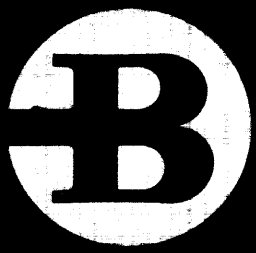
Burroughs

B 2500 and B 3500 SYSTEMS

ASSEMBLERS REFERENCE MANUAL



Burroughs

B 2500 and B 3500

INFORMATION PROCESSING SYSTEMS

ASSEMBLERS

REFERENCE MANUAL

Burroughs Corporation
Detroit, Michigan 48232

$5.00

Printed in U.S. America

4-69

1034949

AA 828690 AA 995117 AA 88919

This Reference Manual also contains information previously contained in "B 2500 and B 3500 Systems Advanced Assemblers Language Manual," form number 1025491, COPYRIGHT©1966 Burroughs Corporation, AA 828688.

This reprint includes the information released under the following:

PCN 1034949-001 (October 30, 1969)
PCN 1034949-002 (November 20, 1970)

Correspondence regarding this document should be forwarded using the Remarks Form at the back of the manual, or may be addressed directly to Systems Documentation, Sales Technical Services, Burroughs Corporation, 6071 Second Avenue, Detroit, Michigan 48232.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (cont)

LIST OF TABLES

INTRODUCTION

The Burroughs B 2500/B 3500 Assembly Language is a symbolic code which makes all of the capabilities of the B 2500 and B 3500 available at the machine language level and offers full flexibility in the specification of instructions and data. Standard linkages to MCP and BCP operating routines may be called out to be included in the object program, as well as any pre-written symbolic code desired from the library file. The machine language output from the assembly translators is ready for execution in the environment of the appropriate Control Program.

To use the Assembly Language properly and efficiently, the programmer must have a greater knowledge of instruction and data conventions in the B 2500/B 3500 Systems than required when using higher-level languages. This manual describes these conventions and also acts as a general introduction to processor operations without requiring prior study of the B 2500/B 3500 Systems Reference Manual.

SECTION 1
DATA FORMATS

GENERAL.

Extreme versatility at processing data in different formats is designed into the hardware of B 2500/B 3500 Series Systems. This permits maximum storage efficiency with very little programming overhead. The processor recognizes mixed-format operations and, in all common cases, automatically performs any necessary data conversion.

Defined within this section are signed and unsigned numeric, floating point, word, and 8-bit EBCDIC and USASCII formats. The features available for efficient input/output data transfer are also described.

INTERNAL STORAGE.

Storage is organized into 4-bit digit locations, each with a unique decimal address. All information stored can be processed under one or several of the following formats:

a. Unsigned 4-bit numeric (UN).
b. Signed 4-bit numeric (SN).
c. Floating point.
d. 8-bit alphanumeric (UA).
e. Word.

UNSIGNED 4-BIT NUMERIC FORMAT (UN).

This format is for high-density storage of data and consists of a string of 1-100 decimal digits or undigits (see note below) starting at any location in storage and occupying one storage position per digit.

Example:

```
(1003)    (1009)
   1055623
```

The length of the example above is seven and the access address of the data is 1003.

NOTE

Four data bits can hold 16 (2^4) different values. In the B 2500/B 3500 System, ten of these values are reserved for the decimal digits 0-9. The other values are undigits and are assigned as shown in the chart below.

BINARY VALUE	CONSOLE DISPLAY	CODING DECLARATION
1010	∅	A
1011	⁄	B
1100	≠	C
1101	∅	D
1110	¼	E
1111	≤	F

SIGNED 4-BIT NUMERIC FORMAT (SN).

This format is for high-density storage of data and consists of a string of 1-100 decimal digits preceded by a sign starting at any location in storage and occupying one storage position per digit plus one for the sign.

Example:

(1003) (1010)
+5567141

The length of the above example is seven and the access address of data is 1003. The sign is not counted in the length of a SN field.

FLOATING POINT FORMAT.

This format is for scientific computation and consists of an exponent sign, two decimal exponent digits, a mantissa sign, and a string of 1-100 decimal mantissa digits starting at any location in storage and occupying one storage position per digit or sign.

Example:

(1003) (1011)
+03+15331

The length of the example above is five and the access address of data is 1003. Only the mantissa digits are counted in the length of the field.

8-BIT FORMAT (UA).

This format provides simple communication with the peripheral devices and consists of a string of 1-100 characters starting in any even-numbered location in storage and occupying two addressable storage positions per character.

Example:

```
(1000-1)       (1018-9)
      ABCDEFGHIJ
```

The length of the example above is ten and the access address of data is 1000. The first digit within a character is called its zone digit and the second its numeric digit. For example, the digit representation of the character X is E7, where E is the zone undigit and 7 is the numeric digit.

WORD FORMAT.

This format provides for fast manipulation of program and data blocks and consists of a string of 1-10,000 words starting in any modulo-4 location in core storage (address evenly divisible by four) and occupying four addressable storage positions per word.

Example 1:

```
(1000-3)                (1020-3)
      AB|CD|EF|GH|IJ|KL|.. ..
```

Example 2:

```
(1000-3)                              (1020-3)
      0012|5641|7785|4109|3218|4476|....
```

Example 3:

```
(1000-3)                    (1020-3)
 A12|6792|12B|CC|2148|62P|
```

The length of each of the above examples is six and the access address of data is 1000.

All data fields, regardless of format, are accessed at the left-most (lowest address) digit location occupied by the field.

INPUT/OUTPUT.

All data transfers between main memory and input/output units are performed at the character or word level depending on the type of peripheral and channel used. Transfers to or from main memory are parallel 8-bit core-image and all necessary conversion to special external representation (card codes, bit-serial disk recording, etc.) is performed within the peripheral control units themselves as a normal hardware function.

The standard internal character set for the B 2500/B 3500 is the 8-bit Extended Binary Coded Decimal Interchange Code (EBCDIC) (refer to appendix D). EBCDIC information can be read from or written to punched cards and 9-channel magnetic tape, thus providing data compatibility with other computer systems.

For data compatibility with other Burroughs machines, and to permit the attachment of proven-design peripheral units to the System, hardware translation between EBCDIC and BCL (Burroughs Common Language) is provided within the Central Control. Translation of output data to line printers or tape listers is automatic. Hardware translation is programmatically selectable for punched cards, paper tape, and magnetic tape. Translation is performed directly upon the data stream as it passes through the Central Control unit without any slowdown of I/O data rate.

To facilitate the use of other character sets for data communications (e.g., UNIVAC XS-3 and IBM 1050 PTTC codes), a powerful hardware translate instruction is provided. Up to 10,000 8-bit characters can be converted from one character set to another by means of a single instruction execution through conversion tables which are easy to set up and understand.

In data communication environments which involve heavy traffic in the USASCII character set, an even greater execution speed can be obtained by use of the USASCII programming mode. This operating mode,

which is programmatically selectable, causes all arithmetic operations and format conversions to produce the USASCII sign/zone bit configurations in the result field, rather than the EBCDIC configurations. Thus, numeric data can be received, processed, and returned without any translation whatsoever.

SECTION 2
GENERAL PROCESSOR DESCRIPTION

GENERAL.

This section provides a detailed description of the following machine-language fundamentals:

a. Instruction format.
b. Index registers.
c. Indirect addressing.
d. Literal operations.
e. Field lengths.
f. Program reserved memory.
g. Base and limit registers.
h. Edit operators.
i. Indirect field length.

INSTRUCTION FORMAT.

Instructions consist of two functional groups with similar formats; processor instructions and I/O descriptors. I/O descriptors directly concern only those users who are controlling sorter-readers or unusual peripheral devices under the BCP, otherwise, physical details of I/O are entirely handled by the software. Descriptors are explained in the section on BCP I/O coding methods (refer to section 5), and also in the B 2500/B 3500 Systems Reference Manual.

Processor instructions are variable in length (from one to four syllables) and contain none, one, two, or three addresses. Each syllable of the instruction has a length of six 4-bit digits.

OPERATION CODE.

The first two 4-bit digit positions of each instruction will be interpreted as an operation code.

VARIANTS.

The next four digit positions are used to specify one or two operational code variants. In most processor instructions, these variants are referred to as AF (A field length) and BF (B field

length), each being two digits in length. The variant digits AF/BF have other uses which will be described in detail when applicable.

ADDRESS FIELD.

The address field (see figure 2-1) is constructed in three parts:

a. Indexing. The two high-order bits of the first digit select an index register to be applied (if any) as follows:

1) 00 - no indexing.
2) 01 - apply Index Register 1.
3) 10 - apply Index Register 2.
4) 11 - apply Index Register 3.

b. Address Controller. The two remaining bits of the first digit specify the data format of the addressed field. Address controllers are as follows:

1) 00 - unsigned numeric.
2) 01 - signed numeric (including floating point).
3) 10 - alphanumeric (8-bit code).
4) 11 - indirect address.

c. The remaining five digits specify a base-relative (to the start of the program's memory area) address in core memory.

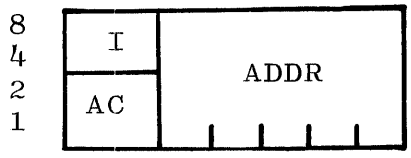


Figure 2-1. Address Syllable of Instruction (Six Digits)

Figures 2-2 through 2-5 provide formats of one, two, three, and four syllable instructions.

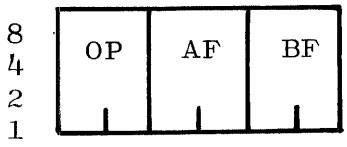


Figure 2-2. One Syllable Instruction (Six Digits)

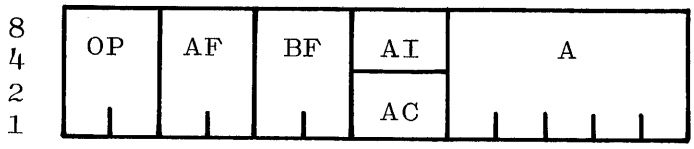


Figure 2-3. Two Syllable Instruction (12 Digits)

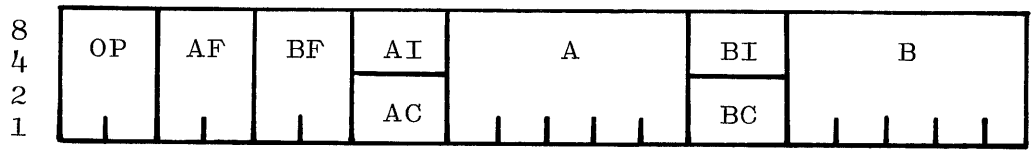


Figure 2-4. Three Syllable Instruction (18 Digits)

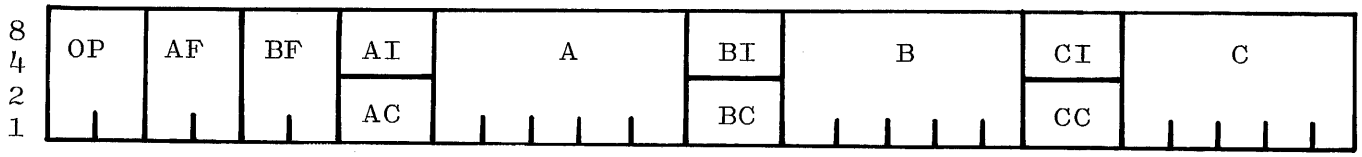


Figure 2-5. Four Syllable Instruction (24 Digits)

BRANCH FORMAT.

Address-branch instructions contain no variant field and the 2-digit operation code is immediately followed by the 6-digit branch address. Values of 0, 1, or 2 in the address controller are regarded as a high-order digit of the branch address. A binary value of 3 in the controller is interpreted as an indirect-address flag. Figure 2-6 provides the format of the address-branch instruction.

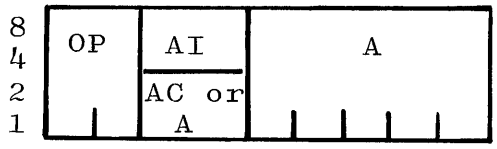


Figure 2-6. Address-branch Instruction Format

INDEX REGISTERS.

If an address field in an instruction specifies that an index register is to be applied, the value contained in the specified index register is algebraically added to the 5-digit address to produce an effective address.

Index registers are 7-digit signed numeric fields at fixed locations in Program Reserved Memory. The digit immediately following the sign must be zero and the remaining six digits may contain any decimal number from 1 through 999999.

The value in an index register always represents machine location displacement, without regard to the data format of the field accessed. For example, to index through successive characters in an alphanumeric field, the index register must be incremented by two since each alphanumeric character occupies two machine locations.

INDIRECT ADDRESSING.

When the address controller in an address field is a binary 3, it specifies an indirect address; i.e., the field pointed to by the address contains, not data to be accessed, but another 6-digit address field which functions as though it had been in the original instruction. Such an address may in turn specify another indirect address, and so on to any depth. The controller of the final (direct) address specifies the format of the operand field to be accessed and must conform to any address controller restrictions for the instruction.

Full generality of indexing is maintained in indirect addressing. Any or all of the indirect addresses in a chain may be indexed. As a practical matter, there are three index registers available so that no more than three different index quantities can apply to any one instruction. However, any or all three can be applied any number of times.

LITERAL OPERATIONS.

The A ADDRESS field of most machine instructions can be used for literals instead of addresses. That is, the actual data to be used in the instruction can be stored in the A ADDRESS field, rather than the address of the data. This option is indicated by an 8-bit and a 2-bit being ON in the high-order four bits of the AF field. The address controller bits must be relocated since the A ADDRESS field contains actual data when a literal is used. The high-order bit of the address controller is represented by the 1-bit of the high-

order digit of the AF field. The low-order bit of the address controller is represented by the 8-bit of the low-order digit of the AF field. The length of the literal is indicated by the 1, 2, and 4-bits of the low-order digit of the AF field in normal binary-coded decimal. The 4-bit of the high-order digit of the AF field must be OFF. Literal data and indirect field length are mutually exclusive.

Example:

AF field = AD_{16} = 101 01 101_2 = 515

The first bit group signals literal operation, the second group an address controller of 01 (signed numeric format), and the third group reflects a length of five digits.

The maximum lengths that can be specified in literals are:

a. Unsigned numeric - six digits.
b. Signed numeric - five digits and sign.
c. Alphanumeric - three characters.

Literal capability can be used in the following instructions:

a. Move Alphanumeric (MVA).
b. Move Numeric (MVN).
c. Move Repeat (MVR).
d. Move Links (MVL).
e. All arithmetic instructions (including floating-point).
f. Compare Alphanumeric (CPA).
g. Compare Numeric (CPN).
h. Logical AND (AND).
i. Logical OR (ORR).
j. Logical NOT (NOT).
k. All Scan instructions.

FIELD LENGTHS.

When arithmetic instructions have a third address, its field length is some combination of the AF and BF values, depending on the

command, since there is no place in the instruction format to contain an explicit C field length. The maximum length of 100 characters/digits is specififed by 00. When two fields in an arithmetic instruction (except multiply and divide) have different field lengths, the shorter of the two is automatically padded with leading zeros in the processor. This padding does not affect stored values.

PROGRAM RESERVED MEMORY.

Base relative locations 00000 through 00063 of each program contain several fixed-location fields used to control the execution of certain instructions and store the results from other instructions. Assignment of these locations is as follows:

Location	Assignment
00000-00007	Unassigned (sometimes used for run-time parameters).
00008-00015	Index Register 1 (used by the SEA instruction).
00016-00023	Index Register 2 (used by MCP "no-work-area" buffer access technique - see FILE, READ, WRIT).
00024-00031	Index Register 3 (used by the NTR and EXT instructions).
	also:
00000-00037	Allowable addresses for indirect-field-length 2-digit counters.
00038-00039	Count storage from Scan instructions (SDE, SDU, SZE, and SZU).
00040-00045	Top-of-stack address for subroutine linkage (NTR and EXT).
00046-00047	Halt breakpoint digit (HBK).
00048-00063	8-character table of edit insertion characters, referred to as table entries 0-7 (EDT).

These fields can be accessed in a program the same as any other core location, and may be used as working storage if such use will not conflict with implicit usage of the area by hardware instructions.

BASE AND LIMIT REGISTERS.

Relocation and memory protection are accomplished on the B 2500/B 3500 Systems by means of the hardware base and limit registers. The values in these registers define the lower and upper bounds of the core memory assigned to a program.

An object program reference (during operation) to a main memory location uses a base-relative address. The hardware will add the indicated address to the contents of the base register to obtain an absolute core address. A program can therefore be loaded into any location in memory and executed without modification of addresses within the program. It can also be "pushed down" to a different location without modification to allow the MCP to bring other programs into the mix when a job has been completed, thus gaining optimum use of core.

For further explanation of the base and limit register functions, refer to the B 2500/B 3500 Systems Reference Manual.

EDIT OPERATORS.

Editing is accomplished on B 2500/B 3500 Systems by means of a highly versatile and compact "edit control" technique. The edit masks which are referenced (but not changed) by EDT commands and used to control data movement from the source field to the edited field are made up of strings of 1-character (double-digit) edit insertion operators.

Three machine flip-flops affect the execution of an EDT instruction:

a. T - initially ZERO and set to ONE when zero-suppression ends (by mask specification or by occurrence of a non-zero source digit).

b. Q - initially ZERO and can be set to ONE to obtain check-protection editing.

c. S - set to ZERO if the A field is negative and to ONE if the A field is zero, positive, or unsigned (UN).

The first (zone) digit of each edit-operator character is the primary operation specifier (called M). The second (numeric) digit (called V) specifies a count for repetition of the operation, an entry number for access to the Edit Insertion Table in Program Reserved Memory, or an extension of the operation specifier.

The M operation specifiers are:

a. 0 - Move Digit. Sets T equal to ONE, moves a digit/character from the source field to the edited field, and sets the zone in the edited character to the EBCDIC/USASCII numeric subset zone. The V digit specifies repetition (see page 2-10).

b. 1 - Move Character. Sets T equal to ONE and moves a digit/character from the source field to the edited field. Characters are moved unchanged. Digits receive the EBCDIC/USASCII numeric subset zone in the edited field. The V digit specifies repetition.

c. 2 - Move Suppress. If T is equal to ONE or ZERO and the source digit (or numeric portion of the source character) is non-zero, the Move Digit operation is performed. If T is equal to ZERO, the source digit (or numeric portion of the source character) is ZERO, and:

   1) Q equals ZERO - a blank is moved to the edited field.

   2) Q equals ONE - table entry 2 (usually *) is moved to the edited table.

   The V digit specifies repetition.

d. 3 - Insert Character. Moves the character specified by the V digit of the edit operator to the edited field. The V digit specifies Table Entry (see page 2-10).

e. 4 - Insert on Plus. If S equals ONE, the character specified by the V digit of the edit operator is moved to the edited field. If S and Q both equal ZERO, a blank is moved to the edited field. If S equals ZERO and Q equals ONE, table entry 2 (usually *) is moved to the edited field. The V digit specifies table entry.

f. 5 - Insert on Minus. If S equals ZERO, the character specified by the V digit of the edit operator is moved to the edited field. If S equals ONE and Q equals ZERO, a blank is moved to the edited field. If S and Q both equal ONE, table entry 2 (usually *) is moved to the edited field. The V digit specifies table entry.

g. 6 - Insert Suppress. If T equals ONE, the character specified by the V digit of the edit operator is moved to the edited field. If T and Q both equal ZERO, a blank is moved to the edited field. If T equals ZERO and Q equals ONE, table entry 2 (usually *) is moved to the edited field. The V digit specifies table entry.

h. 7 - Insert Float. If T equals ONE, the move digit operation is performed. If T equals ZERO and the source digit (or numeric portion of the source character) is non-zero, the character specified by the V digit of the edit operator is moved to the edited field, then the move digit operation is performed. If T equals ZERO, and the source digit (or numeric portion of the source character) is ZERO, and:

1) Q equals ZERO - a blank is moved to the edited field.

2) Q equals ONE - table entry 2 (usually *) is moved to the edited field.

The V digit specifies table entry.

i. 8 - End Float Mode. If T equals ZERO, the character specified by the V digit of the edit operator is moved to the edited field. If T equals ONE, no action occurs and thus proceed to the next edit operator.

j. 9 - Special Operations. If V equals 0, T is set to ZERO. If V equals 1, T is set to ONE. If V equals 2, the setting of Q is reversed (e.g., if Q equals ZERO, it is set to ONE and if it equals ONE, it is set to ZERO). If V equals 3, skip over the next source-field digit/character.

The values of the V digit of the edit operator have two sets of meanings:

a. Repetition. Repeat the edit operation V times. For example, example, the edit operator 05 (move digit, repeat five times) will cause <u>six</u> digits to be moved.

b. Table Entry. The values of V between zero and seven <u>select</u> entries 0 through 7 of the Edit Insertion Table in Program Reserved Memory. The values between 8 and 11 (undigit) specify conditional or special character selects. These selects are as follows:

1) V equals 0 - select entry 0 (usually +).

2) V equals 1 - select entry 1 (usually -).

3) V equals 2 - select entry 2 (usually *).

4) V equals 3 - select entry 3 (usually .).

5) V equals 4 - select entry 4 (usually ,).

6) V equals 5 - select entry 5 (usually $).

7) V equals 6 - select entry 6 (usually 0).

8) V equals 7 - select entry 7 (usually blank).

9) V equals 8 - if S equals ONE, select table entry 0 and if S equals ZERO, select table entry 1 (+ if positive and - if negative).

10) V equals 9 - if S equals ONE, select blank character and if S equals ZERO, select table entry 1 (blank if positive and - if negative).

11) V equals A (undigit value) - if S equals ZERO, select blank character and if S equals ONE, select table entry 0 (+ if positive and blank if negative).

12) V equals B (undigit value) - select the character which immediately follows the edit operator in the mask as an insertion character in place of a table entry. This character will be properly skipped in the mask after the insertion operation has been performed.

NOTE

When operating with the Basic Assembler or if not using the PICT command, the user program must load the Edit Insertion Table. The Advanced Assembler will load the standard values into the Edit Insertion Table if PICT declaratives appear in a source program.

EDIT EXAMPLES.

The following are examples of edit mask construction. The editing desired is described either verbally or as COBOL pictures.

VERBAL DESCRIPTION.

To edit a field containing 0000500010 to form 5-digit fields with a blank between the fields, zero-suppressed up to, but not including the units position of each field, the result should be:

bbbb5bbbb10 (b equals blank)

The edit mask for the operation above is:

a. 23 - move suppress four digits.
b. 00 - move digit once.
c. 90 - reset T equal to ZERO to resume zero suppression.
d. 37 - insert table entry 7 (usually blank).
e. 23 - move suppress four digits.
f. 00 - move digit once.

To edit +000375 to become bbb3.75+ and -000007 to become bbbb.07-, the edit mask is:

a. 23 - move suppress four digits.
b. 33 - insert character period (.).
c. 01 - move digit twice.
d. 38 - insert character (+ on positive and - on negative).

NOTE

The corresponding COBOL PICTURE for the above example would be ZZZZ.99+.

COBOL PICTURES.

The following are COBOL editing pictures and their corresponding machine language edit masks.

The COBOL picture Z(6) would edit 000325 to bbb325, and 000000 to bbbbbb. The edit mask is 25 (move suppress six digits).

The COBOL picture $ZZ,ZZ9.99 would edit 0013775 to $bbb137.75 and 0000050 to $bbbbb0.50. The edit mask is:

a. 35 - insert character $.
b. 21 - move suppress two digits.
c. 64 - insert suppress comma (,).
d. 21 - move suppress two digits.
e. 00 - move digit once.
f. 33 - insert character period (.).
g. 01 - move digit twice.

The COBOL picture $**,**9.99 would edit 0013775 to $***137.75 and 0000050 to $*****0.50. The edit mask is:

a. 35 - insert character $.
b. 92 - turn on Q to obtain * fill instead of blank fill.
c. 21 - move suppress two digits.
d. 64 - insert suppress comma (,).
e. 21 - move suppress two digits.
f. 00 - move digit once.
g. 33 - insert character period (.).
h. 01 - move digit twice.

The COBOL picture $$$,$$9.99 would edit 0013775 to bbb$137.75 and 0000050 to bbbbb$0.50. The edit mask is:

a. 75 - insert float $ (COBOL PICTURE specifies one extra position).

b. 75 - insert float $.

c. 64 - insert suppress comma (,).

d. 75 - insert float $.

e. 75 - insert float $.

f. 85 - end float $ to force out $ if it has not yet been inserted.

g. 00 - move digit once.

h. 33 - insert character period (.).

i. 01 - move digit twice.

Machine language edit masks should be coded as strings of numeric digits and undigits in a CNST statement, then associated with a UA label and length by means of the EQUIV operation. For example, the last example above would be coded as shown in figure 2-7.

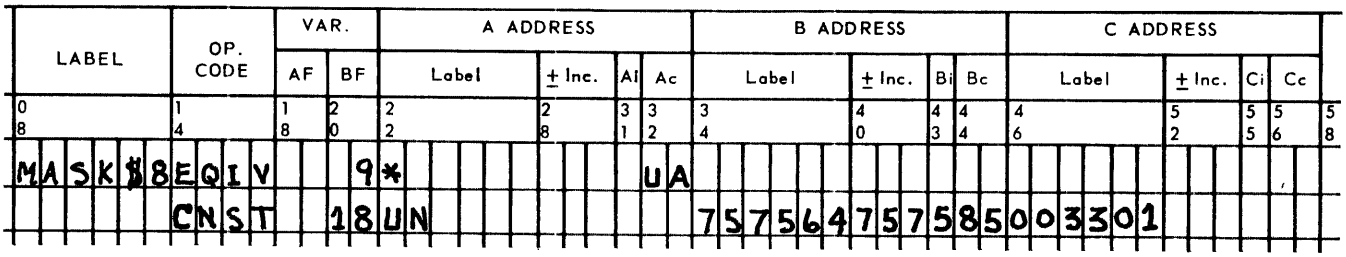


Figure 2-7. Coding for Machine Language Edit Masks

INDIRECT FIELD LENGTH.

Indirect field length is indicated when the two high-order bits of the AF or BF variant length field are ON. The processor considers the low-order six bits of the AF or BF field (grouped 2-bits and 4-bits) to be the two low-order digits of a 5-digit address (high-order digits equal 000) where the 2-digit length (or a similarly-formated pointer for another indirect-length field) is to be found. This address must be an even number between 00000 and 00038 (within the Program Reserved Memory Area).

