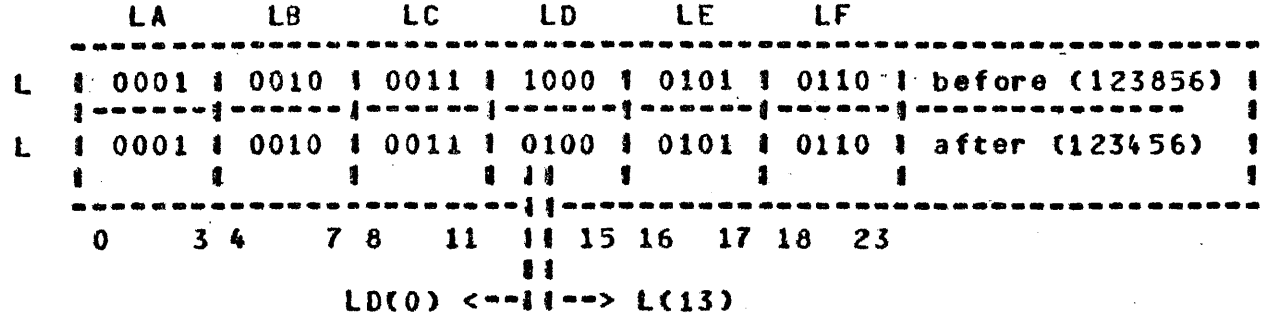
The literal has a decimal range from 0 to 3 for a 4-bit register; from 0 to 15 for the FL register; and from 0 to 23 for the FB, L, and T registers.

COMPLEMENT
cont

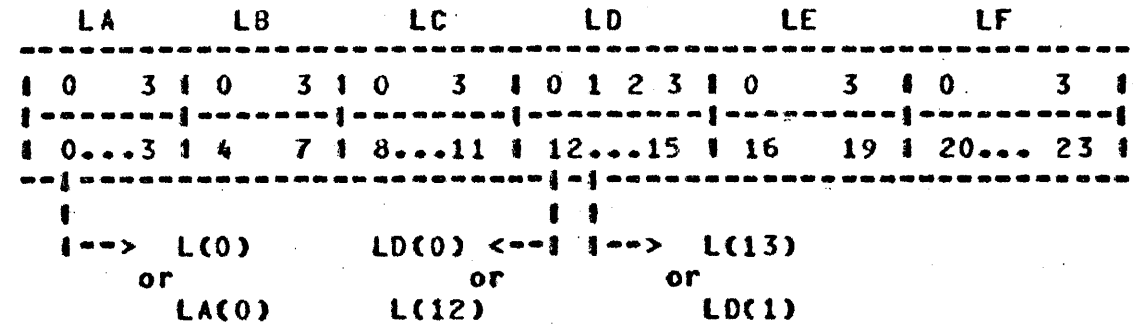
COMPLEMENT
cont

EXAMPLE:

COMPLEMENT LD(0) AND L(13)



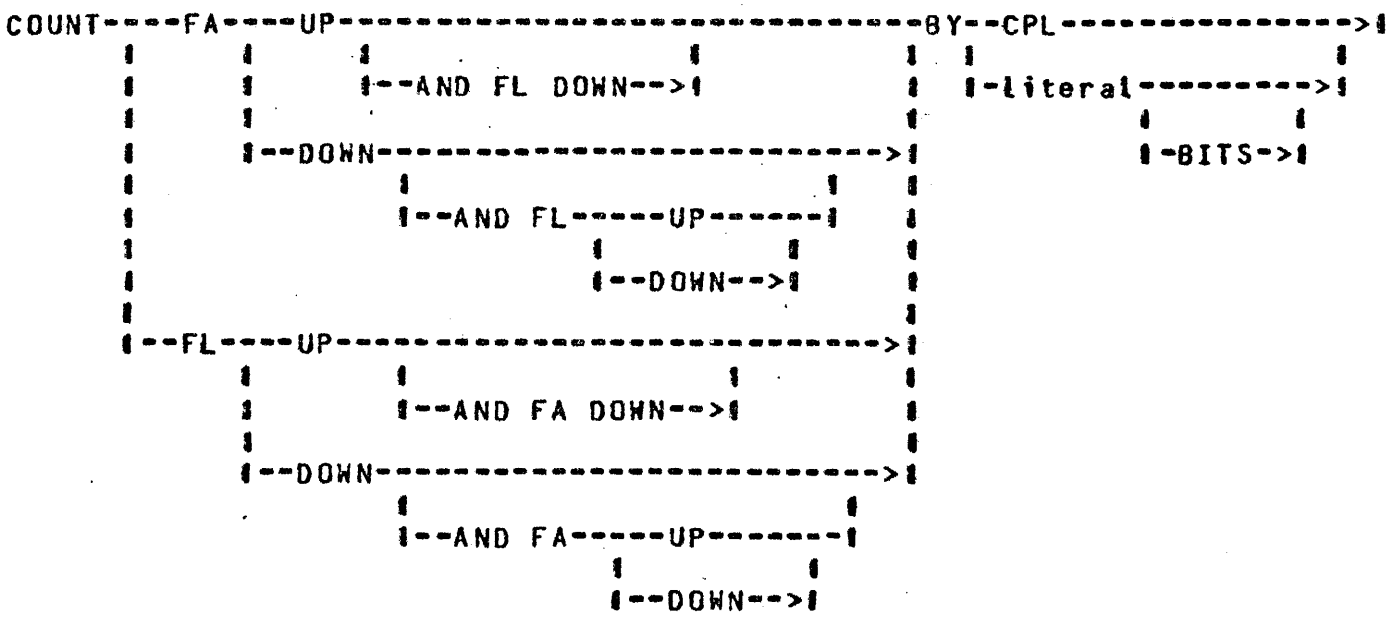
It should be noted that most registers can be adressesed in either of two ways:



COUNT

COUNT

SYNTAX:



SEMANTICS:

This instruction increments or decrements the designated registers by the value of the literal or the contents of the Control Parallel Length (CPL) register. If the value of literal is 0, the value contained in the CPL register is used.

If the FA register is counted down, it may pass through 0 (i.e., if FA=0 and is counted down by 1, it will be set to hex FFFF). If the FL register is counted down, it will not become less than 0.

If either the FA or FL register overflows, wraparound to or through 0 will occur; e.g., if either is equal to the maximum value it can contain and is counted up by 1, it becomes equal to 0.

The literal has a maximum decimal value of 72.

COUNT
cont

COUNT
cont

EXAMPLE:

COUNT FA UP AND FL DOWN BY 10

FA	0000	1001	1010	0111	1111	1011	before (09A7FB)
	--	--	--	--	--	1010	literal + A
FA	0000	1001	1010	1000	0000	0101	after (09A805)

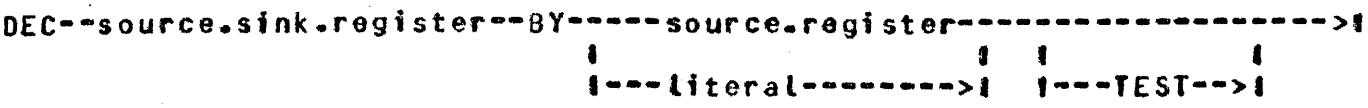
FL	0000	0000	0000	1000	before (0008)
	--	--	--	1010	literal - A
FL	0000	0000	0000	0000	after (0000)

FA is counted up by decimal 10 (hexadecimal A), while FL is counted down by 8 to its minimum value.

DEC

DEC

SYNTAX:



SEMANTICS:

This instruction decrements the contents of a 4-bit source and sink (destination) register by the value of the literal or the contents of a 4-bit source register. The result is placed in source.sink.register; the contents of source.register remain unchanged. (See also: INC.)

The register may be any of the following:

source.sink.register	source.register
CA CB *CC *CD	source.sink.register
FT FU	BICN
FLC FLD FLE FLF	FLCN
LA LB LC LD LE LF	INCN (available on B1720 only)
TA TB TC TD TE TF	XYCN
TOPM (avaiable on B1720 only)	XYST

* CC and CD represent processor interrupts and flags

The literal has a decimal range from 0 to 15.

If the TEST option is used and source.sink.register underflows (is decremented beyond 0, the smallest value it can contain), the next microinstruction is skipped. If underflow does not occur or if the TEST option is not used, the next microinstruction is executed.

NOTE: All 4-bit registers count modulo 16; e.g., if a register contains a value of 0 and is decremented by 2, it underflows to a value of 14.

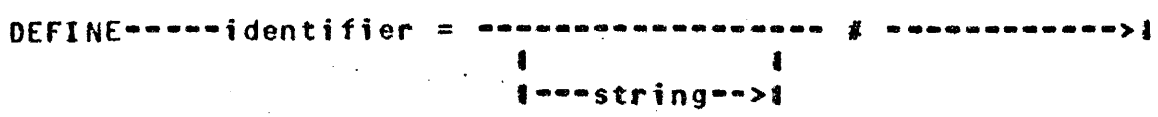
EXAMPLE:

```
DEC TB BY 7
DEC FLD BY LC TEST
```

DEFINE

DEFINE

SYNTAX:



SEMANTICS:

This declaration assigns a name (identifier) to a string of characters. Any subsequent reference to the identifier is replaced by the string.

String may be a scratchpad name (24 or 48-bit); a register name; a literal; a part of one instruction; an entire instruction, part of which may have been previously DEFINED; or empty. It may neither begin with a pound sign (#) nor contain any embedded pound signs.

The entire DEFINE declaration must be contained on one card, and all DEFINEs must be declared prior to any executable instruction.

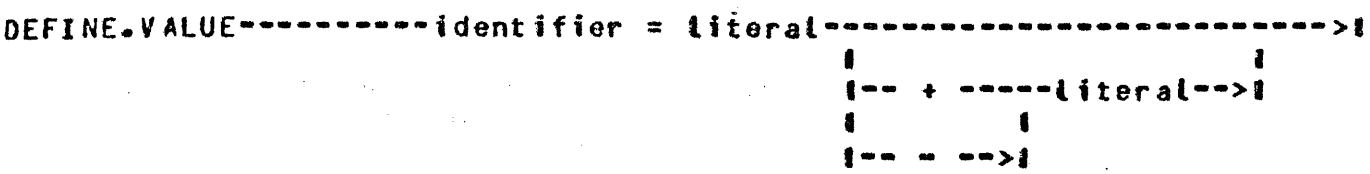
Nested DEFINEs are allowed up to 13 levels.

EXAMPLE:

```
DEFINE SOURCE.POINTER = S3#          % LOAD F FROM SOURCE.POINTER
DEFINE OP.REG = L#                   % CLEAR OP.REG
DEFINE TEST.OP = 08000000#           % MOVE TEST-OP TO OP.REG
DEFINE HINT = CC(3)#                 % RESET HINT
DEFINE IGNORE.HALT = RESET HINT#     % IGNORE.HALT
```

DEFINE.VALUE

SYNTAX:



SEMANTICS:

This instruction assigns the value of the arithmetic result of the literals to the identifier. Any occurrence of the identifier in the program is replaced by its assigned value.

DEFINE.VALUE creates up to a 24-bit literal. Values less than zero are in 2's complement notation and are 24 bits long.

Previously defined identifiers may be used as literals.

The literal may be a hex value, a binary value, or a character used as two hex values.

EXAMPLE:

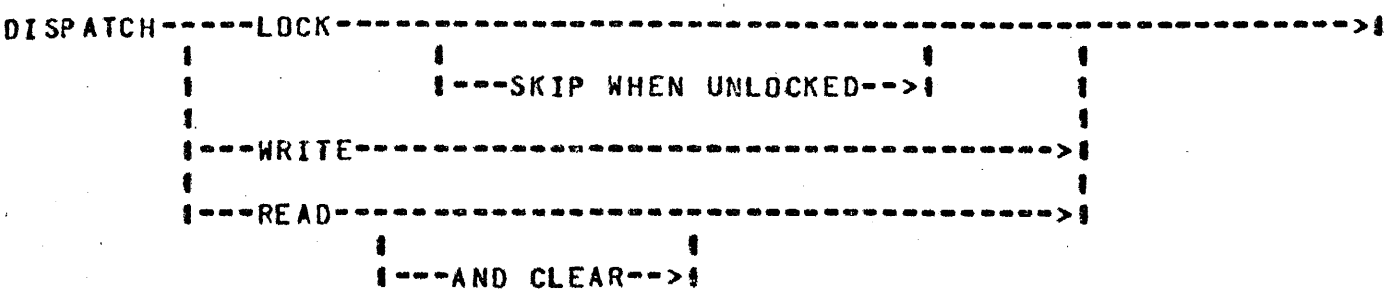
```
DEFINE.VALUE AA = @50@                  % VALUE is hex 000050
DEFINE B = AA + 1                       % VALUE is hex 000051
DEFINE C = AA - 3                       % VALUE is hex 00004D
DEFINE.VALUE F03 = @(1)00100@ + 4       % VALUE is hex 000006
```

DISPATCH

DISPATCH

(available on B1720 systems only)

SYNTAX:



SEMANTICS:

This instruction sends a message (e.g., an I/O descriptor address) from the processor to a device on an I/O port.

Before sending a message to a port, the processor should first attempt to gain control of the interrupt system with a DISPATCH LOCK. This is necessary because the interrupt system is shared by all ports.

DISPATCH LOCK locks (marks as "in use") the interrupt system. If the interrupt system is already locked, the next microinstruction is skipped.

DISPATCH LOCK SKIP WHEN UNLOCKED locks the interrupt system or skips the next microinstruction if the interrupt system is already unlocked.

DISPATCH WRITE sends a 24-bit message to a port. Before a DISPATCH WRITE is executed, the L register must contain the 24-bit message; the seven least-significant bits of the T register must contain the destination port (bits 17-19) and channel numbers (bits 20-23). The contents of the L register are then stored in the Dispatch buffer (main memory locations 0-23), and the port and channel numbers are transferred to a hardware register (Dispatch register) in the port interchange. The contents of the L and T register remain unchanged.

DISPATCH
cont

DISPATCH
cont

SEMANTICS cont:

DISPATCH READ transfers both a 24-bit message from the Dispatch buffer to the L register, and the source port and channel numbers to the seven least-significant bits of the T register.

NOTE: If T(23) is found set after a DISPATCH READ and the source port is an I/O multiplexor, a main memory parity error was encountered during the fetch of an I/O descriptor address or an I/O descriptor, or during a RESULT SWAP operation. Consequently, the message transferred to the L register will be the address +24 of the parity error.

DISPATCH READ AND CLEAR does everythin a DISPATCH READ will do and in addition clears the Interrupt Condition (INCN) register. That is, it RESETs all INCN bits to zero.

Only the least-significant seven bits of the T register are involved in any DISPATCH operation.

If the SKIP WHEN UNLOCKED option is used with any variant other than a DISPATCH LOCK, the next-micro instruction is skipped.

EMIT.RETURN.TO.EXTERNAL

EMIT.RETURN.TO.EXTERNAL

SYNTAX:

EMIT.RETURN.TO.EXTERNAL----------->1

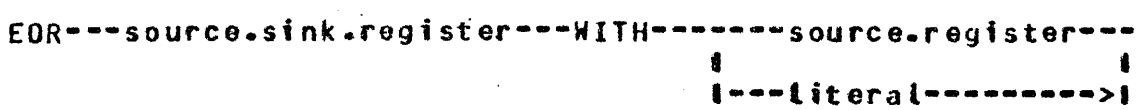
SEMANTICS:

This instruction causes the compiler to emit the common code necessary to get back to the main segment from the external segment. This code also includes the return code when the segment is exited for the last time. (See: Segmentation.)

EOR

EOR

SYNTAX:



SEMANTICS:

This instruction logically EXCLUSIVE ORs the bits in a 4-bit source and sink (destination) register with the value of the literal or the contents of a 4-bit source register. The result is placed in source.sink.register; the contents of source.register remain unchanged. (See also: AND and OR.)

The register may be any of the following:

source.sink.register	source.register
CA CB *CC *CD	source.sink.register
FT FU	BICN
FLC FLD FLE FLF	FLCN
LA L8 LC LD LE LF	INCN (available on B1720 only)
TA T8 TC TD TE TF	PERR (available on B1720 only)
TOPM (avaiable on B1720 only)	XYCN
	XYST

* CC and CD represent processor interrupts and flags

The literal has a decimal range from 0 to 15.

EOR
cont

EOR
cont

SEMANTICS cont:

TABLE 8-2 EOR Truth Table

Source & Sink Register	Literal Source Register			Source & Sink Register
0	EOR	0	Yields	0
0	EOR	1	Yields	1
1	EOR	0	Yields	1
1	EOR	1	Yields	0

EXAMPLE:

EOR TB WITH 3

	TA	TB	TC	TD	TE	TF	
T	0000	0101	1111	0011	0001	0010	before (05F312)
	--	0011	--	--	--	--	EOR (030000)
T	0000	0110	1111	0011	0001	0010	after (06F312)

EXIT

EXIT

SYNTAX:

EXIT---------->|

SEMANTICS:

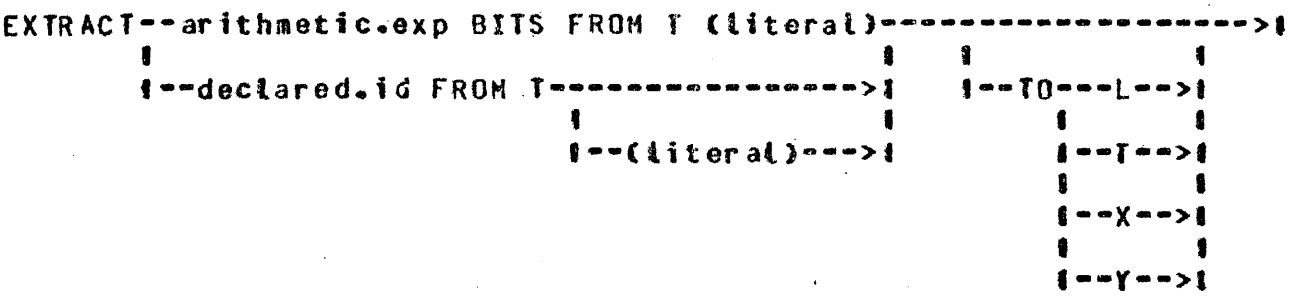
This instruction returns program control to the calling routine by causing the compiler to generate a MOVE TAS TO A operation.

The top of the A stack (TAS) is moved to the ADDRESS (A) register, which is used by the hardware logic as the address of the next microinstruction to be fetched. The stack is decremented automatically by the hardware after the move.

NOTE: MOVE TAS TO A may be used instead of EXIT with the same result.

EXTRACT

SYNTAX:



SEMANTICS:

This instruction isolates the specified bits from the T register and moves them to a destination register (L, T, X, Y). If a destination register is not specified, T is assumed.

The value of the following combinations may not exceed 24 bits:

arithmetic.expression + literal
CPL + literal
DATA.LENGTH of declared.identifier
DATA.LENGTH of declared.identifier + literal
DATA.LENGTH of declared.identifier + DATA.ADDRESS of declared.identifier

NOTES:

1. If arithmetic.expression = 0, CPL indicates the number of bits extracted.

2. If the starting bit for declared.identifier is not specified, its DATA.ADDRESS is used.

EXTRACT
cont

EXTRACT
cont

EXAMPLE:

EXTRACT 4 BITS FROM T(20) TO L

	TA	TB	TC	TD	TE	TF	
T	0000	0001	0011	1000	1110	0100	before (0138E4)
					T(20)		
	LA	LB	LC	LD	LE	LF	
L	1001	1110	0011	1001	1111	1100	before (1E39FC)
L	0000	0000	0000	0000	0000	0100	after (000004)

Register T remains unchanged while its four extracted bits are placed in the L register. The bits are right-justified; leading zeroes are added.

NOTE: EXTRACT 0 BITS FROM T(23) TO a destination register may be specifed, but the programmer must OR into the M register the number of bits to be extracted. Caution must be exercised, however, when ORing into the M register: the machine hardware instruction requires the right-bit pointer for the extraction field, not the left. The hardware also indexes the T register from 1 to 24, left to right, not 0 to 23; the assembler performs this conversion.

FA.POINTS

SYNTAX:

FA.POINTS TO---------arithmetic.expression---------->|

SEMANTICS:

This pseudo-operation does not generate any code. It merely informs the compiler of the current contents of FA. This information is then used when compiling the POINT constructs in the READ, WRITE and POINT instructions.

The FA.POINTS and POINT constructs are provided so that the user may symbolically reference the memory structures declared in a declaration statement. Such references will show up in a cross-reference listing and can often result in automatic code changes when the declaration changes.

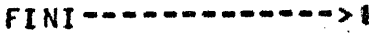
EXAMPLE:

```
DECLARE
    01 STRUCTURE,
         02  DATA.A BIT(10),
         02  DATA.B CHARACTER (20),
         02  DATA.C FIXED;

FA.POINTS TO DATA.A
READ DATA.LENGTH (DATA.A) BITS TO X POINT FA TO DATA.B
POINT FA TO STRUCTURE
MOVE DATA.C TO FA
WRITE DATA.LENGTH (DATA.C) BITS FROM Y POINT FA TO DATA.B
```

FINI

SYNTAX:

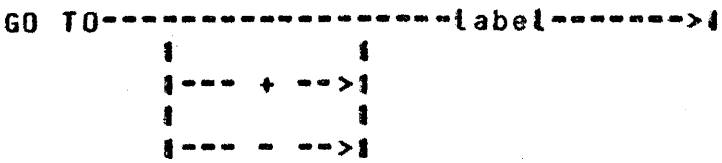


SEMANTICS:

This instruction signals the compiler that the end of the input record has been reached. It should be the last statement in the source program.

GO TO

SYNTAX:



SEMANTICS:

This instruction transfers control to the location specified by label.

Label must be associated with a run time address that has a displacement from the GO TO instruction of less than 4096 microinstructions.

EXAMPLE:

```
GO TO SORT.ROUTINE
GO TO -LOOP.1
GO TO +LOOP.2
```

HALT

HALT

SYNTAX:

HALT---------->|

SEMANTICS:

This instruction brings the processor to an orderly halt. The settings of the Console switches determine the register displayed.

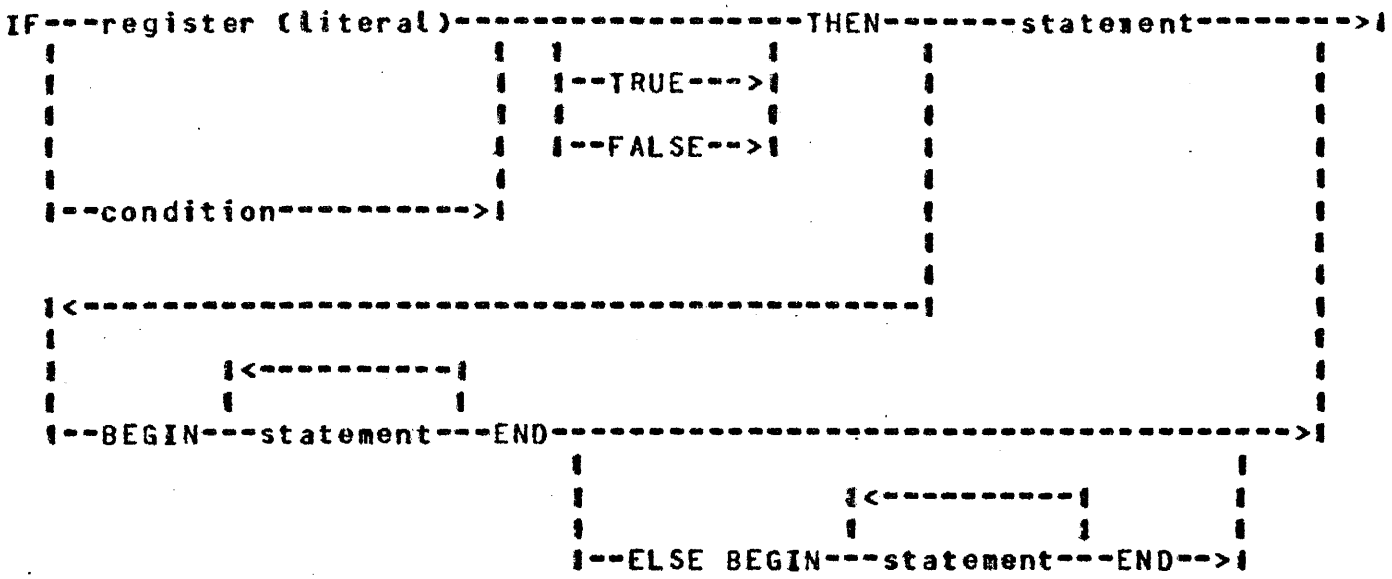
Pressing the START pushbutton on the system Console will cause the processor to again begin executing microinstructions. If the STEP/RUN switch is in the STEP position, only one microinstruction is executed.

IF

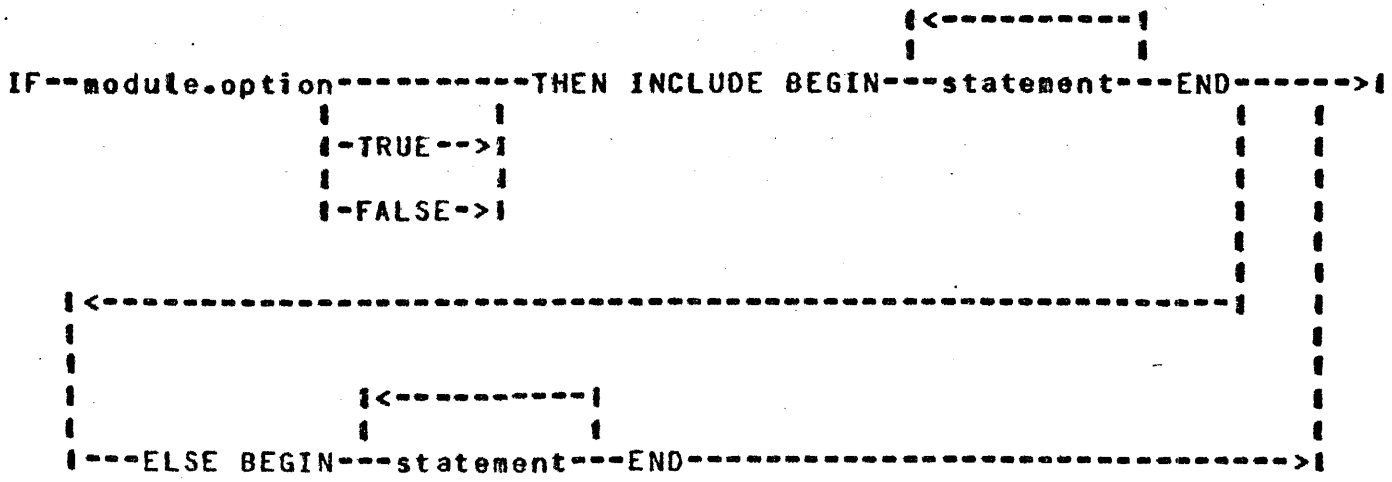
IF

SYNTAX:

FORMAT 1: CONDITIONAL PROGRAM CONTROL



FORMAT 2: CONDITIONAL COMPILATION CONTROL



IF
cont

SEMANTICS:

FORMAT 1: CONDITIONAL PROGRAM CONTROL

This instruction tests a bit(s) for TRUE (one) or FALSE (zero). If the test condition is met, either the specified single statement or the specified BEGIN/END statement(s) is executed. If the test condition is not met, a branch around the first BEGIN/END pair is taken, and the ELSE BEGIN/END statement(s) is executed. (See also: COMPLEMENT, SET and RESET.)

Logical operators are valid on the registers immediately following the IF, with the following restrictions:

1) All registers logically related must be within the same 4-bit group: IF T(0) and T(3) is valid, IF T(2) and T(4) is not.

2) Only two register elements may be logically related: IF T(2) or T(0) is valid, IF T(2) and T(1) and T(0) is not.

3) NOT logic may be applied anywhere: IF NOT (L(3) or NOT L(0)) is valid.

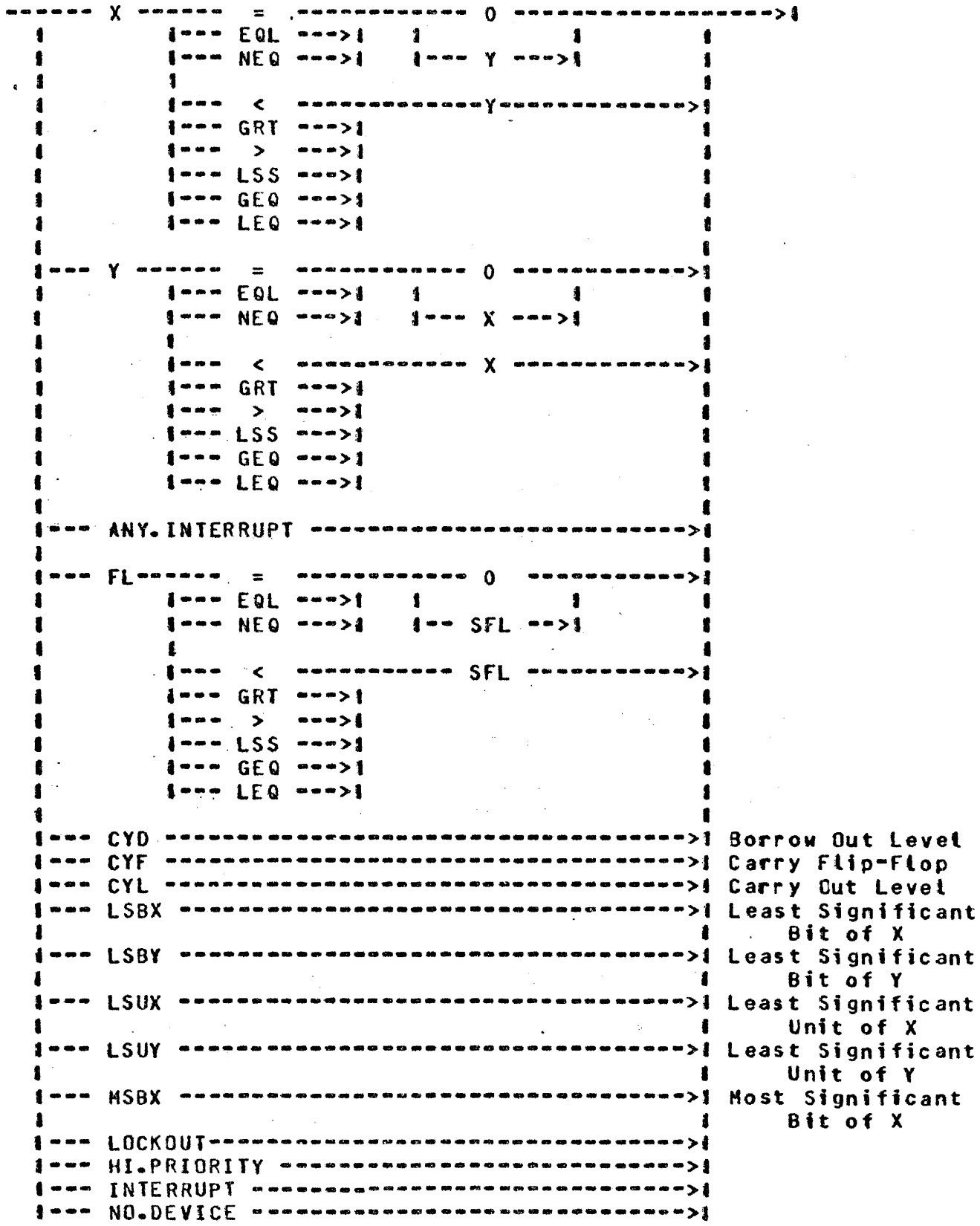
The register may be any 4-bit source and sink (destination) register below:

CA CB CC CD (CC and CD represent processor interrupts and flags)
FT FU
FLC FLD FLE FLF
LA LB LC LD LE LF
TA TB TC TD TE TF
TOPM (avaiable on B1720 only)

The register may also be the FL, FB, L, or T register: all bits must then be in the same 4-bit subfield.

The literal points to the bit position which is to be tested. It has a decimal range from 0 to 3 for a 4-bit register; from 0 to 15 for the FL register; and from 0 to 23 for the FB, L and T register.

The condition may be any of the following conditions available from the condition registers:


X
=
0
EQL
NEQ
Y
<
GRT
>
LSS
GEQ
LEQ
Y
=
0
EQL
NEQ
X
<
GRT
>
LSS
GEQ
LEQ
ANY.INTERRUPT
FL
=
0
EQL
NEQ
SFL
<
GRT
>
LSS
GEQ
LEQ
CYD
Borrow Out Level
CYF
Carry Flip-Flop
CYL
Carry Out Level
LSBX
Least Significant Bit of X
LSBY
Least Significant Bit of Y
LSUX
Least Significant Unit of X
LSUY
Least Significant Unit of Y
MSBX
Most Significant Bit of X
LOCKOUT
HI.PRIORITY
INTERRUPT
NO.DEVICE

IF
cont

SEMANTICS cont:

Any combination of conditions that is contained in one condition registor can be tested using AND/OR logic if all bits can be tested for TRUE (on) or FALSE (off). For example, the following are valid conditions:

```
CYL AND LSUY
CYL OR CYD
```

```
Example:  IF CYL AND LSUY TRUE THEN GO TO END
          IF CYL OR CYD FALSE THEN GO TO BEGIN
```

If TRUE or FALSE is not specified, TRUE is assumed.

Example: IF TD(2) THEN GO TO LABL7

Register TD	Branch To LABL7
0101	NO (bit position two is OFF)
1101	NO (bit position two is OFF)
0111	YES (bit position two is ON)
0011	YES (bit position two is ON)

Note: TD(2) could have been referred to as T(14)

EXAMPLE:

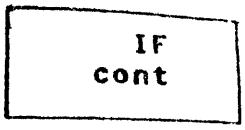
The following examples illustrate Format 1: Conditional Program Control:

```
IF X = Y THEN GO TO *A

IF TB(1) OR TB(3) THEN EXIT

IF LF(2) THEN
         MOVE X TO Y
IF FU(1) FALSE THEN
         COMPLEMENT T(10)
ELSE
         RESET FL(5)
```

IF
cont



EXAMPLE cont:

```
    IF FLF(3) FALSE THEN
       BEGIN
             RESET FB(1) AND FB(3)
             CLEAR S14A
       END

    IF LA(0) THEN
       BEGIN
             MOVE TAS TO T
       END ELSE
             MOVE FA TO T

    IF TD(3) THEN
             MOVE  L TO X
    ELSE
       BEGIN
             MOVE T TO X
             MOVE SUM TO X
       END

    IF LA = 14 THEN
       BEGIN
             MOVE 512 TO X
       END
```

FORMAT 2: CONDITIONAL COMPILATION CONTROL

This instruction should be used for conditional inclusion of code, depending upon the setting of a user-defined, module.option toggle. This module.option toggle is declared and SET or RESET via a module option $ card. (See Appendix A: MIL Compiler Operation.)

More than one module.option toggle can be tested with the same IF statement by using AND/OR logic. If NOT is used in front of any module.option toggle, that module.option toggle is checked for the RESET state. If both TRUE and FALSE are omitted, TRUE is assumed.

Note: A conditional inclusion-block may not be used to include or exclude a BEGIN statement when the associated END statement is not part of the block.

IF
cont

EXAMPLE:

The following are examples of conditional inclusion of code:

```
$ SET DEBUG, RESET TRACE
$ SET TRACE, RESET B1700
```

After processing these $ cards, the module options will be set TRUE or FALSE as follows:

```
DEBUG = TRUE
TRACE = TRUE
B1700 = FALSE
```

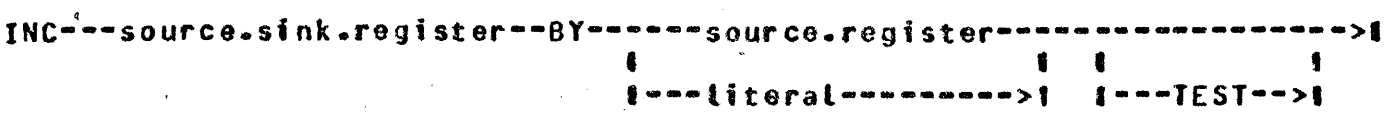
```
IF DEBUG THEN INCLUDE
        CALL DEBUG.ROUTINE
IF TRACE THEN INCLUDE
   BEGIN
        CALL SAVE.REGISTERS
        CALL TRACE.ROUTINE
   END
IF DEBUG AND NOT B1700 INCLUDE
   BEGIN
        MOVE T TO X
   END ELSE
   BEGIN
        MOVE L TO X
        MOVE T TO S0A
   END
IF NOT TRACE OR B1700 INCLUDE
   BEGIN
        MOVE L TO X
        MOVE T TO S1A
   END ELSE
   BEGIN
      CALL TRACE.ROUTINE
      MOVE T TO X
   END
```

Any of the preceding examples may be nested within any of the above BEGIN/END pairs up to a maximum of 15 levels. That is, at any given time during a compilation there may be at most 15 BEGINs that have not been paired with their respective ENDs.

INC

INC

SYNTAX:



SEMANTICS:

This instruction increments the contents of a 4-bit source and sink (destination) register by the value of the literal or the contents of a 4-bit source register. The result is placed in the source and sink register; the contents of the source register remain unchanged. (See also: DEC.)

The register may be any of the following:

source.sink.register	source.register
CA CB *CC *CD	source.sink.register
FT FU	BICN
FLC FLD FLE FLF	FLCN
LA LB LC LD LE LF	INCN (available on B1720 only)
TA TB TC TD TE TF	XYCN
TOPM (avaiable on B1720 only)	XYST

* CC and CD represent processor interrupts and flags

The literal has a decimal range from 0 to 15.

If the TEST option is used and source.sink.register overflows (is incremented beyond 15, the largest value it can contain), the next microinstruction is skipped. If overflow does not occur or if the TEST option is not used, the next microinstruction is executed.

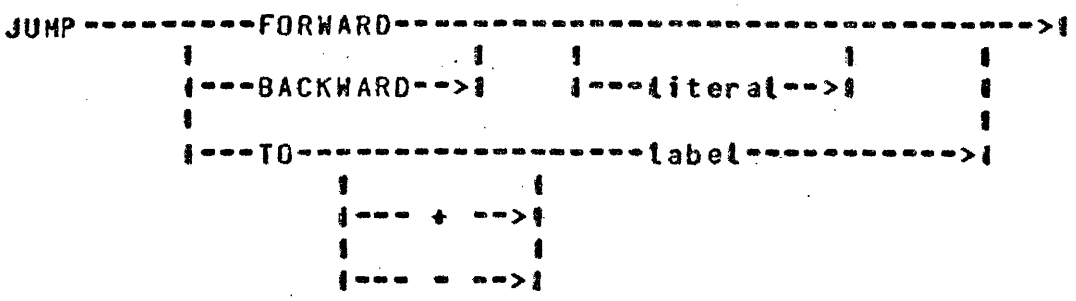
NOTE: All 4-bit registers count modulo 16; e.g., if a register contains a value of 15 and is incremented by 2, it overflows to a value of 1.

EXAMPLE:

```
INC LB BY 7
INC FLD BY BICN TEST
```

JUMP

SYNTAX:



SEMANTICS:

This instruction transfers control to the designated location.

The address of label is limited to a maximum relative displacement of plus or minus 4095 microinstructions.

The literal has a decimal range from 0 to 4095.

If literal is not specified, FORWARD/BACKWARD causes the compiler to generate a JUMP instruction with a displacement of zero and a direction sign of plus or minus. This is to facilitate ORing the actual displacement into the the M register prior to the execution of a JUMP instruction.

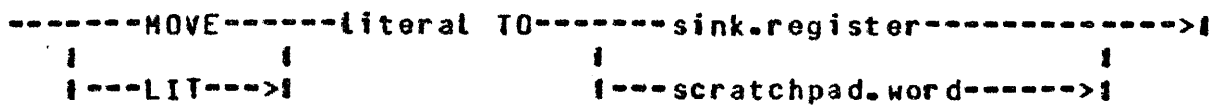
EXAMPLE:

```
JUMP TO +LOOP.1
JUMP TO END.OF.CODE.LABEL
JUMP FORWARD 12
JUMP BACKWARD
```

LIT

LIT

SYNTAX:



SEMANTICS:

This instruction moves a literal to any sink (destination) register (except the M register) or to any 24-bit scratchpad word. (See also: MOVE.)

The literal may be any decimal integer from 0 to 16777215, a hexadecimal number from 0X0 to 0XFFFFF0, a binary number from 0(1)0B to 0(1)111111111111111111111111B, or a character string up to three characters in length. Leading zeros are not required unless the actual value of the literal is zero. The value of the literal should not exceed the maximum value that the sink register can contain; if less, the zero fill occurs.

Literal moves to a 24-bit scratchpad word generate MOVE literal TO TAS followed by MOVE TAS TO scratchpad.word.

PROGRAMMING NOTE

It is recommended that the MOVE instruction be used instead of LIT.

EXAMPLE:

MOVE 12 TO L

	LA	LB	LC	LD	LE	LF	
L	0011	0000	1001	1010	0001	0011	before (309A13)
	--	--	--	--	--	--	LIT (C)
L	0000	0000	0000	0000	0000	1100	after (00000C)

LOAD

SYNTAX:

LOAD F FROM-----double.scratchpad.word----->!

SEMANTICS:

This instruction moves any 48-bit double scratchpad word (S0...S15) to the Field (F) register.

NOTE: The compiler will generate two MOVE instructions for B1710 systems.

EXAMPLE:

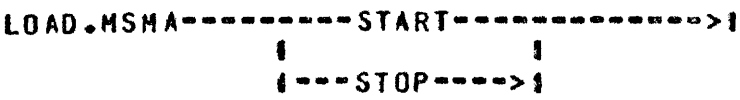
LOAD F FROM S11

LOAD.MSMA

LOAD.MSMA

(available on 81720 systems only)

SYNTAX:



SEMANTICS:

This pseudo-operation causes the compiler to either start or stop prefacing all emitted microcode with the first 16 bits of a MOVE 24 BIT LITERAL TO MSMA instruction.

The above action is required when a microprogram is to be loaded into control memory from a cassette tape while the system is in the TAPE mode. The action of the hardware while in this mode is as follows:

```
.READLOOP
     READ 16 BITS FROM THE CASSETTE TO THE U-REGISTER
     MOVE U TO M
     IF M = FIRST HALF OF 24-BIT LITERAL MOVE, THEN READ 16 BITS
          FROM THE CASSETTE TO U
     EXECUTE THE MICRO-OPERATOR IN M
          (IF M=@9D00@=MOVE 24-BIT LITERAL TO THE CONTROL MEMORY
          WORD ADDRESSED BY THE A-REGISTER; THEN U, WHICH NOW
          CONTAINS THE ACTUAL MICROINSTRUCTION, IS MOVED TO
          CONTROL MEMORY ADDRESSED BY THE A-REGISTER AND A IS
          INCREMENTED BY 1)
     IF M = CASSETTE STOP THEN
          STOP CASSETTE AND HALT PROCESSOR
     ELSE
          JUMP TO -READLOOP
```

No statement between LOAD.MSMA START and its corresponding LOAD.MSMA STOP may reference any label which has not been declared prior to the LOAD.MSMA STOP.

LOAD.MSMA
cont

LOAD.MSMA
cont

EXAMPLE:

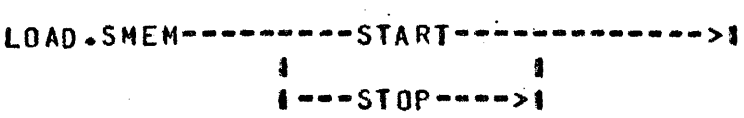
The following source code could be used to enable a microprogram to be loaded from a cassette into control memory, beginning at control memory address zero:

```
MOVE 0 TO A
SEGMENT ANYNAME AT 0
LOAD.MSMA START
        .
        .
(Microprogram)
        .
        .
LOAD.MSMA STOP
MOVE 0 TO A
CASSETTE STOP
```

LOAD.SMEM

LOAD.SMEM

SYNTAX:



SEMANTICS:

This pseudo-instruction causes the compiler to either start or stop appending each microinstruction with the following instructions:

```
MOVE 24 BIT LITERAL TO X
WRITE (25) BITS FROM X
WRITE 16 BITS FROM X INC FA
```

These instructions are required when a microprogram is to be loaded into main memory from a cassette tape while the system is in the TAPE mode.

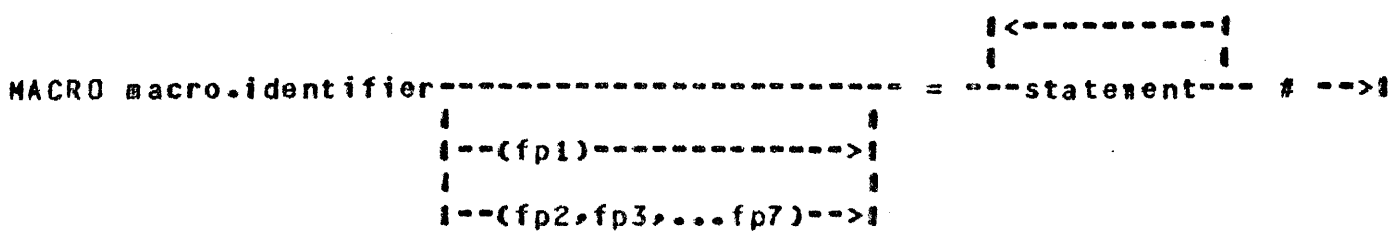
EXAMPLE:

```
MOVE 4096 TO FA          % START ADDRESS
LOAD.SMEM START
      .
      .
(microprogram)
      .
      .
LOAD.SMEM STOP
CASSETTE STOP
```

NOTE: The FA must start at a mod 32 value.

MACRO

SYNTAX:



SEMANTICS:

This declaration assigns a name (macro.identifier) to a series of statements and declares any formal parameters (fp) which may be used in the macro definition. Any subsequent reference to macro.identifier is replaced by the actual parameters in the reference.

The actual parameters used in the reference to a macro must be single identifiers and must not not contain embedded blanks or special characters. The one exception is that an actual parameter could be a DEFINE identifier and therefore could contain an embedded dash. However, the DEFINE identifier itself would then have to define a valid, actual parameter. For example, X, 3, H801F, and TO are valid, actual parameters, but 3 TO X and (are not. Actual parameters may not be omitted; as with formal parameters, they must be enclosed in parentheses and separated by commas.

The macro declaration must be contained on one line and must be terminated with an equal sign (=).

The macro defintion must then follow with one statement per line. The last statement must be terminated by a pound sign (#); for this reason, a MACRO must not itself contain a pound sign. A MACRO may reference another MACRO or a DEFINE which has been previously declared but must not be recursive.

All MACROS must be declared prior to any executable instructions.