Example:

BF field = $D8_{16}$ = 11 01 1000_2 = 318

The first bit group (binary 3) signals indirect field length, and the next two groups indicate memory address 00018 which points to a location in the Program Reserved Memory area.

NOTE

Some instructions (MVW, MVC, TRN, etc.) interpret the variant field as a 4-digit length value. Indirect field lengths are still accessed on a 2-digit basis, however, and the resultant actual field lengths are then put together to form the 4-digit length.

SECTION 3
ASSEMBLER CODING FORM

GENERAL.

This section describes the assembler coding form (Form 1024627, figure 3-1) used with the assembler.

The form is divided into eight general fields: Sequence Number, Label, Operation Code, Variants, three Address Fields (A, B, and C), and Remarks. The three Address Fields are further divided into: the Address Label, Address Increment, Address Index, and Address Controller. The following general rules apply.

SEQUENCE NUMBER (COLUMNS 2-7).

Sequence numbers are used for the sequential numbering of the symbolic input.

LABEL (COLUMNS 8-13).

This field is used for entering symbolic names and program points, left-justified. Normal label names must start with an alphabetic character and may be up to six characters in length. The second through sixth character may be any combination of alphabetic, numeric, or special characters. Labels are reuseable, but not within a given segment.

Program points are normal label names of up to five characters in length (for the Basic Assembler, only one alphanumeric character in length) preceded by a period (.). These program points have the property of unrestricted reusability and need not be unique. They are usually, but not necessarily, used for short distance references (definition and reference less than 20 statements apart).

Every label and program point definition is associated with:

a. Corresponding base-relative address.
b. Length of the field named by the label (normal labels only).
c. Data format of the field (normal labels only).
d. Segment.

Program Id. _____ **Burroughs** ASSEMBLER CODING FORM Page _____ of _____

SEQ. NO.	LABEL	OP. CODE	VAR.		A ADDRESS			B ADDRESS				C ADDRESS				REMARKS
			AF	BF	Label	± Inc.	Ai Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 2	0 8	1 4	1 8	2 0	2 2	2 8	3 1 3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 6 6 7 8 8 6 5 0

Figure 3-1. Assembler Coding Form

The following labels are reserved and can be referred to, but not reused:

Label	Location	Description
BASE	00000-undefined length	4-bit unsigned numeric format.
IX1	00008-00015	4-bit signed numeric format (refered to as Index Register 1).
IX2	00016-00023	4-bit signed numeric format (refered to as Index Register 2).
IX3	00024-00031	4-bit signed numeric format (refered to as Index Register 3).
IOCxxx		BCP translator only. 6-character labels starting with IOC are used in the SIOC input-output package and should be avoided by the programmer.

OP. CODE (COLUMNS 14-17).

This field is used for entering the particular mnemonic instruction to be assembled, macro to be expanded, or assembler control function. All 3-character mnemonic operation codes will produce a single instruction, while the number of instructions generated by a macro operation depends upon the particular macro specified and the operating software.

VARIANT FIELD (COLUMNS 18-21).

This field is used for specifying variants to the basic instructions and/or additional specifications for operations. It may be left blank when label-associated lengths are correct for the instruction. The AF length must always be specified for literals (other than label literals).

ADDRESS LABEL FIELDS A, B, AND C (COLUMNS 22-27, 34-39, AND 46-51).

These fields are used to specify operands to be associated with given instructions. Entries to these fields must be left-justified

and can be any of the following forms:

a. A name which is defined in the label field of some statement, or a reserved label.

b. A program point, with the period replaced by a plus (+) to denote forward reference (next definition) or a minus (-) to denote backward reference (last definition) to a program point defined in the label field of some statement.

c. A single asterisk to denote the address of the first digit of the current instruction.

d. An unsigned 5-digit base-relative address.

e. A literal, when acceptable by the instruction (A ADDRESS label only).

ADDRESS INCREMENT FIELDS A, B, AND C (COLUMNS 28-30, 40-42, AND 52-54).

These fields are used to specify positive or negative increments which will alter the value as defined in the address label field. The range of permissable increments is from -999 to +999. Plus signs need not be written and minus increments above -99 are coded with the letters J thru R to decrement from 100-900. Signs must be left-justified and the integer right-justified. This increment may not be in conjunction with a literal (except label literal).

The increment is understood as being in number of digits/characters rather than in machine locations. Thus, for an alphanumeric label, an increment value of +5 would increase the label-equivalent address by 10 rather than by 5 since each alphanumeric character occupies two machine locations.

ADDRESS INDEX FIELDS A, B, AND C (COLUMNS 31, 43, AND 55).

These fields are used to specify that the contents of a designated index register are to be added to the address given or implied in the address label field during the execution of the program. The acceptable codes are:

a. Blank - no index register.
b. 1 - Index Register 1.
c. 2 - Index Register 2.
d. 3 - Index Register 3.

NOTE

The address index field may not be used with a literal.

ADDRESS CONTROLLER FIELDS A, B, AND C (COLUMNS 32, 33, 44-45, AND 56-57).

These fields are used to specify the format of the data referred to by the operand in the address label field and need only be specified if it differs from that declared in the data field declarative, in which case it overrides the other. Only those address controllers permitted by the individual instructions are valid. If a certain address controller is legal, its corresponding literal also is legal unless explicitly excluded.

The address controller notations and their meanings are as follows:

a. Blank - as defined in the data declaration.
b. UN - unsigned numeric.
c. SN - signed numeric.
d. UA - unsigned alphanumeric.
e. IA - indirect address.
f. NL - unsigned numeric literal.
g. SL - signed numeric literal.
h. AL - alphanumeric literal.
i. LL - label literal.
j. SG - segment number literal.

} Legal only in the A ADDRESS controller

NOTE

LL (label literal) yields an unsigned numeric literal equal to the equivalent address of the referenced operand (label, program point, asterisk, or 5-digit address). The length of the field need not be given and will be assumed to be six digits if not specified in the variant field. If a field length less than

six is specified, the high-order digits of the label literal will be used. Index and controller bits associated with the label are not assembled into the literal. NL, SL, and AL must be used when a literal is the operand. AL literals may not contain untranslatable EBCDIC characters when the source program is maintained on a seven-track magnetic tape. SG yields a 3-digit unsigned numeric literal equal to the segment number of the referenced program segment. An asterisk (*) will indicate the segment in which the instruction is contained.

REMARKS FIELD (COLUMNS 58-80).

The remarks field may be used to furnish documentational remarks or a description of the operation being performed.

SECTION 4
MULTI-STATEMENT STRUCTURES AND CONTROL INFORMATION

GENERAL.

An entire Assembly language source program can be regarded as a coding structure made up of substructures. Just as there are divisions and sections in COBOL programs, and subroutines and functions in FORTRAN programs, there are coding structures in Assembly programs which are made up of several elementary statements. These structures are used to define program entries which cannot be described in a single elementary statement (e.g., record descriptions), or which by their nature consist of many elementary statements (e.g., program segments). These structures are described in this section.

PROGRAM STRUCTURE.

An Assembly source program is structured in roughly the same fashion as a COBOL program. The same sequence of identification, environment, data, and procedure divisions should be followed, though the Assembler is more flexible than COBOL in permissible statement sequences. The following statement sequence is recommended for source programs:

a. The first statements should be SPEC and IDNT.

b. FILE statements along with their associated RECD statements should follow.

c. Following the last FILE block, working-storage records, areas, and constants should be declared.

d. Executable instructions and macros follow the working-storage declarations.

Statement categories b through d above are repeated for each segment in a multi-segment program. Any category can be omitted if it is not needed.

LOWER-LEVEL STRUCTURES.

There are several lower levels of multi-statement structuring within a typical source program:

a. FILE declarations and their associated records (see below).

b. Data fields within a record (see below).

c. Segments within a program (see below).

d. Parameters in a subroutine-call instruction (refer to the NTR instruction).

FILE AND RECORD DECLARATION.

Figure 4-1 provides a skeleton example of the FILE declaration and working-storage portion of the symbolic input deck for the B 2500/B 3500 Systems Assemblers. The following are rules that should be followed when constructing this deck:

a. Record declarations following a FILE declaration are assembled into the file's work area (if there is one), or into location 00000 if the file has no work area.

b. If several record blocks follow a single FILE declaration, they are assembled into the same area, thus redefining each other.

c. Both Assemblers require the ENDR statement to terminate a working-storage record block.

d. The Basic Assembler requires and the Advanced Assembler permits an ENDR to terminate a file-associated record block.

e. Both Assemblers require the ENDF statement to terminate the <u>last</u> file block.

f. The Basic Assembler requires and the Advanced Assembler permits an ENDF to terminate file blocks <u>other</u> than the last.

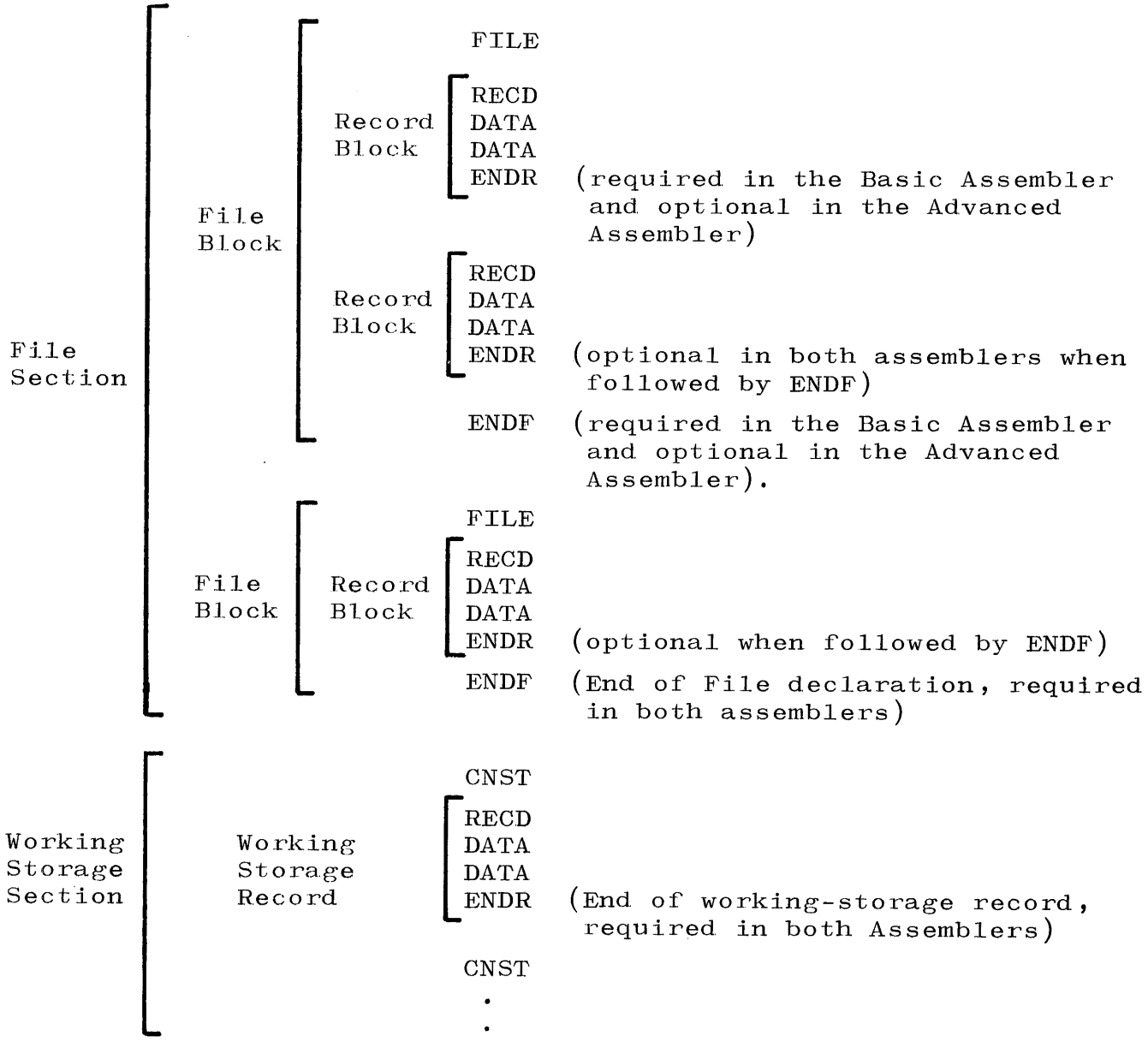


Figure 4-1. FILE Declaration and Working-Storage Portion of Symbolic Deck

g. Only the DATA statement may be used to define fields within a file-associated record.

h. Both Assemblers count the number of characters defined by DATA statements within a record, and give a syntax error if the total does not equal the length declared in the RECD statement.

PROGRAM SEGMENTATION.

Segmentation is required when a program and its data areas are too large for available core memory. It is also desirable when a program consists of several sequentially-executed parts (e.g., housekeeping, main processing routine, and recapping) of long duration. Segmentation consists of separating the program into mutually exclusive parts, with no data or control cross-references, at as many levels as necessary. These parts are then declared as overlayable segments and share common core storage.

Segments are declared by means of the SEGM and ENSG pseudo operations. Refer to the description of these pseudos for the restrictions on segment coding.

Segment loading and inter-segment communication is accomplished by the OVLY pseudo operation. The Assemblers perform extensive checks on the validity of inter-segment references.

The total core memory required for a segmented program is equal to the length of the main (permanently-resident) segment plus the length of the largest chain of nested subsegments. The Assemblers will automatically compute the memory requirement.

Figure 4-2 provides an example of a segmented program in terms of Assembly coding and memory allocation at object time.

Figure 4-3 shows the amount of core memory required for the example segmented program in figure 4-2.

```
          IDNT  PRØGRAM  (5000 digits)
            .
            .
SEGA        SEGM  (2000 digits)
            .
            .
EXAMIN      SEGM  (2000 digits)
            .
            .
SEGZ        SEGM  (3000 digits)
            .
            .
          ENSG
CHECK       SEGM  (1000 digits)
            .
            .
          ENSG
          ENSG
UPDAT       SEGM  (2000 digits)
            .
            .
CLOSE       SEGM  (1000 digits)
            .
            .
          ENSG
OPEN        SEGM  (1000 digits)
            .
            .
          ENSG
COPY        SEGM  (1000 digits)
            .
            .
REJECT      SEGM  (1000 digits)
            .
            .
          ENSG
NEW#        SEGM  (1000 digits)
            .
            .
          ENSG
          ENSG
          ENSG
          ENSG
REPORT      SEGM  (4000 digits)
            .
            .
          ENSG
          FINI
```

Figure 4-2. Example of a Segmented Program

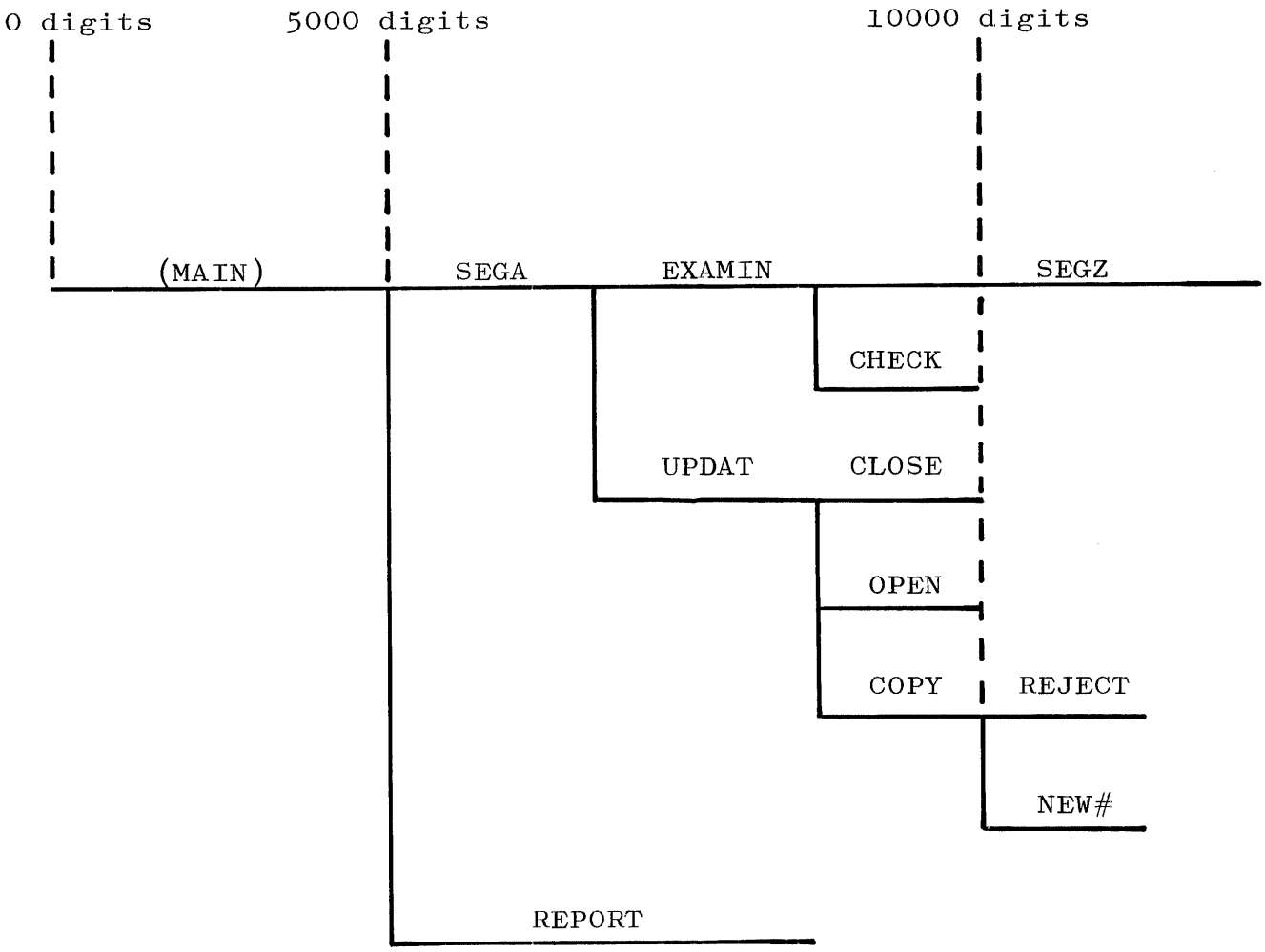


Figure 4-3. Core Memory Requirements for Figure 4-2

CONTROL INFORMATION FOR BASIC AND ADVANCED ASSEMBLER.

The following Program Control Cards are necessary for assembly of a source program by the Advanced Assembler.

The first input control card notifies the MCP to assemble the indicated Program-Name (P-N) using one of the following options:

a. ?COMPILE P-N WITH ASMBLR (compile and run object program).

b. ?COMPILE P-N WITH ASMBLR LIBRARY (object program left on disk and entered in disk directory).

c. ?COMPILE P-N WITH ASMBLR SAVE (combines features of compile and go and compile for library).

d. ?COMPILE P-N WITH ASMBLR SYNTAX (for syntax check only).

e. ?COMPILE P-N WITH ASMBLR AFTER JOB (wait to compile until completion of execution of another program).

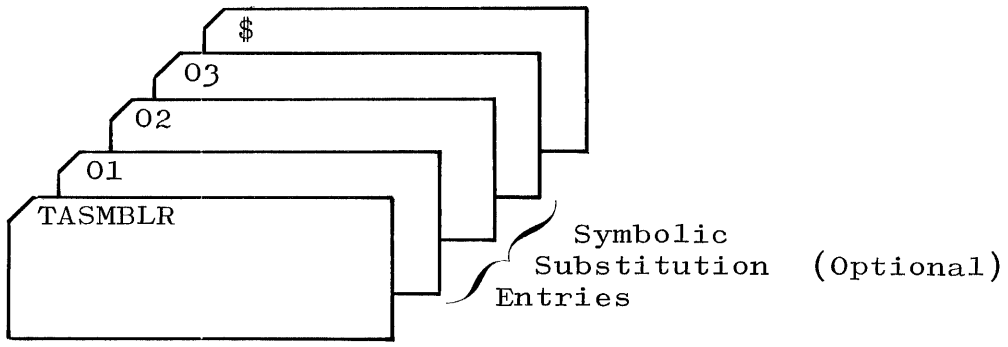
The second control card is the data card which provides file identification. It is formatted as follows:

a. ?DATA CARDS (for EBCDIC source language input).
b. ?DATAB CARDS (for BCL source language input).

The End Card must be the last card in the source deck. Its format is:

?END

The Basic Assembler requires the following control cards for assembly of a source program:



The first card specifies the input unit where ASMBLR is to be found. A "T" (as in the above example) indicates magnetic tape. A "C" would indicate the control unit, i.e., card reader or paper tape reader.

The symbolic substitution entries are optional and can be loaded in place of, or in addition to, the symbolic entries loaded at BCP load time. See table 4-1 for symbolic substitution entries.

Table 4-1

Symbolic Substitution Entries

Symbolic Entry No.		Device	Recommended Unit No.	Card Format
00		Systems tape	1	00CC029140
01	or	Card reader	-	01CC220000
01	or	Card reader	-	01CC200000
02		Printer	0	02CC102000
03	or	Card punch	-	03CC230000
03	or	Card punch	-	03CC250000
04		Output tape	2	04CC069240
05		Output tape	3	05CC069340
06		Output tape	4	06CC069440
07		Input tape	5	07CC029540
08		Output tape	6	08CC069640
09		Paper tape reader	-	09CC4000V0*
10		Paper tape punch	-	10CC4800V0*

CC = Channel Number

*V = 1 BCL to EBCDIC translation; V = 2 EBCDIC

The card format for the symbolic substitution is:

COLUMN	
1-2	Symbolic Reference Number
3-4	Channel Number
5-10	First Syllable of I/O Descriptor

The $ card signifies the end of the control cards. The symbolic input, beginning with a SPEC card, immediately follows the $ card.

The last card of the symbolic deck must be an END card in the following format:

?bEND	where b = blank

SECTION 5
BASIC ASSEMBLER INPUT/OUTPUT CODING

GENERAL.

The Basic Assembler differs in several ways from the Advanced Assembler due to the differences in the operating programs. Whereas the Master Control Program will provide many functions to the user's program such as blocking, label checking, and comprehensive error recovery procedures, none of these functions are provided in the Basic Control Program. The programmer, therefore, must understand the logic of these functions and provide this logic in his program if he requires these functions. The purpose of this section is to outline the areas of consideration that the programmer must know in order to write programs for use with the Basic Control Program.

It is also necessary for the programmer to fully understand the functions of the hardware in order to implement the logic desired. Appendix C contains a summary of the I/O and result descriptors plus other subject matter that must be fully understood by the user. A more detailed explanation of descriptors can be found in the Burroughs B 2500/B 3500 Systems Reference Manual.

BASIC ASSEMBLER AND ADVANCED ASSEMBLER DIFFERENCES.

The differences between the two languages of this text are primarily in the area of the input and output instructions. The following are differences between the two:

a. The following mnemonics are not available in the Basic Assembler:

MACR	MEXT	RITE	PCKT	SGNM
MEND	BOOL	WCRC	SPAS	ADSL
MEQL	SETB	WTRC	SKIP	SEAE
MLSS	CLRB	WCRT	SLEW	SEAL
MGTR	SYST	FILL	FLOW	SLST
MLEQ	SEEK	LSTR	SRTR	REDY
MGEQ	OVLY	WAIT	ABLE	ACPR
MNEQ	USER	INTR	CWNT	DISR
MBUN	DATE	UNCL	LGHT	INTA
MNOP	PICT	CNCL	CORE	SEND
MERR	REED	ENBL	DOZE	RECV

b. With the assumption that the SIOC macro is used, the following mnemonics are available in the Basic Assembler language:

FILE	ENDF	READ
ACPT	OPEN	WRIT
DISP	CLOS	POSN

c. The following mnemonics are not available in the Advanced Assembler:

INIT	SIOC
TIOC	RUNN
INER	VALU
IOCU	

BASIC ASSEMBLER INPUT/OUTPUT INSTRUCTIONS.

The Basic Assembler has two methods of initiating input/output commands to the BCP. One method, SIOC, allows the program to be upward compatible with the advanced software. This method covers only the magnetic tape, line printer, paper tape, and card units. The second method will allow the program to initiate any device available to the B 2500/B 3500 hardware. The two methods are outlined below.

USE OF SPECIAL I/O CONTROL (SIOC) ROUTINES FOR BASIC ASSEMBLER.

Unlike the MCP, the BCP has to be provided with the hardware environment through the loading of a machine language program. This environment can change from time to time, therefore, the BCP contains a Symbolic Substitution Table which each program can access to provide the current descriptor to itself for later execution. The basic software provides this information to an object program through the OPEN statement. The OPEN statement contains a symbolic substitution number that refers to the Symbolic Substitution Table provided in the BCP. This table has a symbolic number that has an associated channel number and I/O descriptor for a specific unit on the system. By referencing the symbolic number in the OPEN statement, the channel number and I/O descriptor are automatically

provided to the user's program. The Symbolic Substitution Table needs to be loaded only once with each cold start, or it may be reloaded with the initialization of each program. It should be noted that if both options are used, the OPEN statement will override the I/O descriptor in the FILE statement.

SIOC ROUTINES.

The SIOC pseudo Op Code (refer to section 6) calls out a standard input/output control package and assembles it into the object program. The routine occupies approximately 1500 bytes of storage. This pseudo allows the use of the following Advanced Assembler constructs:

FILE	CLOS	POSN
ENDF	READ	
OPEN	WRIT	

For Basic Assembler variations, refer to these constructs in section 6 of this manual.

The above constructs can be used for the following peripheral devices:

a. Magnetic tape.
b. Card reader.
c. Card punch.
d. Line printer.
e. Paper tape reader.
f. Paper tape punch.

All other peripheral devices will have to use the INIT and INER options.

FILE CONSTRUCT.

The FILE construct provides the following outline for a file:

a. File Information Block - 72 digits in length.
b. File Label - 80 bytes in length.
c. File Buffer - defined size.

d. Delimiter - two digits in length.

e. File Work Area - defined size.