MACRO
cont

MACRO
cont

EXAMPLE:

The declaration

```
MACRO WRITEM(WRITEM1, WRITEM2, WRITEM3)=
    XCH WRITEM1 F WRITEM1
    WRITE 24 BITS FROM WRITEM2 WRITEM3 FA AND DEC FL
    XCH WRITEM1 F WRITEM1#
```

when referenced as

```
WRITEM(SO, X, INC)
```

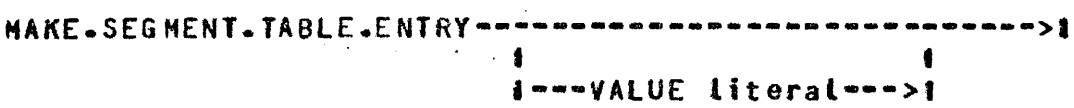
results in the reference being replaced by the following in-line code:

```
XCH SO F SO
WRITE 24 BITS FROM X INC FA AND DEC FL
XCH SO F SO
```

MAKE.SEGMENT.TABLE.ENTRY

MAKE.SEGMENT.TABLE.ENTRY

SYNTAX:

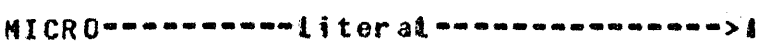


SEMANTICS:

This instruction causes an entry to be written to a segment table which connects labeled segments with their sequence number. This facilitates the creation of the segment table required for normal state segmented execution. (See: Segmentation.)

MICRO

SYNTAX:



SEMANTICS:

This instruction places a 16-bit constant in line. The programmer is responsible for providing any protection that may be needed to prevent a MICRO from executing: Therefore this instruction should be used rarely.

The literal has a decimal range from 0 to 65535.

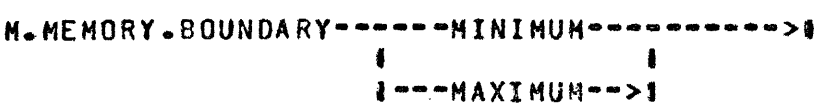
EXAMPLE:

```
MICRO @0001@      %  THIS PLACES A HALT MICROINSTRUCTION IN LINE
MICRO @83AA@      %  THIS IS EQUIVALENT TO "MOVE @AA@ TO L"
MICRO "22"        %  THIS IS EQUIVALENT TO @F2F2@
MICRO "HI"        %  "HI" = @C8C9@
```

M.MEMORY.BOUNDARY

(available on B1720 systems only)

SYNTAX:



SEMANTICS:

This instruction sets the M.MEMORY boundary within the IPB (Interpreter Parameter Block) of a HIL program.

MINIMUM specifies that the program will be loaded into M-Memory at the current code address.

MAXIMUM specifies that the M-Memory loading will stop at the current code address.

MONITOR

MONITOR

SYNTAX:

MONITOR-----literal----->1

SEMANTICS:

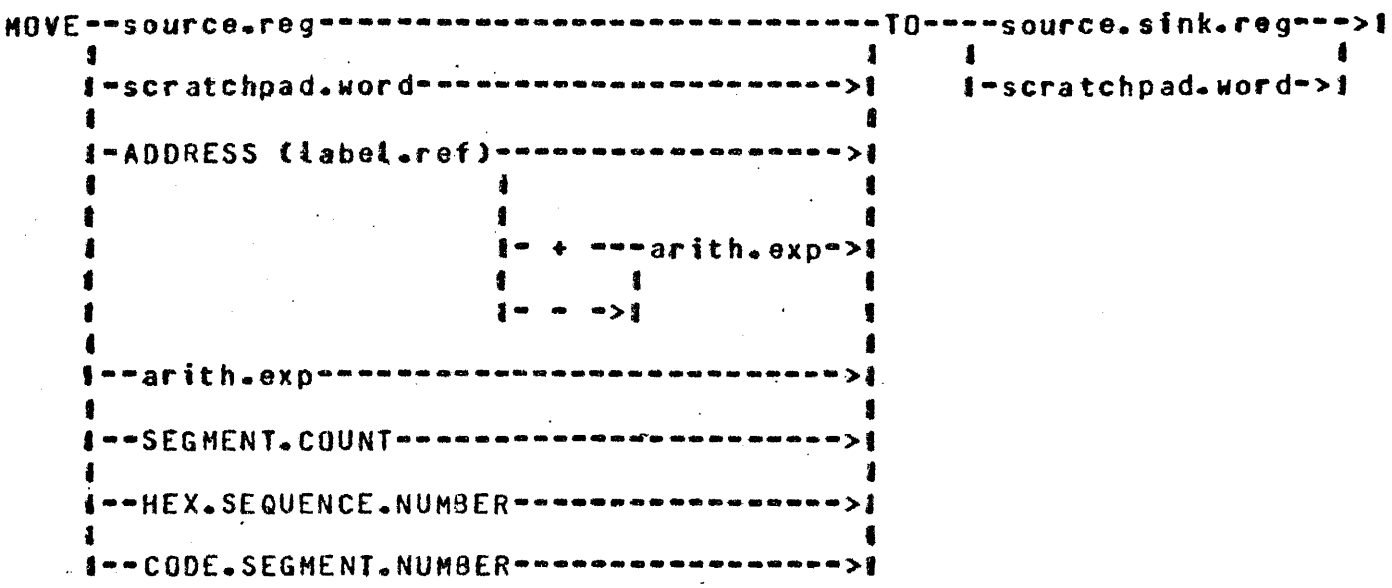
This instruction emits the monitor micro-operator with the literal occurrence identifier. (See also Appendix B: MONITOR.)

The literal has a decimal range from 0 to 255.

EXAMPLE:

```
MONITOR 5
```

SYNTAX:



SEMANTICS:

This instruction copies the specified information into a source and sink (destination) register or scratchpad word.

ADDRESS (label.ref) is a literal value equal to the code address of the label reference.

SEGMENT.COUNT is a literal value equal to the number of times a segment statement has occurred.

HEX.SEQUENCE.NUMBER is a literal value equal to the last six digits of the source statement sequence number.

CODE.SEGMENT.NUMBER is a literal value equal to the current code segment number.

MOVE
cont

SEMANTICS cont:

The following are restrictions on an S-Memory Processor.

1. If ADDRESS or arith.exp has a value greater than 255, and source.sink.reg is CP, the move will not take place.

2. If source.reg is U, source.sink.reg may not be TAS, M, or A.

3. If source.reg is A, CP, M, or DATA, source.sink.reg may not be a 4-bit register.

4. If source.reg is SUM or DIFF, source.sink.reg may not be CMND or DATA.

The following are restrictions on both an S-Memory and M-Memory Processor.

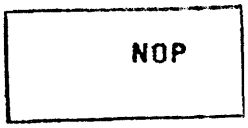
1. When source.reg is DATA, source.sink.reg may not be DATA or CMND.

2. When source.sink.reg is M, the operation is changed to a BIT-OR which modifies the next micro-operation; it does not modify the instructions stored in memory. In tape mode no BIT-OR takes place. A literal value generated from ADDRESS, arith.exp, or SEGMENT.COUNT may not be moved to the M register.

EXAMPLE:

```
MOVE X TO Y
MOVE 48 TO S1A
MOVE ADDRESS (+ GLOP) TO T
MOVE 10 TO TA
MOVE S12A TO S10B
MOVE ADDRESS (BLAH) + 16 * 8 - 1 TO FA
MOVE SEGMENT.COUNT TO T
MOVE (81+(3*10)-1)/2 TO Y
```

NOP



SYNTAX:

SEMANTICS:

This NO OPERATION instruction does nothing except use one clock cycle and take up one word of control or main memory.

NORMALIZE

NORMALIZE

SYNTAX:

NORMALIZE------------------>|

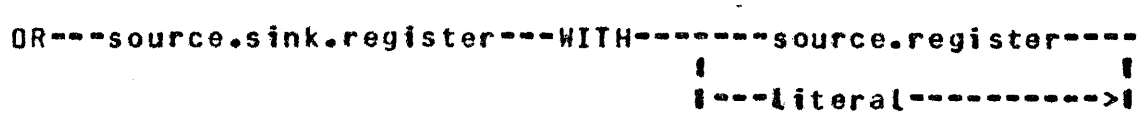
SEMANTICS:

This instruction shifts the contents of the X register left while counting the FL register down until either the most-significant bit of X (determined by CPL) equals 1 or FL equals 0. If the most-significant bit of X is already 1, or if FL is already 0, then no shift takes place.

OR

OR

SYNTAX:



SEMANTICS:

This instruction is used to logically OR the contents of a 4-bit source and sink (destination) register with the value of the literal or the contents of a 4-bit source register. The result is placed in source.sink.register; the contents of source.register remain unchanged. (See also: AND and EOR.)

The register may be any of the following:

source.sink.register	source.register
CA CB *CC *CD	source.sink.register
FT FU	BICN
FLC FLD FLE FLF	FLCN
LA LB LC LD LE LF	INCN (available on B1720 only)
TA TB TC TD TE TF	PERR (available on B1720 only)
TOPM (avaiable on B1720 only)	XYCN
	XYST

* CC and CD represent processor interrupts and flags

The literal has a decimal range from 0 to 15.

OR
cont

OR
cont

SEMANTICS cont:

TABLE 5-3 OR Truth Table

Source & Sink Register		Literal Source Register		Source & SinK Register
0	OR	0	Yields	0
1	OR	0	Yields	1
0	OR	1	Yields	1
1	OR	1	Yields	1

EXAMPLE:

OR TB WITH 3

	TA	TB	TC	TD	TE	TF	
T	0000	0101	1111	0011	0001	0010	before (05F312)
	--	0011	--	--	--	--	literal
T	0000	0111	1111	0011	0001	0010	after (07F312)

OVERLAY

OVERLAY

(Available on 81720 systems only)

SYNTAX:

OVERLAY---------->|

SEMANTICS:

This instruction overlays control memory from main memory. Before an overlay is initiated the L register must contain the first control memory overlay address, the FA register must contain the beginning main memory address, and the FL register must contain the length in bits to be overlayed. Overlay will continue until the FL register equals 0 or the A register is out of bounds. If the A register goes out of bounds, FA contains the address of the next microinstruction in main memory; FL contains the length in bits of unfetched data.

The action of the hardware executing this instruction is as follows:

```
MOVE A TO TAS
MOVE L TO A
READ 16 BITS TO L INC FA AND DEC FL
MOVE L TO CONTROL MEMORY ADDRESSED BY A
INC A
TEST FL=0 OR A OUT OF BOUNDS
NO LOOP TO READ EVENT
YES END INSTRUCTION
```

PAGE

PAGE

SYNTAX:

PAGE----------->|

SEMANTICS:

This instruction causes the source listing to skip to the top of a new page at compile time. Code is not generated.

POINT

POINT

SYNTAX:

POINT FA TO------arithmetic.expression------>|

SEMANTICS:

This pseudo-operation causes the compiler to generate an instruction that adjusts the value of FA to the value of the arithmetic expression.

Prior to the execution of this instruction, the compiler must have been given some knowledge of the contents of FA. This can be done via:

MOVE arithmetic.expression TO FA

or

FA.POINTS TO arithmetic.expression

FA will be adjusted by up to 144 bits as a result of this command. (A warning message will result if the adjustment is greater than 72 bits). (See also: READ and WRITE.)

EXAMPLE:

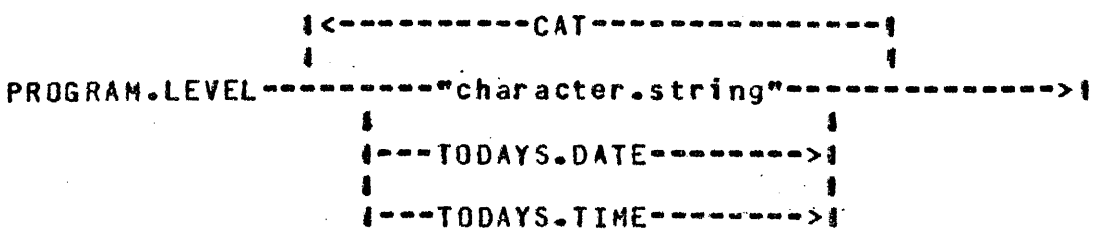
```
DECLARE
     01  STRUCTURE,
          02  DATA.A BIT(10),
          02  DATA.B CHARACTER(20),
          02  DATA.C FIXED;

FA.POINTS TO DATA.A
READ DATA.LENGTH (DATA.A) BITS TO X POINT FA TO DATA.B
POINT FA TO STRUCTURE
MOVE DATA.C TO FA
WRITE DATA.LENGTH (DATA.C)) BITS FROM Y POINT FA TO DATA.B
```

PROGRAM.LEVEL

PROGRAM.LEVEL

SYNTAX:



SEMANTICS:

This instruction places forty characters of information into the PROGRAM.LEVEL location of the IPB (Interpreter Parameter Block).

If the TITLE statement is unused, the title headings of the program listing will reflect the PROGRAM.LEVEL information.

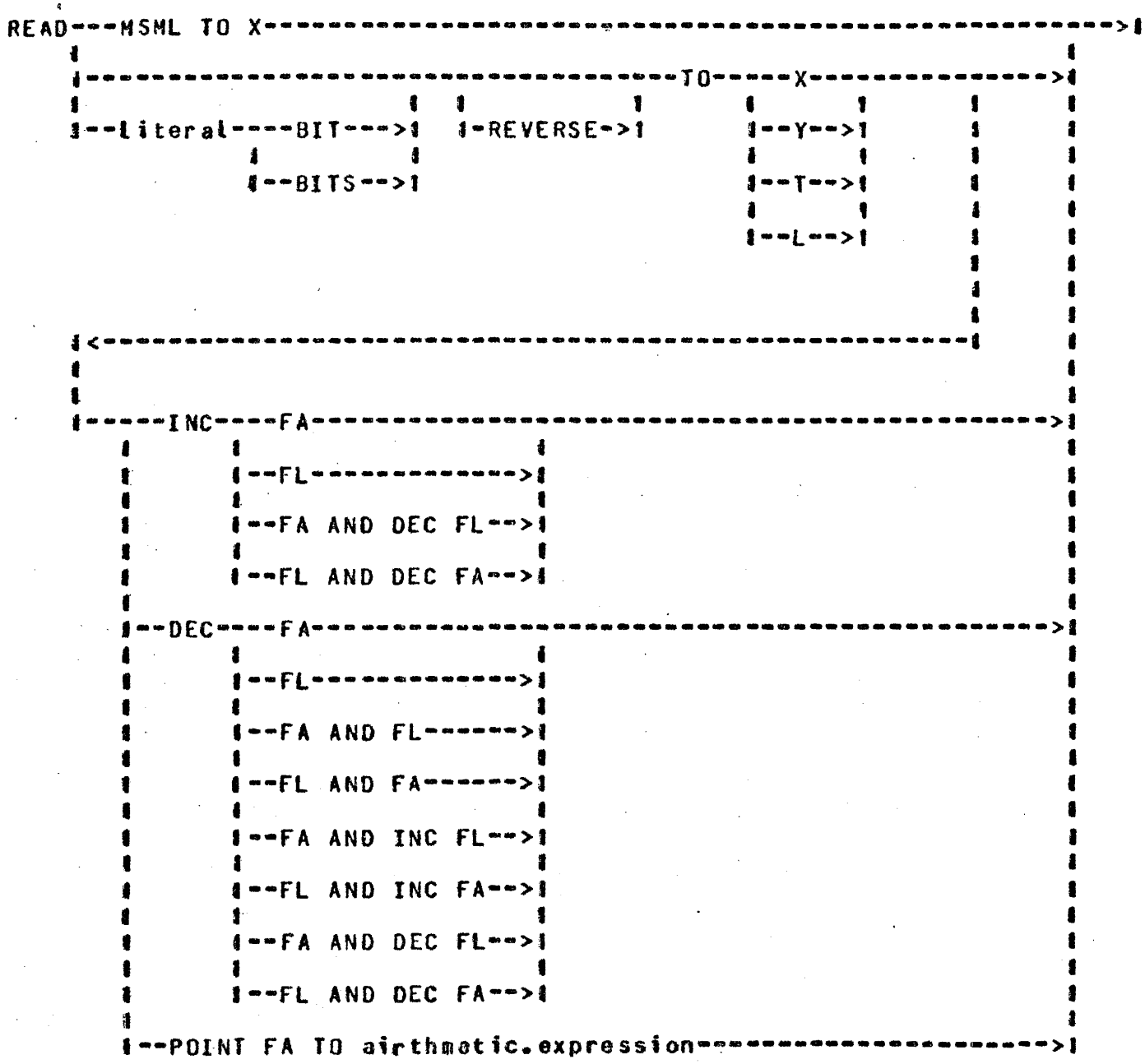
EXAMPLE

```
PROGRAM.LEVEL "THIS IS A SUBHEADING" CAT TODAYS.TIME
```

READ

READ

SYNTAX:



READ
cont

READ
cont

SEMANTICS:

An M-Memory READ (MSML TO X) instruction reads to the X register the 16 bits in M-Memory pointed to by the contents of the L register. The contents of L must be modulo 16. This facility is not available on S-Memory Processors.

An S-Memory READ instruction reads from 0 to 24 bits of information from S-Memory into one of the allowable sink (destination) registers: X, Y, T, or L.

If the literal is zero or is not specified, the field length is given by the contents of CPL. The read data will be right justified in the selected sink register. If the field length is zero then X, Y, T, or L will be set to zero.

Normally, on an S-Memory read, the contents of the FA register point to the first bit of the field to be read. If the REVERSE option is used, the contents of the FA register point to the last bit + 1 of the field to be read. The sink register receives the contents of this field as if it had been read in a forward direction.

INC/DEC adjusts FA/FL by the field length after the operation but in the same microinstruction.

POINT FA adjusts FA by up to 144 + field length bits after the operation. (A warinig message will be issued if the adjustment is greater than 72 + field length bits). The POINT FA option can be used only if literal BIT(S) is specified and is greater than 0. (See also: FA.POINTS and POINT.)

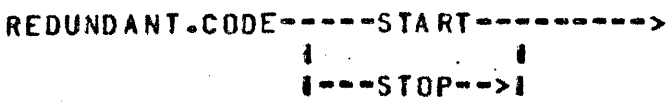
EXAMPLE:

```
READ MSML TO X
READ 24 BITS TO X
READ TO Y INC FA
READ 2 BITS REVERSE TO T DEC FA AND FL
READ REVERSE TO L INC FL
READ 10 BITS TO T POINT FA TO 100
```

REDUNDANT.CODE

REDUNDANT.CODE

SYNTAX:



SEMANTICS:

This instruction causes the compiler to duplicate each micro. It is used to facilitate loading programs to memory.

RESERVE.SPACE

RESERVE.SPACE

SYNTAX:

RESERVE.SPACE FOR-----arithmetic.expression-----BITS---------->

SEMANTICS:

This instruction causes the compiler to emit a sufficient number of NOP's (i.e., 300003) to allow for the number of bits specified by arithmetic.expression.

The actual amount of space reserved will always be MOD 16; therefore up to 15 bits more than that specified by the arithmetic.expression may be reserved.

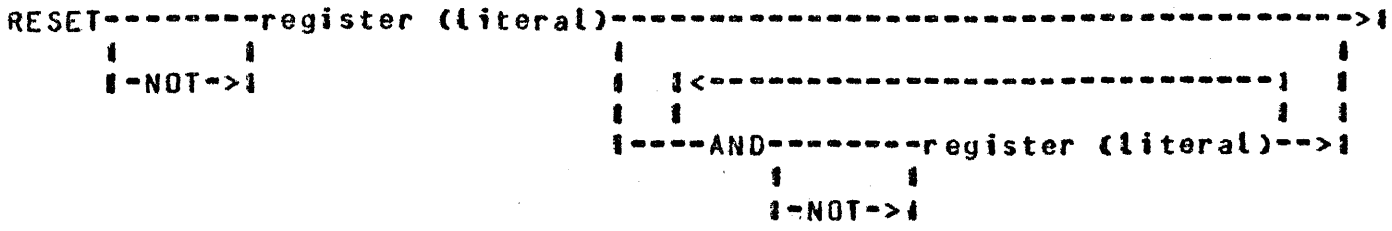
EXAMPLE:

```
  DECLARE IO.DESCRIPTOR BIT(188);
  .
  .
  .
DESC.LOCN
  RESERVE.SPACE FOR DATA.LENGTH(IO.DESCRIPTOR) BITS
```

RESET

RESET

SYNTAX:



SEMANTICS:

This instruction RESETs (sets to zero) the bit specified by the literal into the register. By using the options, more than one bit in any one register can be reset with the same instruction if ALL BITS are in the SAME 4-BIT REGISTER. (See also: COMPLEMENT and SET.)

The register may be any 4-bit source and sink (destination) register below:

CA CB CC CD (CC and CD represent processor interrupts and flags)
FT FU
FLC FLD FLE FLF
LA LB LC LD LE LF
TA TB TC TD TE TF
TOPM (avaiable on B1720 only)

It may also be the FL, FB, L, or T register: all bits must then be in the same 4-bit subfield.

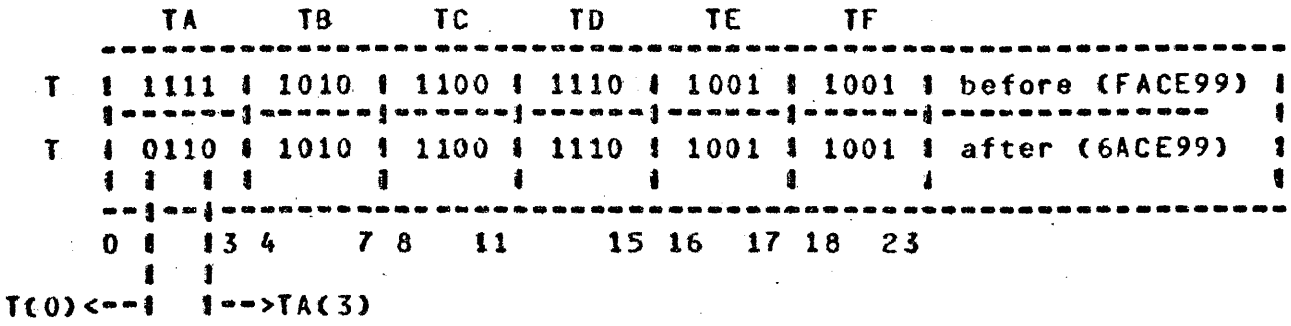
The literal has a decimal range from 0 to 3 for a 4-bit register; from 0 to 15 for the FL register; and from 0 to 23 for the FB, L, and T registers.

RESET
cont

RESET
cont

EXAMPLE:

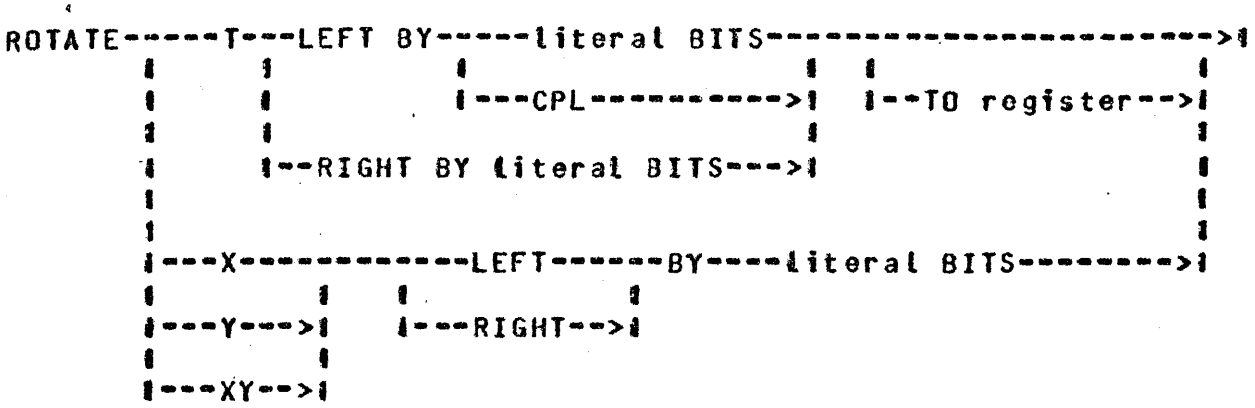
RESET T(0) AND TA(3)



ROTATE

ROTATE

SYNTAX:



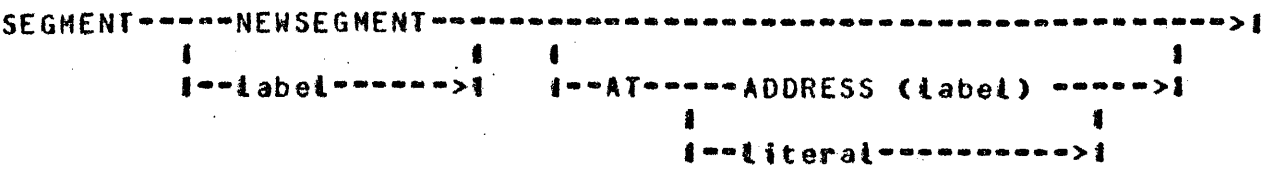
SEMANTICS:

See SHIFT/ROTATE T and SHIFT/ROTATE X/Y/XY.