Table 5-1 shows the component fields of the File Information Block.

TABLE 5-1

Component Fields of the File Information Block

Label	Length	Function
IOCNXT	Six digits	Next record address
IOCWRK	Six digits	Work area address
IOCSIZ	Four digits	Record size in words
IOCIO	One digit	I/O flag
IOCDIG	Five digits	Record size in digits
IOCERR	Six digits	Use routine tape parity
IOCLAB	Six digits	Standard label
IOCAEA	Six digits	Actual end address
IOCCHN	Two digits	Channel
IOCRD	Four digits	Result descriptor
IOCOP	Two digits	Op code
IOCYAR	Four digits	Variants
IOCBEG	Six digits	Begin address
IOCEND	Six digits	End address
IOCDFA	Eight digits	Reserved

Table 5-2 shows the component fields of an External File Label.

LABEL HANDLING.

This routine will verify correct values in the first several fields of the standard label format. These include label identifier, multi-file ID, and file ID.

Non-standard labels will have to be handled by the user. This can be accomplished in two ways:

TABLE 5-2

Component Fields of External File Label

Position	Value or Function
1-8	bLABELbb
9	0
10-16	Multi-file ID or zeros
18-23	Identifier
24	0
25-27	Reel number
28-32	Date written
33-34	Cycle
35-39	Purge date
40	Sentinel (zero equals End-of-File and one equals End-of-Reel)
41-45	Block count (ending label)
46-52	Record count (ending label)
53	Memory-dump-key, beginning label (one means memory dump follows)
54-58	Physical-reel-number
59-80	User's portion

a. Alter the label area in core programmatically before the OPEN statement for that file.

b. Ignore the halt-breakpoint error on the console or error message on the supervisory printer (SPO).

Multi-file tape reels will have to be programed by the user.

The following are the formats of the two types of labeled tape reels:

Standard Labeled Reels	Multi-File Labeled Reels
Begin Label	Begin Label
EOF Mark	EOF Mark
Data File	Data File
EOF Mark	EOF Mark
End Label	End Label
	Begin Label
	Etc.

BLOCKING AND VARIABLE SIZE RECORDS.

The blocking and unblocking of files will have to be handled by this routine. Variable size records and variable size records blocked will have to be handled by the user.

SIOC ERROR RECOVERY PROCEDURES.

MAGNETIC TAPE READ. This procedure will attempt to read the record ten times and then indicate an I/O error.

CARD READ AND PUNCH. This procedure will check for an End-of-File condition. If not at End-of-File condition, an I/O error is indicated.

LINE PRINTER. This procedure will test for an End-of-Page condition. If End-of-Page condition, it will branch to the user's routine. If not, it will perform a skip to channel one. If there is some condition other than End-of-Page, it will ignore the error.

USE OF INIT, INER, IOCU, TIOC, AND RTRN CONSTRUCTS.

When working with a limited memory system or when using types of peripherals other than those specified in the SIOC macro, the user will have to construct his own file area. In this case, he will have to provide the following:

a. Memory allocation for each file (including buffers).
b. Constant areas for each channel number.
c. His own error procedure with each channel.

An example of this is shown in figure 5-2. The only requirement to be concerned with when designing this file area is that a copy of the result descriptor is placed in the program by the BCP, four digits in front of the I/O descriptor. The rest of the file requirements may be designed at the discretion of the programmer.

A thorough knowledge of the I/O descriptors and their associated result descriptors is required. These are provided in Appendix C of this manual in summary form. For a more detailed explanation of result descriptor meanings, refer to the Burroughs B 2500/B 3500 Systems Reference Manual.

USER ROUTINES.

The design of an exception routine will take two general formats. These are the INIT and TIOC commands and the INER command. These are governed by the type of option the user requests in his instructions. These options are as follows:

a. Where an INIT or TIOC command is used, a simple branch to an ERROR label will occur and no de-link of the channel will take place.

b. When the user issues an INER command and column 19 (AF) of the statement equals 0-4, the BCP will send 16 digits of information to the USE routine when an exception condition exists. These are as follows:

1) Result descriptor address - six digits.

2) Channel number - two digits.

3) End address code - one digit (code 1 means "ending address returned" and code zero means "no ending address requested").

4) Error handling code - one digit (code one means "no error handling requested" and code zero means "error handling requested").

5) End address location - six digits.

Coding at the start of an exception routine should look like this:

```
ENTRY LABEL     BUN             +A
                CNST    16      UA      I/O COMPLETION INFO
.A              XXX
```

When an INER is used and an exception condition does arise, the BCP will pass control to the use routine. The BCP can control but one exception condition at a time. The BCP allows all I/O descriptors to finish and will service new ones on the error channel only. A RTRN or a pocket select descriptor use routine clears this condition in the BCP. When a RTRN or a pocket select descriptor is initiated, control is returned to the current program point.

An INER statement that has a 5-9 in the AF field (column 19) will have the BCP branch to the USE routine on any I/O complete. This condition is used for I/O devices where timing is critical (e.g., sorter/readers). The 16 digits of BCP information is not transferred as in option b above.

Figures 5-1 and 5-2 provide samples of I/O coding under the Basic Control Program (BCP).

Program Id. _____ **Burroughs** ASSEMBLER CODING FORM Page 1 of 1

BASIC ASSEMBLER

SEQ. NO.	LABEL	OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ADDRESS ± Inc.	A ADDRESS Ai	A ADDRESS Ac	B ADDRESS Label	B ADDRESS ± Inc.	B ADDRESS Bi	B ADDRESS Bc	C ADDRESS Label	C ADDRESS ± Inc.	C ADDRESS Ci	C ADDRESS Cc	REMARKS
		SPEC			CARD					TAPE							LIST
		SIOC															
	CARD	FILE	0080										Ø				1 BUFFER, UNLABELED
	CWA	REC	D0080	UA													CARD WORK AREA
		ENDR															
		ENDF															
	PRINT	FILE	0132	PRINT									S				1 BUFFER, LABELED
	PWA	REC	D0132	UA													PRINT WORK AREA
		ENDR															
		ENDF															
	STACK	DATA	0200	UN													
		MVN	0606	STACK				LL	00040								INITIALIZE STACK
		OPENIN		CARD					01								DESC, CHAN FROM ENTRY #1
		OPENOT		PRINT					02								DESC, CHAN FROM ENTRY #2
		SPRD	4040	PWA													BLANK PRINTER WORK AREA
	X	READ		CARD					END								
		MVW	40	CWA					PWA								CARD TO PRINT WORK AREA
		WRIT	01	PRINT													
		BUN		-X													GO TO GET NEXT CARD
	END	CLOS		PRINT													PRINT ENDING LABEL
		STOP															END OF JOB
		FINI															
END																	

CARD TO PRINT NO 1

Figure 5-1. Sample 1 of BCP I/O Coding

Program Id. _____ **Burroughs** ASSEMBLER CODING FORM BASIC ASSEMBLER Page 1 of 3

SEQ. NO.	LABEL	OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ADDRESS ± Inc.	A ADDRESS Ai	A ADDRESS Ac	B ADDRESS Label	B ADDRESS ± Inc.	B ADDRESS Bi	B ADDRESS Bc	C ADDRESS Label	C ADDRESS ± Inc.	C ADDRESS Ci	C ADDRESS Cc	REMARKS CARD TO PRINT NO 2
		SPEC			CARD					TAPE							LIST
		LOCN			00046												
		CNST		2	UN				FF								ALLOW HAILTS
					CARD FIB AND BUFFER												
	C-CHAN	DATA		2	UN												CHANNEL
	C-RD	DATA		4	UN												RESULT DESCRIPTOR
	CDESC	DATA		6	UN												I/O DESCRIPTOR
		ACON			C-BUFF												BEGIN ADDRESS
		ACON			C-BUFF 80												END ADR
	CWA	RECD	80	U	A												CARD WORK AREA
		ENDR															
	C-BUFF	RECD	80	U	A												CARD BUFFER
		ENDR															
					PRINTER FIB AND BUFFER												
	P-CHAN	DATA		2	UN												CHANNEL
	P-RD	CNST		4	UN				8000								RESULT DESCRIPTOR
	PDESC	DATA		6	UN												I/O DESCRIPTOR
		ACON			P-BUFF												BEGIN ADR
	PWA	RECD	132	U	A												PRINT WORK AREA
		ENDR															
	P-BUFF	RECD	132	U	A												PRINT BUFFER
		ENDR															
	PGRD	DATA		4	UN												PG OUT RESULT DESC
	PGDESC	CNST		6	UN				110001								SKIP TO CHANNEL 1

Figure 5-2. Sample 2 of BCP I/O Coding (Sheet 1 of 3)

Program Id. _____ **Burroughs** ASSEMBLER CODING FORM Page 2 of 3

BASIC ASSEMBLER

SEQ. NO.	LABEL	OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ± Inc.	A Ai	A Ac	B ADDRESS Label	B ± Inc.	B Bi	B Bc	C ADDRESS Label	C ± Inc.	C Ci	C Cc	REMARKS CARD TO PRINT NO 2
					***SET UP STACK FOR NTR OP ***												
	STACK	DATA	16		UN												
		MVN	06	06	STACK				LL00040								INITIALIZE STACK
					***EQUIV TO OPEN STATEMENT ***												
		IOCU	01		CDESC				C-CHAN				BASE				INIT. CARDS FROM ENT #1
		IOCU	02		PDESC				P-CHAN				BASE				INIT. PRINT FROM ENT #2
		MVN	01	01	PDESC	03			PGDESC 03								SET PAGE DESC UNIT
		INIT			PGDESC				P-CHAN								SET NEW PG
		TIOC			PGRD												
					***START OF PROGRAM LOGIC, INITIALIZATION ***												
		SPRD	40	40	PWA												BLANK PRINT WORK AREA
	OPEN	INIT			CDESC				C-CHAN								
		TIOC			C-RD				OP-ERR								
		BUN			+B												
					***CARD TO PRINT LOGIC ***												
	.A	TIOC			C-RD				C-ERR								CARD READ DONE. NO WAIT
	.B	MVW	00	40	C-BUFF				CWA								YES MOVE C BUFF TO WA
	READ	INIT			CDESC				C-CHAN								READ A CARD TO BUFFER
		MVW	00	40	CWA				PWA								MOVE CRD WA TO PRN WA
		TIOC			P-RD				P-ERR								PRINT WRITE DONE WAIT
	.C	MVW	00	66	PWA				P-BUFF								YES MOVE P WA TO P BUFF
	WRITE	INIT			PDESC				P-CHAN								WRITE PRN BUFF
		BUN			-A												LOOP BACK

Figure 5-2. Sample 2 of BCP I/O Coding (Sheet 2 of 3)

Program Id. _____ **Burroughs ASSEMBLER CODING FORM** Page 3 of 3

BASIC ASSEMBLER

SEQ. NO.	LABEL	OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ADDRESS ± Inc.	A ADDRESS Ai	A ADDRESS Ac	B ADDRESS Label	B ADDRESS ± Inc.	B ADDRESS Bi	B ADDRESS Bc	C ADDRESS Label	C ADDRESS ± Inc.	C ADDRESS Ci	C ADDRESS Cc	REMARKS
					***EXCEPTIØN SUB-RØUTINES AND STØP												***
	CRDERRMVN	0202			C-CHAN				C-HALT	02							
	MVN	0101			CDESC	03			C-HALT	04							
	C-HALTHBK	000F															
	EXT																
	ØP-ERRNTR				CRDERR												
	BUN				ØPEN												
	C-ERR CPA	0303			END				ALC-BUFF	02							
	EQL				STØP												
	NTR				CRDERR												
	INIT				CDESC				C-CHAN								
	BUN				-A												
	P-ERR BUN				-C												IGNØRE PRINT CHECKS (MAY DØ PAGING ØUT HERE, WHERE APPLICABLE).
	STØP STØP																
	FINI																

Figure 5-2. Sample 2 of BCP I/O Coding (Sheet 3 of 3)

SECTION 6
ASSEMBLY OPERATIONS CODES

GENERAL.

This section contains all of the operation codes used in the B 2500/B 3500 Assembler Language, with the following exceptions: Data Communications, MICR sorter-reader and lister. A description of these instructions and the Advanced Assembler macro facility are contained in separate sections. These operation codes are described in alphabetical sequence, and each description is presented in a standard format which is keyed to the assembly coding form. First is the symbolic operation code (i.e., machine code, macro, pseudo, or declarative) and then a general description of the instructions function. This is followed by a coding example of the operation code and then a description of LABEL entry restrictions and entries (if any). After the LABEL entry description, the conventions (if any) defining the length of the C field in a three-address instruction are defined. Next, the coding requirements (if any) for the A, B, and C ADDRESS fields along with their subfields are defined. Any changes to the comparison and OVERFLOW indicators and Program Reserved Memory are described next. This is then followed by any programming notes, semantic connections with other operations in a program, and finally operational examples of the operation code (only in some cases).

DECLARE ADDRESS CONSTANT (ACON) - DECLARATIVE.

The function of this declarative is to specify that the base-relative address corresponding to the specified label is to be assembled as a constant, with all associated address controller and index bits.

The format for the ACON instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
TABADRACØN				TABLE	+	6	1	U	A						

There are no LABEL restrictions when using this instruction.

No VAR length is specified. The assembled address constant has an implicit length of six digits and is an unsigned numeric constant.

Specified in the A ADDRESS field is the label whose base-relative address is to be assembled. Full generality of increment, index, and controller apply. If an address controller is associated with the label, it will be assembled into the address constant unless overridden. The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The ACON command may be used to specify a parameter following a NTR instruction within executable coding, or independently within working storage. The assembled address constant will be synchronized modulo-2.

ACCEPT SPO TYPE-IN (ACPT) - PSEUDO.

The function of this pseudo is to read an operator-supplied message from the Supervisory Printer.

The format of the ACPT instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
READIN	ACPT	20		PARAMS											

There are no LABEL restrictions when using this instruction.

The AF field specifies the number of characters to be read. If it is blank, the length associated with the A ADDRESS label is used.

The A ADDRESS field points to an alphanumeric area into which the typed-in message is copied. Indexing is not permitted, and the address controller must be UA. The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

Under control of the MCP, an ACPT causes the program to suspend execution and type a "waiting message" notification. The operator then responds with an AX text type-in, and the program resumes operation.

When using with BCP, SIOC assumes that the supervisory printer is on symbolic substitution entry 11.

THREE-ADDRESS ADD (ADD) - MACHINE CODE-02.

The function of this instruction is to perform a three-address addition by adding the contents of the A field to the B field and storing the result in the C field.

The format of this instruction is

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
.DA	ADD	2		+20				SLCØUNT1				CØUNT2			

There are no LABEL restrictions when using this instruction.

The AF field specifies the number of digits/characters in the A field and the BF field specifies the number of digits/characters in the B field. The C field length is assumed to be equal to the larger of the AF and BF values. If the number of significant digits in the sum is greater than C field length, the addition will not be performed. If the A and B field lengths are unequal, the shorter of the two is assumed to be left-zero-filled until the lengths are equal.

The A ADDRESS field points to the addend field and all address controllers are valid. UN fields are assumed to have a positive value. Only the numeric digits of a UA field will be accessed and the sign will be assumed positive.

The B ADDRESS field points to the augend field and the address controller conventions are the same as for the A ADDRESS.

The C ADDRESS field points to the field into which the sum is stored. If the C field is UN, the sign of the sum will be lost and the absolute value will be stored. If the C field is UA, the sign of the sum will be lost, the absolute value of the sum will be stored into

the numeric-digit portion of each character position, and the zone-digit portion will be set to the numeric-subset zone.

If the sum is too large to fit into the C-field, the OVERFLOW indicator is set and the comparison indicators remain unchanged; otherwise, the comparison indicators are set according to the sign of the sum (even if the sign is not stored).

These settings are as follows:

Setting	Sign
LOW	-
EQUAL	0
HIGH	+

Program Reserved Memory is not changed by this instruction.

Operational examples of the Three-Address Add instruction are as follows:

Unsigned Numeric To Signed Numeric Giving Signed Numeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
ADD	02	05	20(NL)	COUNT1(SN)	COUNT2(SN)

	BEFORE	AFTER
A ADDRESS	20	20
B ADDRESS	+00015	+00015
C ADDRESS	xxxxxx	+00035
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Unsigned Numeric To Signed Numeric Giving Unsigned Alpha-numeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
ADD	02	05	INCR(UN)	COUNT1(SN)	COUNT2(UA)

	BEFORE	AFTER
A ADDRESS	10	10
B ADDRESS	-00050	-00050
C ADDRESS	xxxxx	00040
COMP. INDC.	xxx	LOW
OVERFLOW	unchanged	

Add Producing Overflow Condition.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
ADD	02	02	COUNT1(UN)	COUNT2(UN)	SUM(SN)

	BEFORE	AFTER
A ADDRESS	61	61
B ADDRESS	53	53
C ADDRESS	unchanged	
COMP. INDC.	unchanged	
OVERFLOW	xxx	ON

ALLOCATE STORAGE FOR SEGMENT DICTIONARY (ADSD) - DECLARATIVE.

This instruction is defined for the Advanced Assembler only. Its function is to allocate storage area for the segment dictionary at the current storage location.

The format for the ADSD instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	ADSD	0	1	0	5										

LABELS are not permitted with this instruction.

The VAR field is used to specify the number of segments in the program. This 4-digit right-justified count should correspond to, or exceed, the number of SEGM statements in the program. Leading zeros are optional.

The remaining fields are not used by this instruction.

Ordinarily, the Assembler automatically assigns space for the segment dictionary at the end of the program area. If, however, the program is longer than 50KB (100K digits), the segment dictionary will not be reachable because of addressing limitations, and all OVLY commands will cause syntax errors in the assembly. The ADSD declarative makes it possible to locate the segment dictionary within the first 50KB of long programs, thus avoiding the addressing problem. ADSD must be the last statement of the main segment.

DECLARE EXTENDED ALPHA CONSTANT (ALFA) - DECLARATIVE INSTRUCTION. This instruction permits the declaration of an alphanumeric constant of up to 59 characters, and it is defined for Advanced Assembler only.

The format for the ALFA instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
LØNG	ALFA		40	ENTIRE LINE				WILL BE LØADED				AS A CØNSTANT				

There are no LABEL restrictions when using this instruction.

The BF field specifies the number of characters to be allocated for the constant; a range of 01 through 59 is acceptable. If omitted, the Assembler will assume 59.

The constant data is specified, left-justified, in columns 22-80.

The remaining fields are not used by this instruction. Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

ADJUST LOCATION COUNTER (ALOC) - PSEUDO.

The function of this pseudo is to algebraically add the integer in the A ADDRESS label field to the current value of the location counter.

The format of the ALOC instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	ALØC				-00010											

LABELS are not permitted with this instruction. A signed 5-digit integer is specified in the A ADDRESS field and the sign must be present.

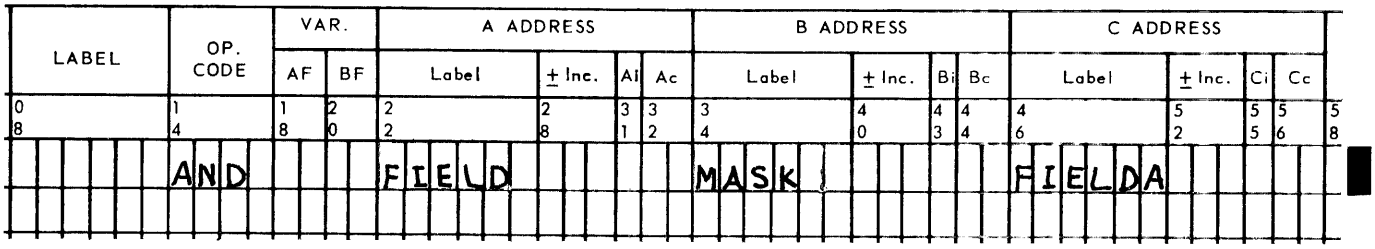
The remaining fields are not used for this instruction.

If this pseudo is coded within an overlayable segment, the adjusted value of the Location Counter must not be less than the beginning address of the segment.

LOGICAL AND (AND) - MACHINE CODE-42.

The function of this instruction is to compare the A field bits with the corresponding B field bits and store a 1 bit into the C field if the corresponding A and B field bits are both on.

The format of the AND instruction is as follows:



There are no LABEL restrictions when using this instruction.

The AF field specifies the number of digits/characters in the A field and the BF field specifies the number of digits/characters in the B field. The C field length is assumed to equal the larger of the AF and BF values. If the A and B fields are not of equal length, the shorter of the two is padded by assuming trailing 0-bit digits/characters (0/00).

The A ADDRESS field points to the first field to be AND'ed. The A address controller may not be SN and the final A address controller must be the same as the final B and C address controllers.

The B ADDRESS field points to the second field to be AND'ed. The B address controller may not be SN. The final B address controller must be the same as the final A and C address controllers.

The C ADDRESS field points to the field into which the AND'ed result is to be stored. The C address controller may not be SN, and the final C address controller must be the same as the final A and B address controllers.

The comparison indicator is set to HIGH if the last result bit (i.e., the low-order bit of the last C field digit/character) is one. The comparison indicator is set to EQUAL if the last result bit is zero.

The OVERFLOW indicator is not changed by this instruction. Program Reserved Memory is not changed by this instruction.

The AND instruction processes bit strings, with no implicit conversion between formats. An example of this follows:

Operands		Result
A Field	B Field	C Field
0	0	0
0	1	0
1	0	0
1	1	1

The following are operational examples of the AND instruction.

Unsigned Numeric To Unsigned Numeric Giving Unsigned Numeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
AND	02	03	FIELDA(UN)	FIELDB(UN)	FIELDC(UN)

	BEFORE	AFTER	
A ADDRESS	F6	F6	111101100000
B ADDRESS	235	235	001000110101
C ADDRESS	xxx	220	001000100000
COMP. INDC.	xxx	EQUAL	
OVERFLOW		unchanged	

Unsigned Alphanumeric To Unsigned Alphanumeric Giving Unsigned Alphanumeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
AND	03	03	FIELD1(UA)	FIELD2(UA)	FIELD3(UA

	BEFORE	AFTER	
A ADDRESS	XYZ	XYZ	111001111110100011101001
B ADDRESS	MQJ	MQJ	110101001101100011010001
C ADDRESS	xxx	DHA	110001001100100011000001
COMP. INDC.	xxx	HIGH	
OVERFLOW		unchanged	

BLANK (bbbb) - PSEUDO.

The function of this pseudo is to provide remarks for documentation purposes. Its function is identical to that of the REMK statement.

The format of the bbbb instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
SECT#1		THIS PSEUDO GIVES GREATER READABILITY													

A label or program point may be entered in the LABEL field. It will be equated to the current value of the location counter (next available storage location), but will not have any field length or data format associated with it.

Documentation text may appear anywhere in columns 18 through 80.

Program Reserved Memory of the object program and the comparison and OVERFLOW indicators are not changed by this instruction.

BCT

BRANCH COMMUNICATE TO CONTROL PROGRAM (BCT) - MACHINE CODE-30.

The function of this instruction is to store program address and status registers, change execution mode to control state, and branch to a specified address.

The format for the BCT instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	BCT	0	2	5	4										

There are no LABEL restrictions when using this instruction.

The VAR field specifies the low-order four digits of an absolute machine address (high-order digits equal 00). If this address contains an undigit F, the five digits immediately following it are interpreted as another absolute machine address, and control is transferred to that address. If the address specified by the B variant field does not contain an undigit F, a processor interrupt is generated.

The remaining fields are not used by this instruction.

The settings of the comparison, OVERFLOW, normal/control state, and EBCDIC/USASCII mode indicators are stored into fixed locations in Systems Reserved Memory, and then reset.

The instruction address register and certain status indicators are stored into Systems Reserved Memory upon execution of this command. The base and limit registers are changed to include all of core storage.

If a processor or I/O interrupt occurs during normal state processing, the processor in effect generates a BCT 0094 command and executes it as though it had been issued within the normal state program. This command is used for all program/MCP communicating.

BIT ONE TEST (BOT) - MACHINE CODE-41.

This instruction tests the A field (in 8-bit groups) for 1-bits in the bit positions selected by the BF field mask.

The format for the BOT instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
TSTSWT	BØT			FOSWTCH1											

There are no LABEL restrictions when using this instruction.

The AF field specifies the number of digits/characters to be tested in the A field. The BF field specifies an 8-bit selection mask. 1-bits within this mask select those bit positions to be tested for a 1-bit within each 8-bit group of the A field. A through F may be used to specify undigits in the mask. The A ADDRESS points to the field to be examined and its controller may <u>not</u> be SN. If the A address controller is UN, the A field will be tested against the mask as consecutive pairs of 4-bit digits. If the A field length is odd, the last (low-order) single digit will be tested against the high-order four bits of the mask.

The remaining fields are not used by this instruction.

The comparison indicators are set to EQUAL if any tested bit in the A field is one. They are set to HIGH if all tested bits in the A field are zero.

Program Reserved Memory and the OVERFLOW indicator are not changed by this instruction.

The BOT instruction essentially treats the A field as a string of bits to which the BF mask is repetitively applied. A 1-bit in the

mask causes a test of the corresponding bit in the A field, and the comparison indicators are set according to the results.

The following are operational examples of the BOT instruction.

Bits Found.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
BOT	03	F0	SWITCH(UN)		

	BEFORE	AFTER
A ADDRESS	001	001
B ADDRESS	——	——
C ADDRESS	——	——
COMP. INDC.	xxx	EQUAL
OVERFLOW	unchanged	

Bits Not Found.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
BOT	02	03	AREA(UA)		

	BEFORE	AFTER
A ADDRESS	DD	DD
B ADDRESS	——	——
C ADDRESS	——	——
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

BRE

BRANCH REINSTATE (BRE) - MACHINE CODE-90

This instruction may not be used in normal state programs. It reinstates the machine registers and transfers control back to the point it had reached when a BCT command was issued. The BCT command may have been issued as a program instruction, or it may have been generated by the hardware. The execution mode is changed from control back to normal.

The format of the BRE instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
RETURN	BRE	0														

There are no LABEL restrictions when using this instruction.

The AF field (column 18) specifies whether or not interrupts are to be masked. Coding is as follows:

Code	Description
0	Interrupts enabled. If an interrupt is pending when the BRE command is issued, it will immediately be converted into a BCT itself.
1	Interrupts masked. Interrupts cannot be serviced until the instruction sequence which receives control from the BRE executes a BCT itself.

The remaining fields are not used by this instruction.

The machine address and condition registers (base, limit, EBCDIC/USASCII mode, and comparison) are restored from Systems Reserved Memory where they were stored when the last BCT command was issued.

Program Reserved Memory is not changed by this instruction.

Non-interruptable normal state execution mode (AF digit equals 1) is usually invoked for special I/O routines such as pocket selection on the sorter-reader. If an invalid instruction or address is attempted in this mode, the machine stops completely.

BRANCH UNCONDITIONAL (BUN) - MACHINE CODE-27

This instruction causes control to be transferred to an address specified by the A field.

The format for the BUN instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	BUN			-C1											

There are no LABEL restrictions when using this instruction.

Any defined branch address may be coded in the A ADDRESS field. Indexing is unrestricted. Address controller bits are interpreted as a high-order digit of the branch address, except that A/C 11_2 retains the indirect address meaning. Thus, a base-relative branch address up to 299998 may be assembled.

The remaining fields are not used by this instruction. The Comparison and OVERFLOW indicators and Program Reserved Memory are not changed by this instruction. The instruction is of 8-digit address-branch format.

BIT ZERO TEST (BZT) - MACHINE CODE-40.

The function of this instruction is to test the A field (in 8-bit groups) for zero bits in the bit positions selected by the BF field mask.

The format of the BZT instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
.1	BZT		12	SWITCH											

There are no LABEL restrictions when using this instruction.

The AF field specifies the number of characters/digits to be tested in the A field. The BF field specifies an 8-bit selection mask. 1-bits in this mask select those bit positions to be tested for zero bits within each 8-bit group of the A field. A through F may be used to specify undigits in the mask.

The A ADDRESS field points to the field to be examined and its address controller may <u>not</u> be SN.

If the A address controller is UN, the A field will be tested against the mask as consecutive <u>pairs</u> of 4-bit digits. If the A field length is odd, the last (low-order) single digit will be tested against the high-order four bits of the mask.

The remaining fields are not used by this instruction.

The comparison indicator is set to EQUAL if any tested bit in the A field is zero. It is set to HIGH if all tested bits in the A field are one (1).

Program Reserved Memory and the OVERFLOW indicators are not changed by the instruction.

The BZT instruction essentially treats the A field as a string of bits to which the BF mask is repetitively applied. A 1 bit in the mask causes a test of the corresponding bit in the A field, and the comparison indicators are set according to the results.

The following are operational examples of the BZT instruction.

Test For All Alphanumeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
BZT	05	C0	FIELD(UA)		

	BEFORE	AFTER
A ADDRESS	CAX#D	CAX#D
B ADDRESS	———	———
C ADDRESS	———	———
COMP. INDC.	xxx	EQUAL
OVERFLOW	unchanged	

Test For All Alphanumeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
BZT	05	C0	FIELD(UA)		

	BEFORE	AFTER
A ADDRESS	BRACE	BRACE
B ADDRESS	———	———
C ADDRESS	———	———
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

CLOSE FILE (CLOS) - PSEUDO.

The function of this pseudo is to terminate processing of a file or reel (including end-label creation, device release, locking, etc.).

The format for the CLOS instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
ENDALLCLØS	L			TAPE2												

There are no LABEL restrictions when using this instruction.

The AF field (column 18) specifies whether a logical file (any device) or a physical reel (magnetic tape only) is to be closed. Coding is as follows:

Code	Description
blank	Close file
F	Close file
R	Close reel

When a close reel operation is performed, the file remains logically open, the current reel is terminated, then the next reel is located and made available to the program.

The AF field (column 19) specifies the disposition of the file. Coding is as follows:

Code	Description
blank	End label, rewind, retain device assignment to program (e.g., for a subsequent re-open within the program).

Code	Description
R	End label, rewind, release to system (e.g., for subsequent use by another program).
L	End label, rewind, lock (e.g., for removal and storage).
N	End label, no rewind.
P	Rewind, purge, release to system.

NOTES

An output disk file must be closed with lock or release if it is to be passed to another program. If it is not closed with lock or release, it will be purged at the end of the current program.

End-of-Job procedures perform a close release on any unclosed file other than disk.

In the basic language, blank and R both mean release to system.

The A ADDRESS label field contains the internal file-name of the file to be closed. Incrementing, indexing, and address controllers are not permitted.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

Files are assumed closed at the beginning of a program execution and must be opened to permit processing of them. They should be closed when processing is complete, and must be closed if they are to be re-opened in a different mode.

If a file is declared and opened within an overlayable segment, it should be closed before control exits to a higher-level segment (nearer to the main routine). If a file within a segment is left open when control exits from that segment, its status at a later re-entry to the segment is uncertain (depending on whether the segment has been overlaid in the meantime).

DECLARE CONSTANT (CNST) - DECLARATIVE.

This instruction allocates storage for and loads a numeric or alphanumeric constant declared in the statement.

The format for the CNST instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
MAX	CNST		10	SN				-000000	1370	5					

There are no LABEL restrictions when using this instruction.

The VAR field contains a 4-digit count of characters/digits to be allocated for the constant. This length should agree with the data format length conventions. Only the first 24 positions of a constant area are loaded from this statement. If the declared count is greater, storage will be allocated and initialized to zero's if the controller is UN or SN. It will be initialized to blanks if the controller is UA.

The data format of the constant is declared left-justified in the A ADDRESS label field. Coding is as follows.

Code	Description
blank	Unsigned numeric
UN	Unsigned numeric
SN	Signed numeric
UA	Alphanumeric

The constant data is specified left-justified in the B and C ADDRESS fields (columns 34 through 57) for a maximum of 24 digits/characters.

In an unsigned or signed numeric constant, A through F may be coded to denote undigits.

In a signed numeric constant, column 34 (first position) must be plus (+) or minus (-). In numeric constants, blank columns within the length will assemble as zeroes.

NOTE

If a 7-track symbolic (SYMTIN) tape is being created by the assembly, CNST data should not contain untranslatable EBCDIC characters (refer to appendix D).

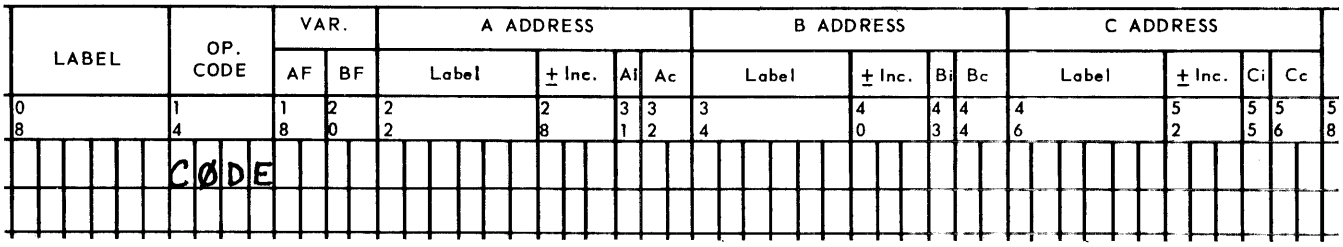
The remaining fields are not used by this instruction. Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

If an alphanumeric constant is declared, the location counter will be automatically synchronized to a character boundry (modulo-2).

ENABLE PRINTED OBJECT LISTING (CODE) - PSEUDO.

This instruction starts or resumes printing of the assembled machine code.

The format for the CODE instruction is:



LABELs are not permitted with this instruction, nor are the remaining fields used. Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

There are no operands for this instruction. CODE may be used in conjunction with, or instead of, the CODE option on the SPEC command.

CORE

OBTAIN AMOUNT OF ASSIGNED CORE (CORE) - PSEUDO.

This pseudo places the value of the amount of core assigned to the object program in a specified 6-digit field.

The format for the CORE instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	CORE			CØRCNT											

There are no LABEL restrictions when using this instruction.

The A ADDRESS field must contain a label referencing a 6 UN field into which the value is to be placed.

The remaining fields are not used by this instruction Program Reserved Memory, and the comparison and OVERFLOW indicators are not changed by this instruction.

COMPARE ALPHANUMERIC (CPA) - MACHINE CODE-45

This instruction compares the characters in the A and B fields according to the binary collating sequence, left-justified, and sets the comparison indicators accordingly.

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
TSTTØPCPA		3	3	9999				ALTABLE			1				

There are no LABEL restrictions when using this instruction.

The AF field specifies the length of the A field and the BF variant specifies the length of the B field. If the field lengths are unequal, the shorter field is padded in the processor with trailing blank fill to equal the length of the longer.

The A ADDRESS points to the first field to be compared and its controller must be UA or IA. The B ADDRESS points to the second field to be compared and its controller must be UA or IA. The remaining fields are not used by this instruction.

The comparison indicator is set to HIGH if the binary value of the A field is greater than that of the B field. The comparison indicator is set to EQUAL if the two fields have exactly the same bit pattern (including trailing blanks), and to LOW if the binary value of the A field is less than that of the B field.

Program Reserved Memory and the OVERFLOW indicators are not changed by this instruction.

The address controller restriction is important. The processor treats each field as a string of bits, with no format conversion

involved. The instruction terminates when an unequal condition is detected, or when the fields are exhausted.

The following are operational examples of the CPA instruction.

Compare Unequal Lengths.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
CPA	05	03	NAME	CODE	

	BEFORE	AFTER
A ADDRESS	ATSbb	ATSbb
B ADDRESS	ATS	ATS
C ADDRESS	———	———
COMP. INDC.	xxx	EQUAL
OVERFLOW	unchanged	

Compare Equal Lengths.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
CPA	02	02	X	Y	

	BEFORE	AFTER
A ADDRESS	AN	AN
B ADDRESS	BN	BN
C ADDRESS	—	—
COMP. INDC.	xxx	LOW
OVERFLOW	unchanged	

COMPARE NUMERIC (CPN) - MACHINE CODE-46.

This instruction algebraically compares the A field against the B field and sets the comparison indicators according to the results.

The format for the CPN instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
CTDØWNCPN		1		0				NLPGCNT							

There are no LABEL restrictions when using this instruction.

The AF field specifies the number of digits/characters in the A field and the BF field specifies the number of digits/characters in the B field. If the A and B fields are not equal in length, the shorter of the two is padded in the processor by assuming leading zeros until their lengths are equal.

The A ADDRESS points to the first field to be compared and the B ADDRESS points to the second field to be compared. UN and UA fields are considered positive. Only the numeric digits of a UA field are used in the comparison.

The remaining fields are not used by this instruction.

The comparison indicator is set to HIGH if the algebraic value of the A field is greater than that of the B field. The comparison indicator is set to EQUAL if the algebraic value of the two fields is equal and to LOW if the algebraic value of the A field is less than that of the B field. Plus zero compares equal to minus zero.

Program Reserved Memory and the OVERFLOW indicator are not changed by this instruction.

The following are operational examples of the CPN instruction.

Compare Signed Numeric Against Unsigned Numeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
CPN	02	05	+20(SL)	FIELD(UN)	

	BEFORE	AFTER
A ADDRESS	+20	+20
B ADDRESS	00015	00015
C ADDRESS		
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Compare Unsigned Numeric Against Signed Numeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
CPN	06	02	000012(NL)	FIELDB(SN)	

	BEFORE	AFTER
A ADDRESS	000012	000012
B ADDRESS	+25	+25
C ADDRESS		
COMP. INDC.	xxx	LOW
OVERFLOW	unchanged	

Compare Unsigned Numeric Against Unsigned Alphanumeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
CPN	03	03	FIELD1(UN)	FIELD2(UA)	

	BEFORE	AFTER
A ADDRESS	213	213
B ADDRESS	KAM	KAM
C ADDRESS		
COMP. INDC.	xxx	LOW
OVERFLOW	unchanged	

ALLOCATE RECORD FIELDS (DATA) - DECLARATIVE.

This instruction causes allocation of storage, without loading any value into the allocated area.

The format for this instruction is:

LABEL	OP CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
TØTAL	DATA		25	UA											

There are no LABEL restrictions when using this instruction.

The number of digits/characters to be allocated is specified as a 4-digit right-justified number in the VARiant field. Leading zeroes are not required.

The data format is specified left-justified in the A ADDRESS label field. Coding is as follows:

Code	Description
blank	Unsigned numeric
UN	Unsigned numeric
SN	Signed numeric
UA	Alphanumeric

The remaining fields are not used by this instruction. Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The DATA statement may be used to allocate storage either independently or within a logical record. Alphanumeric DATA allocations are automatically synchronized modulo-2.

DATE

OBTAIN SYSTEMS DATE (DATE) - PSEUDO.

This instruction is defined for the Advanced Assembler only. It copies the systems date at execution time into a specified location.

The format for the DATE instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	DATE			SYSDAT												

There are no LABEL restrictions when using this instruction.

The A ADDRESS field receives the systems run date and it must be a 6-digit UN field. The coding format is MMDDYY (e.g., 101967).

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

TWO-ADDRESS SUBTRACT (DEC) - MACHINE CODE-03.

This instruction subtracts the contents of the A field from the contents of the B field and leaves the result in the B field.

The format for the DEC instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
DØWNERDEC		1		+5				SLTØTAL							

There are no LABEL restrictions when using this instruction.

The AF field specifies the number of digits/characters in the A field, and the BF field specifies the same for the B field. If the AF and BF lengths are unequal, the shorter of the two is assumed left-zero-filled until their lengths are equal. If the calculated difference is too large to fit into the B field, the subtraction is not performed.

The A ADDRESS points to the subtrahend, and any address controller is valid. A UN field is assumed to be positive. Only the numeric digits of a UA field enter into the subtraction and the field is assumed to be positive.

The B ADDRESS points to the minuend/difference, and any address-controller is valid. A UN field is assumed to be positive as the minuent and the sign of the difference is lost. Only the numeric digits of a UA field enter into the subtraction. The difference is stored in the numeric digits, the zone digits are set to the numeric-subset zone, and the sign of the difference is lost.

The remaining fields are not used by this instruction.

If the difference is too large to fit into the B field, the OVERFLOW indicator is set, and the comparison indicators remain unchanged; otherwise, the comparison indicators are set according, to the sign of the difference (even if the sign is not stored). The settings are as follows:

Setting	Sign
LOW	-
EQUAL	0
HIGH	+

Program Reserved Memory is not changed by this instruction.

The following are operational examples of the Two-Address Subtraction.

Normal Two-Address Subtraction.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
DEC	3	3	BB100	BB200	

	BEFORE	AFTER
A ADDRESS	+014	+014
B ADDRESS	+062	+048
C ADDRESS		
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Two-Address Subtraction With Negative A And B Fields.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
DEC	3	3	BB100	BB200	

	BEFORE	AFTER
A ADDRESS	-035	-035
B ADDRESS	-029	+006
C ADDRESS		
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Two-Address Subtraction With Negative A Field.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
DEC	2	3	BB100	BB200	

	BEFORE	AFTER
A ADDRESS	-71	-71
B ADDRESS	+121	+192
C ADDRESS		
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Two-Address Subtraction With A Greater Than B

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
DEC	3	3	BB100	BB200	

	BEFORE	AFTER
A ADDRESS	+259	+259
B ADDRESS	+138	-121
C ADDRESS		
COMP. INDC.	xxx	LOW
OVERFLOW	unchanged	

Two-Address Subtraction Causing Overflow.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
DEC	3	3	BB100	BB200	

	BEFORE	AFTER
A ADDRESS	-556	-556
B ADDRESS	+942	+942
C ADDRESS		
COMP. INDC.	unchanged	
OVERFLOW	xxx	ON

DECR

DECREMENT BY ONE (DECR) - PSEUDO.

This instruction decrements the contents of the A field by one.

The format for the DECR instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	DECR			REGIST											

There are no LABEL restrictions when using this instruction.

The A ADDRESS points to the field to be decremented.

The remaining fields are not used by this instruction.

This pseudo yields a decrement instruction. The length of the field to be decremented, and its controller are obtained from the declaration of the data field referenced by the A ADDRESS.

DELIMIT CONSTANTS (DELM) - PSEUDO.

The function of this pseudo is to stop the passage of constants to the program's STACK when a DELM is encountered in a list of constants following an NTR instruction.

The format for the DELM pseudo is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	DELM															

Only the OP CODE field is used for this instruction.

The return address stored in the program's stack for the NTR preceding this pseudo will be the address of the constant following the DELM.

The remaining fields are not used by this instruction. Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

DISPLAY MESSAGE ON SPO (DISP) - PSEUDO.

This instruction causes a message to be typed out on the console supervisory printer.

The two formats allowed for the DISP instruction are:

LABEL	OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ADDRESS ± Inc.	A ADDRESS Al	A ADDRESS Ac	B ADDRESS Label	B ADDRESS ± Inc.	B ADDRESS Bi	B ADDRESS Bc	C ADDRESS Label	C ADDRESS ± Inc.	C ADDRESS Ci	C ADDRESS Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
ERRØR1	DISP			ERMSG1											
	DISP			"INPUT LABEL FOR DISK FILE OMITTED											"

There are no LABEL restrictions when using this instruction.

The AF field specifies the number of characters to be typed out. If it is blank, the length associated with the A ADDRESS label is used.

The A ADDRESS points to an alphanumeric text field to be typed out. Incrementing is permitted; indexing is <u>not</u>. Literals are not permitted and the address controller must be UA.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

If column 22 = ("), the character string* following the quote will be displayed. The string must be delimited with a right (") somewhere prior to column 58. Embedded quotes are allowed, i.e., DISP "No. "xxx"" will display No. "xxx". The size of the string need not be given.

When used with BCP, SIOC assumes that the supervisory printer is on symbolic substitution entry 11.

* Maximum size is 34 characters.

DIV

DIVIDE (DIV) - MACHINE CODE-06.

This instruction divides the value of the B field by that of the A field and stores the quotient into the C field. The remainder is left in the B field.

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	DIV	02		+16				SLØUNCES				PØUNDS				

There are no LABEL restrictions when using this instruction.

The AF field specifies the length of the A field and the BF field specifies the length of the B field. The length of the B field must be greater than that of the A field. The C field length is the difference in length of the A and B field. If the quotient is to large to fit into the C field, the division is not performed.

The A ADDRESS points to the divisor field, and any address controller is valid. UN fields are considered to be positive. Only the numeric digits of a UA field enter into the division; the field is considered to be positive.

The B ADDRESS points to the dividend/remainder field, and any address controller is valid. UN fields are considered to be positive. Only the numeric digits of a UA enter into the division; the field is considered to be positive. The sign of the remainder is the same as the sign of the dividend. If the B field is UA, the remainder is stored into the numeric digits and the zone digits are set to the numeric-subset zone.

The C ADDRESS points to the quotient field, and any address controller is valid. If the C field is UN, the quotient sign is lost. If the field is UA, the quotient is stored in the numeric digits, the zone digits are set to the numeric-subset zone, and the quotient sign is lost. If the C field is SN, the quotient sign will be algebraically correct.

If the quotient is to large to fit into the C field (e.g., the absolute value of the divisor is not greater than the absolute value of a corresponding number of leading digits of the dividend), the OVERFLOW indicator is set on and the comparison indicators are not set.

The comparison indicators are set according to the sign of the quotient, even if the sign is not stored. The settings are; LOW if negative, EQUAL if zero, and HIGH if positive.

Program Reserved Memory is not changed by this instruction.

The following are operational examples of the Divide instruction.

Normal Division With Positive Operands.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
DIV	1	4	BB100	BB200	BB300

	BEFORE	AFTER
A ADDRESS	+9	+9
B ADDRESS	+0101	+0002
C ADDRESS	xxxx	+011
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Division With Minus Operands.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
DIV	2	5	BB100	BB200	BB300

	BEFORE	AFTER
A ADDRESS	-12	-12
B ADDRESS	-00187	-00007
C ADDRESS	xxxx	+015
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Division With Unlike Signed Fields.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
DIV	3	5	BB100	BB200	BB300

	BEFORE	AFTER
A ADDRESS	-180	-180
B ADDRESS	+03920	+00140
C ADDRESS	xxx	-21
COMP. INDC.	xxx	LOW
OVERFLOW	unchanged	

Alphabetic Division.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
DIV	3	4	BB100	BB200	BB300

	BEFORE	AFTER
A ADDRESS	9QD	9QD
B ADDRESS	AR1B	0928
C ADDRESS	xx	+1
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Division Where The Absolute Value Of The Divisor Is Not Greater Than The Absolute Value Of The Corresponding High Order Positions Of The Dividend.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
DIV	3	5	BB100	BB200	BB300

	BEFORE	AFTER
A ADDRESS	+017	+017
B ADDRESS	+29451	+29451
C ADDRESS	unchanged	
COMP. INDC.	unchanged	
OVERFLOW	xxx	ON

DELETE SOURCE STATEMENTS IN UPDATE (DLET) - PSEUDO.

This instruction causes deletion of input source-program statements between the sequence number bounds specified in this statement.

The format for the DLET instruction is:

SEQ. NO.	LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDR	
			AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	±
0 2	0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2
020510		DLET			020545									

The starting sequence number for deletion is specified in columns 2 through 7 of the SEQ. NO. field.

LABEL entries are not permitted with this instruction.

The ending sequence number for deletion is specified in the A ADDRESS label field. If only one statement is to be deleted, this field may be left blank.

The remaining fields are not used by this instruction.

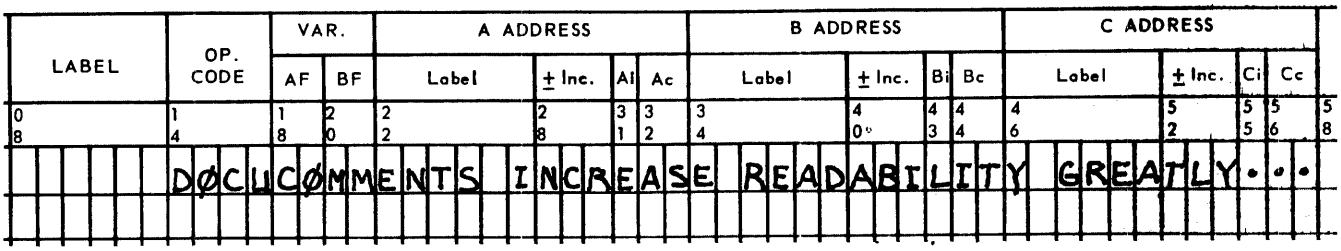
Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

This statement is used when performing an assembly with source card changes being posted against a library input file contained on magnetic tape or disk. It can only be used when the input file is sequenced since it causes the deletion of one contiguous sequence of records. The DLET pseudo causes deletion of all statements whose sequence numbers fall between the starting number and the ending number inclusive. Thus, the numbers specified in the pseudo need not precisely equal the first and last sequence numbers to be deleted. The DLET pseudo itself has only a clerical function and will not appear in the final assembly listing.

COMMENT (DOCU) - PSEUDO.

The function of this pseudo is to carry remarks for documentation purposes. Its function is identical to that of the REMK statement.

The format of the DOCU instruction is:



LABELs are not permitted with this instruction.

The remarks text may be located anywhere in columns 18 through 80.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

SUSPEND PROGRAM TEMPORARILY (DOZE) - PSEUDO.

This instruction is defined for the Advanced Assembler only. It causes execution of the program to be suspended for a specific number of seconds.

The format of the DOZE instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
PAUSE	DØZE			AMINIM												

There are no LABEL restrictions when using this instruction.

The A ADDRESS points to a 5-digit unsigned numeric field containing the number of seconds the program is to be suspended. Literal values are not permitted. The maximum allowable value in this field is 86399 (23 hours, 59 minutes, and 59 seconds).

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

DUMP PROGRAM MEMORY (DUMP) - PSEUDO.

This instruction causes a temporary suspension of program execution while the entire contents of program memory (base to limit) are dumped to the printer. Execution then resumes.

The format for this instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
SNPSHT	DUMP															

There are no LABEL restrictions when using this instruction.

The remaining fields are not used by the instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

There are no operands for the command. The command produces the same dump format as the keyboard DM and DP commands. It may be used any number of times in the program, and at any point.

EDIT (EDT) - MACHINE CODE-49.

This instruction moves digits or characters from the A field to the C field under control of the edit mask in the B field. Additional characters may be inserted according to the specifications of the edit mask.

The format of this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
.A301	EDT			TØTAL				PC9(7)				UAPRTREC	+50		

There are no LABEL restrictions when using this instruction.

The AF field is not used for execution and may contain any value. The BF field specifies the number of characters in the edit mask. If the edit mask has been defined by a PICT declarative (Advanced Assembler only), this field should be left blank. The length of the C field depends on the specifications of the edit mask.

Data movement and editing are stopped by the exhaustion of the edit mask.

The A ADDRESS points to the source field to be edited, and all address controllers are legal. If the A field is UA, the zone portion of the first (most-significant) character is treated as a sign. If the field is UN, it is considered positive.

The B ADDRESS points to the edit mask, and its address controller must be UA or IA. The B field consists of a string of double-digit edit operators which control data movement. These operators are explained in section 2 of this manual.

The C ADDRESS points to the area into which the edited data is stored; its address controller must be UA or IA.

The comparison indicator is set to HIGH if at least one digit or character was fetched from the A field and the A field is non-zero positive. It is set to EQUAL if no digits or characters were fetched from the A field, or if the field is zero. It is set to LOW if at least one digit or character was fetched from the A field and the field is non-zero negative. The OVERFLOW indicators are not changed by this instruction.