SEGMENT

SEGMENT

SYNTAX:



Note: The literal must be MOD 16.

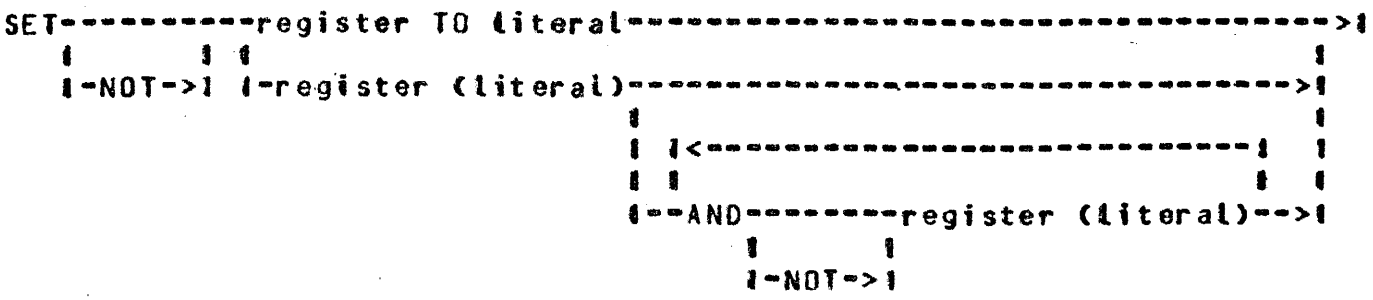
SEMANTICS:

See: Segmentation.

SET

SET

SYNTAX:



SEMANTICS:

This instruction SETs the register to the value of the literal or SETs (bit=one) the bit specified by the literal into the register. By using the options more than one bit in any one register can be set with the same instruction if ALL BITS are in the SAME 4-BIT REGISTER. (See also: COMPLEMENT and RESET.)

SET register TO literal: The register may by any 4-bit source and sink (destination) register listed below.

CA CB CC CD (CC and CD represent processor interrupts and flags)
FT FU
FLC FLD FLE FLF
LA LB LC LD LE LF
TA TB TC TD TE TF
TOPH (available on B1720 systems only)

It may also be the CPU register. If CPU is used, the literal has a decimal range from 0 to 3; otherwise the literal has a range from 0 to 15.

SET register (literal): The register may be any 4-bit source and sink register listed above. It may also be the FL, FB, L, or T register: all bits must then be in the same 4-bit subfield. The literal has a decimal range from 0 to 3 for a 4-bit register; from 0 to 15 for the FL register; and from 0 to 23 for the FB, L, and T registers.

SET
cont

SET
cont

EXAMPLE:

SET TA TO 3

	TA	TB	TC	TD	TE	TF	
T	1111	0100	0101	0110	0111	1000	before (F45678)
T	0011	0100	0101	0110	0111	1000	after (345678)

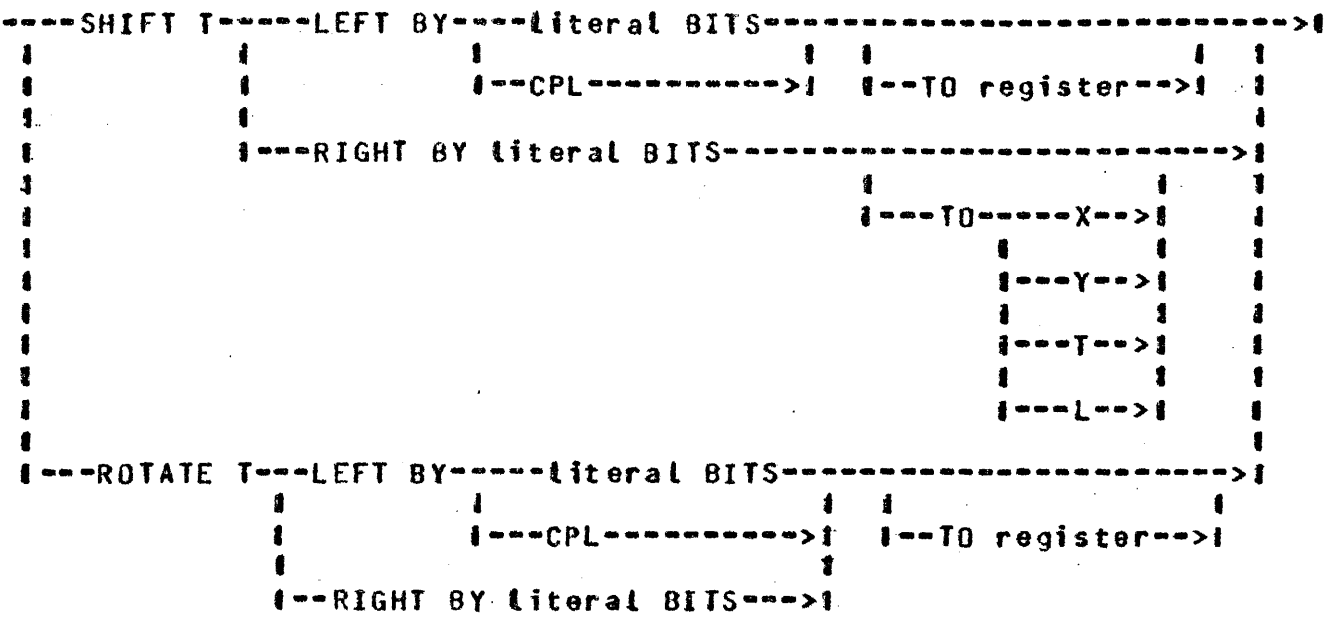
SET TC(2) AND T(11)

	TA	TB	TC	TD	TE	TF	
T	0001	0010	0000	0100	0101	0110	before (120456)
T	1001	0010	0011	0100	0101	0110	after (923456)

TC(2) <--||--> T(11)

SHIFT/ROTATE T

SYNTAX:



SEMANTICS:

This instruction SHIFTs or ROTATEs the contents of the T register and places the result either in T or in some other source and sink (destination) register. If the result is not placed in the T register, T remains unchanged. SHIFT will zero fill; ROTATE will put the most-significant bit in the least-significant bit position and shift the remaining bits.

The literal has a decimal range from 0 to 24.

SHIFT/ROTATE T LEFT: If 0 or CPL is used, a shift or rotation by the value of the CPL register will occur. If CPL is greater than 24, 24 is used.

TO register: places the shifted or rotated results in the specified source and sink register; the T register remains unchanged. If the TO register option is not used, the result is placed in the T register. The register may any source and sink register except DATA or MBR (refer to: Registers And Scratchpad). If the register is M, the result of the SHIFT/ROTATE operation is BIT-ORed into the M register and modifies the next microinstruction.

SHIFT/ROTATE T
cont

SEMANTICS cont:

ROTATE T RIGHT: Because the hardware can only rotate the T register to the left, the compiler converts this instruction to the proper left rotate to accomplish the same result as the rotate right.

SHIFT T RIGHT: Because the hardware can only shift the T register left, the compiler will generate an EXTRACT to accomplish the same result. Therefore, the T register may be shifted right only to the X, Y, T or L register. If the TO... option is not used, the result is placed in the T register; otherwise, the T register remains unchanged.

PROGRAMMING NOTE

It is recommended that the EXTRACT instruction itself be used, rather than SHIFT T RIGHT.

EXAMPLE:

ROTATE T LEFT BY 4 BITS

	TA	TB	TC	TD	TE	TF	
T	0110	0011	1000	0101	1111	0000	before (6385F0)
T	0011	1000	0101	1111	0000	0110	after (385F06)

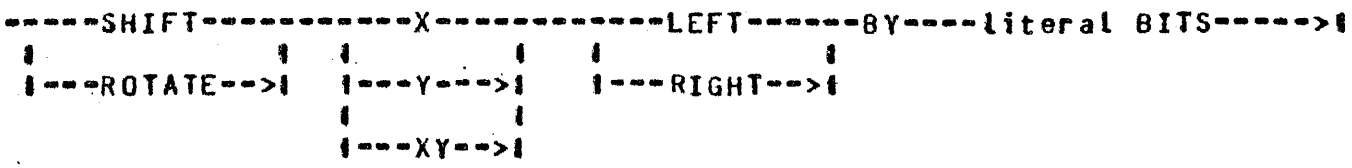
SHIFT T LEFT BY 4 BITS

	TA	TB	TC	TD	TE	TF	
T	0110	0011	1000	0101	1111	0000	before (6385F0)
T	0011	1000	0101	1111	0000	0000	after (385F00)

SHIFT/ROTATE X/Y/XY

SHIFT/ROTATE X/Y/XY

SYNTAX:



SEMANTICS:

This instruction shifts or rotates the X, Y, or XY register (X concatenate Y) a specified number of bits to the right or left. Zero fill will occur with the SHIFT instruction.

The literal has a decimal range from 0 to 23 for the X and Y register; and from 0 to 47 for the XY register.

NOTE: The literal has a maximum value of 1 on the B1710 systems with the concatenated XY register.

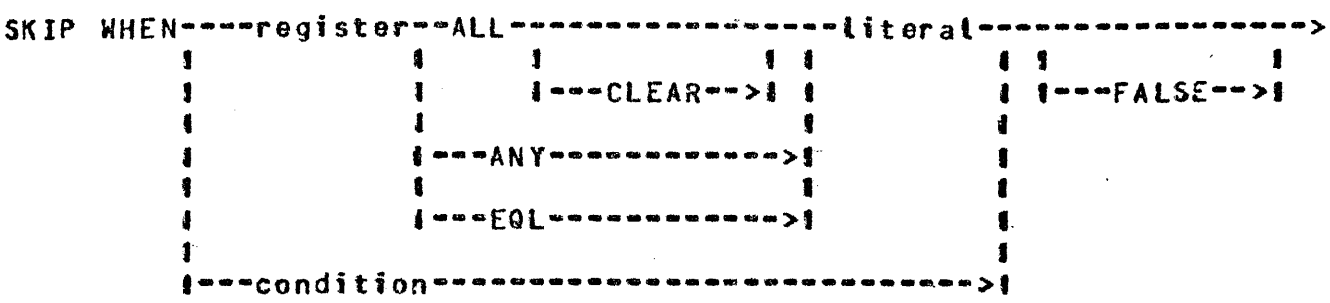
EXAMPLE:

```
SHIFT X LEFT BY 5 BITS
ROTATE XY RIGHT BY 40 BITS
```

SKIP

SKIP

SYNTAX:



SEMANTICS:

This instruction causes one microinstruction to be skipped if the designated condition is satisfied. (See also: IF.)

SKIP WHEN register: The literal contains a 4-bit mask and may be comprised of decimal, binary, or heaxadecimal entries.

ALL is considered to be true only if all the bits in the register corresponding to one bits in the mask are true. That is, only the designated bit positions are tested to see if they contain ones. ANY is true if at least one bit in the register corresponding to a one bit in mask is true. EQL is true if all the register bits equal the corresponding bits in the mask. That is, the register must be exactly like the mask.

ALL CLEAR causes the masked bits of the register to be set to zeros after testing the ALL condition. Only the bits tested are cleared, and the clearing action always occurs whether the SKIP is taken or not. If ALL is used with a mask of 0000, the result is always false.

SKIP
cont

SEMANTICS cont:

FALSE causes a skip when the whole condition is false.

SKIP WHEN condition: The condition may be any condition available from the condition registers. (See: IF.)

The register may be declared as follows:

FU	TA	LA	CA	BICN
FT	TB	LB	CB	FLCN
FLC	TC	LC	CC	INCN
FLD	TD	LD	CD	XYCN
FLE	TE	LE		XYST
FLF	TF	LF		

PROGRAMMING NOTE

The use of the IF...THEN...ELSE instruction is recommended rather than the SKIP instruction. The SKIP is limited to one, 4-bit grouping mask in one register and may only skip one microinstruction. The IF is capable of testing any combination of bits in many registers or skipping blocks of microinstructions and will generate a SKIP WHEN hardware microinstruction whenever possible.

S.MEMORY.LOAD

S.MEMORY.LOAD

SYNTAX:

SEMANTICS:

This instruction specifies the location for beginning statements in S.MEMORY. Code is not generated, but the code address of the last statement is placed in the IPB (Interpreter Parameter Block) at RESERVED.M.MEMORY.

STORE

SYNTAX:

STORE F INTO------double.scratchpad.word----->1

SEMANTICS:

This instruction MOVEs the Field (F) register into any double scratchpad word (S0...S15); the F register remains unchanged.

NOTE: The compiler generates two MOVE instructions on B1710 systems.

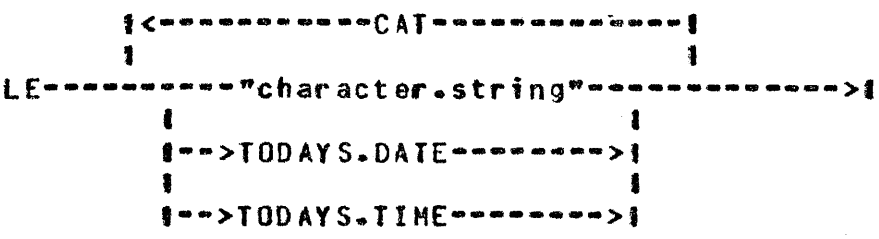
EXAMPLE:

STORE F INTO S6

SUB.TITLE

SUB.TITLE

SYNTAX:



SEMANTICS:

This instruction modifies program title information.

If "character.string" exceeds 72 characters, right-hand truncation will occur.

$ HEADINGS and either $ LINES.PER.PAGE [#] or $ PAGE.NUMBERS must be specified if subtitles are required on following pages.

EXAMPLE:

```
SUB.TITLE TODAYS.DATE CAT "PROB.A" CAT TODAYS.TIME
```

SUBTRACT SCRATCHPAD

SUBTRACT SCRATCHPAD

SYNTAX:

SUBTRACT-----scratchpad.word-----FROM FA--------->

SEMANTICS:

This instruction subtracts the left half of any scratchpad word (S0A...S15A) from the Field Address (FA) register. The result is placed in FA; the contents of scratchpad.word remain unchanged. (See also: ADD SCRATCHPAD.)

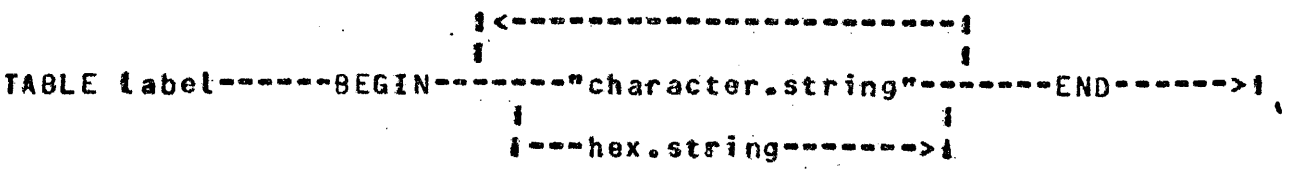
EXAMPLE:

SUBTRACT S3A FROM FA

TABLE

TABLE

SYNTAX:



SEMANTICS:

This instruction creates in-line character-strings.

Only one string is allowed per line. The charcter.string must be enclosed within quotation marks; the hex.string must be enclosed within @ signs.

The BEGIN/END pair must surround all strings in the TABLE. The characters are grouped two per address, i.e., 16 bits.

The label must be unique; its use references the first 16 bits of the table.

EXAMPLE:

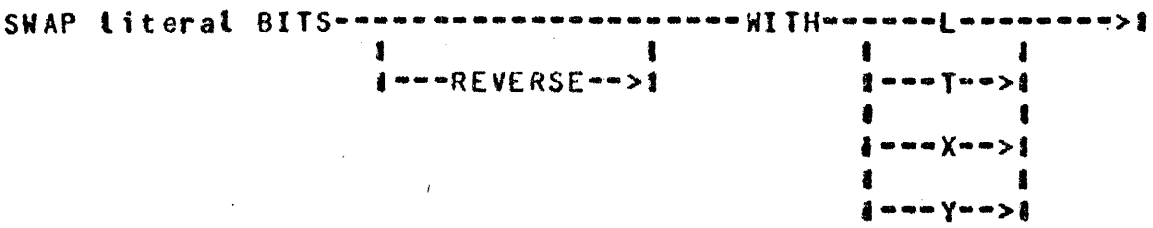
```
TABLE REF                          Code generated:
BEGIN
     "AB"                               C1C2
     @ABC@                              C1C2
     "D"                                C3C4
     "45"                               F4F5
END
MOVE ADDRESS (REF) TO Y            % The address of the table (REF) will
                                   % be loaded into the Y register
```

SWAP

SWAP

(Available on 81720 systems only)

SYNTAX:



SEMANTICS:

This instruction swaps the specified number of bits between main memory and the specified register.

The FA (Field Address) register must have been previously set to the proper main memory address.

The literal has a decimal range from 0 to 24. If the value of the literal is zero, the contents of the CPL register are used. If the CPL register is also 0, the register is cleared to all zeros. If less than 24 bits are swapped, the leading bits of the register are zero.

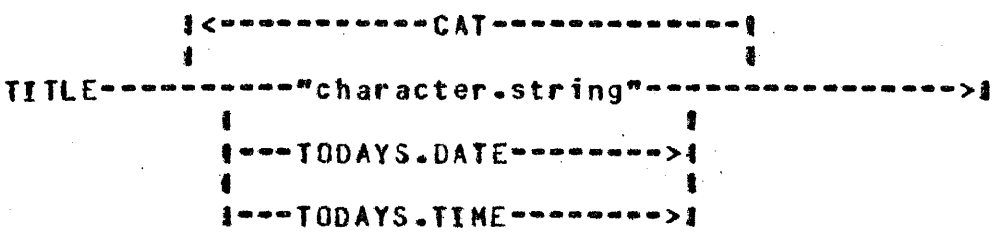
Normally the contents of the FA register point to the first bit of the field to be swapped. If the REVERSE option is used, the contents of FA point to the last bit + 1 of the main memory field involved. The specified register (L, T, X or Y) receives the contents of this field as if it had been read in a forward direction.

PROGRAMMING NOTE

Incrementing or decrementing of the FA or FL registers is not allowed with the SWAP instruction.

TITLE

SYNTAX:



SEMANTICS:

This instruction modifies program title information.

If "character.string" exceeds 72 characters, right-hand truncation will occur.

$ HEADINGS and either $ LINES.PER.PAGE [#] or $ PAGE.NUMBERS must be specified if titles are required on following pages.

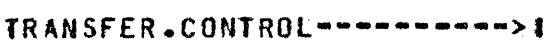
EXAMPLE:

```
TITLE TODAY.DATE CAT "PATCHES"
```

TRANSFER.CONTROL

TRANSFER.CONTROL

SYNTAX:



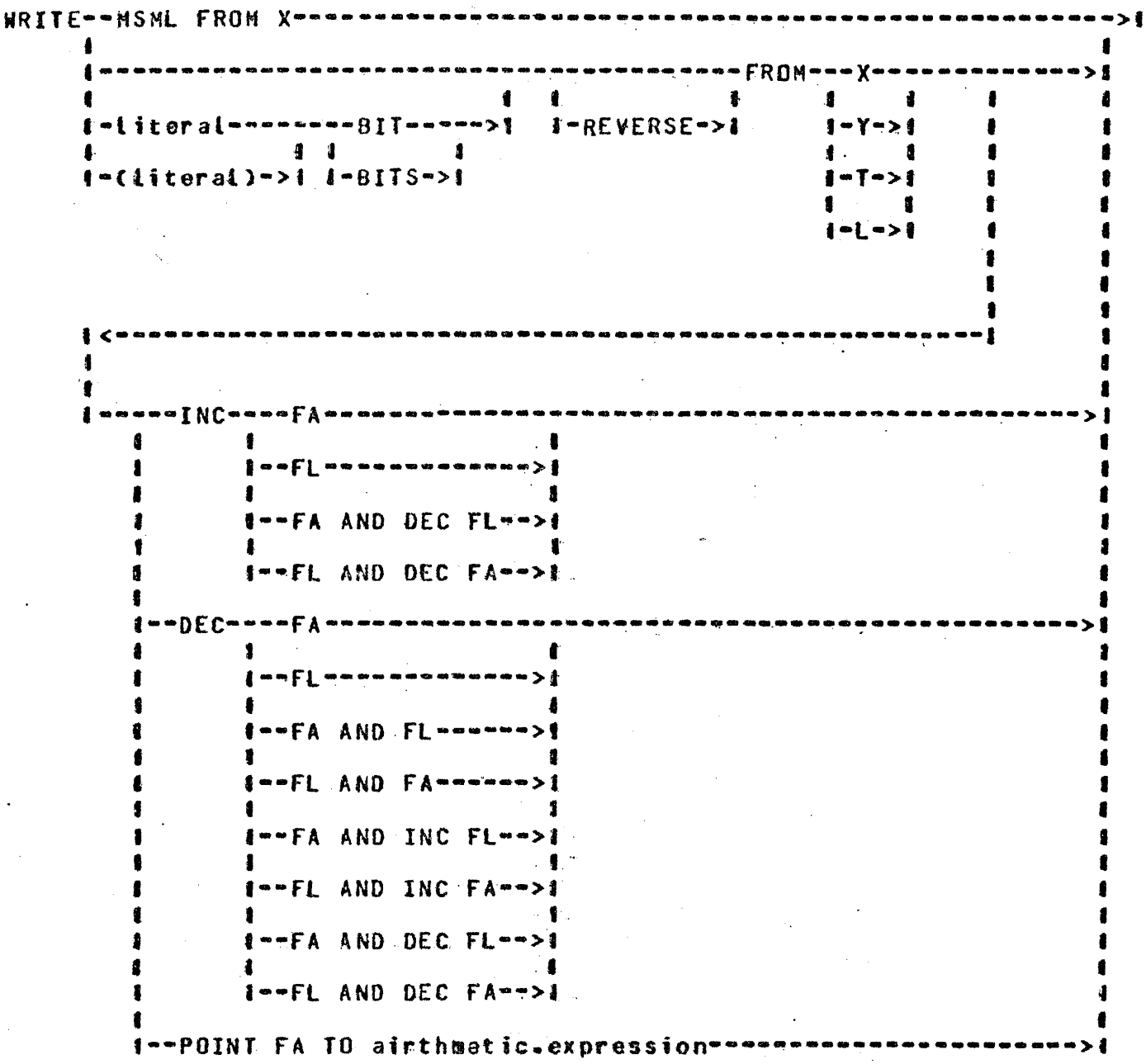
SEMANTICS:

This instruction generates the BIND hardware instruction. (See Appendix B: BIND)

WRITE

WRITE

SYNTAX:



WRITE
cont

SEMANTICS:

An M-Memory WRITE (MSML TO X) instruction writes from the X register the 16 bits in M-Memory pointed to by the contents of the L register. The contents of L must be modulo 16. This facility is not available on S-Memory Processors.

An S-Memory WRITE instruction writes from 0 to 24 bits of information into S-Memory from one of the allowable source registers: X, Y, T, or L.