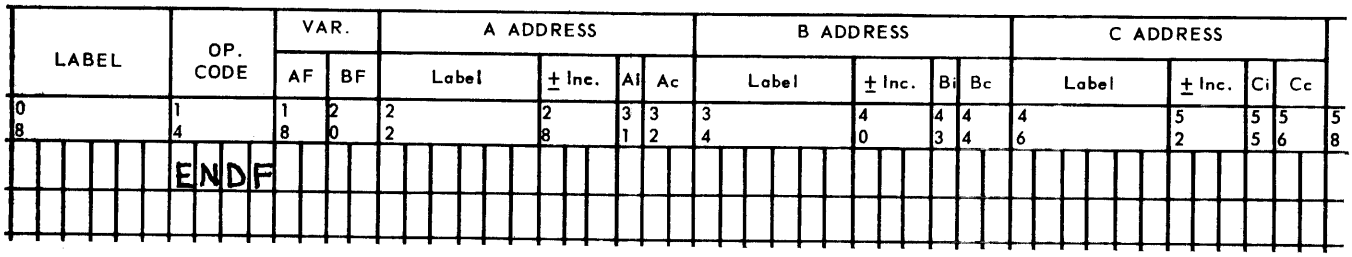
This command makes use of the Edit Insertion Table (Program Reserved Memory locations 00048-63) if the edit operators specify insertion of punctuation characters. The standard punctuation set in the Edit Insertion Table (the set which is loaded by the Advanced Assembler if the PICT declarative is used) is +, -, *,',',$,0, and b (blank). The user may load a different punctuation set into the Edit Insertion Table, but should be extremely careful with his "housekeeping" coding.

The EDT command may be used to edit <u>several</u> data fields in one instruction providing they are in adjacent source fields and are to be adjacent in the edited field. Proper specification of the edit operators will accomplish this multiple editing quite easily.

END FILE BLOCK (ENDF) - PSEUDO.

This pseudo signals the Assembler that all declaratives of one FILE block have been processed, or that a single FILE declarative is now being delimited.

The format for the ENDF instruction is:



LABELs are not permitted, nor are the remaining fields used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

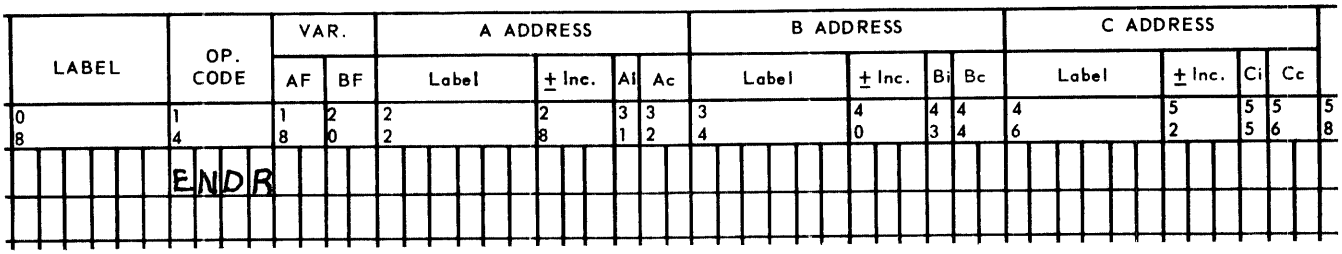
In the Advanced Assembler this pseudo is optional if the next command is FILE or USER, and required otherwise.

The Basic Assembler requires that each file be delimited with an ENDF.

END RECORD BLOCK (ENDR) - PSEUDO.

This pseudo defines the end of assembly statements which comprise one record definition.

The format for this instruction is:



LABELs are not permitted, nor are the remaining fields used by this instruction. Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The instruction has no operands. Under the Advanced Assembler, the ENDR statement is not required if the next statement is USER, FILE, ENDF, or RECD.

The Basic Assembler requires that each RECD be delimited with an ENDR.

END OVERLAYABLE SEGMENT (ENSG) - PSEUDO.

This instruction defines the end of an overlayable coding segment.

The format of this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	ENSG			START											

LABELs are not permitted with this instruction.

A segment-entry label is given in the A ADDRESS label field if control should enter the segment (via OVLY from another segment) at <u>other</u> than the first executable instruction. Indexing is not permitted. The A ADDRESS increment field may be used.

The remaining fields are not used by the instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

Every SEGM card must have a corresponding ENSG card. The main routine (code preceding the first SEGM card) is not considered to be a segment and, thus, must <u>not</u> have an ENSG card.

EQIV

DEFINE SYMBOL OF EQUIVALENCE (EQIV) - PSEUDO.

This instruction defines a label according to the values in the variant and A ADDRESS fields.

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
MASK	EQIV	0005	*				UA								

Only LABELs (not program points) may be defined by this OP code.

If the A ADDRESS field does not have a length associated with it, or if the length is to be changed, a new 4-digit value can be entered in the VAR field.

The A ADDRESS label field may contain any one of the following:

a. A normal label.
b. An asterisk reference (current value of the Location Counter).
c. An unsigned 5-digit base-relative address.
d. A forward or backward reference to a program point.
e. May reference files and records associated with files.

The A ADDRESS increment field may contain a positive or negative value. The A ADDRESS index field may be used. The A address controller field may contain a data format (UN, SN, or UA) if the A ADDRESS label has no format associated with it, or if its format is to be overridden.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

Any references to a Label defined in an EQIV statement must <u>follow</u> the EQIV statement.

BRANCH EQUAL (EQL) - MACHINE CODE-22

The EQL instruction causes control to be transferred to the specified address if the comparison indicator is set to EQUAL.

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	EQL			FØUND			IA								

There are no LABEL restrictions when using this instruction.

Any defined branch address may be coded in the A ADDRESS field. Indexing is unrestricted, The address controller bits are interpreted as a high-order digit of the branch address except that A/C 11_2 retains the indirect address meaning. Thus, a base-relative branch address up to 299998 can be assembled.

The remaining fields are not used by this instruction. Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

This instruction is of 8-digit address-branch format.

EXT

EXIT FROM SUBROUTINE (EXT) - MACHINE CODE-32.

This instruction reverses the actions performed by the NTR instruction, and thus accomplishes exiting from a subroutine.

The format of this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
EXIT	EXT															

There are no LABEL restrictions when using this instruction.

The A ADDRESS is optional. If it is omitted, the Assembler will generate a return branch to the location specified in the subroutine stack entry (the address of the first instruction after the corresponding NTR). If the A ADDRESS is specified, control will be returned to the A ADDRESS location after the top entry in the subroutine stack has been removed.

The remaining fields are not used by this instruction.

The settings for the comparison, OVERFLOW, and EBCDIC/USASCII indicators that were stored in the subroutine stack will be restored into the machine indicators.

The address located in IX3 (loaded by the NTR instruction) is copied into location 00040. The transfer-control address is interpreted (BASE + IX3 Indirect unless overridden) and stored. The previous value of IX3 is copied from the stack into location IX3. The stored transfer-control address is then used to access the next instruction.

Any paramter values left in the stack entry are lost; they are <u>not</u> copied back to the area following the NTR instruction which loaded them into the stack.

The following is an example of the execution of the EXT instruction.

010970 EXT (assembled as 32F00000)

assume that the contents of storage before execution of the EXT are:

IX3	+0001024
00040	001046
001024	003034
	+0000010
	00
	203010

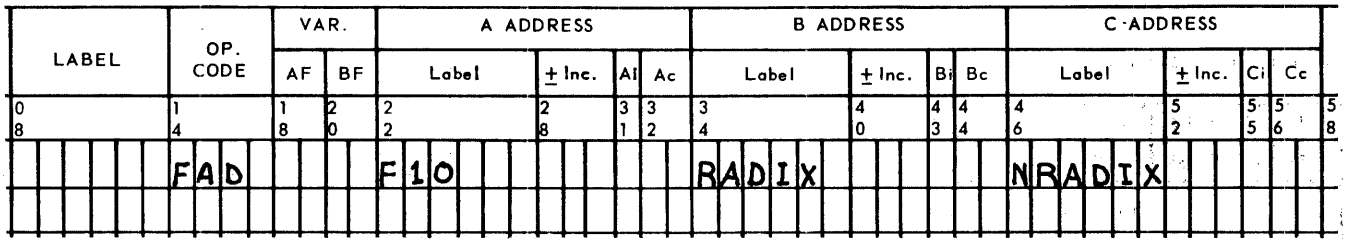
after execution of the EXT instruction, the values will be:

next instruction address:	003034
IX3	+0000010
00040	001024
001024	(unchanged)

FP ADD (FAD) - MACHINE CODE-80.

The function of this instruction is to add one floating-point value to another and store the result into a third field.

The format for the FAD instruction is:



There are no LABEL restrictions when using this instruction.

The AF and BF fields specify the number of mantissa digits in their corresponding floating-point numbers. The C field result contains as many mantissa digits as the longer of the A and B fields.

The A ADDRESS field points to the first floating-point number to be added, and its address controller must be SN or IA. The number need not be normalized.

The B ADDRESS field points to the second floating-point number to be added, and its address controller must be SN or IA. The number need not be normalized.

The C ADDRESS field points to the field into which the sum is to be stored, and its address controller must be SN or IA. The sum is normalized. If the result exponent is less than -99 (underflow), floating zero is stored. The C field may not overlap the A or the B field.

The OVERFLOW indicator is set if floating-point overflow (result exponent greater than +99) occurs. If overflow occurs, the result will <u>not</u> be stored.

The comparison indicator is set HIGH if the result is positive or if floating-point overflow occurs. The comparison indicator is set to EQUAL if the result is zero and to LOW if the result is negative.

Floating-point zero is stored as -99+0000... (number of mantissa digits dictated by the A and B field-lengths).

The following are operational examples of the FP Add instruction.

Normal Floating Point Add.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
FAD	02	05	A(SN)	B(SN)	C(SN)

	BEFORE	AFTER
A ADDRESS	-04+20	-04+20
B ADDRESS	-05+67501	-05+67501
C ADDRESS	xxxxxxxxx	-04+26750
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Floating Point Add Causing Overflow.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
FAD	01	03	A(SN)	B(SN)	C(SN)

	BEFORE	AFTER
A ADDRESS	+99+1	+99+1
B ADDRESS	+99+999	+99+999
C ADDRESS	unchanged	
COMP. INDC.	xxx	HIGH
OVERFLOW	xxx	ON

FDV

FP DIVIDE (FDV) - MACHINE CODE-83.

This instruction divides the floating-point number of the B field by that of the A field. The floating-point quotient is stored in the C field and the remainder is stored in the B field.

The format of this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	FDV			CØS				SIN				TAN				

There are no LABEL restrictions when using this instruction.

The AF and BF fields specify the number of mantissa digits in their corresponding fields. The B field must be longer than the A field or overflow will occur. The length of the C field is the difference between the A and B field lengths.

The A ADDRESS points to the field containing the floating-point divisor and its address controller must be SN or IA. The divisor must be normalized.

The B ADDRESS points to the field containing the floating-point dividend and also receives the remainder. The address controller for this field must be SN or IA. The dividend must be normalized and the remainder will not be normalized.

The C ADDRESS points to the field which receives the quotient, and its address controller must be SN or IA. The quotient is normalized.

The OVERFLOW indicator is set on if the mantissa of the B field is shorter than the mantissa of the A field or, if floating-point overflow (quotient exponent is greater than +99) or underflow

(quotient exponent less than -99 and quotient mantissa is non-zero) occurs. The quotient and remainder are not stored.

The comparison indicator is set to HIGH if the quotient is positive, floating-point overflow occurs if the B field mantissa is shorter than the A field mantissa. It is set to EQUAL if the quotient is zero, and LOW if the quotient is negative or floating-point underflow occurs.

Program Reserved Memory is not changed by this instruction.

Division by zero will cause floating-point overflow. The following are operational examples of the FP Divide instruction.

Normal Floating Point Divide.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
FDV	02	05	AFP	BFP	CFP

	BEFORE	AFTER
A ADDRESS	+00+20	+00+20
B ADDRESS	+00+60000	-99+00000
C ADDRESS	xxxxxxx	+01+300
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Floating Point Divide Causing Underflow.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
FDV	02	05	AFP	BFP	CFP

	BEFORE	AFTER
A ADDRESS	+50+20	+50+20
B ADDRESS	-60+60000	-60+60000
C ADDRESS	unchanged	
COMP. INDC.	xxx	LOW
OVERFLOW	xxx	ON

FILE

DECLARE LOGICAL FILE (FILE) - DECLARATIVE. (ADVANCED ASSEMBLER)

This instruction is defined for the Advanced Assembler only. It declares a logical input/output file and specifies the device, access method, etc.

The format of the FILE instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
DSKØUTFILE				NEWMST				DSKS	100020	S	1		002	W	

A normal LABEL must be used, and this LABEL is the internal file-name used in MCP control operations.

The VAR field is used for OCR and MICR reader files only. Column 19 specifies the buffer technique and coding is as follows:

Code	Description
0	MICR read with non-alternating buffers.
1	MICR read with alternating buffers.
2	OCR read with non-alternating buffers (demand or flow).
3	OCR read with alternating buffers (flow only).
4	OCR and MICR read with non-alternating buffers (demand only).
5	OCR/MICR read with alternating buffers.

Column 21 specifies record formatting and read-error suppression. Coding is as follows:

Code	Description
0	No record formatting; report all read errors.
1	No record formatting. Ignore the cannot-read errors after the second transit-field delimiter.
2	No record formatting. Ignore reading after the second transit-field delimiter.
4	Format the record. Report the amount-field and transit-field errors separately.
5	Format the record. Report the amount-field and transit-field errors separately. Ignore the cannot-read errors after the second transit-field delimiter.
6	Format the record. Report the amount-field and transit-field errors separately. Ignore reading after the second transit-field delimiter.

Unless "labels omitted" is specified in column 46, the A ADDRESS field must contain the external file identification. This identification must begin with a letter followed by zero to five alphanumeric characters. It is the name that the MCP uses to associate the logical FILE declaration with a physical data file.

The A ADDRESS increment/index controller contains an auxiliary file identification for multiple-file tape reels only. The auxiliary file-name identifies the reel, and the ordinary file-name identifies the file within the reel.

Columns 34 through 36 of the B ADDRESS must contain the peripheral equipment type. Coding is as follows:

Code	Description
blank	7- or 9-track magnetic tape.
MTP	7- or 9-track magnetic tape.
MT7	7-track magnetic tape.
MT9	9-track magnetic tape.
MPE	Phase encoded tape.
CRD	Card reader.
PRØ	Printer only.
CPU	Card punch.
PBT	Printer back-up tape.
PRN	Line printer.
PRF	Printer special forms.
DSK	Disk file.
DKF	Assign disk by file number.
DKA	Assign disk by area number.
Edd	Assign disk to Designated Electronics Unit (where dd = specified unit).
PTR	Paper tape reader.
PTP	Paper tape punch.
SØR	MICR sorter-reader.
MTL	Multiple tape lister.
SPØ	Supervisory printer.
TYP	9350 Typewriter.
ØLB	B 606.
TWX	TWX.
T50	1050 (IBM) terminal.
T30	1030 (IBM) terminal.
D20	UNIVAC DCT-2000.
B35	B 3500.
B25	B 2500.
AA1	8A1.
TLX	Telex.
BTT	Burroughs audio response.
BDD	Burroughs digital display.

Code	Description
A3B	83B3 (Teletype 28).
F73	Friden 7311.
TC5	Burroughs TC 500.
TC7	Burroughs TC 700.
B05	Burroughs 500 series computer.
VDD	B 9352 visual display.

Column 37 of the B ADDRESS field contains either the code translation for tape or data communications, or the access method for disk. Coding is as follows:

Code	Description
N	No code translation for 7-track magnetic tape.
blank or T	BCL/EBCDIC translation for 7-track magnetic tape.
N	No code translation for data communications.
blank or T	Device code/EBCDIC translation for data communications.
F	Upper and lower case PTTC-6/EBCDIC translation for IBM 1050 (device code T50).
F	Upper and lower case USASCII-EBCDIC translation.
blank or S	Sequential access to disk records.
R	Random access to disk records.

A "W" in column 38 declares work files and will cause the ⟨mix index⟩ number assigned to the object program, while under MCP control, to

be placed into the file-name. This creates a unique name and will allow the same program to multiprocess with itself.

An S in column 39 specifies a shared disk file. Any shared disk file must also be declared random.

For disk files, the number of logical records per area is coded in columns 40 through 43 (maximum value of 9999) of the B ADDRESS field and the number of areas in the file is coded in columns 44 through 45 (maximum value of 20). If more than 9999 records per area are desired, columns 40 through 43 must be left blank. For tape with a non-standard label, the length of the label in characters is specified in columns 40-43. The minimum length is 80, and if no length is specified, 80 characters are assumed.

NOTE

During program execution, disk space is assigned to a file (as needed) in blocks of contiguous physical segments called areas. A file may be checkerboarded across the disk in a maximum of 20 areas. By this means, dedicated but unused disk space is held to a minimum. As files are purged, the areas are released back to the system. For a more detailed explanation, refer to "format of files on disk" in the B 2500/B 3500 Master Control Programs Information Manual (1031218).

The physical label convention of the external file is declared in column 46 of the C ADDRESS field. Coding is:

Code	Definition
0 or Ø	Omitted label.
blank or S	Standard label.
U	USASCII label.
I	Installation label.

Non-standard labels must be declared omitted and they are verified or created by the program. Refer to the USER command.

Labels other than the above may be created or verified by the program through the USER pseudo. When this option is exercised, the labels should be declared omitted.

The external recording mode of the file is declared in column 47 of the C ADDRESS field. This specification is only required for magnetic tape, cards, and paper tape. Coding for card input/output is as follows:

Code	Definition
N or A	BCL card code.
S, E, 2, or blank	EBCDIC card code.
U, 1 or B	Binary card code.

NOTE

The recording mode of an input card file is entirely determined by the file header card unless the file is attached to the program by means of an UL console command.

EXAMPLE:
?DATA or ?LABEL - the card code is EBCDIC.
?DATAB - the card code is BCL.

The MCP currently cannot read binary-coded cards because of the control card recognition problem.

The coding for 7-track tape input/output is as follows:

Code	Description
N or A	Even parity (BCL or BCD).
S or blank	Odd parity (binary).

The coding for paper tape is as follows:

Code	Description
A	USASCII

Code	Description
B or blank	BCL
E	EBCDIC

The number of alternate buffers (input/output) is declared in column 48 of the C ADDRESS field. Coding is:

Code	Definition
0 or blank	No alternate buffers (one buffer is always assigned). Sorter-reader files must use this technique.
1-9	Number of alternate buffers desired.

NOTE

A tape lister file (MTL device) requires at least three alternate buffers if a work area is declared, or at least four if no work area is declared.

The file retention period (in days) is declared in columns 49 through 51 of the C ADDRESS field. A blank or zero value will indicate a zero retention period; the result of this will depend on the particular MCP's handling of a zero day retention period.

The number of logical records per block is declared in columns 52 through 54 of the C ADDRESS field. A blank or zero implies 001 (unblocked records).

The buffer access technique is specified in column 55 of the C ADDRESS field. Coding is as follows:

Code	Description
Blank, 0, or W	Work area and buffer.
1 or B	Buffer only.

If no work area is assigned, the MCP will supply the address of the next logical record by way of Index Register 2.

The record type is specified in column 56 of the C ADDRESS field. Coding is:

Code	Description
Blank or F	Fixed-length.
V	Variable-length.

The first four characters of a variable-length record must contain the length of the record, in 8-bit numeric characters. For example, if a record contained 170 data characters (exclusive of the length count), its first four characters would be 0174.

Columns 58 through 63 of the REMARKS field contain a control label or value for various special files. These labels or values are:

a. Random access disk files - a label addressing an 8-digit UN field containing the actual key.

b. Variable-length records - the maximum block length in digits.

c. Multiple tape lister file - a label addressing a 4-digit UN field containing the lister unit and tape designation.

d. Sorter-reader file - necessary branch-address label for manual End-of-File.

e. On-line banking - label addressing a location into which the MCP stores the terminal unit number upon completion of each I/O operation.

Optional files may be declared by coding OPT in columns 64 through 66. For a disk file, if more than 9999 logical records per area are desired, the number is coded, left-justified, in columns 67 through 74. For the Assembler to recognize this field, columns 40 through 43 must be blank.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

Any FILE statement must be followed by a RECD statement (regardless of work-area conventions) to define (maximum) logical record length.

FILE

DECLARE LOGICAL FILE (FILE) - DECLARATIVE. (BASIC ASSEMBLER)

This instruction is defined for Basic Assembler only. It declares parameters for file processing while under control of the SIOC routine in the Basic Control Program environment.

The format of the FILE instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
TAPEINFILE	01	00		MASTER								S	005		

The name of the file is given in the LABEL field. This is the name used to refer to the file in executable macros.

The VAR field specifies a 4-digit maximum record length in characters and must be an even number.

The external file identification is specified in the A ADDRESS label field unless "labels omitted" is declared in column 46.

The multi-file identification, if necessary, is provided in columns 28 through 33 of the A ADDRESS field.

A 2-digit actual channel number for the device carrying the file, may optionally be declared in columns 38-39 of the B ADDRESS field. If a symbolic substitution number is specified in the B ADDRESS of the OPEN to this file, the actual channel number specified here will be overridden.

The first syllable of an I/O descriptor for the file may optionally be given in columns 40-45 of the B ADDRESS field. This descriptor is overridden if the file's OPEN specifies a symbolic substitution entry. For a more detailed explanation of coding under SIOC, refer to section 5.

The external label convention is declared in column 46 of the C ADDRESS field. Coding is as follows:

Code	Definition
0 or ∅	Labels omitted.
blank or S	Standard labels.

The number of logical records per physical block is specified in columns 52 through 54. Zero or blank is equivalent to 001.

The remaining fields are not used with this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

FINI

END OF SYMBOLIC INPUT (FINI) - PSEUDO.

This instruction signals the end of the symbolic input file.

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	FINI														

LABEL entries are not permitted, nor are the remaining fields used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

The instruction has no operands. It is required and must be the last instruction in the symbolic input.

FP MULTIPLY (FMP) - MACHINE CODE-82.

This instruction multiplies the floating-point number in the A field by the number in the B field and stores the result in the C field.

The format of the FMP instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	FMP			PI2				DIAM				CIRCUM				

There are no LABEL restrictions when using this instruction.

The AF field specifies the number of mantissa digits in the A ADDRESS field number and the BF field does the same for the B ADDRESS field number. The C ADDRESS mantissa field-length is the sum of the A and B mantissa field-lengths.

The A ADDRESS field points to the field containing one of the factors. The B ADDRESS field points to the field containing the second factor and the C ADDRESS field points to the field which receives the product. The address controllers for these fields must be SN or IA. The factor must be normalized for the A and B fields and the product will be normalized for the C field. The C field may not overlap the A or B fields.

The OVERFLOW indicator is set on if floating-point overflow (result exponent greater than +99) or underflow (result exponent less than -99 and result mantissa non-zero) occurs. If over/underflow occurs, the result will not be stored.

The comparison indicator is set HIGH if the result is positive or if floating-point overflow occurs. It is set EQUAL if the result is zero, and LOW if the result is negative or if floating-point underflow occurs.

Program Reserved Memory is not changed by the instruction.

Floating-point zero is stored as -99+0000 (number of mantissa digits dictated by the A and B field lengths).

The following are operational examples of the FMP instruction.

Normal Floating Point Multiply.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
FMP	02	04	A	B	C

	BEFORE	AFTER
A ADDRESS	+01+20	+01+20
B ADDRESS	-01-5050	-01-5050
C ADDRESS	xxxxxxxxx	+00-101000
COMP. INDC.	xxx	LOW
OVERFLOW	unchanged	

Floating Point Multiply Causing Overflow.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
FMP	02	02	A	B	C

	BEFORE	AFTER
A ADDRESS	+50+40	+50+40
B ADDRESS	+60+20	+60+20
C ADDRESS	unchanged	
COMP. INDC.	xxx	HIGH
OVERFLOW	xxx	ON

FP SUBTRACT (FSU) - MACHINE CODE-81.

This instruction subtracts the A field floating-point number from that of the B field and stores the result in the C field.

The format of the FSU instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
.3	FSU			ASQUAR				BSQUAR				CSQUAR			

There are no LABEL restrictions when using this instruction.

The AF field specifies the number of mantissa digits in the A field number and the BF field does the same for the B field number. The C field result contains as many mantissa digits as the longer of the A and B fields.

The A ADDRESS field points to the subtrahend field (number to be subtracted) and its controller must be SN or IA. The subtrahend need not be normalized. The B ADDRESS field points to the minuend field (number to be subtracted from) and its controller must be SN or IA. The minuend need not be normalized.

The C ADDRESS field points to the field into which the difference is to be stored, and its controller must be SN or IA. The difference is normalized. The C field may not overlap the A field or the B field. If the result exponent is less than -99 (underflow), floating zero is stored.

The OVERFLOW indicator is set on if floating-point overflow (result exponent greater than +99) occurs. If overflow occurs, the result will not be stored.

The comparison indicator is set to HIGH if the result is positive or floating overflow occurs. It is set to EQUAL if the result is zero, and LOW if the result is negative.

Program Reserved Memory is not changed by this instruction.

Floating-point zero is stored as -99+000 (number of mantissa digits dictated by the A and B field-lengths).

The following are operational examples of the FP Subtract instruction.

Normal Floating Point Subtract.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
FSU	03	02	A	B	C

	BEFORE	AFTER
A ADDRESS	+02+500	+02+500
B ADDRESS	+03+20	+03+20
C ADDRESS	xxxxxxx	+03+150
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Floating Point Subtract Causing Overflow.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
FSU	03	03	A	B	C

	BEFORE	AFTER
A ADDRESS	+98-100	+98-100
B ADDRESS	+99+993	+99+993
C ADDRESS	unchanged	
COMP. INDC.	xxx	HIGH
OVERFLOW	xxx	ON

BRANCH NOT LESS (GEQ) - MACHINE CODE-26.

This instruction transfers control to the specified address if the comparison indicator is set to HIGH or EQUAL.

The format of this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	GEQ			+B											

There are no LABEL restrictions when using this instruction.

Any defined branch address may be coded in the A ADDRESS field, and indexing is unrestricted. The address controller bits are interpreted as a high-order digit of the branch address except that A/C 11_2 retains the indirect address meaning. Thus, branch address up to 299998 base-relative may be assembled.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

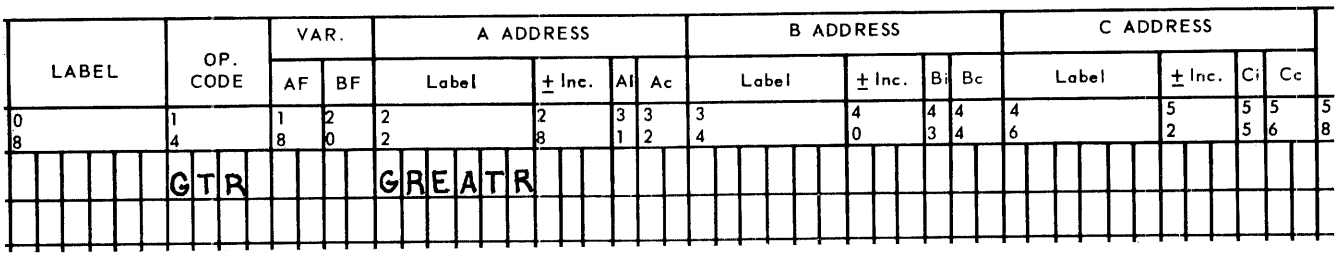
The instruction is of 8-digit address-branch format.

GTR

BRANCH GREATER (GTR) - MACHINE CODE-24.

This instruction transfers control to the specified address if the comparison indicator is set to HIGH.

The format for this instruction is as follows:



There are no LABEL restrictions when using this instruction.

Any defined branch address may be coded in the A ADDRESS field, and all indexing is unrestricted. The address controller bits are interpreted as a high-order digit of the branch address except that A/C 11_2 retains the indirect address meaning. Thus, a base-relative branch address up to 299998 can be assembled.

The remaining fields are not used by this instruction.

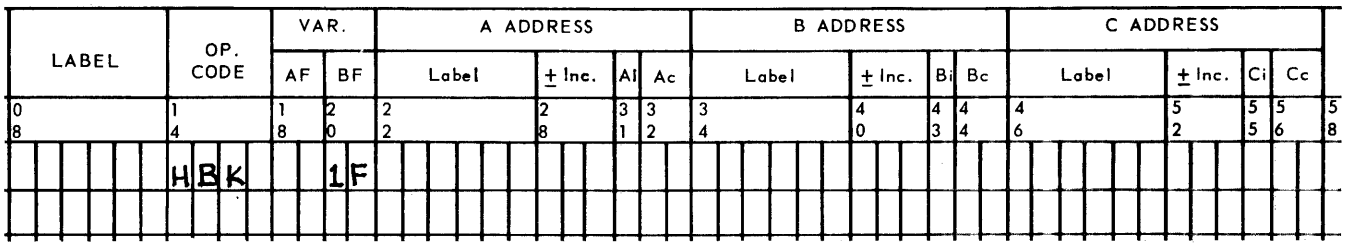
Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The instruction is of 8-digit address-branch format.

HALT ON BREAKPOINT (HBK) - MACHINE CODE-48.

This instruction performs a mask test against a halt character in Program Reserved Memory and examines the execution digit in Systems Reserved Memory to determine the halt/proceed choice if the mask test is satisfied.

The format of this instruction is as follows:



There are no LABEL restrictions when using this instruction.

The BF field specifies an 8-bit breakpoint control mask. This mask is tested bit for bit against the program's breakpoint control character (Program Reserved Memory location 00046). If any 1 bit in the mask matches a 1 bit in the breakpoint control character, the breakpoint test is satisfied. The processor then examines the halt execution digit in Systems Reserved Memory location 00077 to determine what kind of halt should be executed. Undigit values may be used in the BF field, but the mask should be specified as two 4-bit digits, not a character.

The remaining fields are not used, nor are the comparison and OVERFLOW indicators changed by this instruction.

This instruction examines the breakpoint character in Program Reserved Memory location 00046.

See the HBR command for an explanation of the System Halt execution digit options. The program's breakpoint control character may be set by the BK console command while under control of the MCP.

If the breakpoint test is not satisfied, or if the System Halt execution digit specifies ignoring halts, this command functions like a NOP and control passes immediately to the next instruction in sequence.

This command should not be used to suspend a program temporarily when operating under the MCP because it halts all other programs as well. A DISP/ACPT instruction paired with an informative operator message is better. This instruction might be used in conjunction with an appropriate setting of the System Halt execution digit to cause an invalid instruction interrupt and thus end a program abnormally when unpassable errors occur.

HALT/BRANCH (HBR) - MACHINE CODE-29.

This instruction conditionally executes a halt and then transfers control to a specified address.

The format of the HBR instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	HBR			ERRPRT												

There are no LABEL restrictions when using this instruction.

The A ADDRESS specifies the instruction to which control is to be transferred after the machine has been started (if a halt occurred).

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

This instruction is of 8-digit address-branch format. The halt execution digit at absolute location 00077 (in Systems Reserved Memory) is tested to determine if the processor should halt. The possible values at this location are:

Digit	Action
0	All halts are executed.
1	All normal state halts are ignored. Control state halts are executed.
2	All control state halts are ignored. Normal state halts are executed.

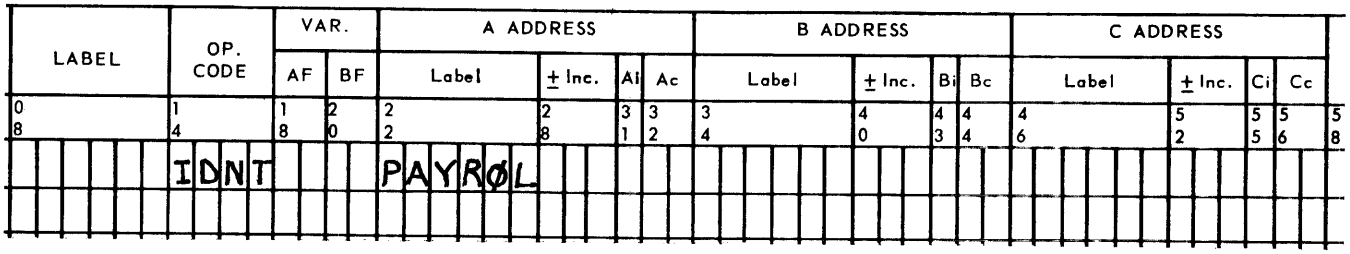
Digit	Action
3	All halts are ignored.
4	All halts are treated as invalid instructions (processor interrupt).
5	All normal state halts are ignored, and control state halts are considered as invalid instructions.
6	All control state halts are ignored, and normal state halts are considered as invalid instructions.
7	All halts are ignored.

While under control of the MCP, the System halt execution digit is set by means of the EX console command. The HBR command should not be used to suspend a program temporarily when operating under the MCP because it stops all other programs as well. Instead, a DISP/ACPT instruction paired with a message like TYPE GO WHEN READY is greatly preferred. Any I/O operations in progress when the halt occurs will be completed.

PROGRAM NAME (IDNT) - PSEUDO.

This instruction establishes a name for the program being assembled.

The format of this instruction is as follows.



LABEL entries are not permitted with this instruction.

The program name is declared in the A ADDRESS label field and must be made in the normal label form.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

While under control of the Advanced Assembler, IDNT is used only to provide a program name for the assembly listing. The file-name for the generated object program is obtained from the ? COMPILE card. IDNT is optional in the Advanced Assembler language.

In the Basic Assembler language, IDNT provides the file-name for the generated program and should be present if the object program is being generated to magnetic tape. IDNT is optional if the generated object program is going to be written into punched cards.

INITIATE I/O (IIO) - MACHINE CODE-94.

This instruction may not be used in normal state programs. It delivers an I/O descriptor to a specified peripheral control unit for initiation, then continues program execution.

The format for this instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
WRITAPIIØ			05	TDESCR											

There are no LABEL restrictions when using this instruction.

The BF field specifies the channel number whose peripheral control unit (PCU) is to receive the I/O descriptor. The PCU performs all subsequent operations involved with the I/O request.

The A ADDRESS field points to the I/O descriptor which is to be delivered to the channel's PCU. The final A ADDRESS must be even and its controller may only be UN or IA.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

All input/output operations are started by means of this instruction.

TWO-ADDRESS ADD (INC) - MACHINE CODE-01.

This 2-address instruction adds the contents of one location to another and stores the result into the second location.

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
INCR	INC	1	0	5	1			NLCØUNTR								

There are no LABEL restrictions when using this instruction.

The AF and BF fields specify the lengths of the incrementing and incremented fields. If the sum is longer (because of carrying) than the specified length of the incremented field, the result is <u>not</u> stored. Both fields retain their prior contents. If the two fields are of unequal length, the shorter field is assumed to be zero-filled until it is equal to the longer field.

The A ADDRESS points to the incrementing field. If the address-controller is UN or UA, the field sign is assumed to be positive. Only the numeric digits of an alphanumeric A field enter into the operation.

The B ADDRESS points to the incremented field and, if its address controller is UN or UA, the sign of the field is assumed to be positive. Only the numeric digits of an alphanumeric B field enter into the operation and the result sign is <u>not</u> stored back into it. The standard EBCDIC/USASCII form of the result sign is stored back into a SN field.

The remaining fields are not used by this instruction.

If the sum is too large to fit into the B field, the OVERFLOW indicator is set and the comparison indicators remain unchanged; otherwise, the comparison indicators are set according to the sum (even if the sign is not stored). The settings are as follows:

Setting	Sign
LOW	-
EQUAL	0
HIGH	+

Program Reserved Memory is not changed by this instruction.

The following are operational examples of the INC instruction.

Addition Of Unsigned Numeric To Signed Numeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
INC	02	04	FIELDA	FIELDB	

	BEFORE	AFTER
A ADDRESS	AX	AX
B ADDRESS	+0257	+0274
C ADDRESS		
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Addition With Overflow.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
INC	02	03	FIELD1	FIELD2	

	BEFORE	AFTER
A ADDRESS	18	18
B ADDRESS	985	985
C ADDRESS		
COMP. INDC.	unchanged	
OVERFLOW	xxx	ON

INITIATE I/O (INER) - PSEUDO.

This pseudo is defined for Basic Assembler only. It is used to initiate an I/O operation, with automatic entry to a user-coded I/O complete routine.

The format for the INER instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
READ	INER	9		MCRDSC				MCRCHN				PKTSELENDADR			

There are no LABEL restrictions for this instruction.

If the AF field contains a value between zero and four, the BCP invokes a USE routine at I/O complete time only if an exception occurred. If the AF value is between five and nine, the BCP always invokes a USE routine at I/O complete time.

A label referencing a descriptor is specified in the A ADDRESS label field. Indexing is not permitted.

A label referencing a 2-digit channel number is specified in the B ADDRESS label field. Indexing is not permitted.

The entry label of the user's I/O complete routine is specified in the C ADDRESS label field. A label referencing the area into which the BCP should store the ending address as specified in columns 52 through 57.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

When the I/O complete branch is executed, the Basic Control Program will lock out the servicing of I/O result descriptors of all other channels. In this case, the users program can initiate I/O commands for this channel and expect results back out. Normal processing of channel result descriptors is restored with the RTRN command. This exception branch will essentially give the programmer a non-interruptable normal state condition which is important where the I/O device timing is critical.

INDIRECT FIELD LENGTH (INFL) - PSEUDO.

This instruction causes the AF and/or BF fields of the machine instruction that immediately follows this command to be assembled as indirect field lengths, referring to locations declared in this command.

The format of this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	INFL							BASE	+38						

The A ADDRESS field entry converts into the AF indirect field-length address for the next instruction. It may be omitted if the AF indirect-length is not desired. The value in this field may be:

a. A label or program point, with increment or decrement permitted.

b. An unsigned 2- or 5-digit base-relative address.

The assembled value must be even and between 00000 and 00038 inclusive. The address controller may be UA or UN.

The B ADDRESS field entry converts into the BF indirect field-length address for the next instruction. It may be omitted if the BF indirect-length is not desired. Coding restrictions are the same as for the A field entry.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

INITIATE I/O OPERATION (INIT) - PSEUDO.

This pseudo is defined for Basic Assembler only. It is used to initiate an I/O operation. When the I/O operation is completed, no special handling is performed by the BCP if an exception occurred.

The format of the INIT instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
RDTAPEINIT				TDESC				TCHAN				ENDADR			

There are no LABEL restrictions when using this instruction.

A label referencing an I/O descriptor is specified in the A ADDRESS label field. The result descriptor will be stored, four digits to the left of this address. Indexing is not permitted.

A label referencing a 2-digit channel number is specified in the B ADDRESS label field. Indexing is not permitted.

A label referencing where the ending address is to be stored is specified in the C ADDRESS label field. The BCP will insert the I/O ending address in the referenced field as a 6-digit unsigned numeric value. The C ADDRESS label field may be blank if no I/O ending address is required. Indexing is not permitted.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

OBTAIN SYMBOLIC CHANNEL AND UNIT (IOCU) - PSEUDO.

This pseudo is defined for Basic Assembler only. It fetches the actual channel number and the first syllable of the I/O descriptor from a specified entry in the Symbolic Substitution Table.

The format of this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	IØCU06			TDESC				TCHAN				LØADSW			

There are no LABEL restrictions when using this instruction.

The AF field specifies a 2-digit symbolic device number for accessing the Symbolic Substitution Table.

The A ADDRESS field points to a 6-digit field into which the first syllable of the I/O descriptor is stored. The B ADDRESS field points to a 2-digit field into which the physical channel number is stored. Indexing is not permitted, and the address controller must be UN or IA for either field.

The C ADDRESS field points to a 1-digit area which functions as a logical switch. The Basic Control Program will store a 1 into this position if the specified Symbolic Substitution entry has been previously loaded, or a zero if it has not been loaded. Indexing is not permitted, and the address controller must be UN or IA.

The remaining fields are not used, by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

KEYA

DEFINE ASCENDING SORT KEY (KEYA) - DECLARATIVE.

This instruction is defined for Advanced Assembler only. It loads a constant (12 UN), at the current value of the location counter, which describes an ascending sort key. This value is used by the MCP sort intrinsic through the sort pseudo.

The format for the KEYA instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	KEYA		110001				UA	*							

There are no LABEL restrictions when using this instruction.

The VAR field specifies the length of the key. The maximum size permitted for a signed alphanumeric field is 50 characters; all others may have a length of 99.

The A ADDRESS field contains a 5-digit number, left-justified, which is the location of the key within the record. Permitted values are in the range 00000 to 99999. The A ADDRESS controller specifies the type of data on which to sort:

Code	Definition
UA	Unsigned alphanumeric.
SN	Signed numeric.
SA	Signed alphanumeric.
blank	Unsigned numeric.

An asterisk (*) in the B ADDRESS field indicates the final key definition.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

A maximum of 49 keys (total KEYA and KEYD declarations) may be defined. The SKEY declaration must directly precede the first KEYA or KEYD definition.

See figure 6-2 (page 6-162A) which illustrates a sample program using the SORT pseudo.

KEYD

DEFINE DESCENDING SORT KEY (KEYD) - DECLARATIVE.

This instruction is defined for Advanced Assembler only. It loads a constant (12 UN), at the current value of the location counter, which describes a descending sort key. This value is used by the MCP sort intrinsic through the sort pseudo.

The format for the KEYD instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	KEYD		3	0 0 0 3 0											
							S	A *							

There are no LABEL restrictions when using this instruction.

The VAR field specifies the length of the key. The maximum size permitted for a signed alphanumeric field is 50 characters; all others may have a length of 99.

The A ADDRESS field contains a 5-digit number, left-justified, which is the location of the key within the record. Permitted values are in the range 00000 to 99999. The A ADDRESS controller specifies the type of data on which to sort:

Code	Definition
UA	Unsigned alphanumeric.
SN	Signed numeric.
SA	Signed alphanumeric.
blank	Unsigned numeric.

An asterisk (*) in the B ADDRESS field indicates the final key definition.

The remaining fields are not used by this instruction. Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

A maximum of 49 keys (total KEYA and KEYD declarations) may be defined. The SKEY declaration must directly precede the first KEYA or KEYD definition.

See figure 6-2 (page 6-162A) which illustrates a sample program using the sort pseudo.

LEQ

BRANCH NOT GREATER (LEQ) - MACHINE CODE-23.

This instruction transfers control to the specified address if the comparison indicator is set to LOW or EQUAL.

The format of the LEQ instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	LEQ			AHTBL	+	42	IA									

There are no LABEL restrictions when using this instruction.

The A ADDRESS field is used to code any defined branch address, and indexing is unrestricted. The address controller bits are interpreted as a high-order digit of the branch address except that A/C 11_2 retains the indirect address meaning. Thus, a base-relative branch address up to 299998 can be assembled.

The remaining fields are not used by this instruction.

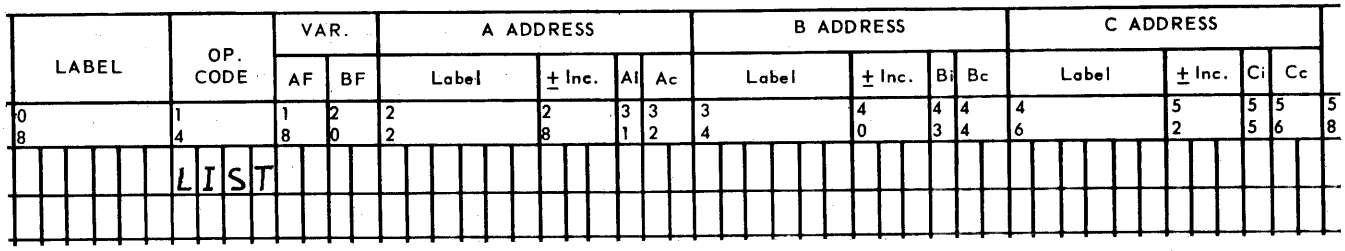
Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The instruction is of 8-digit address-branch format.

ENABLE PRINTED SOURCE LISTING (LIST) - PSEUDO.

This pseudo causes the printing of symbolic input to be started or resumed.

The format of the LIST instruction is:



LABEL entries are not permitted with this instruction. The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

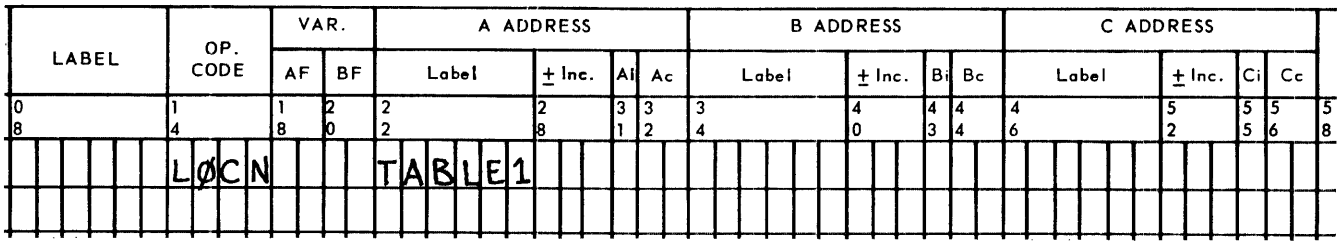
This pseudo has no operands and may be used in conjunction with, or instead of, the LIST option in the SPEC statement. The LIST card itself is printed.

LOCN

SET NEW LOCATION COUNTER VALUE (LOCN) - PSEUDO.

This pseudo sets the location counter to the value represented in the A ADDRESS label field.

The format of this instruction is:



LABEL entries are not permitted with this instruction.

The A ADDRESS label field may contain any of the following:

a. An unsigned 5-digit base-relative address.

b. A backward-refering label or program point (i.e., the label or program point has been defined in a prior statement). A ADDRESS increment may be used. Labels defined by EQIV statements may not be used here.

c. An asterisk reference (current value of the location counter), with the A ADDRESS increment specified.

The A ADDRESS controller specifies whether the A ADDRESS increment indicates digits or characters.

The remaining fields are not used by this instruction.

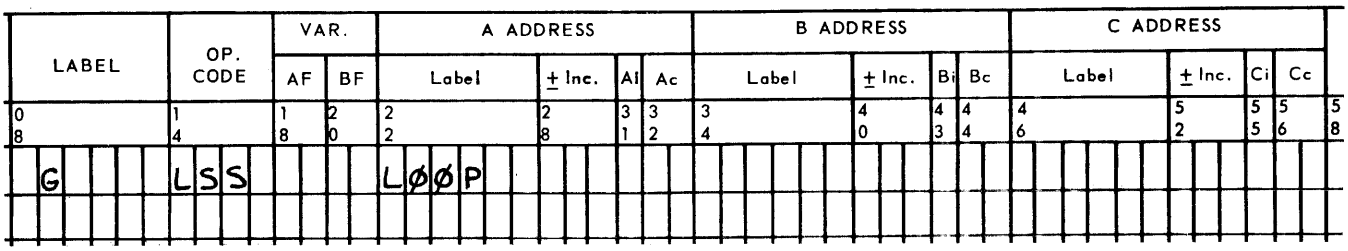
Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The current value of the location counter is saved before the new value is loaded to permit possible later reinstatement by a RLOC command.

BRANCH LESS (LSS) - MACHINE CODE-21.

This instruction transfers control to the specified address if the comparison indicator is set to LOW.

The format of this instruction is:



There are no LABEL restrictions when using this instruction.

Any defined branch address may be coded in the A ADDRESS field. Indexing is unrestricted. The address controller bits are interpreted as a high-order digit of the branch address, except that A/C 11_2 retains the indirect address meaning. Thus, a base-relative branch address up to 299998 can be assembled.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

This instruction is of 8-bit address-branch format.

MULTIPLY (MPY) - MACHINE CODE-05.

This instruction multiplies the value of the A field by that of the B field and stores the result in the C field.

The format for the MPY instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	MPY	3		525				NLPRINCP				INTRST			

There are no LABEL restrictions for this instruction.

The AF field specifies the length of the A field and the BF field does the same for the B field. The C field length is assumed to equal the sum of the A and B field lengths.

- The A ADDRESS field points to the multiplicand field and any address-controller is valid. UN fields are assumed to be positive. Only the numeric digits of a UA field enter into the multiplication and the field is assumed to be positive.

- The B ADDRESS field points to the multiplier field and the address-controller conventions are the same as for the A ADDRESS.

The C ADDRESS field points to the field into which the product is stored. If the address controller for this field is UN, the sign of the product is lost and the absolute value of the product is stored. If the address controller is UA, the sign of the product is lost, the absolute value of the product is stored into the numeric digits of the C field, and the zone digits are set to the numeric-subset zone.

The remaining fields are not used by this instruction.

The OVERFLOW indicator is not affected by this instruction. The comparison indicators are set according to the sign of the product. These settings are, LOW if negative, EQUAL if zero, and HIGH if positive.

Program Reserved Memory is not changed by this instruction.

The following are operational examples of the MPY instruction.

Multiply - Mixed Operands.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MPY	02	05	FIELDA(UA)	FIELDB(UN)	FIELDC(SN)

	BEFORE	AFTER
A ADDRESS	JK	JK
B ADDRESS	00011	00011
C ADDRESS	xxxxxxxx	+0000132
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Multiply - Signed Numeric Operands.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MPY	02	02	FIELD1(SN)	FIELD2(SN)	FIELD3(SN)

	BEFORE	AFTER
A ADDRESS	-15	-15
B ADDRESS	-17	-17
C ADDRESS	xxxxx	+0255
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

MUL

MULTIPLY (MUL) - MACHINE CODE-05.

This instruction is an alternate mnemonic for the MPY command (refer to the explanation of the MPY command).

The format of the MUL instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	MUL		3	475			N	LPRINCP				INTRST			

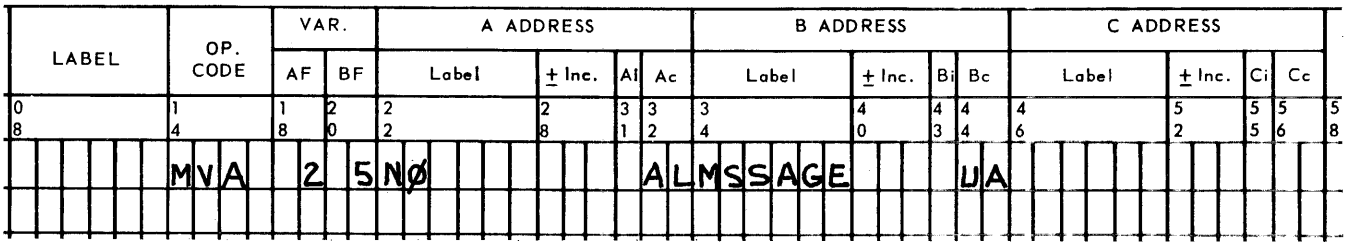
There are no LABEL restrictions when using this instruction.

The function of the remaining fields, Program Reserved Memory, comparison indicators and OVERFLOW indicators is identical to those of the MPY command.

MOVE ALPHANUMERIC (MVA) - MACHINE CODE-10.

This instruction moves digits or characters from one field to another, left-justified.

The format of this instruction is:



There are no LABEL restrictions when using this instruction.

The AF and BF fields specify the number of digits/characters in their respective fields. If the sending field is longer than the receiving field, the data is transmitted right-truncated. If the sending field is shorter than the receiving field, the excess low-order positions of the receiving field are filled with zeroes (UN or SN) or blanks (UA).

The A ADDRESS field points to the sending field and the B ADDRESS points to the receiving field. The remaining fields are not used by this instruction.

When the A and B fields are both of UN format, each digit is moved.

When the A field is UN and the B field is of SN format, each digit is moved and the sign of the B field is set to the standard EBCDIC/USASCII positive-sign code.

When the A field is UN and the B field is of UA format, each digit is moved and the zone digits of the receiving field are set to the EBCDIC/USASCII numeric-subset code.

When the A field is SN and the B field is of UN format, each digit is moved and the A field sign is not moved.

When the A and B fields are both of SN format, each digit is moved and the sign of the B field is set to the standard EBCDIC/USASCII form of the A field sign.

When the A field is SN and the B field is of UA format, each digit is moved. The zone digits of the receiving field are set to the E/A numeric-subset code except that the zone digit of the most-significant character in the B field receives the standard EBCDIC/USASCII form of the A field sign.

When the A field is UA and the B field is of UN format, each numeric digit of the A field is moved and the A field zone digits are ignored.

When the A field is UA and the B field is of SN format, each numeric digit of the A field is moved. The A field zone digits are ignored, except that the sign of the B field is set to the standard EBCDIC/USASCII form of the sign contained in the zone digit of the most-significant A field character.

When the A and B fields are both of UA format, each character is moved.

If the length of the A field is greater than the length of the B field, the OVERFLOW indicator is set to on.

The comparison indicators are set to HIGH if the numeric digits moved are non-zero, and:

a. The A field is UN, or

b. The A field is SN and signed positive, or

c. The A field is UA and the B field is UN or UA, or

d. The A field is UA, the B field is SN, and the high-order zone digit of the A field is positive.