The amount of data written (field length) is determined by the literal/(literal) BIT(S) option. If this is equal to 0 or is empty, then the field length is given by the contents of CPL is right justified in the selected source register. If the field length is zero then nothing is written.

Normally the contents of the FA register point to the first bit of the field to be written. If the REVERSE option is used, the contents of the FA register point to the last bit + 1 of the field to be written to memory. Memory contains the rightmost contents of the source register as if it had been written in a forward direction.

INC/DEC adjusts FA/FL by the field length after the operation but in the same microinstruction.

POINT FA adjusts FA by up to 144 + field length bits after the operation. (A warning message will be issued if the adjustment is greater than 72 + field length bits). This option can be used only if literal/(literal) BIT(S) is specified and is greater than 0. (See also: FA.POINTS and POINT.)

The unparenthesized literal has a decimal range from 0 to 24. (literal) has a decimal range from 0 to 26: a value of 25 will cause 24 bits to be written with correct parity; a value of 26 will cause 24 bits to be written with incorrect parity.

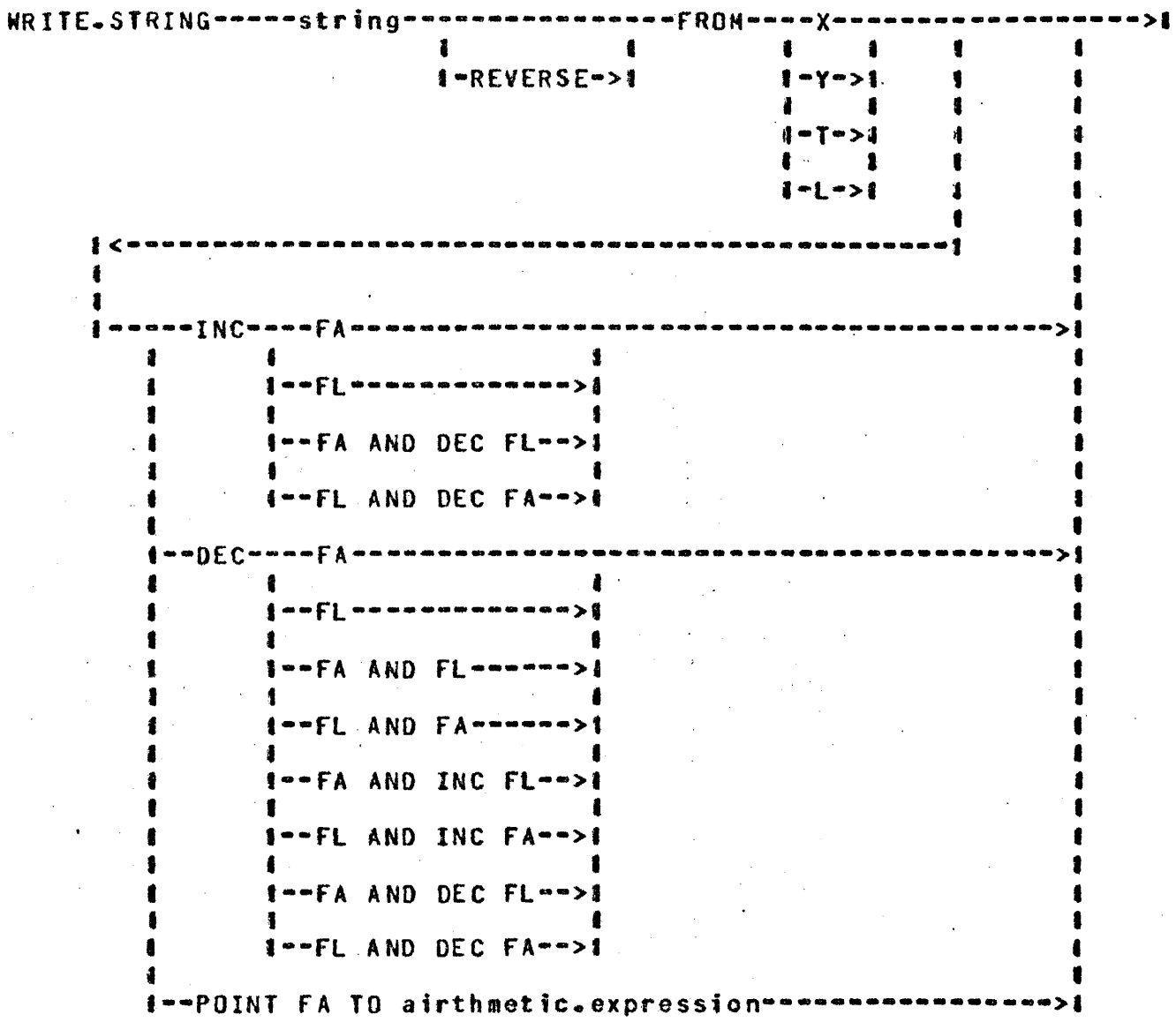
EXAMPLE:

```
WRITE MSML FROM X
WRITE 24 BITS FROM X
WRITE FROM Y INC FA
WRITE 2 BITS REVERSE FROM T DEC FA AND DEC FL
WRITE REVERSE FROM L DEC FL
WRITE 10 BITS FROM T POINT FA TO 25
```

WRITE.STRING

WRITE.STRING

SYNTAX:



WRITE.STRING
cont

WRITE.STRING
cont

SEMANTICS:

This instruction generates the necessary in-line literals for a string, with moves to the indicated register. It also generates the WRITE commands to write the string into main memory, beginning at the address in the FA register.

The length of the string is limited to the remainder of the source card image. It may be any of the following data types.

Type	Start-Stop Symbol	Length of Each Unit	Example
Character	"	8 bits	"APC128JKL"
Hex	@	4 bits	@124ADF@
Octal	@(3)	3 bits	@(3)1235670
Quartal	@(2)	2 bits	@(2)123321@
Unary	@(1)	1 bit	@(1)110011010

EXAMPLE:

```
WRITE.STRING "APC" REVERSE FROM X
WRITE.STRING POINT FA TO 64
```

XCH

XCH

SYNTAX:

XCH----double.scratchpad.word.1----F----double.scratchpad.word.2---->1

SEMANTICS:

This instruction moves the Field (F) register to double scratchpad word.2 (S0...S15); double scratchpad word.1 (S0...S15) is then moved to the F register. The two words may be the same.

EXAMPLE:

```
XCH S0 F S0        %  equivalent to:          MOVE FA TO S0A
                   %                          MOVE FL TO S0B
                   %
                   %  and simultaneously:     MOVE S0A TO FA
                   %                          MOVE S0B TO FL
```

PROGRAMMING TECHNIQUES

VIRTUAL-LANGUAGE DEFINITIONS

A set of virtual-instructions for the virtual machine must first be defined as each being a unique string of bits. This definition may be chosen according to any relevant criteria. For example, COBOL verbs may be encoded according to their frequency of usage, the higher frequency verbs being encoded in three bits with one escape code that specifies the next eight bits as an extended code string. Another approach might be to accept directly the source language as in a time-sharing, "line-at-a-time," interactive mode. After the S-instructions and their operand fields have been defined, any standard location or technique should be selected. For example, the base values of S-instructions and S-data might be in S4A and S5A of the scratchpad; or all routines are to be referenced with CALL and end with an EXIT instruction to facilitate subrouting. The microprogrammer is now ready to begin creating the microroutines needed to perform each of the events in the S-language.

ASSEMBLY CODING FORM

The compiler accepts card images consisting of one symbolic microinstruction per card. The source program must reflect the following format:

Column	Usage
1-5	Reserved for label declarations which, if used, must begin somewhere within this field.
1-72	A percent sign (%) anywhere within this field indicates that everything to the right is a comment.
6-72	Microinstructions may appear anywhere within this field. At least one blank must be used between words except in those cases where a special character (e.g., a parenthesis or relational operator) is required, in which case blanks are optional.

Both point labels and uniques labels are allowed, with a limit of 25 characters and no imbedded blanks. A blank is the separator between the label and the beginning of the microinstruction.

73-80 Reserved for sequence numbers.

Source code maintenance as well as other compiler options may be specified by the use of either a $ (dollar sign) or & (ampersand) in column 1. (See Appendix A: MIL Compiler Operation.)

PROGRAM EXAMPLES

Examples A, B and C first explain S-language statements; there is assumed to exist some basic driver routine which is in control at the beginning and end of each S-instruction. This control routine performs the Virtual Machine functions of maintaining an Instruction register and fetching the next S-instruction.

EXAMPLE A:

Assume the following:

1. The 3 bits 010 imply an S-instruction of ADD 6 decimal digits, Indirect address-1, Indirect address-2 and store the answer in Indirect address-2.

2. Indirect addresses are displacements from the beginning of a table; the actual base value of the table is the current setting of the Base Register (BR).

3. The lengths of the Indirect addresses are 9 bits.

4. All data is in 4-bit decimal form and is 6 decimal digits (24 bits) long.

5. Overflow is to be ignored.

The instruction might appear as follows:

010 000011001000010110 0 in main memory.

This bit string represents an S-instruction compiled from a source language statement such as the following:

```
'ADD SUM TO ROLLTOTAL
```

That portion of an interpreter which would perform the addition might appear as follows:

```
NEXTSOP  READ 3 BITS TO X INC FA      % GET OP-CODE
         MOVE X TO M                  % PREPARE TO DECODE
         JUMP FORWARD                 % GO TO DECODER
         GO TO ROUTINE.FETCH
         GO TO ROUTINE.STORE
         GO TO ROUTINE.ADD            % ADD ROUTINE
                  .
                  .
                  .
ROUTINE.ADD                           % LABEL FIRST LINE
          MOVE BR TO S1A              % SET BASE FOR ADD
          READ 9 BITS TO T INC FA     % READ FIRST INDEX
          READ 9 BITS TO L INC FA     % READ SECOND INDEX
          MOVE L TO FA                % LOAD INDEX
          ADD S1A TO FA               % ADD ACTUAL BASE TO INDEX
          READ 24 BITS TO X           % GET DATA-2
          MOVE T TO FA                % LOAD INDEX
          ADD S1A TO FA               % ADD ACTUAL BASE TO INDEX
          READ 24 BITS TO Y           % GET DATA-1
          MOVE 2(1)001110002 TO CP    % CLEARS CARRY
                                      % SETS CPU AND CPL CORRECTLY
          MOVE SUM TO T               % GET SUM READY TO WRITE
          WRITE 24 BITS FROM T        % WRITE SUM
          MOVE 24 TO CP               % MUST RESTORE CPU IN GENERAL
          GO TO NEXTSOP               % GO TO NEXT S-INSTRUCTION
```

EXAMPLE B:

If the source language statement was

```
MOVE INVERTING FIELD 1 TO FIELD 2
```

The S-language might be:

OP-CODE BIT STRING	DATA TYPE	ADDRESS-1 LEFT-MOST UNIT ABSOUTE ADDRESS IN BITS	LENGTH-1 IN BITS	ADDRESS-2 RIGHT-MOST UNIT ADDRESS IN BITS	LENGTH-2 IN BITS
	0001=BINARY 0100=4-BIT DECIMAL(BCD) 1000=8-BIT DECIMAL(EBCDIC)				

NOTES:

1. The size of each element in the S-language is 24 bits or less.

2. The left-most-address-1 points to the beginning of field-1 and the data will be accessed with READ FORWARD commands. The right-most-address-2 points to the end of field-2 and the data will be accessed with READ REVERSE commands.

Assume the following events have been performed in a manner similar to that used in Example A:

1. The Op-code has been properly decoded and the correct routine has been entered.

2. The address and length of Field 1 are in the F register.

3. The address and length of Field 2 are in scratchpad word S0.

The following code then performs the INVERTED-MOVE operation and properly pads if the receiving field (field 2) is longer than the sending field (field 1).

```
INMOV    BIAS BY UNIT                      % SET CPU AND CPL
         READ 4 BITS REVERSE TO L          % GET TYPE INDICATOR
  TOP    IF FL = 0 THEN GO TO PAD          % TEST LIMIT FIELD1
         IF SFL = 0 THEN GO TO ENDOP       % END OF FIELD2 STOP
         READ TO X INC FA AND DEC FL       % GET A UNIT OF DATA
         XCH S0 F S0                       % EXCHANGE S0 AND F REG
         WRITE REVERSE FROM X DEC FA AND DEC FL
                                           % PUT A UNIT OF DATA
         XCH S0 F S0                       % EXCHANGE FOR GET
         GO TO TOP
  PAD    LOAD F FROM S0                    % GET ADDRESS INTO F REG
         MOVE 0 TO X                       % SET ZERO FOR PAD
         IF LF(4) THEN                     % TEST FOR EBCDIC
            MOVE @40@ TO X                 % ADD PAD SPACES
  A.     WRITE FROM X DEC FA DEC FL        % WRITE A SPACE
         IF FL NEQ 0 GO TO -A              % TEST LIMIT
         GO TO ENDOP                       % END OF OPERATION
```

The resultant data movement in memory would be:

Alpha Data String

	FIELD1	FIELD2
Before	A B C D E	4 5 6 7 8
After	A B C D E	E D C B A

Bit Strings

	FIELD1	FIELD2
Before	1 1 0 0 0	1 1 1 1 1
After	1 1 0 0 0	0 0 0 1 1

Notice that the same microinstruction sequence will work in either case.

EXAMPLE C

Given: A list of data items.

Problem: Sort the data items into ascending sequence in place. (Do a bubble sort.)

Assume:

1. The S-operator has the following general format:

S-op	Type Indicator	Left-most Address In Bits	Length Of List In Bits
A BIT STRING	4 = 4-bit decimal 8 = 8-bit alpha (EBCDIC) any other value from 0 to 15	A BIT STRING	A BIT STRING

2. The S-op has been decoded (see Example A) and the necessary routine has been entered.

3. Scratchpad word S5 contains the most-significant (left-most) address in S5A and the TYPE and length in S5B.

Then the following routine will perfrom the bubble sort:

```
BSORT   CLEAR L                          % CLEAR SWITCH
CYCLE   LOAD F FROM S5                   % FETCH BEGINNING ADDRESS
        IF FL = 0 THEN GO TO ENDOP       % DEGENERATE CASE TEST
        BIAS BY UNIT                     % SET CPL TO UNIT FOR RD/WT
.A      COUNT FA UP AND FL DOWN BY COP   % PLACE BETWEEN ITEMS
        IF FL = 0 GO TO ENDOP            % LAST ITEM TEST
        READ REVERSE TO X                % GET ITEM ON LEFT
        READ TO Y                        % GET ITEM ON RIGHT
        IF X GEQ Y THEN GO TO -A         % LEAVE ALONE
        WRITE REVERSE FROM Y             % REPLACE RIGHT TO LEFT
        WRITE FROM X                     % REPLACE LEFT TO RIGHT
        MOVE @F@ TO LF                   % MARK NOT ALL SORTED SWITCH
        GO TO -A                         % GO GET NEXT
ENDOP   IF LF(0) TRUE GO TO EXITR        % EXIT ROUTINE
        CLEAR L                          % RESET SWITCH
        GO TO CYCLE                      % TRY WHOLE LAST AGAIN
EXITR   EXIT
```

```
NL BLOCK        CODE   MEMORY                                                                               SEQUENCE      PATCH
   NAME                ADDRESS                                                                                            INFO

0                                  :%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%: 00000200
0                                  :%                    EXAMPLE  OF A MIL PROGRAM                         %: 00000300
0                                  :%     THIS PROGRAM DEMONSTRATES SOME DESIRABLE CHARACTERISTICS OF A    %: 00000400
0                                  :%  MIL PROGRAM.  THE PROGRAM "INTERPRETS" ONE BIT OP-CODE INSTRUCTIONS %: 00000500
0                                  :%  FROM A FIELD IN MEMORY AND PERFORMS THE INDICATED OPERATION.  SINCE %: 00000600
0                                  :%  THERE ARE ONLY TWO VALUES OF A ONE BIT OP-CODE, THERE ARE ONLY TWO  %: 00000700
0                                  :%  INSTRUCTIONS TO DECODE.  (IN THIS CASE "ADD" AND "END.OF.JOB" ARE   %: 00000800
0                                  :%  THE ONLY INSTRUCTIONS.)                                             %: 00000900
0                                  :%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%: 00001000
0                                  :%                                                                       : 00001100
0                                  :%                                                                       : 00001200
0                                  :%                                                                       : 00001300
0                                  :       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%             : 00001400
0                                  :       %%%%%%%%%%          GLOBAL DECLARES          %%%%%%%%%%             : 00001500
0                                  :       %%%%%%%%%%                                   %%%%%%%%%%             : 00001600
0                                  :       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%             : 00001700
0                                  :         DECLARE                                                        : 00001800
0                     [000000]     :                 01 SYSTEM.INFORMATION           BIT(1668),             : 00001900
0                     [000000]     :                    02  DEVICE.STATUS.TABLE      BIT(256),              : 00002000
0                     [000100]     :                    02  AVAILABLE.SYSTEM.MEMORY  BIT(8),                : 00002100
0                     [000108]     :                    02  AVAILABLE.CONTROL.MEMORY BIT(4),                : 00002200
0                     [00010C]     :                    02  OPTIONS.TABLE            BIT(48),               : 00002300
0                     [00010C]     :                        03 SWITCHES              BIT(8),                : 00002400
0                                  :                        03 FILLER                BIT(40),               : 00002500
0                                  :                    02  FILLER                   BIT(1352);             : 00002600
0                                  :                                                                        : 00002700
0                                  :                                                                        : 00002800
0                                  :       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%             : 00002900
0                                  :       %%%%%%%%%%          GLOBAL DEFINES           %%%%%%%%%%             : 00003000
0                                  :       %%%%%%%%%%                                   %%%%%%%%%%             : 00003100
0                                  :       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%             : 00003200
0                                  :         DEFINE        ADDR1                    = S0#                   : 00003300
0                                  :         DEFINE        ADDR2                    = S1#                   : 00003400
0                                  :         DEFINE        ADDR3                    = S2#                   : 00003500
0                                  :         DEFINE        NEXT.INSTRUCTION.POINTER = S4A#                  : 00003600
0                                  :                                                                        : 00003700
0                                  :                                                                        : 00003800
0                                  :                                                                        : 00003900
0                                  :                                                                        : 00004000
0                                  :%%%%%%%%%%%%%%%%%%%%%                                                   : 00004100
0                                  :BEGINNING.OF.PROGRAM%                                                   : 00004200
0                                  :%%%%%%%%%%%%%%%%%%%%%                                                   : 00004300
0                                  :%                                                                       : 00004400
0                                  :%  THIS PART OF THE INTERPRETER ARBITRARILY SETS THE STARTING ADDRESS OF: 00004500
0                                  :%  THE NEXT INSTRUCTION ALONG WITH THE PROGRAM BASE AND LIMIT REGISTERS.: 00004600
0                                  :%  (THIS INFORMATION WOULD NORMALLY BE FOUND BY SOME OTHER METHOD.)     : 00004700
0                                  :%                                                                       : 00004800
0                                  :%                                                                       : 00004900
0               9800 @ [00000]     :         MOVE @3000@ TO NEXT.INSTRUCTION.POINTER                        : 00005000
0               0300 @ [00010]     :                                                                        :             G
0               2884 @ [00020]     :                                                                        :             G
0               9600 @ [00030]     :         MOVE @6000@ TO BR                                              : 00005100
0               0600 @ [00040]     :                                                                        :             G
0               9700 @ [00050]     :         MOVE @18000@ TO LR                                             : 00005200
0               1800 @ [00060]     :                                                                        :             G
0                                  :%                                                                       : 00005300
```

```
0                                   :FETCH%       BINARY; READS IN[illegible]T OF CODE, AND PERFORMS A      : 00005500
0                                   :%%%%%%       CALCULATED BRANC[illegible]O THE INDICATED EXECUTION ROUTINE.  : 00005600
0                                   :%                                                                       : 00005700
1 FETCH                             :          BEGIN FETCH                                                   : 00005800
1 FETCH      8C18 2 [00070]         :               MOVE 24 TO CP                                            : 00005900
1 FETCH      28A4 2 [00080]         :               MOVE NEXT.INSTRUCTION.POINTER TO FA                      : 00006000
1 FETCH      7101 2 [00090]         :               READ 1 BIT TO X INC FA              %GET OP CODE.        : 00006100
1 FETCH      2884 2 [000A0]         :               MOVE FA TO NEXT.INSTRUCTION.POINTER                      : 00006200
1 FETCH      10A5 2 [000B0]         :               MOVE X TO M                                              : 00006300
1 FETCH      C000 2 [000C0]         :               JUMP FORWARD                                             : 00006400
1 FETCH      C001 2 [000D0]         :                   GO TO ADD                       %OP _ 0.             : 00006500
1 FETCH      C015 2 [000E0]         :                   GO TO END.OF.JOB                %OP _ 1.             : 00006600
1 FETCH                             :          END FETCH                                                     : 00006700
0                                   :%                                                                       : 00006800
0                                   :%%%%         THIS EXECUTION ROUTINE PERFORMS AN ABSOLUTE, 3 ADDRESS,    : 00006900
0                                   :ADD%         BINARY OR DECIMAL ADD. OVERLAPPING FIELDS ARE PERMITTED.   : 00007000
0                                   :%%%%         LEFT TRUNCATION OR LEFT ZERO FILL WILL OCCUR IN THE        : 00007100
0                                   :%            RESULT FIELD IF IT IS RESPECTIVELY LONGER OR SHORTER       : 00007200
0                                   :%            THAN IS REQUIRED TO CONTAIN THE RESULTANT SUM. ALL 3       : 00007300
0                                   :%            FIELDS MUST BE OF THE SAME TYPE(BINARY, 4-BIT DECIMAL      : 00007400
0                                   :%            OR 8-BIT DECIMAL) OR THE RESULT IS NOT DEFINED.            : 00007500
0                                   :%                                                                       : 00007600
1 ADD                               :          BEGIN ADD                                                     : 00007700
1 ADD        3182 2 [000F0]         :               MOVE 2 TO FT            %INDICATES A 3 ADDRESS INSTRUCTION: 00007800
1 ADD        8890 2 [00100]         :               MOVE 144 TO FA          %LENGTH OF 3 ADDRESSES       : 00007900
1 ADD        E013 2 [00110]         :               CALL LOAD.ADDRESSES                                      : 00008000
1 ADD        0050 2 [00120]         :               LOAD F FROM ADDR1                                        : 00008100
1 ADD                               :.LOOP                                                                   : 00008200
1 ADD        E01F 2 [00130]         :               CALL GET.DATA           %GET ADDEND.                     : 00008300
1 ADD        11A0 2 [00140]         :               MOVE Y TO X                                              : 00008400
1 ADD        0701 2 [00150]         :               XCH ADDR2 F ADDR1                                        : 00008500
1 ADD        E01C 2 [00160]         :               CALL GET.DATA           %GET AUGEND.                     : 00008600
1 ADD        0712 2 [00170]         :               XCH ADDR3 F ADDR2                                        : 00008700
1 ADD        0033 2 [00180]         :               BIAS BY F TEST                                           : 00008800
1 ADD        D013 2 [00190]         :                   GO TO FETCH    %END OF RESULT FIELD AND INSTRUCTION. : 00008900
1 ADD        10E2 2 [001A0]         :               MOVE SUM TO T                                            : 00009000
1 ADD        0064 2 [001B0]         :               CARRY SUM                                                : 00009100
1 ADD        60A8 2 [001C0]         :               IF FU EQL 8 THEN    %SUPPLY EBCDIC NUMERIC ZONES IF 8-BIT : 00009200
1 ADD        C003 2 [001D0]         :                                                                        :  I
2 ADD                               :               BEGIN                                                    : 00009300
2 ADD        300F 2 [001E0]         :                   MOVE 3FA TO TA                                       : 00009400
2 ADD        1002 2 [001F0]         :                   MOVE TA TO TC                                        : 00009500
2 ADD        1004 2 [00200]         :                   MOVE TA TO TE                                        : 00009600
2 ADD                               :               END                                                      : 00009700
1 ADD        7FA0 2 [00210]         :               WRITE REVERSE FROM T DEC FA AND FL   %WRITE SUM.         : 00009800
1 ADD        0720 2 [00220]         :               XCH ADDR1 F ADDR3                                        : 00009900
1 ADD        D011 2 [00230]         :               GO TO -LOOP                                              : 00010000
1 ADD                               :          END ADD                                                       : 00010100
0                                   :%                                                                       : 00010200
0                                   :%%%%%%%%%%%                                                            : 00010300
0                                   :END.OF.JOB%                                                            : 00010400
0                                   :%%%%%%%%%%%                                                            : 00010500
0                                   :%                                                                       : 00010600
1 END.OF.JOB                        :          BEGIN END.OF.JOB                                              : 00010700
1 END.OF.JOB 0001 2 [00240]         :               HALT                                                     : 00010800
1 END.OF.JOB                        :          END END.OF.JOB                                                : 00010900
0                                   :%                                                                       : 00011000
0                                   :%%%%%%%%%%%%%%%    THIS GENERAL ROUTINE LOADS 1, 2 OR 3 ADDRESSES TO    : 00011100
0                                   :LOAD.ADDRESSES%    SCRATCHPAD AND UPDATES THE NEXT.INSTRUCTION.POINTER. : 00011200
0                                   :%%%%%%%%%%%%%%%    ALL ADDRESSES ARE IN THE FOLLOWING FORMAT;           : 00011300
0                                   :%                        A) A 24 BIT, BASE RELATIVE, S-MEMORY ADDRESS   : 00011400
0                                   :%                           WHICH IS CONVERTED TO AN ABSOULUTE ADDRESS   : 00011500
0                                   :%                           POINTING AT THE END OF A FIELD   1.          : 00011600
```

```
0                                   :%                  B) A 4 BIT TYPE FIELD(0001 FOR BINARY, 0100      : 00011700
0                                   :%                     FOR 4-BIT DECIMAL AND 1000 FOR 8-BIT          : 00011800
0                                   :%                     DECIMAL);                                     : 00011900
0                                   :%                                                                   : 00012000
0                                   :%                  C) 4 UNUSED BITS                                 : 00012100
0                                   :%                                                                   : 00012200
0                                   :%                  D) A 16 BIT FIELD LENGTH INDICATING THE NUMBER   : 00012300
0                                   :%                     OF BITS CONTAINED IN EACH FIELD.              : 00012400
0                                   :%                                                                   : 00012500
1 LOAD.ADDRE                        :         BEGIN LOAD.ADDRESSES                                       : 00012600
1 LOAD.ADDRE 0804 2 [00250]         :              ADD NEXT.INSTRUCTION.POINTER TO FA  %---> END OF ADDRESSES: 00012700
1 LOAD.ADDRE 2884 2 [00260]         :              MOVE FA TO NEXT.INSTRUCTION.POINTER                   : 00012800
1 LOAD.ADDRE                        :.LOOP                                                               : 00012900
1 LOAD.ADDRE 16A1 2 [00270]         :              MOVE BR TO Y                                          : 00013000
1 LOAD.ADDRE 7588 2 [00280]         :              READ 24 BITS REVERSE TO T DEC FA    %GET TYPE AND LENGTH. : 00013100
1 LOAD.ADDRE 7538 2 [00290]         :              READ 24 BITS REVERSE TO X DEC FA    %GET ADDRESS.         : 00013200
1 LOAD.ADDRE 10E0 2 [002A0]         :              MOVE SUM TO X                       %ADDRESS   BASE.      : 00013300
1 LOAD.ADDRE 8C30 2 [002B0]         :              EXTRACT 16 BITS FROM T(8) TO Y      %GET LENGTH           : 00013400
1 LOAD.ADDRE 1165 2 [002C0]         :              MOVE FT TO M                        %SELECT A SCRATCHPAD  : 00013500
1 LOAD.ADDRE 20C0 2 [002D0]         :              MOVE SUM TO SOA                     %ADDRESS   LENGTH     : 00013600
1 LOAD.ADDRE 1165 2 [002E0]         :              MOVE FT TO M                                          : 00013700
1 LOAD.ADDRE 2290 2 [002F0]         :              MOVE T TO SOB       %TYPE AND LENGTH.                 : 00013800
1 LOAD.ADDRE 31F1 2 [00300]         :              DEC FT BY 1 TEST                                      : 00013900
1 LOAD.ADDRE D008 2 [00310]         :                 GO TO -LOOP                                        : 00014000
1 LOAD.ADDRE 18A4 2 [00320]         :              EXIT                                                  : 00014100
1 LOAD.ADDRE                        :         END LOAD.ADDRESSES                                         : 00014200
0                                   :%                                                                   : 00014300
0                                   :%%%%%%%%%      THIS GENERAL ROUTINE WILL READ FROM ZERO TO 24 BITS INTO : 00014400
0                                   :GET.DATA%      THE M-MACHINE, SET CPU TO THE ENCODED DATA TYPE AND SET  : 00014500
0                                   :%%%%%%%%%      CPL TO THE NUMBER OF BITS READ.                      : 00014600
0                                   :%                                                                   : 00014700
1 GET.DATA                          :         BEGIN GET.DATA                                             : 00014800
1 GET.DATA   0032 2 [00330]         :              BIAS BY F                                             : 00014900
1 GET.DATA   7760 2 [00340]         :              READ REVERSE TO Y DEC FA AND FL                       : 00015000
1 GET.DATA   18A4 2 [00350]         :              EXIT                                                  : 00015100
1 GET.DATA                          :         END GET.DATA                                               : 00015200
0                                   :         FINI                                                       : 00015300
0            000F 2 [00360]         :                                                                    :          G
0            0000 2 [00370]         :                                                                    :          G
0            0000 2 [00380]         :                                                                    :          G
0            0000 2 [00390]         :                                                                    :          G

NO ERRORS DETECTED

MICRO INSTRUCTION COUNT = 00058

CAUTION: $ SUBSET WAS NOT SPECIFIED; THEREFORE, THIS
         PROGRAM SHOULD NOT BE USED ON A B1712/B1714.
```