The comparison indicator is set to EQUAL if the numeric digits moved are all zero. It is set to LOW if the numeric digits moved are non-zero and:

a. The A field is SN and signed negative, or

b. The A field is UA, the B field is SN, and the high-order digit of the A field is negative.

Program Reserved Memory is not changed by this instruction.

The following are operational examples of the MVA instruction.

Move Alphanumeric - Unsigned Numeric To Signed Numeric Causing Overflow.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVA	5	3	FIELD1(UN)	FIELD2(SN)	

	BEFORE	AFTER
A ADDRESS	23511	23511
B ADDRESS	xxxx	+235
C ADDRESS		
COMP. INDC.	xxx	HIGH
OVERFLOW	xxx	ON

Move Alphanumeric - Signed Numeric To Unsigned Alphanumeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVA	3	3	FIELDA(SN)	FIELDB(UA)	

	BEFORE	AFTER
A ADDRESS	+823	+823
B ADDRESS	xxx	H23
C ADDRESS		
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Move Alphanumeric - Unsigned Alphanumeric to Signed Numeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVA	3	5	FIELDX(UA)	FIELDZ(SN)	

	BEFORE	AFTER
A ADDRESS	MNO	MNO
B ADDRESS	xxxxxx	-45600
C ADDRESS		
COMP. INDC.	xxx	LOW
OVERFLOW	unchanged	

Move Alphanumeric - Unsigned Alphanumeric to Unsigned Alphanumeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVA	3	5	FIELDJ(UA)	FIELDK(UA)	

	BEFORE	AFTER
A ADDRESS	XYZ	XYZ
B ADDRESS	xxxxx	XYZbb
C ADDRESS		
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

MOVE AND CLEAR WORDS (MVC) - MACHINE CODE-13.

This instruction moves words (refer to Data Formats) from one field to another and clears the sending field to numeric zeroes after the move has been performed.

The format of this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	MVC	0040		TABLE1				TABLEA							

There are no LABEL restrictions with this instruction.

The VAR field specifies the 4-digit count of words to be moved and cleared. If this field is blank, the word count is generated from the length associated with the A ADDRESS label. Generation is as follows:

Label	Action
Alphanumeric	The length is divided by two. A non-zero remainder will cause a syntax error.
Unsigned numeric	The length is divided by four. A non-zero remainder will cause a syntax error.

The A ADDRESS field points to the sending field. This address must be evenly divisible by four (modulo-4) after all indexing and indirect addressing have been applied or a processor interrupt will occur at execution time. The address controller for this field must be UN or UA if the word count is to be calculated. If the word count is specified, the address controller may be SN. The hardware does not use the address controller, and literals are not permitted.

The B ADDRESS field points to the receiving field. Coding restrictions on modulo-4 and the address controllers are the same as for the A ADDRESS field.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

After all data has been moved, the A field is cleared to an all-zero bit pattern regardless of the data format.

The following is an operational example of the MVC instruction.

Normal Move And Clear.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVC	00	04	RECA(UN)	RECB(UA)	

	BEFORE	AFTER
A ADDRESS	C1C2C3C4C5C6C7C8	0000000000000000
B ADDRESS	xxxxxxxx	ABCDEFGH
C ADDRESS	———	———
COMP. INDC.		unchanged
OVERFLOW		unchanged

MOVE LINKS (MVL) - MACHINE CODE-09.

This instruction rotates the contents of the three data fields to the left. The contents of the B field are moved to the A field, the contents of the C field to the B field, and the contents of the A field to the C field.

The format of the MVL instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
LISTUP	MVL	05		ATAG				BLABEL				CNAME			UA

There are no LABEL restrictions for this instruction.

The AF field provides the length of all the fields and the BF field is ignored by the hardware.

The A ADDRESS field points to the first field to be rotated. Its address controller must be UN or UA and must agree with the address controllers of both the B and C fields.

The B ADDRESS field points to the second field to be rotated. Its address controller must be UN or UA and must agree with the address controllers of both the A and C fields.

The C ADDRESS field points to the third field to be rotated. Its address controller must be UN or UA and must agree with the address controllers of both the A and B fields.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The hardware picks up the C address controller and controls all data formatting. The A and B address controllers are not interpreted by the hardware and they are assumed to be identical to the C address controller.

The following are operational examples of the MVL instruction.

Move Links - Unsigned Numeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVL	05		ATAG(UN)	BTAG(UN)	CTAG(UN)

	BEFORE	AFTER
A ADDRESS	12345	67890
B ADDRESS	67890	ABCDE
C ADDRESS	ABCDE	12345
COMP. INDC.	unchanged	
OVERFLOW	unchanged	

Move Links - Unsigned Alphanumeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVL	03		AAREA(UA)	BAREA(UA)	CAREA(UA)

	BEFORE	AFTER
A ADDRESS	XYZ	MNO
B ADDRESS	MNO	GHI
C ADDRESS	GHI	XYZ
COMP. INDC.	unchanged	
OVERFLOW	unchanged	

MOVE NUMERIC (MVN) - MACHINE CODE-11.

The function of this instruction is to move digits or characters from one field to another, right-justified.

The format of the Move Numeric (MVN) instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	MVN	3		502				NLPØINTR							

There are no LABEL restrictions when using this instruction.

The AF and BF fields specify the number of digits/characters in the A and B fields. If the B field is longer than the A field, the excess high-order positions of the B field are filled with numeric or alphanumeric zeros. If the B field is shorter than the A field, the excess high-order positions of the A field are scanned and, if any of these positions contain a non-zero digit or character, the move will not be performed. If all the excess A field positions are zero, the A field data will be moved (left-truncated) to the B field.

The A ADDRESS points to the field from which data is moved, the B ADDRESS points to the field into which the data is moved, and the C ADDRESS is not used.

When the A and B fields are both of UN format, each digit is moved.

When the A field is UN and the B field is of SN format, each digit is moved and the sign of the B field is set to the EBCDIC/USASCII positive-sign code.

When the A field is UN and the B field is of UA format, each digit is moved and the zone digits of the B field characters are set to the EBCDIC/USASCII numeric-subset code.

When the A field is SN and the B field is of UN format, each digit is moved and the sign of the A field is not moved.

When the A and B fields are both of SN format, each digit is moved and the B field sign is set to the standard EBCDIC/USASCII form of the A field sign.

When the A field is SN and the B field is of UA format, each digit is moved. The zone digits of the B field characters are set to the EBCDIC/USASCII numeric-subset code, except that the zone digit of the most-significant B field character receives the standard EBCDIC/USASCII form of the A field sign.

When the A field is UA and the B field is of UN format, each digit of the A field is moved and the A field zone digits are ignored.

When the A field is UA and the B field is of SN format, each numeric digit of the A field is moved and the sign of the B field is set to the most standard EBCDIC/USASCII form of the sign in the zone digit of the most-significant A field character.

When the A and B fields are both of UA format, each numeric digit of the A field is moved and the zone digits of the B field characters are set to the standard EBCDIC/USASCII numeric-subset code.

If the A field is longer than the B field and the excess high-order positions of the A field contain non-zero digits/characters, the OVERFLOW indicator is turned on. If the OVERFLOW indicator is turned on, the comparison indicators are not changed by the instruction.

The HIGH comparison indicator is turned on if the numeric data moved is non-zero and:

a. The A field is UN, or
b. The A field is SN and signed positive, or
c. The A field is UA and the B field is UN or UA, or
d. The A field is UA, the B field is SN, and the high-order zone digit of the A field is positive.

The EQUAL comparison indicator is turned on if the numeric data moved is zero. The LOW comparison indicator is turned on if the numeric data moved is non-zero and:

a. The A field is SN and signed negative, or
b. The A field is UA, the B field is SN, and the high-order zone digit of the A field is negative.

Program Reserved Memory of the object program is not used or changed by this instruction.

Operational examples of the Move Numeric instruction are as follows:

Unsigned Numeric To Unsigned Numeric Move With B Field Shorter.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVN	5	3	FIELDA(UN)	FIELDB(UN)	

	BEFORE	AFTER
A ADDRESS	00123	00123
B ADDRESS	xxx	123
C ADDRESS		
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Unsigned Numeric To Unsigned Numeric Move Causing Overflow.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVN	5	3	FIELDA(UN)	FIELDB(UN)	

	BEFORE	AFTER
A ADDRESS	12300	12300
B ADDRESS	unchanged	
C ADDRESS		
COMP. INDC.	unchanged	
OVERFLOW	xxx	ON

Unsigned Numeric To Unsigned Numeric Move With B Field Longer.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVN	3	5	FIELDC(UN)	FIELDE(UN)	

	BEFORE	AFTER
A ADDRESS	123	123
B ADDRESS	xxxxx	00123
C ADDRESS		
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Unsigned Alphanumeric To Signed Numeric Move.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVN	3	5	FIELDA(UA)	FIELDB(SN)	

	BEFORE	AFTER
A ADDRESS	RYZ	RYZ
B ADDRESS	xxxxxx	D00989*
C ADDRESS		
COMP. INDC.	xxx	LOW
OVERFLOW		

* D is negative-sign undigit

Unsigned Alphanumeric To Un-
Signed Alphanumeric Move.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVN	5	7	FIELDA(UA)	FIELDB(UA)	

	BEFORE	AFTER
A ADDRESS	bbbbb	bbbbb (blanks)
B ADDRESS	xxxxxxx	0000000
C ADDRESS		
COMP. INDC.	xxx	EQUAL
OVERFLOW	unchanged	

MOVE REPEATED (MVR) - MACHINE CODE-14.

This instruction moves digits/characters from one field to another, and repeats the move into successive receiving locations a specified number of times.

The format for the MVR instruction is.

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	MVR		120A					ALMSSAGE								

There are no LABEL restrictions when using this instruction.

The AF field specifies the length of the sending field and the BF field specifies the number of times the move is to be performed. The BF field may not be left blank.

The effective length of the receiving field is the product of the AF field times the BF field (the number of digits/characters moved multiplied by the number of times the move is performed).

The A ADDRESS field specifies the sending field and its address controller may be UN or UA.

The B ADDRESS field specifies the receiving field and its address controller may be UN or UA, independent of the format of the A field.

The following are various address controller conventions and their corresponding move description for the A and B fields:

A ADDRESS	B ADDRESS	Move Description
UN	UN	Each digit is moved.
UN	UA	Each sending digit is moved and the EBCDIC/USASCII numeric subset

A ADDRESS	B ADDRESS	Move Description
		zone digit is inserted in each receiving character.
UA	UN	Only the numeric digits of the sending field are moved to the receiving field.
UA	UA	Each character is moved.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The following are operational examples of the MVR instruction.

Move Repeated - Unsigned Numeric To Unsigned Numeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVR	3	4	ADF(UN)	CGT(UN)	

	BEFORE	AFTER
A ADDRESS	057	057
B ADDRESS	xxxxxxxxxxxx	057057057057
C ADDRESS		
COMP. INDC.	unchanged	
OVERFLOW	unchanged	

Move Repeated - Unsigned Alphanumeric To Unsigned Numeric.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVR	3	2	TAG(UA)	NUMTAG(UN)	

	BEFORE	AFTER
A ADDRESS	MNO	MNO
B ADDRESS	xxxxxx	456456
C ADDRESS		
COMP. INDC.	unchanged	
OVERFLOW	unchanged	

MVW

MOVE WORDS (MVW) - MACHINE CODE-12.

This instruction moves words (refer to Data Formats) from one location to another.

The format of the MVW instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
CØPY	MVW	0040		CARDIN				CARDØT								

There are no LABEL restrictions for this instruction.

The VAR field specifies the 4-digit count of words to be moved. If the variant field is blank, the word count is generated from the length associated with the A ADDRESS label. If the label is alphanumeric, its length is divided by two and a non-zero remainder on the divide causes a syntax error. If the label is unsigned numeric, its length is divided by four and a non-zero remainder causes a syntax error.

The A ADDRESS field points to the sending field and this address must be evenly divisible by four (modulo-4) after all indexing and indirect addressing has been applied. If not, a processor interrupt will occur at execution.

The address controller for this field must be UN or UA if the move length is to be calculated. If the word count is specified, it may be SN. The address controller is not used by the hardware. Literals are not permitted.

The B ADDRESS field points to the receiving field. The coding restrictions on modulo-4 and the address controllers are the same as for the A ADDRESS field.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

This is a high-speed data move. It is twice as fast as MVA on fields of the same length and can move much longer fields with a single instruction execution.

The following are operational examples of the MVW instruction.

Move Words - Numeric Fields.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVW	00	04	FIELDA(UA)	FIELDB(UN)	

	BEFORE	AFTER
A ADDRESS	01020304	01020304
B ADDRESS	xxxxxxxx	01020304
C ADDRESS		
COMP. INDC.	unchanged	
OVERFLOW	unchanged	

Move Words - Alphanumeric Fields.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
MVW	00	03	FIELDC(UA)	FIELDE(UA)	

	BEFORE	AFTER
A ADDRESS	VWXYZA	VWXYZA
B ADDRESS	xxxxxx	VWXYZA
C ADDRESS		
COMP. INDC.	unchanged	
OVERFLOW	unchanged	

NEQ

BRANCH NOT EQUAL (NEQ) - MACHINE CODE-25.

This instruction transfers control to the specified address if the comparison indicator is set to HIGH or LOW.

The format of this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	NEQ			UNEQL												

There are no LABEL restrictions for this instruction.

The A ADDRESS field is used to code any defined branch address. Indexing is unrestricted, and the address controller bits are interpreted as a high-order digit of the branch address, except that A/C 11_2 retains the indirect address meaning. Thus, a base-relative branch address up to 299998 may be assembled.

The remaining fields are not used by this instruction.

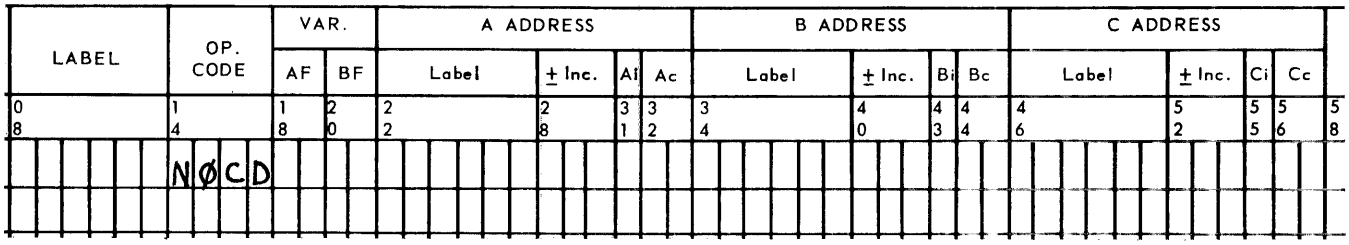
Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

This instruction is of 8-digit address-branch format.

DISABLE PRINTED OBJECT LISTING (NOCD) - PSEUDO.

This pseudo instruction inhibits printing of the assembled machine code.

The format of this instruction is:



LABEL entries are not permitted with this instruction. The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

This instruction has no operands. Printing of assembled machine code may be resumed with the pseudo instruction CODE.

NOLI

DISABLE PRINTED SOURCE LISTING (NOLI) - PSEUDO.

This instruction inhibits the printing of symbolic input.

The format of the NOLI instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	NØLI														

LABEL entries are not permitted for this instruction. The remaining fields are not used by this instruction.

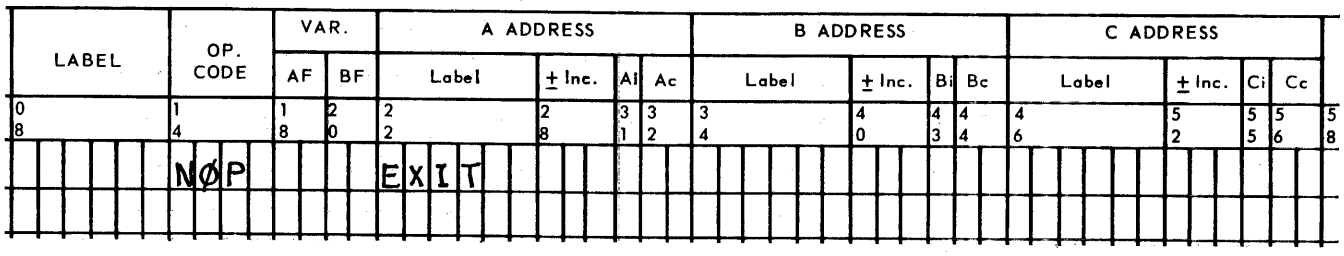
Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

This command has no operands. The NOLI card itself is printed. Printing of symbolic input may be resumed with the pseudo instruction LIST.

NO OPERATION (NOP) - MACHINE CODE-20.

This instruction causes control to pass to the next sequential instruction.

The format of the NOP instruction is:



There are no LABEL restrictions when using this instruction.

Any address may be coded into the A ADDRESS field. The address controller bits are interpreted as a high-order digit of the address, except that 11_2 retains the indirect address meaning, thus address up to 299999 base-relative may be assembled.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The NOP instruction may be used as a program switch. Since this instruction is of 8-digit address-branch format, it can only be converted into a conditional/unconditional branch or an EXT command, and not into a data handling instruction.

NOT

LOGICAL EXCLUSIVE OR (NOT) - MACHINE CODE-44.

This instruction compares the A field bits with the corresponding B field bits and stores a 1 bit into the C field position if the A and B bits are not equal. Otherwise, a 0 bit is stored into the field.

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	NØT	02	03	ASWIT				BSWIT				CØMSWT			

There are no LABEL restrictions when using this instruction.

The AF and BF fields are used to specify the number of digits/characters in their corresponding fields. The C field length is assumed to be equal to the larger of the AF and BF values. If the A and B fields are not of equal length, the shorter of the two is padded by assuming trailing 1-bit digits/characters (F or FF).

The A ADDRESS field points to the first field to be Exclusively ORed. The address controller may not be SN and the final controller must be the same as the final address controller for the B and C fields.

The B ADDRESS field points to the second field to be Exclusively ORed. The address controller may not be SN and the final controller must be the same as the final A and C field address controllers.

The C ADDRESS field points to the field into which the result is to be stored. The address controller may not be SN and the final controller must be the same as the final address controllers for the A and B fields.

The remaining fields are not used by this instruction.

The comparison indicator is set to HIGH if the last result bit (i.e., the low-order bit of the last C field digit/character) is 1. The comparison indicator is set to EQUAL if the last result bit is 0. The OVERFLOW indicator is not changed by this instruction.

Program Reserved Memory is not changed by this instruction.

The NOT instruction processes bit string's with no implicit conversion between formats. For example:

Operand Bits		Result Bit
A field	B field	C field
0	0	0
0	1	1
1	0	1
1	1	0

The following are operational examples of the NOT instruction.

Exclusive OR - Unsigned Numeric Operands.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
NOT	03	03	INVERT(UN)	FIELD(UN)	IFIELD(UN)

	BEFORE	AFTER	
A ADDRESS	FFF	FFF	111111111111
B ADDRESS	6A1	6A1	011010100001
C ADDRESS	xxx	95E	100101011110
COMP. INDC.	xxx	EQUAL	
OVERFLOW	unchanged		

Exclusive OR - Unsigned Alphanumeric Operands.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
NOT	02	02	PLUS(UA)	CHARS(UA)	PCHARS(UA)

	BEFORE	AFTER	
A ADDRESS	&&	&&	0101000001010000
B ADDRESS	GP	GP	1100011111010111
C ADDRESS	xx	pg	1001011110000111
COMP. INDC.	xxx	HIGH	
OVERFLOW	unchanged		

NTR

ENTER SUBROUTINE (NTR) - MACHINE CODE-31.

This instruction causes control information and parameters to be copied into the subroutine stack, and transfers control to a specified address.

The format for the NTR instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	NTR			FSQRT											

There are no LABEL restrictions when using this instruction.

The VAR field should be left blank. The assembler calculates a 4-digit count of characters in the parameters to be transferred to the subroutine and puts this count into the variant field of the assembled machine instruction. The assembler assumes that all CNST and ACON statements immediately following the NTR instruction are parameters.

The A ADDRESS label field specifies the address to which control is to be transferred after the parameters have been copied into the subroutine stack.

The remaining fields are not used by this instruction.

The settings of the comparison and OVERFLOW indicators are stored, then cleared by this instruction.

The address contained in Program Reserved Memory location 00040 is assumed to point to the first available location in the subroutine stack. This address must be initialized by the program, before the first NTR is executed, to point to an area (which must be modulo-2 oriented) within the program that is large enough to hold the maximum

expected nesting of subroutine calls. The following information is copied into the location specified by this address:

Digit	Information
0-5	The return address, i.e., the address of the instruction which follows the parameters.
6-13	The current setting of IX3.
14	Zero.
15	The current settings of the comparison, OVERFLOW, and EBCDIC/USASCII indicators. Bit 8: 1 = USASCII 0 = EBCDIC Bit 4: 1 = OVERFLOW 0 = NO OVERFLOW Bits 2 and 1: 1 = High, 2 = Low, 3 = Equal

The remaining digits of the stack entry are the parameters following the NTR instruction.

The current value of the address in Program Reserved Memory location 00040 is then copied into IX3, and the new value of "next available location in subroutine stack" is copied into location 00040.

When coding subroutine references to the parameters, note that the first parameter character/digit is located at "contents of IX3 plus 16", since the IX3 value points to the beginning of the entire current stack entry and linkage information occupies the first 16 positions of the entry. This is shown in the example below.

OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ADDRESS ± Inc.	A ADDRESS Ai	A ADDRESS Ac	B ADDRESS Label	B ADDRESS ± Inc.	B ADDRESS Bi	B ADDRESS Bc	C ADDRESS Label	C ADDRESS ± Inc.	C ADDRESS Ci	C ADDRESS Cc	
1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8 6 16
MVA	3	3	BASE	+8	3	UAC	ØDE								ALPHA REF 1
MVN	5	5	BASE	+163		UNC	ØUNT								NUM REF 1

Subroutines may call themselves (recurse) to any depth permitted by the size of the area allocated for the subroutine stack. Subroutines return to the level that called them by means of the EXT instruction. This instruction requires that the value loaded into location IX3 by the corresponding NTR instruction be still present. Therefore, any subroutine which uses IX3 for processing must save its value before altering it and restore the saved value before exiting.

EXAMPLE FORMAT:

```
001560  NTR   SUBRTN  (at location 003016, assembled 310003020166)
001570  ACON  ARGUMT  (at location 003028, assembled 203010)
```

Assume that before execution of the NTR instruction, IX3 contains +0000010 and 00040 contains 001024. After execution of the NTR instruction, the field contents will be:

next instruction address:	020166
IX3:	+0001024
00040:	001046
001024:	003034 (instruction address after parameters)
	+0000010 (stored value of IX3)
	00 (stored value of machine indicators)
	203010 (ACON parameter)

DECLARE EXTENDED NUMERIC CONSTANT (NUMR) - DECLARATIVE.

This instruction permits the declaration of a numeric constant of up to 59 digits, and it is defined for Advanced Assembler only.

The format for the NUMR instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
08	14	18	20	22	28	31	32	34	40	43	44	46	52	55	56
EXTRA	NUMR		31												

There are no LABEL restrictions when using this instruction.

The BF field specifies the number of digits to be allocated for the constant; a range of 01 through 59 is acceptable. If omitted, the Assembler will assume 59.

The constant data is specified, left-justified, in columns 22-80.

Any non-numeric character other than A through F will have the zone bits removed, and a number will be assembled.

The remaining fields are not used by this instruction. Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

BRANCH ON OVERFLOW (OFL) - MACHINE CODE-28.

This instruction transfers control to the specified address providing the OVERFLOW indicator is set to on.

The format of the OFL instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
TSTØVFØFL				ØVRRUN											

There are no LABEL restrictions when using this instruction.

Any defined branch address may be coded in the A ADDRESS field. Indexing is unrestricted, and the address controller bits are interpreted as a high-order digit of the branch address, except that A/C 11_2 retains the indirect address meaning. Thus, a base-relative branch address up to 299998 may be assembled.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison indicators are not changed by this instruction. The OVERFLOW indicator is reset by this instruction.

This instruction is of 8-digit address-branch format.

OPEN

OPEN FILE (OPEN) - PSEUDO.

This pseudo initializes a logical file for reading or writing (including device assignment, label creation, etc.).

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
START	ØPENIN			CARDIN												

There are no LABEL restrictions when using this instruction.

The AF field specifies the direction of data flow on the file. Coding is:

Code	Description
IN	Input.
OT	Output.
IO	Input/output (Advanced Assembler only - disk file updating).
OI	Output/input (Advanced Assembler only - disk file updating).

A file may be OPENed in one direction, processed, CLOSed, and then OPENed in the opposite data-flow direction.

Column 20 of the BF field provides extra information pertaining to magnetic tape files and disk files. Coding is as follows:

Code	Description
blank	Rewind at OPEN.
N	No rewind.

Code	Description
R	OPEN for reverse reading (Magnetic tape).

Column 21 of the BF field specifies the MICR sorter-reader processing mode. Coding is:

Code	Description
D	Demand } (Advanced Assembler only).
F	Flow } (Advanced Assembler only).

For a disk file, an F in column 21 specifies an OPEN WITH LOCK (Advanced Assembler only). This allows the user to have unrestricted INPUT and OUTPUT capabilities on the specified file while restricting subsequent users of that file to INPUT activity only.

The A ADDRESS field contains the internal file-name (as defined in the label field of a FILE statement) of the file to be OPENed. Incrementing, indexing, and address controlling are not permitted.

The B ADDRESS field (in the Basic Language) may optionally contain a left-justified 2-digit reference to the symbolic substitution table. The BCP will obtain a channel number and an I/O descriptor syllable from this table, and override any entry in the FILE statement. Refer to the Basic I/O Section.

NOTE

The Advanced Assembler ignores this field.

The C ADDRESS field (in the Basic Assembler) may optionally contain an error branch address. Control branches to this address if the BCP is unable to recover from an I/O error on the file.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

A logical file must be OPENed before any I/O commands (READ, WRIT, POSN, SEEK, etc.) are issued to it.

LOGICAL OR (ORR) - MACHINE CODE-43.

This instruction compares the A field bits with corresponding B field bits and stores a 1 bit into the C field if either or both of the corresponding A and B field bits are on.