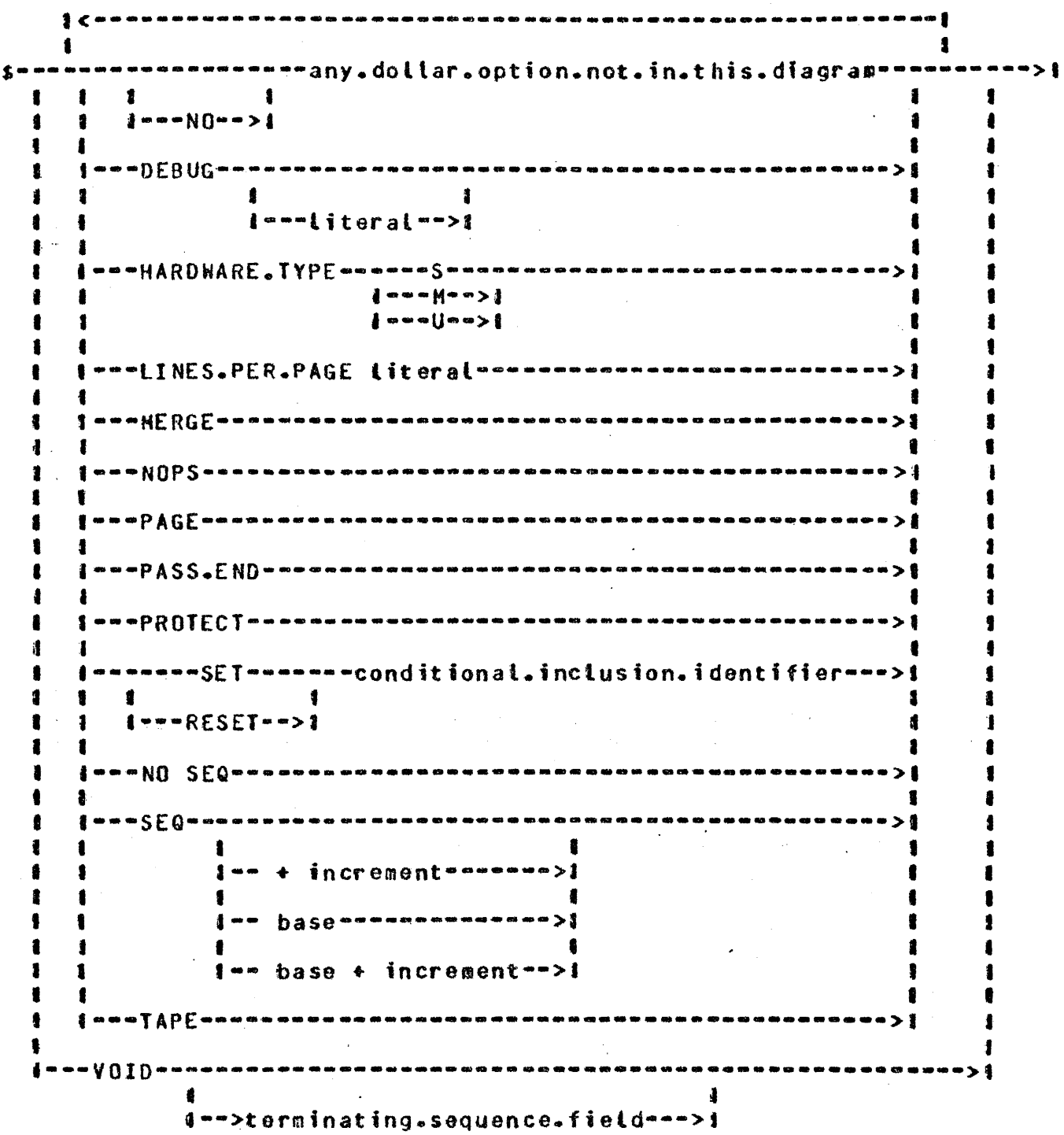
APPENDIX A: MIL COMPILER OPERATION

CONTROL CARDS

The purpose of the compiler control card is to allow the programmer to specify options to the compiler I/O files.

Every MIL control card has either a $ (dollar sign) in column 1 and is called a "dollar card", or has an & (ampersand) in column 1 and is called an "ampersand card". Columns 73-80 may be used as a sequence field.

DOLLAR CARD SYNTAX:



DOLLAR CARD SEMANTICS:

ALLCODE	lists all code generated by each MIL statement (default on)
AMPERSAND	lists all ampersand cards (default on)
ANALYZE.CODEFILE	prints an analysis of the code file at end of source listing
B710	generates code for B1710 (S-Memory) Processors; same as $SUBSET, $HARDWARE TYPE=S
CHECK	checks for sequence errors (default on)
COMPILE	when reset a fast source listing will be produced with no code generation or syntax checking (default on)
CONTROL	prints all dollar cards on listing; same as $DOLLAR
DEBUG	for compiler debugging use
DECK	punches an object deck
DOLLAR	prints all dollars cards on listing
DOUBLE	double spaces listing when printing
ERRORFILE	lists errors and warning separately
EXPAND	prints all statements (including comments) in a macro invocation
EXTERNAL	generates external segment branching code
FORCE	generates a code file regardless of syntax errors
FRAME	lists all IF, BEGIN...END statements which conditionally exclude code (default on)
HARDWARE.TYPE	specifies which hardware processor type will be used: S (S-Memory); M (M-Memory); U (Universal). Example: $HARDWARE.TYPE = U

HEADINGS	prints all title and subtitle headings at the beginning of each page
LINES.PER.PAGE	prints maximum number of lines per page of listing if PAGE.NUMBERS is unspecified
LIST	lists all source records excluding macro record (default on)
LISTALL	lists all unconditionally excluded records to be printed
LIST.NOW	lists source records when read; same as $LISTP
LIST.PATCHES	lists all patches when read ;used with $MERGE (default on)
LISTP	same as $ LIST.NOW
MERGE	merges a secondary source file ("CARDS") with the primary source file ("SOURCE") replacing primary source records by secondary records with the same sequence numbers
NEW	creates a new source file ("NEWSOURCE")
NO	resets any specified dollar option if allowed
NOPS	generates a NOP for external linking code
OLD.LISTING.FORMAT	uses listing for pre-5.1 version of the compiler
PAGE	skips to a new page before printing
PAGE.NUMBERS	puts page number on each new page of listing
PARAMETER.BLOCK	punches a parameter block with the object deck if used with $DECK; otherwise only code is punched
PASS.END	displays compiler pass information on Console
PROTECT	protects SKIP when specified

RESET	resets any specified conditional inclusion option
SEQ	resequences source records
SET	sets any specified condtional inclusion options
SINGLE	prints single-space listings (default on)
SUBSET	generates code for B1710 (S-Memory) Processors; same as $B710, $HARDWARE.TYPE=S
SUPPRESS	suppresses printing of warning messages
TAPE	specifies that the source file is on tape
VOID	deletes a specified range of source records. The specified range must be exactly 8 characters; the first character must not be blank or %
XREF	sets XREF.LABELS and XREF.NAMES
XREF.ALL	sets XREF.LABELS, XREF.NAMES and XREF.REGISTERS
XREF.LABELS	cross-references all labels
XREF.NAMES	cross-references all names
XREF.REGISTERS	cross-references all registers

NOTES AND RESTRICTIONS:

A. Unless otherwise specified (through the MERGE option), the only source of input is the card reader. Once $ MERGE has been specified, it is not possible to again indicate "CARDS ONLY".

B. If no dollar cards are used the default options are: ALLCODE, AMPERSAND, CHECK, COMPILE, FRAME, LIST and SINGLE. All input will be from cards.

C. Options are turned off only through the appearance of NO followed by the option word. Note that NO and the option word are separated by at least one blank.

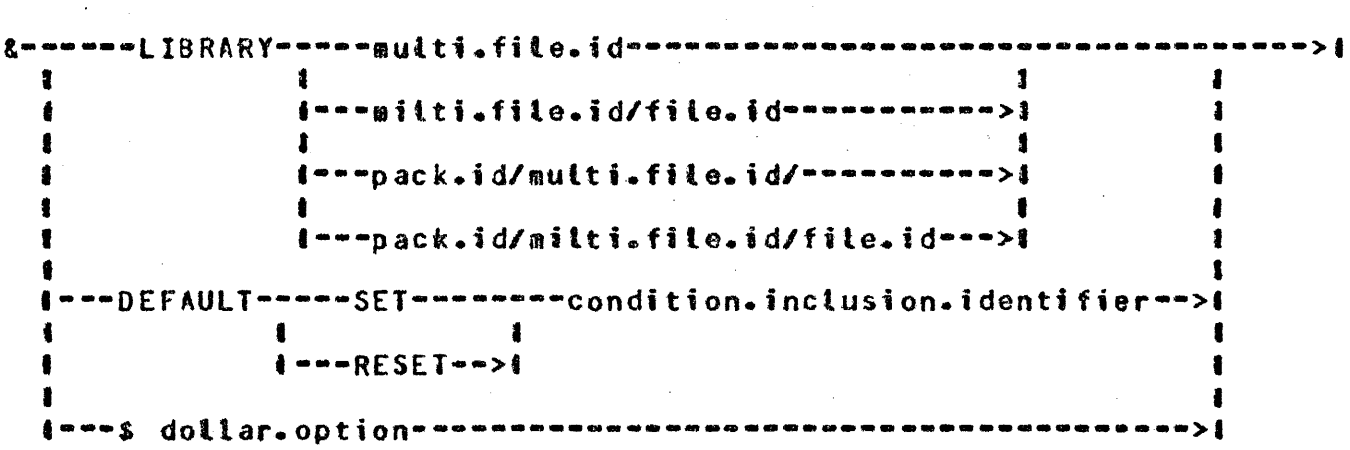
D. Comments may appear on dollar cards by preceeding the comment with a % (percent sign).

E. Dollar cards are not included as part of a "NEWSOURCE" file when $ NEW is specified.

F. In the case of redundant $ cards, the last card in the equivalence is perferred.

```
$ HARDWARE TYPE = $ SUBSET = $B710
$ CONTROL = $ DOLLAR
$ LISTP = $ LIST.NOW
```

AMPERSAND CARD SYNTAX:



AMPERSAND CARD SEMANTICS:

LIBRARY — Causes the specified file to be opened and compiled. Compilation proceeds to the end of the library file with no contribution from any standard primary or secondary input file. At end of file, compilation is resumed from the standard input files.

DEFAULT — Specifies default settings for one or more conditional inclusion toggles. The default setting for a particular toggle will take effect only if no previous $ or & card specified a setting for that toggle.

EXAMPLE: & DEFAULT SET TOG.A RESET TOG.B...

$ Any valid dollar statement may be used.

NOTES AND RESTRICTIONS:

A. A library file is assumed to be a disk file with 4 records/block.

B. The last record in a library file that is to be compiled must be FINI: This record cannot be omitted.

C. All & records are included as part of a "NEWSOURCE" file when $ NEW is specified.

D. &$ records are listed only when both $ DOLLAR and $ AMPERSAND are specified.

E. LIBRARY, DEFAULT and $ statements may not be intermixed on a single & card.

EXECUTION DECKS

STANDARD EXECUTION DECKS

Following are examples of some execution decks that may be used to compile MIL programs on a B1700.

If the source code is on cards only, the following deck may be used:

```
? COMPILE object.name MIL LIBRARY
? DATA CARDS
                .
                .
        (MIL Source Cards)
                .
                .
? END
```

If the source code is on disk, the following deck may be used:

```
? COMPILE object.name MIL LIBRARY
? FILE SOURCE NAME source.name;
? DATA CARDS
$ MERGE
                .
```

```
                    .
          (Patch Cards If Any)
                    .
                    .
    ? END
```

If it is desired to make a new source file on disk as the result of a compilation, one of the following decks may be used:

```
    ? COMPILE object.name MIL LIBRARY
    ? FILE NEWSOURCE NAME new.source.name;
    ? DATA CARDS
    $ NEW
                    .
                    .
          (MIL Source Cards)
                    .
                    .
    ? END
```

or if the source file is already on disk then:

```
    ? COMPILE object.name MIL LIBRARY
    ? FILE SOURCE NAME source.name;
    ? FILE NEWSOURCE NAME new.source.name;
    ? DATA CARDS
    $ MERGE NEW
                    .
                    .
         (Patch Cards If Any)
                    .
                    .
    ? END
```

Each of the names object.name, source.name, and new.source.name may take one of the following formats:

```
    multi.file.id
    multi.file.id/file.id
    pack.id/multi.file.id/
    pack.id/multi.file.id/file.id.
```

where

```
    multi.file.id = identifier
    file.id = identifier
    pack.id = identifier
```

EXAMPLES:

```
? COMPILE MINNI MIL LIBRARY
? FILE SOURCE NAME MOUSE/SOURCE;
? FILE NEWSOURCE NAME MICKEY/MOUSE/SOURCE;
```

or

```
? COMPILE MICKEY/MINNI/OBJECT MIL SYNTAX
? FILE SOURCE NAME MICKEY/MOUSE/;
? FILE NEWSOURCE NAME TRASH;
```

INTERNAL FILE NAMES

Some of the compiler's internal file names and their uses are listed below. This information is provided for use with ? FILE statements.

CARDS	Input file containing control and source records. Default device = Card Reader
LINE	Output file for the listing. Default device = Printer or Backup
PUNCH	Output file for the object deck; used when $ DECK is specified. Default device = Punch or Backup
SOURCE	Secondary input file for source records when $ MERGE is specified. Default device = Disk. USE.INPUT.BLOCKING
NEWSOURCE	Output file for new source records when $ NEW is specified. Default device = Disk, 4 records/block
LIBSOURCE	Input file for source records when & LIBRARY is encountered. Default device = Disk. USE.INPUT.BLOCKING
LINESAVE	A temporary work file containing a copy of the listing.
CODE.FILE	A temporary work file containing a copy of the object code.
PARAM.FILE	A temporary work file containing parameters affecting the object code and the listing.
MILXREF	Output file containing cross referencing information to be passed to MIL/XREFER.

NOTES:

A. The question mark (?) or an invalid character on 80 column cards, the ampersand (&), and the dollar sign ($) must appear in column 1 of the card.

B. The word LIBRARY or SYNTAX must appear after the word MIL on the compile card.

1. LIBRARY means the object MIL file will be saved on disk at the completion of a compile with no syntax errors.

2. SYNTAX means the object file will not be saved on disk.

C. The ? FILE card(s) must appear immediately after the ? COMPILE card. Each element in the file statement must be separated by a space and the statement must be terminated with a semicolon.

D. If the "SOURCE" or "NEWSOURCE" file is on magnetic tape then the word TAPE should appear on the file card. Example:

```
? FILE SOURCE NAME EBCSOR TAPE;
                or
? FILE SOURCE TAPE;
```

E. If disk cartridge is used and there is not enough room on the system cartridge for backup files etc., then some of the compiler's temporary files may be label equated to the users pack using the file card. Example:

```
? FILE LINESAVE NAME MYPACK/LINESAVE/;
```

F. Cards with a $ in column 1 are not transferred to the new source file when $ NEW is specified.

G. The MIL command FINI must be the last physical record in any source file. The word FINI must appear in column 6 or greater on this record.

APPENDIX B: HARDWARE INSTRUCTION FORMATS AND TABLES

NOTE: This Appendix will follow the hardware bit numbering convention: bits are numbered right to left, 0 through 23. As noted earlier, this is at variance with the software convention, where the most-significant (left-most) bit in any register was identified in the MIL syntax as bit 0 (zero), the next most significant as bit 1, etc.

B1700 HARDWARE TABLES

Table B-1: Register Addressing

GROUP (Row) NUMBER	SELECT (Column) NUMBER 0	1	2	3
0	TA	FU	X	SUM
1	TB	FT	Y	CMPX
2	TC	FLC	T	CMPY
3	TD	FLD	L	XANY
4	TE	FLE	A	XEQY
5	TF	FLF	M	MSKX
6	CA	BICN	BR	MSKY
7	CB	FLCN	LR	XORY
8	LA	*TOPM	FA	DIFF
9	LB	RESERVED	FB	MAXS
10	LC	RESERVED	FL	*MAXM
11	LD	*PERR	TAS	U
12	LE	XYCN	CP	*MBR
13	LF	XYST	*MSM	DATA
14	CC	*INCN	READ	CMND
15	CD	RESERVED	WRIT	NULL

* Available on B1720 systems only

Table B-2: Condition Registers

Software Bits

	0	1	2	3
BICN	LSUY	CYF	CYD	CYL
XYCN	MSBX	X=Y	X<Y	X>Y
XYST	LSUX	INT	Y NEQ 0	X NEQ 0
FLCN	FL=SFL	FL>SFL	FL<SLF	FL NEQ 0
*INCN	PORT DEVICE MISSING	PORT HIGH PRIORITY	PORT INTERRUPT	PORT LOCKOUT
CC	STATE LIGHT	TIMER INTERRUPT	I/O INTERRUPT	CONSOLE INTERRUPT
CD	MEMORY READ DATA PARITY ERROR INTERRUPT	MEMORY * WRITE/SWAP ADDR OUT OF BOUNDS OVER-RIDE	MEMORY * READ ADDR OUT OF BOUNDS INTERRUPT	MEMORY * WRITE/SWAP ADDR OUT OF BOUNDS INTERRUPT
	3	2	1	0

Hardware Bits

* Available on B1720 systems only

NOTES:

1. BICN, FLCN, XYST, and XYCN are addressable as source registers only.
2. The TOPM, MBR, and A registers are used to determine the memory (control or main) and location of the next microinstruction.
3. MSMA is control memory and may be addressed only from the maintenance Console during tape mode.
4. CPU is a destination register only.
5. NULL always contains a value of 0. Any register or scratchpad word to which it is moved will be cleared to 0.

Table B-3: Microinstructions

MICRO NAME	MC 15	14	13	12	MD 11	10	9	8	ME 7	6	5	4	MF 3	2	1	0
REGISTER MOVE	0	0	0	1	REG 1 GROUP SOURCE REGISTER				REG 1 SELECT		REG 2 SELECT		REG 2 GROUP SINK REGISTER			
SCRATCHPAD MOVE	0	0	1	0	REGISTER GROUP SOURCE OR SINK				REG SELECT		MOV DIR	DPW 1/2	DOUBLE PAD WORD ADDRESS			
4 BIT MANIPULATE	0	0	1	1	REGISTER GROUP, 4 BIT SOR & SNK				REG SEL	MANIPULATE VARIANTS			4 BIT MANIP. LITERAL			
BIT TEST REL BRANCH FALSE	0	1	0	0	REGISTER GROUP, 4 BIT SOURCE				REG SEL	TESTBIT NUMBER		DSP SGN	RELATIVE BRANCH DISPLACEMENT MAG			
BIT TEST REL BRANCH TRUE	0	1	0	1	REGISTER GROUP 4 BIT SOURCE				REG SEL	TESTBIT NUMBER		DSP SGN	RELATIVE BRANCH DISPLACEMENT MAG			
SKIP WHEN	0	1	1	0	REGISTER GROUP 4 BIT SOR & SNK				REG SEL	SKIP TEST VARIANTS			4 BIT TEST MASK			
READ/WRITE MEMORY	0	1	1	1	R/W VAR	COUNT FA/FL VARIANTS			DATAREG CODE		TW SGN		DATA TRANSFER WIDTH MAGNITUDE			
MOVE 8 BIT LITERAL	1	0	0	0	REGISTER GROUP, REG SEL IS 2.				ENTIRE 8 BITS OF 8 BIT LITERAL							
MOVE 24 BIT LITERAL	1	0	0	1	REGISTER GROUP, REL SEL IS 2.				8 MOST SIGNFICANT BITS OF FULL 24 BIT LITERAL							
SHIFT/ROTATE T REG	1	0	1	0	SINK REGISTER GROUP				SNK REG SELECT		S/R VAR		LEFT SHIFT/ROTATE COUNT			
EXTRACT FROM T REG	1	0	1	1	RIGHT BIT POINTER FOR EXTRACTION FLD				SNK REG CODE				EXTRACTION FIELD WIDTH			
BRANCH RELATIVE	1	1	0	DSP SGN	RELATIVE DISPLACEMENT MAGNITUDE											
CALL RELATIVE	1	1	1	DSP SGN	RELATIVE CALLED ADDRESS MAGNITUDE											
SWAP MEMORY	0	0	0	0	0	0	1	0	DATAREG CODE		TW SGN		DATA TRANSFER WIDTH MAGNITUDE			
CLEAR REGISTERS	0	0	0	0	0	0	1	1	L REG	T REG	Y REG	X REG	FA REG	FL REG	FU REG	CP REG
SHIFT/ROTATE X OR Y	0	0	0	0	0	1	0	0	S/R DIR VARIANT		X/Y VAR		LEFT OR RIGHT, X OR Y SHIFT/ROTATE COUNT			
SHIFT/ROTATE X AND Y	0	0	0	0	0	1	0	1	S/R, DIR VARIANT		LEFT OR RIGHT X AND Y SHIFT/ROTATE COUNT					
COUNT FA/FL	0	0	0	0	0	1	1	0	COUNT FA/FL VARIANTS				COUNT SCALAR MAGNITUDE			
EXCHANGE DPW	0	0	0	0	0	1	1	1	SINK DPW ADDRESS				SOURCE DPW ADDRESS			
SCRATCHPAD RELATE FA	0	0	0	0	1	0	0	0				DSP SGN	LEFT HALF PAD WORD ADDRESS			
MONITOR	0	0	0	0	1	0	0	1	LITERAL OCCURRENCE IDENTIFIER							
DISPATCH	0	0	0	0	0	0	0	0	0	0	0	1	DISPATCH VARIANTS			SKP FLG
CASSETTE CONTROL	0	0	0	0	0	0	0	0	0	0	1	0	CASSETTE MANIP VARIANTS			
BIAS	0	0	0	0	0	0	0	0	0	0	1	1	BIAS VARIANTS			TST FLG
STORE F INTO DPW	0	0	0	0	0	0	0	0	0	1	0	0	SINK DPW ADDRESS			
LOAD F FROM DPW	0	0	0	0	0	0	0	0	0	1	0	1	SOURCE DPW ADDRESS			
CARRY FF MANIPULATE	0	0	0	0	0	0	0	0	0	1	1	0	CYF CYD	CYF CYL	CYF 1	CYF 0
EXERCISE MSM(L)	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	R/W VAR
HALT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
OVERLAY M-STRING	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
NORMALIZE X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
BIND A, TOPM, MBR	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
NO OPERATION	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

VARIANTS	0 00 000	1 01 001	10 010	11 011	100	101	110	111
MOV DIR: 1/2 DPW:	P←R LEFT	R←P RIGHT						
MANIP. VARIANTS:	SET	AND	OR	EOR	INC	INC TEST	DEC	DEC TEST
DSP SIGN:	+	−						
DSP SIGN:	+	−						
SKIP TEST VARIANTS:	ANY CLR/	ALL CLR/	EQL CLR/	ALL CLR	ANY/ CLR/	ALL / CLR /	EQL / CLR /	ALL / CLR
R/W VAR: CNT VAR: REG SEL: T.W SIGN:	READ NØP X +	WRT FA↑ Y −	FL↑ T	FA↑ FL↑ L	FA↑ FL↑	FA↑	FL↑	FA↑ FL↑
S/R VAR:	SHFT	ROT						
SINK REG CODE:	X	Y	T	L				
DSP SIGN:	+	−						
DSP SIGN:	+	−						
TW SIGN; REG CODE:	+ X	− Y	T	L				
X/Y VAR: S/R, DIR:	X SFT←	Y SFT→	ROT←	ROT→				
S/R, DIR VARIANTS:	SFT←	SFT→	ROT←	ROT→				
COUNT FA/FL VAR:	NØP	FA↑	FL↑	FA↑ FL↑	FA↑ FL↑	FA↑	FL↑	FA↑ FL↑
DSP SIGN:	+	−						
SKP FLAG: DISP VAR:	FAIL LOCK	SUCC WRTLO	READ	R&C	WRTHI	ABSNT	UNDEF	UNDEF
CASSETTE MANIP:	START TAPE	STOP GAP	STOP ON X+Y	UNDEF	UNDEF	UNDEF	STOP ON X+Y	UNDEF
TEST FLG: BIAS VAR:	TST/ UNIT	TEST F	S	FS	NØP	FCP	NØP	NØP
R/W VAR	READ	WRIT						

Table 8-4: Variant Field Definitions

FOUR-BIT MANIPULATE (3nnn) VARIANTS

BITS 4-6	CONDITIONS
000	SET
001	AND
010	OR
011	EOR
100	INC
101	INC/TEST
110	DEC
111	DEC/TEST

SKIP WHEN (6nnn) SKIP TEST VARIANTS

BITS 4-6	CONDITIONS
000	ANY.SKIP
001	ALL.SKIP
010	EQU.SKIP
011	ALL CLR.SKIP
100	NOT ANY.SKIP
101	NOT ALL.SKIP
110	NOT EQU.SKIP
111	NOT ALL.CLR.SKIP

READ/WRITE MEMORY (7nnn) VARIANTS

BITS 6-7	CONDITIONS
00	X REG.
01	Y REG.
10	T REG.
11	L REG.

BITS 8-10	CONDITIONS
000	NOP
001	FA UP
010	FL UP
011	FA UP FL DN
100	FA DN FL UP
101	FA DN
110	FL DN
111	FA DN FL DN

EXTRACT FROM T REG. (8nnn) VARIANTS

BITS 5-6	CONDITIONS
00	X REG.
01	Y REG.
10	T REG.
11	L REG.

SWAP MEMORY (02nn) VARIANTS

BITS 6-7	CONDITIONS
00	X REG.
01	Y REG.
10	T REG.
11	L REG.

CASSETTE CONTROL (002n) VARIANTS

BITS 3-1	CONDITIONS
000	START TAPE
001	STOP ON GAP
010	STOP ON X NEQ Y
011-111	RESERVED

COUNT FA AND FL (06nn) VARIANTS

BITS 5-7	CONDITIONS
000	NOP
001	FA UP
010	FL UP
011	FA UP FL DN
100	FA DN FL UP
101	FA DN
110	FL DN
111	FA DN FL DN

DISPATCH (001n) VARIANTS

BITS 1-3	CONDITIONS
000	DISPATCH LOCK
001	DISPATCH WRITE
010	DISPATCH READ
011	DISPATCH RD & CLR
100	RESERVED
101	RESERVED
110	RESERVED
111	RESERVED

BIAS (003n) VARIANTS

BITS 3-1	CONDITIONS
000	FU
001	24 OR FL
010	24 OR SFL
011	24 OR FL OR SFL
100	NOP
101	24 OR CPL OR FL
110	NOP
111	24 OR CPL OR FL OR SFL

B1700 HARDWARE INSTRUCTION FORMATS

BIAS

OP CODE	BIAS VARIANTS (V)	TEST CPL NEQ 0 FLAG
0000 0000 0011	0...7	0 - NO TEST 1 - TEST CPL RESULT

15 4 3 1 0

This instruction sets CPU to the value 1 if the value of FU is 4 or 8 and to 0 otherwise, unless V = 2. If V = 2, the value of the CPU is determined by SFU in lieu of FU. SFU is the first 4 bits of the scratchpad word S08. (On the B1710, FU = 8 will set CPU = 0.)

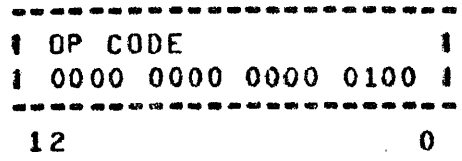
The value of CPL is also set to the smallest of the values denoted in the following table.

V	VALUES
0	FU
1	24 or FL
2	24 or SFL
3	24 or FL or SFL
4	CPL
5	24 AND CPL AND FL
6	CPL
7	CPL (not defined on the B1710)

If the test flag equals 1 and the final value of CPL is not 0, the next microinstruction is skipped.

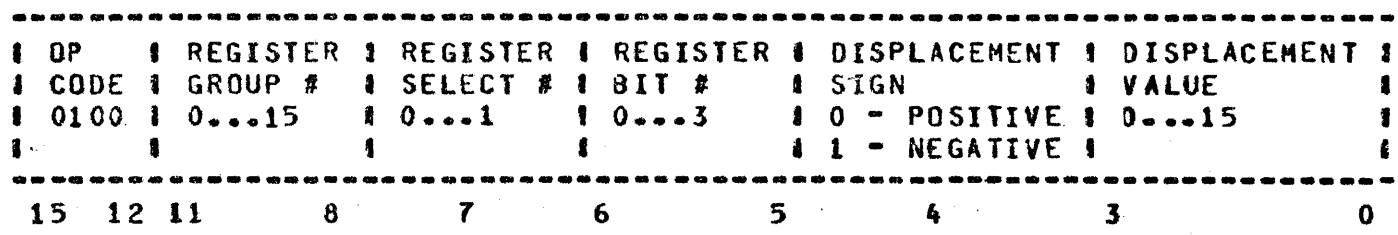
BIND

(Available on B1720 systems only)



This instruction moves the 24-bit value from the L register to the MBR register; moves the least significant 4 bits from the T register to the TOPM register; and moves the most significant 20 bits from the T register to the A register, truncating the left most 6 bits of the source.

BIT TEST BRANCH FALSE



This instruction tests the designated bit within the specified register and branches (relative to the next instruction) by the amount and direction of the signed displacement value if the bit is 0. If the bit is 1, a displacement value of 0 is assumed, and control passes to the next in-line M-instruction. A displacement value indicates the number of 16-bit words from the next in-line instruction. A negative sign indicates lower addresses in control memory (backword displacement). The maximum displacement is 15 microinstructions.

NOTE: Register Bit # is read from right to left, 0-3 according to the hardware bit numbering convention.

BIT TEST BRANCH TRUE

OP CODE	REGISTER GROUP #	REGISTER SELECT #	REGISTER BIT #	DISPLACEMENT SIGN	DISPLACEMENT VALUE
0101	0...15	0...1	0...3	0 - POSITIVE 1 - NEGATIVE	0...15
15 12	11 7	6	5	4	3 0

This instruction tests the designated bit within the specified register and branches (relative to the next instruction) by the amount and direction of the signed displacement value if the bit is 1. If the bit is 0, a displacement value of 0 is assumed, and control passes to the next in-line M-instruction. A displacement value indicates the number of 16-bit words from the next in-line instruction. A negative sign indicates lower addresses in control memory (backword displacement). The maximum displacement is 15 microinstructions.

NOTE: Register Bit # is read right to left, 0-3 according to the hardware bit numbering convention.

BRANCH

OP CODE	DISPLACEMENT SIGN 0 = POSITIVE 1 = NEGATIVE	DISPLACEMENT VALUE
110		0...4095
15 13	12	11 0

This instruction fetches the next microinstruction from the location obtained by adding the signed displacement value given in the instruction to the address of the next in-line microinstruction.

A displacement value indicates the number of 16-bit words.

CALL

OP CODE	DISPLACEMENT SIGN 0 = POSITIVE 1 = NEGATIVE	DISPLACEMENT VALUE
111		0...4095
15 13	12	11 0

This instruction pushes the address of the next in-line microinstruction (already contained in A register) into the A stack and then fetches the next microinstruction from the location obtained by adding the signed displacement value given in the instruction to the address of the next in-line microinstruction.

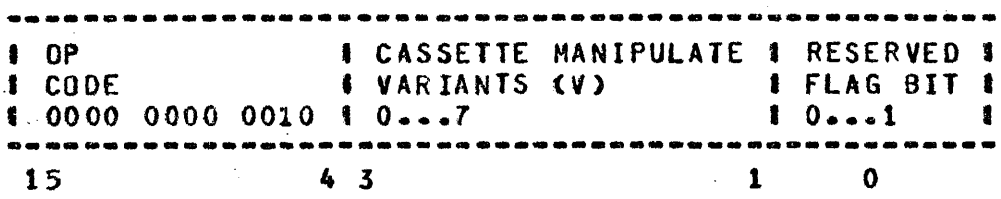
A displacement value indicates the number of 16-bit words.

NOTES:

1. EXIT, the opposite of CALL, is accomplished by employing the MOVE register instruction with TAS as the source register and A as the sink register.

2. When the A address is stored in the A stack, it is multiplied by 16 and stored as a bit address.

CASSETTE CONTROL



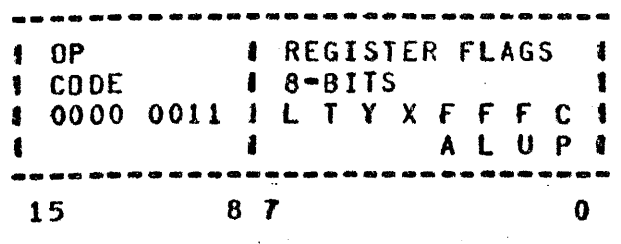
This instruction performs the indicated operation on the tape cassette.

V = 0 Start Tape
1 Stop Tape
2 Stop Tape if X NEQ Y
3 Reserved
4 Reserved
5 Reserved
6 Reserved
7 Reserved

All Stop Tape variants cause the tape to halt in the next available gap.

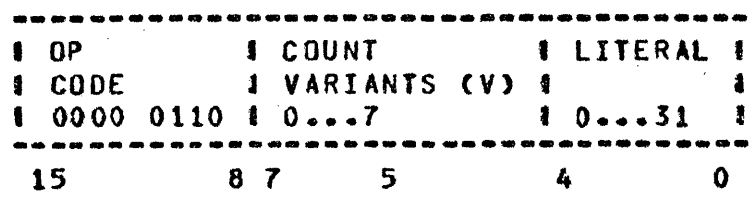
CLEAR REGISTERS

(Available on B1720 systems only)



This instruction clears the specified register(s) to 0 if the respective flag bit is 1.

COUNT FA/FL



This instruction increments (decrements) binarily the designated register(s) by the value of the literal contained in the instruction or by the value of CPL if the value of the literal is 0.

Neither overflow nor underflow of FA is dectected. The value of FA may go through its maximum value or its minimum value and wrap around.

Overflow of FL is not detected. The value of FL may go through its maximum value and wrap around. Underflow of FL is detected and will not wrap around. The value 0 is left in FL.

Literal values (or CPL values if LIT=0) of 25 through 31 are truncated to the value 24.

Count variants are as follows:

V = 000 No Count
001 Count FA Up
010 Count FL Up
011 Count FA Up and FL Down
100 Count FA Down and FL Up
101 Count FA Down
110 Count FL Down
111 Count FA Down and FL Down

PORT ABSENT is executed by the processor when necessary to return a Port Device Absent level signal to another port indicating the absence of the designated channel.

Dispatch operations in the case of Processor-2 and Processor Adapter-1 (direct connect to memory) are limited to the following:

1. LOCKOUT + SKIP-IF-NOT-ALREADY-LOCKED: always skips.
2. WRITE LOW: always sets Port Device Absent level true (true indicates absence).
3. READ AND CLEAR: always sets the Port Device Absent level false (false indicates present).

No changes occur in the T and L registers. In the INCN register only the Port Device Absent bit can change. The Lockout, the Interrupt, and High Priority bits will always be false. No other dispatch operations are defined.

EXTRACT FROM REGISTER T

OP CODE	ROTATE BIT COUNT	DESTINATION REGISTER	EXTRACT BIT COUNT
1011	0...24	00 - X	0...24
		01 - Y	
		10 - T	
		11 - L	

15 12 11 7 6 5 4 0

This instruction rotates the T register contents left by the ROTATE count, extracts the bits specified and moves the result to the sink register. If the extract bit count is less than 24, the data is right-justified with the left (most-significant) zero bits supplied.

The contents of the T register are unchanged unless it is also the sink register.

A rotate value of 24 is equal to 0 and is equivalent to a NO OPERATION.

DISPATCH

(Requires a hardware I/O subsystem available on the B1720 only)

OP CODE	DISPATCH VARIANTS	SKIP VARIANT (Applies only to lockout variant)
0000 0000 0001	000-LOCKOUT	0-SKIP IF ALREADY LOCKED
	001-WRITE	1-SKIP IF NOT ALREADY LOCKED
	010-READ	
	011-READ & CLEAR	
	100-WRITE HIGH	
	101-PORT ABSENT	

15 4 3 1 0

This instruction sends/receives interrupt and interrupt information to/from other ports.

Since the interrupt system is shared by all ports, the processor should gain control of the interrupt system by successfully completing a LOCKOUT prior to a DISPATCH WRITE.

LOCKOUT sets the lockout bit in the DISPATCH register and allows, via the skip variant, skipping or not skipping the next 16-bit instruction based upon the success or failure (already set) of the LOCKOUT.

WRITE (High or Low) DISPATCH sets the Lockout and Interrupt flip flops in the port interchange. It also stores the contents of the L register into memory location 0 to 23 and the contents of the least-significant seven bits of the T register (designating the destination port # and channel #) into the appropriate port interchange register. In addition, it sets (Write High) or resets (Write Low) the high Interrupt flip flop in the port interchange.

READ DISPATCH stores the contents of memory locations 0 through 23 into the L register and the contents of the Port Channel register into the least significant 7 bits of the T register. The other 17 bits of T are unaffected.

READ AND CLEAR DISPATCH in addition to performing the READ DISPATCH operation clears the lockout flip flop, the two interrupt flip flops and the Port Device Absent flip flop in the port interchange. It does not clear any memory locations.

NOTE: The microprogramming assembler uses the left-most bit to be extracted and calculates the rotate bit count to be used by the hardware circuits. The assembler addresses the bits within the T register left to right as 0 through 23; hardware addresses the bits right to left as 0 through 23.

FOUR-BIT MANIPULATE

OP CODE	REGISTER GROUP #	REGISTER SELECT #	MANIPULATE VARIANTS (V)	LITERAL
0011	0...15	0...1	0...7	0...15
15 12	11 8	7	6 4	3 0

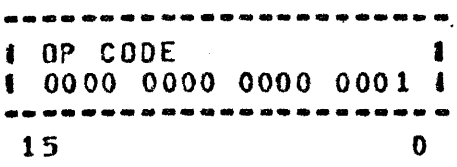
This instruction performs the operation specified by the variants on the designated register.

V = 0 The register is set to the value of the literal.
1 The register is set to the logical AND of the register and literal.
2 The register is set to the logical OR of the register and literal.
3 The register is set to the logical EXCLUSIVE-OR of the register and literal.
4 The register is set to the binary sum (modulo 16) of the register and literal.
5 The register is set to the binary sum (modulo 16) of the register and literal, and the next microinstruction is skipped if a carry is produced.
6 The register is set to the binary difference (modulo 16) of the register and the literal.
7 The register is set to the binary difference modulo 16 of the register and literal, and the next microinstruction is skipped if a borrow is produced.

EXCEPTIONS:

BICN, FLCN, XYCN, XYST, INCN (B1720) and CPU (B1710) when specified as operand registers are not changed as a result of this operation. However, the carry or borrow outputs are produced and a skip can result.

HALT

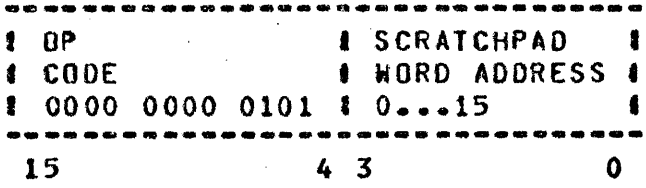


This instruction stops the execution of the microinstructions. In RUN mode the next micro to be executed is fetched and stored in the M register, and the A register points to the next following micro. In TAPE mode the next micro is not fetched and stored in the M register, but the HALT micro is left in the M register.

The register indicated by the register select switch will be displayed.

LOAD F FROM DOUBLEPAD WORD

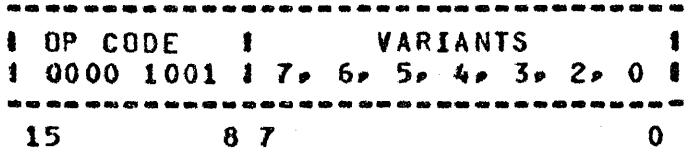
(Available on B1720 systems only)



This instruction moves the contents of the A and B portions of the designated scratchpad word to the FA and FB registers respectively.

MONITOR

(Available on B1720 systems only)



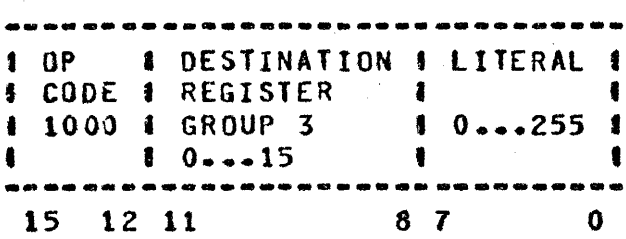
This instruction skips to the next sequential instruction.

During the time this micro-operator is executing the operator and the last two bits (0 and 1) are decoded, ANDed with the system clock and are present in the backplane as follows:

MONITOR	0	True for the OP Code
MONITOR	00R0	True if last two bits are 00
MONITOR	01R0	True if last two bits are 01
MONITOR	02R0	True if last two bits are 10
MONITOR	03R0	True if last two bits are 11

At the backplane, the monitors are one-half clock from leading edge to trailing edge.

MOVE 8-BIT LITERAL



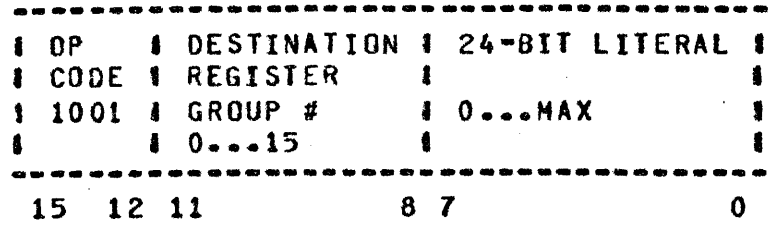
This instruction moves the 8-bit literal given in the instruction to the sink register. If the move is between registers of unequal lengths, the data is right-justified with left (most-significant) zero bits supplied.

EXCEPTIONS:

1. READ and WRIT are excluded as sinks.

2. When M is used as a sink register, the operation is changed to a bit-OR which modifies the next microinstruction. It does not modify the instruction as stored in memory.

MOVE 24-BIT LITERAL

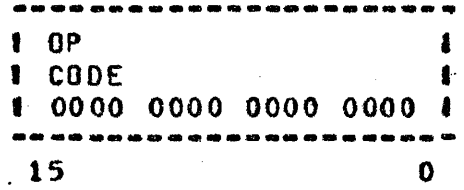


This instruction moves the 24-bit literal given in the double-length microinstruction to the sink register. If the move is between registers of unequal lengths, the literal is truncated from the left.

EXCEPTIONS:

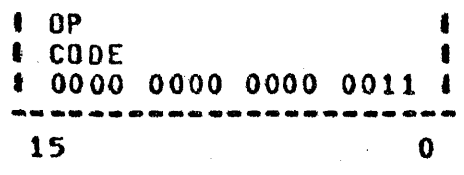
1. READ, WRIT, M and CP (B1710) are excluded as sinks.
2. The MSMA register (available only on the B1720) may be a sink only in the TAPE mode.

NO OPERATION



This instruction initiates a skip to the next sequential instruction.

NORMALIZE X

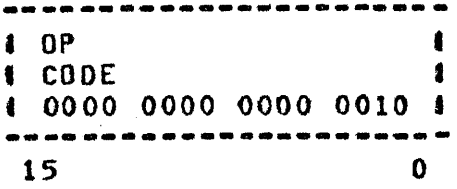


This instruction shifts the X register left while counting FL down until FL = 0 or until the bit in X referenced by CPL = 1. Zeros are shifted into the right-most end of X.

CPL = 1 references the right-most bit of X while CPL = 24 references the left-most bit of X. If CPL = 0, the operation will continue until FL = 0.

OVERLAY CONTROL MEMORY

(Available on B1720 systems only)



This instruction overlays control memory (M-Memory) from main memory.

The starting main memory address is in the FA register; the length of the data to be overlaid, in bits, is in the FL register. The starting control memory address is in the L register.

Execution of the instruction proceeds as follows:

1. The contents of the A register are moved to the TAS register.
2. The contents of the L register are moved to the A register.
3. The first 16 bits of data are read from main memory and stored in the control memory via register L. Register FL is decremented by 16 bits; FA is incremented by 16 bits; and A is incremented by 1 word.
4. Step 3 is repeated until FL = 0 or A > MAXM, at which point the process terminates with a move of TAS to A.
5. The operation then continues with the next microinstruction.

READ/WRITE MEMORY

OP CODE 0111	DIRECTION 0 TO REGISTER 1 TO MEMORY	COUNT VARIANTS 0...7	REGISTER # 00 = X 01 = Y 10 = T 11 = L	FIELD DIRECTION 0 - POSITIVE 1 - NEGATIVE	MEMORY FIELD LENGTH 0...26
15 12	11	10 8	7 6	5	4 0

This instruction moves the contents of the register (memory) to the memory (register). If the value of the memory field length is less than 24, the data from memory is right-justified with left (most-significant) zero bits supplied while the data from the register is truncated from the left.

The contents of the source is unchanged.

Register FA contains the bit address of the memory field while the memory field direction sign and memory field length are given in the instruction.

If the value of the memory field length as given in the instruction is 0, the value in CPL is used.

Memory field length values (or CPL values if MFL = 0) of 25 and 26 are truncated to the value of 24. When used on a WRITE operation, the value 25 and 26 cause odd and even parity respectively to be written into memory regardless of the parity of the read data.

For a description of the count variants, see COUNT FA/FL.

READ/WRITE MSM

(Available on B1720 systems only)

OP CODE 0000 0000 0111	VARIANTS G/B	H/F	S/N	R/W VARIANT 0 TO X 1 FROM x
15 4	3	2	1	0

This instruction (1) moves the contents of the X register to the M-Memory word specified by the address contained in the L register if the R/W variant bit = 1; data is right justified with left (most significant) bits supplied or (2) moves the contents of the M-Memory word specified by the address contained in the L register to the X register if the R/W variant bit = 0; data is right justified with left (most significant) zero bits supplied.

The lower 4 bits and the upper 8 bits of the address in L are ignored.

READ/WRITE MSM causes the A register to be moved to the TAS register and the L register to be moved to the A register before the instruction is executed. The TAS is restored to A after the READ/WRITE MSM operation is completed.

The S variant is used to enable the set/reset of the G/B and H/F flip flops. If S = 1, the G/B and H/F flip flops are set/reset by the G/B and H/F variants. If S = 0, no change is made in the G/B and H/F flip flops.

If the G/B flip flop is true, all READ/WRITE MSM operations will force bad parity in the addressed word. If the G/B flip flop is false, all READ/WRITE MSM operations will force good parity in the addressed word.

If the H/F flip flop is true, the processor upon reading an M-Memory word containing parity error will flag the error condition by setting a CD bit true. It will not halt. If the H/F flip flop is false, the processor upon detection of a parity error in reading an M-Memory word will flag the error condition by setting PERR bit 1 true and then halt. Reading an M-Memory word occurs when fetching a M-op from M-Memory or when moving an M-Memory word to any destination.

The H/F and G/B flip flops are cleared to zero (false) with the CLEAR signal. If S = 1, the G/B and H/F flip flops are set/reset prior to the execution of the READ/WRITE MSM portion of the operation.

REGISTER MOVE

OP CODE 0001	SOURCE REGISTER GROUP # 0...15	SOURCE REGISTER SELECT # 0...3	DESTINATION REGISTER GROUP # 0...3	DESTINATION REGISTER SELECT # 0...15
15 12	11 8	7 6	5 4	3 0

This instruction moves the contents of the source register to the sink register. If the move is between registers of unequal lengths, the data is right-justified with left (most-significant) zero bits supplied or the data is truncated from the left, whichever is appropriate.

The contents of the source register are unchanged unless it is also the sink register.

EXCEPTIONS:

1. WRIT, CMND (and CPU, READ on B1710) are excluded as source registers.

2. When the M register is used as a sink in RUN or STOP mode, the operation is changed to an bit-OR which modifies the next microinstruction. It does not modify the instruction stored as stored in memory. In TAPE mode, no bit-OR takes place.

3. BICN, FLCN, XYCN, XYST, INCN, READ, WRIT, SUM, CMPX, CMPY, XANY, XEOY, XEOR, MSKX, DIFF, MAX, MAXM, and U are excluded as sink registers.

4. U is excluded as a source register in the STEP mode.

5. When DATA (and SUM, DIFF on B1710) is designated as a source, CMND, and DATA are excluded as sinks.

6. On the B1710 when A, M, CP, or DATA is designated as a source, all 4-bit registers are prohibited as sinks.

7. On the B1720, when U or DATA is designated as a source and when the next micro is to be obtained from main memory, M is excluded as a sink.

SCRATCHPAD MOVE

OP CODE 0010	REGISTER GROUP # 0...15	REGISTER SELECT # 0...3	DIRECTION 0-TO SCRATCHPAD 1-FROM SCRATCHPAD	SCRATCHPAD WORD 0-LEFT WORD 1-RIGHT WORD	SCRATCHPAD WORD ADDRESS 0...15
15 12	11 8	7 6	5	4	3 0

This instruction moves the contents of the register (scratchpad) to the scratchpad (register). If the move is between fields of unequal lengths, the data is right-justified with left (most-significant) zero bits supplied or the data is truncated from the left, whichever is appropriate.

The contents of the source register are unchanged.

EXCEPTIONS:

1. When the M register is used as a sink, the operation is changed to a bit-OR which modifies the next microinstruction. It does not modify the instruction as stored in memory.

2. BICN, FLCN, XYCN, XYST, INCN, READ, WRIT, SUM, CMPX, CMPY, XANY, XORY, XEQY, MSKX, MSKY, DIFF, MAXS, MAXM and U are excluded as sink registers.

3. WRIT, CMND (and CPU, READ on B1710) are excluded as source registers.

4. U is excluded as a source in STEP mode.

5. On the B1710 M as a source results in a transfer of 24 zeros.

SCRATCHPAD RELATE FA

OP CODE	RESERVED	SIGN OF	LEFT HALF ADDRESS
0000 1000	000	0 = POSITIVE 1 = NEGATIVE	OF A SCRATCHPAD WORD 0...15

15 8 7 5 4 3 0

This instruction replaces the contents of the FA register by the binary sum of FA and the left half the specified scratchpad word.

Neither overflow nor underflow of FA is detected. The value of FA may go through its maximum value or its minimum value and wrap around.

SET CYF

OP CODE	SET VARIANTS (V)
0000 0000 0110	1,2,4,8

15 4 3 0

This instruction sets the carry flip-flop as specified by the variants.

V = 1 Set CYF to 0
2 Set CYF to 1
4 Set CYF to CYL (carry total from sums)
8 Set CYF to CYD (carry borrow from difference)

NOTES:

1. CYL is generated under the control of the length in CPL.

2. CYF is an input to the arithmetic logic along with the X and Y registers. CYF is the left-most bit of the CP portion of the C register.

SHIFT/ROTATE REGISTER T LEFT

OP CODE	DESTINATION REGISTER GROUP #	DESTINATION REGISTER SELECT #	SHIFT/ROTATE 0 - SHIFT 1 - ROTATE	SHIFT/ROTATE BIT COUNT
1010	0...15	0...3		0...24

15 12 11 8 7 6 5 4 0

This instruction shifts (rotates) register T left by the number of bits specified and then moves the 24-bit result to the sink register. If the move is between registers of unequal lengths, the data is right-justified, with data truncated from the left.

The contents of the T register are unchanged unless it is also the sink register.

Zero fill on the right and truncation on the left occurs with the shift operation. ROTATE is an end-around shift with no truncation or fill.

If the value of the SHIFT/ROTATE COUNT as given in the instruction is 0, the value given in CPL is used.

EXCEPTIONS:

1. When the M register is used as a sink register, the operation is changed to a bit-OR which modifies the next microinstruction. It does not modify the instruction as stored in memory.

2. BICN, FLCN, XYCN, XYST, INCN, READ, WRIT, SUM, CMPX, CMPY, XANY, XEQY, XORY, DIFF, MAXS, MAXM and U are excluded as sink registers.

SHIFT/ROTATE REGISTERS XY LEFT/RIGHT

OP CODE	SHIFT/ROTATE VARIANT	SHIFT/ROTATE DIRECTION	SHIFT/ROTATE BIT
0000 0101	0 - SHIFT 1 - ROTATE	VARIANT 0 - LEFT 1 - RIGHT	COUNT 0...48

15 8 7 6 5 0

This instruction shifts (rotates) register X and Y left (right) by the number of bits specified. The register X is the left-most (most-significant) half of the concatenated 48-bit XY register. Only a count of one may be specified on the B1710 for the concatenated XY register.

Zero fill on the right and truncation on the left occurs with the left shift. Zero fill on the left and truncation on the right occurs with the right shift.

If the value of the SHIFT/ROTATE COUNT as given in the instruction is 0, the operand is shifted/rotated by the amount determined by CPU as follows:

CPU	SHIFT/ROTATE COUNT
00	1 bit
01	4 bits
10	Undefined
11	8 bits (available only on B1720 systems)

NOTE: The shift by CPU option is available only on B1720 systems.

SHIFT/ROTATE REGISTER X/Y LEFT/RIGHT

OP CODE	SHIFT/ROTATE VARIANT	SHIFT/ROTATE DIRECTION	X/Y VARIANT	SHIFT/ROTATE BIT COUNT
0000 0100	0 - SHIFT 1 - ROTATE	0 - LEFT 1 - RIGHT	0 - X REG 1 - Y REG	0...24
15 8	7	6	5	4 0

This instruction shifts (rotates) register X or Y left or right by the number of bits specified.

Zero fill on the right and truncation on the left occurs with the left shift. Zero fill on the left and truncation on the right occurs with the right shift.

If the value of the SHIFT/ROTATE COUNT as given in the instruction is 0, the operand is shifted (rotated) by the amount determined by CPU as follows:

CPU	SHIFT/ROTATE COUNT
00	1 bit
01	4 bits
10	Undefined
11	8 bits (not available on B1710 systems)

NOTE The shift by the CPU option is available on B1720 systems only.

SKIP WHEN

OP CODE	REGISTER ROW #	REGISTER COLUMN #	SKIP TEST VARIANTS (V)	MASK 0...15
0110	0...15	0...1	0...7	

15 12 11 8 7 6 4 3 0

This instruction tests only the bits in the register that are referenced by the 1 bits in the mask and ignores all others. It then performs the actions specified below. Exception: If V = 2 or V = 6, it compares all bits for an equal condition.

V = 0 If any of the referenced bits are 1's, the next M-instruction is skipped.

1 If all the referenced bits are 1's, the next M-instruction is skipped.

2 If the register is equal to the mask, skip the next M-instruction.

3 This is the same as V = 1, but the referenced bits are also cleared to 0 without affecting the non-referenced bits.

4 If any of the referenced bits are 1's, the next M-instruction is not skipped.

5 If all the referenced bits are 1's, the next M-instruction is not skipped.

6 If the register is equal to the mask, the next M-instruction is not skipped.

7 This is the same as V = 5, but the referenced bits are also cleared to 0 without affecting the non-referenced bits.

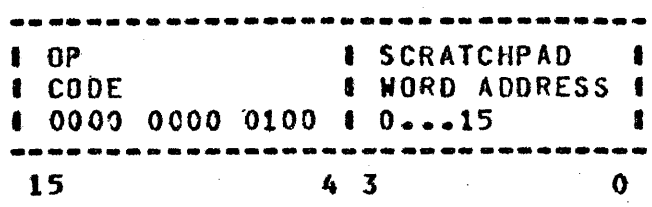
NOTES AND RESTRICTIONS:

1. If the mask equals 0000 the ANY result is false. The skip is made for V = 0 and is not made for V = 4. If the mask equals 0000, the ALL result is true. The skip is made for V=5 and V=7 and is not made for V=1 and V=3.

2. BICN, FLCN, XYCN, XYST, and cannot be cleared with V = 3 or V = 7. However, they can be tested.

STORE F INTO DOUBLEPAD WORD

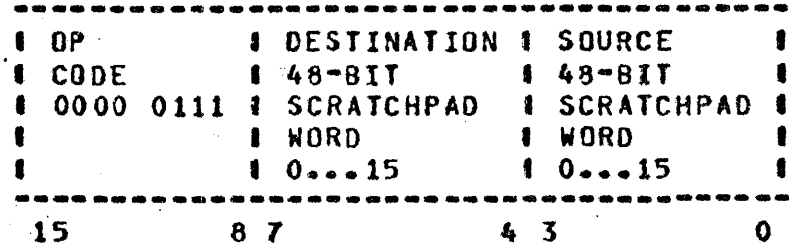
(Available on B1720 systems only)



This instruction moves the contents of the FA and FB registers to the designated scratchpad word. FA is transfered to the A half of the scratchpad word, and FB (which contains FL, FT, and FU) is transferred to the B scratchpad word.

The contents of FA and FB remain unchanged.

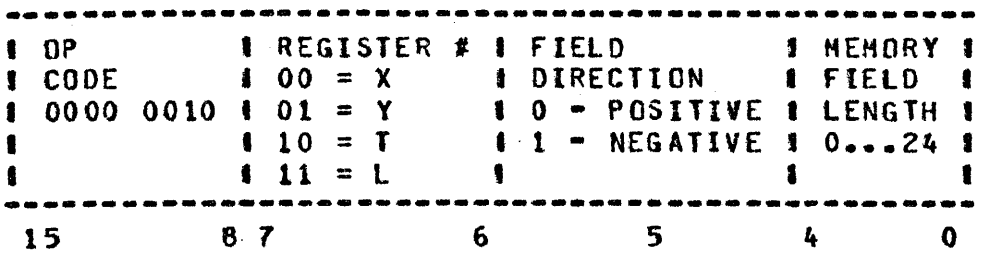
SWAP F WITH DOUBLEPAD WORD



This instruction moves the contents of the FA and FB registers to a hardware holding register. It also moves the contents of the left and right word of the source scratchpad word to the FA and FB register respectively, and moves the contents of the hardware holding register to the destination scratchpad word.

SWAP, MEMORY

(Available on B1720 systems only)



This instruction swaps data from main memory with the data in the specified register. If the value of the memory field is less than 24, the data from memory is right-justified with left (most-significant) zero bits supplied. The data from the register is truncated from the left before entering memory.

Register FA contains the absolute binary address of the main memory field while the field direction sign and field is given in the instruction.

If the value of the memory field length as given in the instruction is 0, the value given in CPL is used.

Table B-6: Microinstruction Timing

B1710 Notes	B1710 Clocks	Microinstructions	B1720 Clocks	B1720 Notes
	2	BIAS	1	
	-	BIND	3	
	2	BIT TEST BRANCH FALSE	1	
	2	BIT TEST BRANCH TRUE	1	
	4	BRANCH	1	1
1	5	CALL	2	
	2	CASSETTE CONTROL	1	
	-	CLEAR REGISTERS	1	
	4	COUNT FA/FL	1	
	-	DISPATCH	6/5	
	3	EXTRACT FROM REGISTER T	1	
	2	FOUR-BIT MANIPULATE	1	
	2	HALT	1	
	-	LOAD F FROM DOUBLEPAD WORD	1	
	2	MONITOR	1	
	2	MOVE 8-BIT LITERAL	1	
	6	MOVE 24-BIT LITERAL	2	
	2	NO OPERATION	1	
3	6	NORMALIZE X	1	2
	-	OVERLAY CONTROL MEMORY	5	3
	8	READ/WRITE MEMORY	5/4	4
	-	READ/WRITE MSM	6	
2	2	REGISTER MOVE	1	
2	2	SCRATCHPAD MOVE	1	
	4	SCRATCHPAD RELATE FA	1	
	2	SET CYF	1	
	3	SHIFT/ ROTATE REGISTER T LEFT	1	
4	6	SHIFT/ ROTATE XY LEFT/RIGHT	1	2
	3	SHIFT/ROTATE X/Y LEFT/RIGHT	1	2
	2	SKIP WHEN	1	
	-	STORE F INTO DOUBLEPAD WORD	1	
	10	SWAP F WITH DOUBLEPAD WORD	2	
	-	SWAP MEMORY	4	5

B1710 NOTES:

The basic clock of the B1710 is 4 mega hertz.

1. This includes the fetch of the called microinstruction.
2. For BCD result register moves, there are three clocks.
3. There are six clocks per bit plus one additional clock.
4. Only a value of one bit is allowed in the B1710.

B1720 NOTES:

The basic clock of the B1720 is 6 mega hertz.

1. If the relative address is not within control memory (therefore in main memory), there are two clocks.
2. There is one clock per bit.
3. There are five clocks per 16 bits (one microinstruction) plus five clocks.
4. READ is five clocks until the processor receives the data. WRITE is four clocks until the processor is released. Some instructions may be performed during the processor READ or WRITE command times if they immediately follow the READ or WRITE commands: this is called "concurrency". Consecutive READ or WRITE commands operate at MAIN MEMORY READ cycle speed (four clocks) or WRITE cycle speed (six clocks) respectively.
5. The data is presented to the processor and is released in one MAIN MEMORY READ cycle. Concurrent execution of certain microinstrcutions is performed if they immediately follow the SWAP command. The WRITE portion of the SWAP command is begun and performed in parallel to the READ portion, and main memory is not available for the duration of a WRITE cycle. For consecutive main memory commands, refer to note 4.

APPENDIX C: RESERVED WORDS AND SYMBOLS

	BEGIN	DEFINE
	BIAS	DEFINE.VALUE
(	BICN	DIFF
.	BIT	DIFFERENCE
<	BITS	DISPATCH
<=	BR	DOWN
(	BRANCH	DUMP
+	BRANCH.EXTERNAL	ELSE
*	BY	EMIT.RETURN.TO.EXTERNAL
)	CA	END
;	CALL.EXTERNAL	EOR
¬	CARRY	EQL
-	CASSETTE	EXIT
/	CAT	EXTRACT
/=	CB	F
,	CC	FA
_	CD	FA.POINTS
>	CHARACTER	FALSE
>=	CLEAR	FB
#	CMND	FINI
@	CMPX	FIXED
'	CMPY	FL
=	CODE.SEGMENT	FLC
"	CODE.SEGMENT.NUMBER	FLCN
A	COMPLEMENT	FLD
ABSOLUTE	CONSOLE.SWITCHES	FLE
ADD	CONSTANT	FLF
ADDRESS	COUNT	FOR
ADJUST	CP	FORWARD
ALL	CPL	FROM
AND	CPU	FT
ANY	CYD	FU
ANY.INTERRUPT	CYF	GEQ
AS	CYL	GO
ASSIGN	DATA	GTR
ASTACK	DATA.LENGTH	HALT
AT	DATA.TYPE	HEX.SEQUENCE.NUMBER
BACKWARD	DATA.USAGE	HI.PRIORITY
BASE.LIMIT	DEC	HIPRI

IF
INC
INCLUDE
INCN
INTERRUPT
INTO
JUMP
L
LA
LB
LC
LD
LE
LEFT
LENGTH.BETWEEN.ENTRIES
LEQ
LF
LIT
LOAD
LOAD.MSMA
LOAD.SMEM
LOCAL.DEFINES
LOCATION
LOCK
LOCKED
LOCKOUT
LR
LSBX
LSBY
LSS
LSUX
LSUY
M
M.MEMORY.BOUNDARY
MACRO
MAKE.SEGMENT.TABLE.ENTRY
MAXIMUM
MAXM
MAXS
MICRO
MINIMUM
MINUS
MOD
MONITOR
MOVE
MSBX
MSKX
MSKY
MSMA
MSML
NEQ
NEWSEGMENT
NO.DEVICE
NODEVICE
NOP
NORMALIZE
NOT
NULL
OR
OVERLAY
PAGE
PLUS
POINT
PORT
PROGRAM.LEVEL
READ
REDUNDANT.CODE
REMAPS
RESERVE.SPACE
RESET
REVERSE
RIGHT
ROTATE
S
S.MEMORY.LOAD
SEGMENT
SEGMENT.COUNT
SET

SFL
SFU
SHIFT
SKIP
SPACE
START
STOP
STORE
SUB.TITLE
SUBTRACT
SUM
SWAP
S0
S0A
S0B
S1
S1A
S1B
S10
S10A
S10B
S11
S11A
S11B
S12
S12A
S12B
S13
S13A
S13B
S14
S14A
S14B
S15
S15A
S15B
S2
S2A
S2B
S3A
S3B
S4
S4A
S4B
S5
S5A
S5B
S6
S6A
S6B
S7
S7A
S7B
S8
S8A
S8B
S9
S9A
S9B
T
TA
TABLE
TAS
TB
TC
TD
TE
TEST
TF
THEN
TITLE
TO
TODAYS.DATE
TODAYS.TIME
TOPM
TRACE
TRANSFER.CONTROL
TRUE
UNIT
UNLOCKED
UP
VALUE
WHEN
WITH
WRITE
WRITE.STRING
X
XANY
XCH
XEQY
XORY
XY
XYCN
XYST
Y
0