The format of the ORR instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
.1	ØRR	2		10				NLSWITCH				UNSWITCH			UN

There are no LABEL restrictions when using this instruction.

The AF and BF fields specify the number of digits/characters in their corresponding fields. The C field length is assumed to equal the larger of the AF and BF values.

If the A and B fields are not equal length, the shorter of the two is padded in the processor by assuming trailing 0-bit digits or characters (0 or 00).

The A ADDRESS field points to the first field to be ORed and its address controller may not be SN. The final A address controller must be the same as the final B and C address controllers.

The B ADDRESS field points to the second field to be ORed and its address controller may not be SN. The final B address controller must be the same as the final A and C address controllers.

The C ADDRESS field points to the field into which the ORed result is to be stored and its address controller may not be SN. The final C address controller must be the same as the final A and B address controllers.

The remaining fields are not used by this instruction.

The comparison indicator is set to HIGH if the last result bit (i.e., the low-order bit of the last C field digit/character) is 1. It is set to EQUAL if the last result bit is 0. The OVERFLOW indicator and Program Reserved Memory are not changed by this instruction.

This instruction processes bit strings, with no implicit conversion between formats. This is shown in the example below.

Operand Bits		Result Bit
A field	B field	C field
0	0	0
0	1	1
1	0	1
1	1	1

The following are operational examples of the ORR instruction.

Logical OR - Unsigned Numeric Operands.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
ORR	02	03	BITS(UN)	SWITCH(UN)	SWITCH(UN)

	BEFORE	AFTER	
A ADDRESS	81	81	100000010000
B ADDRESS	223	223	001000100011
C ADDRESS	xxx	A33	101000110011
COMP. INDC.	xxx	HIGH	
OVERFLOW	unchanged		

Logical OR - Unsigned Alphanumeric Operands.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
ORR	03	02	CHAR1(UA)	CHAR2(UA)	FIELD(UA)

	BEFORE	AFTER	
A ADDRESS	ABD	ABD	110000011100001011000100
B ADDRESS	23	23	111100101111001100000000
C ADDRESS	xxx	33D	111100111111001111000100
COMP. INDC.	xxx	EQUAL	
OVERFLOW	unchanged		

CALL AND ENTER SUBSEQMENT (OVLY) - PSEUDO.

This pseudo is defined for Advanced Assembler only. It tests for the presence of a subsegment, loads it if not present, and transfers control into the subsegment.

The format of the OVLY instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
ENDUP	ØVLY			RECAP												

There are no LABEL restrictions when using this instruction.

The A ADDRESS label field contains the name of the called subsegment, as defined in a SEGM card. The called segment must be an immediate subsegment (nested and one level down) of the segment which contains the OVLY command.

The remaining fields are not used by this instruction.

The previous comparison indicator setting is eliminated by this instruction and the new value is unpredictable.

Program Reserved Memory and the OVERFLOW indicator is not changed by the instruction.

Control enters the subsegment at its first executable instruction <u>unless</u> a label was specified in the ENSG card of the subsegment, in which case control enters at that label. Refer to the discussion of segmentation (section 4), and the description of the SEGM pseudo-operation.

DECLARE COBOL-FORM EDIT MASK (PICT) - DECLARATIVE.

This declarative is defined for Advanced Assembler only. It converts the specified COBOL PICTURE into a string of editing micro-operators.

The format of the PICT instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
EDIT1	PICT		12					Z(5),ZZZ.99+							

The LABEL entry may not be a program point.

The VAR field specifies the number of characters (right-justified) in the COBOL PICTURE (columns 20 through 21). This value is <u>not</u> related to either the length of the field to be edited or the resultant length of the edited field. It also is <u>not</u> related to the length of the converted micro-operator string (the length of the string is the value associated with the PICT label). The maximum length of the COBOL PICTURE is 24 characters.

The COBOL PICTURE is coded in the B and C ADDRESS fields.

There are no restrictions other than the 24-character length limit.

Refer to Section 2, for a full explanation of PICTURE usage.

The remaining fields are not used by this instruction.

The comparison and OVERFLOW indicators are not changed by this instruction.

The standard Edit insertion characters (+, -, *, ., ', $, 0, b) are suitably generated in the Reserved Memory Locations 48-63.

If a PICT declarative is used to define the edit mask for an EDT instruction (this is highly recommended), the following coding rules apply to the EDT command:

a. The label of the PICT command should be coded in the B ADDRESS label field.

b. The increment, index, and address controller fields for the B ADDRESS field should be left blank.

c. The BF field of the EDT instruction should be left blank.

POSITION EXTERNAL FILE (POSN) - PSEUDO.

This pseudo causes forward spacing of the printer paper (both Assemblers) or forward/backward spacing on magnetic or paper tape (Advanced Assembler only).

The format of this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
SKIP	POSN	0005	TAPEIN					EWDTP								

There are no LABEL restrictions for this instruction.

For magnetic tape, the VAR field specifies the number of BLOCKS to be skipped (permitted values are 0001 to 9999 and -001 to -999). For example, if columns 18-21 are coded 0010, the tape will be skipped forward 10 full blocks and positioned to read the eleventh block from the point at which the POSN was issued. Similarly, if columns 18-21 are coded -999, the tape will be skipped backward 999 blocks and positioned to read/write the 999th block from the point at which the POSN was issued.

For the printer, the AF field specifies the number of lines to be spaced. Permitted values when using the Advanced Assembler are 01 to 99; when using the Basic Assembler, code only 01 or 02. The BF field specifies a channel number for the printer control tape. For example, if columns 20-21 are coded 09, the carriage control tape will be skipped to channel 9. Permitted values are 01 to 11.

The A ADDRESS label field specifies the internal file-name of the file to be positioned. Incrementing, indexing, and address controlling are not permitted. The B ADDRESS label field optionally specifies an End-of-Volume branch address.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

READ ADDRESS (RAD) - MACHINE CODE-92.

This is a privileged instruction and may not be used in normal state programs. It reads a 6-digit address from scratch pad memory and stores it at a specified location.

The format of this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	RAD	10	5	TAPEND												

There are no LABEL restrictions when using this instruction.

The AF field (column 19) specifies whether the first or second address (beginning or ending address of an I/O descriptor) is to be read out. Coding is:

Code	Action
0	Read out first address.
1	Read out second address.

The BF field specifies a channel number (from 00 to 19). The scratch pad address will be read out of the corresponding slot.

The A ADDRESS field specifies the location into which the address value is to be stored, and the final address must be even. The address controller may only be UN or IA.

The remaining fields are not used by this instruction.

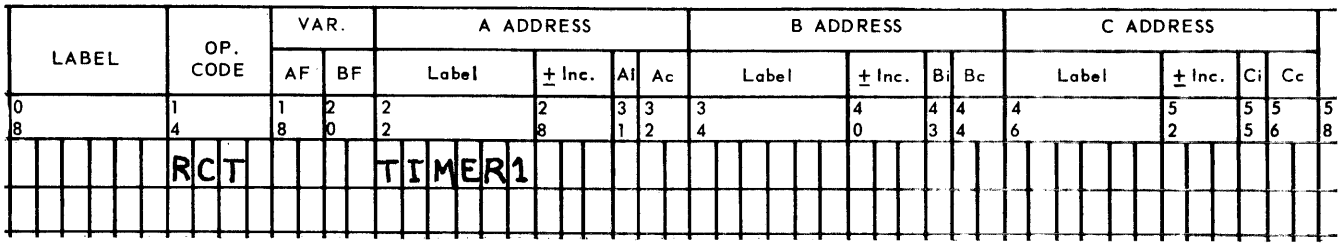
Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

This command is used to determine the ending address of I/O descriptors (short record read etc.).

READ AND CLEAR TIMER (RCT) - MACHINE CODE-96.

This is a privileged instruction and may not be used in normal state programs. It transfers the contents of the first timer word from scratch pad memory to a specified storage address, and then clears the word to zeroes.

The format of the RCT instruction is:



There are no LABEL restrictions when using this instruction.

The A ADDRESS field points to a 6-digit area into which the timer word is to be stored, and its address controller must be either UN or IA. The final A ADDRESS must be even.

The remaining fields are not used, nor are Program Reserved Memory or the comparison and OVERFLOW indicators changed by this instruction.

The first timer word is the one which is automatically incremented by the hardware.

READ TIMER (RDT) - MACHINE CODE-95.

This is a privileged instruction and it may not be used in normal state programs. It transfers the contents of the first timer word from scratch pad memory to a specified address.

The format of the RDT instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	RDT			ELAPSE											

There are no LABEL restrictions for this instruction.

The A ADDRESS field points to a 6-digit area into which the timer word is to be stored, and its address controller must be UN or IA. The final A ADDRESS must be even.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

The first timer word is the one which is automatically incremented by the hardware (real-time clock). This command does <u>not</u> reset the first timer word in scratch pad memory.

READ RECORD (READ) - PSEUDO.

This instruction makes the next record available to the program from an input/output file.

The format of the READ instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	READ			CARDIN				ENDCRD							

There are no LABEL restrictions for this instruction.

A 1 or an L in column 18 of the AF field causes a READ with LOCK to be performed on a shared disk file.

The A ADDRESS Label field specifies the internal file-name from which the record is to be read. Incrementing, indexing, and address controllers are not permitted. For the Advanced Assembler only, a record name associated with the file may be used instead of the file-name. The B ADDRESS field may optionally contain an End-of-File branch address.

The remaining fields are not used, nor are the comparison and OVERFLOW indicators changed by this instruction.

A file must be opened before records can be read from it. If the file declaration specifies "work-area technique," the READ statement causes the next record to be placed into the work area. If the file declaration specifies "no-work-area technique," the READ statement causes the address of the first character of the next record to be loaded into Index Register 2, thus destroying its previous contents. If the file declaration specifies "disk random-access," a key value must be loaded into the actual key field before a READ statement is issued.

RECD

DECLARE RECORD (RECD) - DECLARATIVE.

This instruction declares a logical record, either associated with a file or as a separate working-storage record.

The format of the RECD instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
CRDREC	REC	D0080		UA											

The RECD entry must appear in an Advanced Assembler source deck prior to the first program instruction.

The LABEL entry may not be a program point.

The VAR field contains the 4-digit, right-justified record length. The record length must be an exact number of words (two characters or four digits per word).

The format of the record is declared left-justified in the A ADDRESS label field. Coding is as follows:

Code	Definition
blank	Numeric.
UN	Numeric.
UA	Alphanumeric.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

The address of a logical record will be synchronized modulo-4 by the assembler. If several logical records are associated with a single FILE declaration, they will be assembled into the same storage area.

If the associated FILE declaration specifies no work area, the logical records will be assembled at base-relative location 00000 to facilitate access to record fields within the buffer area by way of Index Register 2 (refer to READ and WRITE).

Fields within a working-storage record may be defined by the DATA and CNST operations. Only the DATA operation may be used to define fields within a record that is associated with a FILE declaration. The total length of all defined fields (if any) within a record must exactly equal the length of the record.

RECV

RECEIVE DATA FROM ANOTHER PROGRAM IN CORE (RECV) - PSEUDO.

This pseudo is defined for the Advanced Assembler only. It receives data from another program, using the core-to-core transfer function in the MCP. When running under a "CP" version of the MCP, the CRCR option must be set, i.e., CRCR=1.

The format for the RECV instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
LISTENRECV			25	PRGNAM				RCVARA				NØSEND			

There are no LABEL restrictions for this instruction.

The VAR field specifies the number of characters to be received. If blank, the length associated with the B ADDRESS label field will be used.

The A ADDRESS label field points to a 6-character alphanumeric field containing the program ID (left-justified) of the program that is sending data. This field may not be indexed. If the referenced program-ID is 6 blanks, data will be transmitted to any program issuing a corresponding SEND. This is referred to as a "global" RECV.

The B ADDRESS label field points to the alphanumeric field which is to receive the data. The B ADDRESS field may not be indexed.

The C ADDRESS field is optional and specifies a branch address to which control should transfer if the sending program has not yet issued the corresponding SEND command. If this operand is not coded and the RECV command is issued before the SEND command in the other program, the program containing the RECV is suspended until the SEND command is issued.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

For a more detailed explanation of this instruction, refer to the B 2500/B 3500 Master Control Programs Information Manual.

REFR

REFERENCE LABEL (REFR) - PSEUDO.

This pseudo when encountered, causes a list of sequence numbers associated with previous instructions that reference the label coded in the A ADDRESS field to be printed.

The format for the REFR pseudo is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	REFR			SYMNAM											

The A ADDRESS field contains the label for which a list of references is desired.

The remaining fields are not used nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

COMMENT (REMK) - PSEUDO.

This pseudo provides remarks text for documentation purposes.

The format for the REMK instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
.A	REMK	REMARKS MAY EXTEND THRU THE WHØLE CARD...														

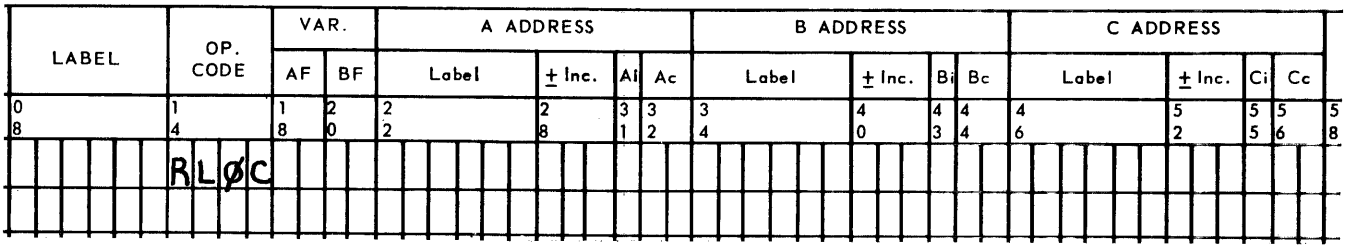
A label or program point may be entered in the LABEL field. It will be equated to the current value of the Location Counter (next available storage location), but will not have any field length or data format associated with it.

The REMARKS text may be coded anywhere within columns 18 through 80.

RESTORE PREVIOUS LOCATION COUNTER VALUE (RLOC) - PSEUDO.

This pseudo restores the location counter to the value it contained before the last LOCN command was given.

The format for the RLOC instruction is:



LABEL entries are not permitted for this instruction.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

This instruction is usually used to resume main-line location assignment after a jump-out which initialized a special location. This command has no operands.

RSET

RESET DATA FIELD TO A ZERO (RSET) - PSEUDO.

This instruction moves a numeric literal of "0" into the data field specified by the A ADDRESS.

The format for the RSET instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	RSET			BØØLM												

There are no LABEL restrictions when using this instruction.

The A ADDRESS points to the field which is to be reset to zero.

The remaining fields are not used by this instruction.

This pseudo yields a MVN instruction. The AF and BF fields and address controller are obtained from the declaration of the data field referenced by the A ADDRESS.

RETURN TO CONTROL PROGRAM FROM USER ROUTINE (RTRN) - PSEUDO.

This pseudo returns control from the user-coded error or label-handling routine to the Control Program.

The format for the RTRN instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
LABENDRTRN				FINAME												

There are no LABEL restrictions for this instruction.

The A ADDRESS field contains the internal file-name for which the user routine was initiated.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

This instruction should not be used for MICR pocket-select routines. The proper return instruction for this is PCKT.

Program control will be returned to the object program immediately after the instruction that caused the user routine to be initiated.

PROCEED TO NEXT PROGRAM (RUNN) - PSEUDO.

This pseudo is defined for Basic Assembler only. It terminates the current program and reads the control records for the next program, without halting.

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
ENDIT	RUNN														

There are no LABEL restrictions for this instruction.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

This instruction has no operators. The control information for the next program must be present and ready at the control device (card or paper tape reader) when this command is issued.

SCAN DELIMITER EQUAL (SDE) - MACHINE CODE-16

This instruction scans each character/digit position of the B field against a list of delimiters in the A field and identifies the first equal condition.

The format of the SDE instruction is:

LABEL	OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ADDRESS ± Inc.	A ADDRESS Ai	A ADDRESS Ac	B ADDRESS Label	B ADDRESS ± Inc.	B ADDRESS Bi	B ADDRESS Bc	C ADDRESS Label	C ADDRESS ± Inc.	C ADDRESS Ci	C ADDRESS Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
SCAN1	SDE	01		=			ALCTLFLD								

There are no LABEL restrictions when using this instruction.

The AF field specifies the number of delimiter characters/digits in the A field, and the BF field specifies the number of characters to be examined in the B field.

The A ADDRESS field points to the delimiter list, and the B ADDRESS field points to the character string to be scanned for the occurrence of delimiters. Their address controller may not be SN and, if the address controller is UN, each digit is converted to a numeric-subset character for comparing.

The remaining fields are not used by this instruction.

The comparison indicator is set to LOW if the <u>first</u> character in the B field equals one of the delimiters in the A field. It is set to EQUAL if some character in the B field (not the first one) is equal to one of the delimiters in the A field. The indicator is set to HIGH if <u>none</u> of the characters in the B field are equal to any of the delimiters in the A field. The OVERFLOW indicator is not changed by this instruction.

This instruction stores a character count (not storage position) into Program Reserved Memory location 00038-39 according to the following rules:

a. 00 is stored if the first B field character is equal to the delimiter.

b. The number of characters in the B field preceding the equal character is stored if the non-first character in the B field is equal to the delimiter.

c. The length of the B field minus one is stored if no B field character is equal to the delimiter.

Upon completion of the SDE command, to branch if a delimiter is found, the programmer should use the LEQ command. To branch if a delimiter is not found, the programmer codes GTR.

The sequence of execution is: the first B field character is compared with all the A field characters and if a match is found the scan is completed. If not, the next B field character is compared and so forth until a match is found, or until the B field is exhausted.

The following are operational examples of the SDE instruction.

Scan Delimiter Equal - First B Field Character Equal.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SDE	01	04	DELIMS(UN)	FIELD	

	BEFORE	AFTER
A ADDRESS	1	1
B ADDRESS	1HDR	1HDR
C ADDRESS	———	———
COMP. INDC.	xxx	LOW
OVERFLOW	unchanged	
00038:	xx	00

Scan Delimiter Equal - Non-First B Field Character Equal.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SDE	01	05	DELIMS	FIELD	

	BEFORE	AFTER
A ADDRESS	=	=
B ADDRESS	AB=5.	AB=5.
C ADDRESS	———	———
COMP. INDC.	xxx	EQUAL
OVERFLOW	unchanged	
00038:	xx	02

Scan Delimiter Equal - No B
Field Character Equal.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SDE	02	05	DELIMS(UA)	FIELD(UN)	

	BEFORE	AFTER
A ADDRESS	12	12
B ADDRESS	45678	45678
C ADDRESS	———	———
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	
00038 :	xx	04

SCAN DELIMITER UNEQUAL (SDU) - MACHINE CODE-17.

This instruction scans each character/digit position of the B field against a list of delimiter character/digits in the A field and identifies the first unequal condition.

The format of this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
.C1	SDU	01		0				ALTFIELD							

There are no LABEL restrictions for this instruction.

The AF field specifies the number of delimiter character/digits in the A field and the BF field specifies the number of characters to be examined in the B field.

The A ADDRESS field points to the delimiter list and the B ADDRESS field points to the character string to be scanned for the occurrence of a non-delimiter. Their address controller may not be SN. If their address controllers are UN, each digit is converted to a numeric-subset character for comparing.

The remaining fields are not used by this instruction.

The comparison indicator is set to LOW if the <u>first</u> character/digit in the B field is not equal to any of the delimiters in the A field. It is set to EQUAL if some character/digit in the B field (not the first one) is not equal to any of the delimiters in the A field. The comparison indicator is set to HIGH if <u>all</u> the characters/digits in the B field are equal to the delimiters in the A field. The OVERFLOW indicator is not changed by this instruction.

This instruction stores a character count (not storage position) into Program Reserved Memory locations 00038-39 according to the following rules:

a. 00 is stored if the first B field character is not equal to any delimiter.

b. The number of characters in the B field preceding the unequal character is stored if the non-first character in the B field is not equal to any delimiter.

c. The length of the B field minus one is stored if all the B field characters are equal to the delimiters.

Upon completion of the SDU command, to branch if a non-delimiter is found, the programmer should use the LEQ command. To branch if a non-delimiter is not found, the programmer codes GTR.

The sequence of execution is: the first B field character is compared with all the A field delimiters and, if a match is found, the scan is complete. If not, the next B field character is compared and so forth until a match is found, or the B field is exhausted.

The following are operational examples of the SDU instruction.

Scan Delimiter Unequal - First B Field Character Not Equal.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SDU	03	05	DELIMS(UA)	FIELD(UN)	

	BEFORE	AFTER
A ADDRESS	123	123
B ADDRESS	61234	61234
C ADDRESS		
COMP. INDC.	xxx	LOW
OVERFLOW	unchanged	
00038:	xx	00

Scan Delimiter Unequal - Non-First B Field Character Not Equal.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SDU	03	04	DELIMS(UA)	FIELD(UA)	

	BEFORE	AFTER
A ADDRESS	ABC	ABC
B ADDRESS	ABCD	ABCD
C ADDRESS		
COMP. INDC.	xxx	EQUAL
OVERFLOW	unchanged	
00038:	xx	03

SEA

SEARCH (SEA) - MACHINE CODE-39.

This instruction compares the A field with the B field, then B+nn, then B+2xnn, etc., until the incremented B ADDRESS reaches the C ADDRESS, or until the condition specified by the C address controller is satisfied.

The format for the SEA instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	SEA	0506		ARG				UNTABLE				UNTABLE	30		UN

There are no LABEL restrictions for this instruction.

The AF field specifies the number of characters/digits to be compared between the A and B fields. The BF field specifies the number of characters/digits by which the B ADDRESS is to be incremented between comparisons[1].

The BF value is independent of the AF value and may be smaller (for scanning purposes).

The A ADDRESS field specifies the location of the argument to be searched for. The A address controller specifies the data format of <u>both</u> the A and B fields in the comparison. The A ADDRESS field may be literal.

The B ADDRESS field specifies the first entry to be compared against the A field. The B address controller bits have no effect on the comparison (which is completely specified by the AF and A address controller), but are used in conjunction with the BF value to determine incrementation between comparisons.

1. nn is the increment to the B ADDRESS.

EXAMPLE:

B Address Controller	Increment
UN	nn
SN	nn + 1
UA	2 x nn

The C ADDRESS field specifies a maximum limit for the incremented value of the B ADDRESS. When the value of the B ADDRESS reaches or exceeds the value of the C ADDRESS, the search operation is terminated immediately. The value of the C ADDRESS should be beyond the end of the table, that is, it should be greater than the last value of the B ADDRESS for which comparing is desired.

The following C address controllers specify the type of search to be performed:

a. If the C address controller is UN, search equal is performed and the search is terminated when a B field entry equal to the A field argument is found, or when the end of the B field is reached.

b. If the C address controller is SN, search low is performed and the search is terminated when a B field entry less then the A field argument is found, or the end of the B field is reached.

c. If the C address controller is UA, search lowest is performed and the search proceeds as with the search low, except that if a B field entry less than the A field argument is found, the search is continued with the low B field entry in place of the A field argument. If a new B field entry is found that is lower than the first B field entry found, the new one takes its place, and so forth. The search is terminated only when the end of the B field is reached.

NOTE

Although the C address controller does not define a C field data format in machine language, the Assembler will still use the specified controller to determine the meaning of the C ADDRESS increment. To avoid conflict and misassembly, define a label just beyond the last entry of the table and use that label in the C ADDRESS.

The setting of the comparison indicator is dependent upon the condition found by a search:

a. If during search equal an equal condition is found, the comparison indicator is set to EQUAL. If an equal condition is not found, the comparison indicator is set to HIGH.

b. If during search low a low condition is found, the comparison indicator is set to EQUAL. If a low condition is not found, the indicator is set to HIGH.

c. If during search lowest at least one low is found, the comparison indicator is set to EQUAL. If no low condition is found, the indicator is set to HIGH.

The OVERFLOW indicator is turned off at End-of-Instruction.

The address of the B entry is loaded into Index Register 1 (IX1) if a B entry is found that fulfills the search conditions. These conditions are as follows:

a. If during search equal an equal condition is found, the B address entry is loaded into IX1. If an equal condition is not found, IX1 remains unchanged.

b. If during search low a low condition is found, the B ADDRESS entry is loaded intl IX1. If low is not found, IX1 remains unchanged.

c. If during search lowest at least one low is found, the lowest B ADDRESS entry is loaded into IX1. If no low is found, the A ADDRESS or the address of the A ADDRESS field (if A ADDRESS is a literal) is loaded into IX1.

Because comparison length is independent of the entry length, the Search instruction may be used for scanning as well as table lookup. This is accomplished by addressing a keyword constant in the A ADDRESS and a text field in the B ADDRESS, with a 1-character increment. When this occurs, the entire text field will be scanned for the occurrence of the keyword. However, the C ADDRESS value must be carefully set to avoid overscanning of the field and possibly producing false results. For example, if an 80-character card image is being scanned for a 5-character keyword, the C ADDRESS should point to column 77 since that is the first column for which a 5-character compare will overrun the field.

The following are operational examples of the SEA instruction.

Search Equal - Equal Found.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SEA	01	02	ARG(UA)	1000(UA)	1020(UN)

	BEFORE	AFTER
A ADDRESS	A	A
B ADDRESS	A1B2C3D2E1	A1B2C3D2E1
C ADDRESS	———	———
COMP. INDC.	xxx	EQUAL
OVERFLOW	xxx	OFF
IX1:	xxx	+0001000

Search Low - Low Not Found.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SEA	1	1	ARG(UN)	1000(UN)	1010(SN)

	BEFORE	AFTER
A ADDRESS	2	2
B ADDRESS	3459876345	3459876345
C ADDRESS	———	———
COMP. INDC.	xxx	HIGH
OVERFLOW	xxx	OFF
IX1:	unchanged	

Search Lowest.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SEA	1	1	ARG(UA)	1000(UA)	1020(UA)

	BEFORE	AFTER
A ADDRESS	E	E
B ADDRESS	EBCDICASCI	EBCDICASCI
C ADDRESS	———	———
COMP. INDC.	xxx	EQUAL
OVERFLOW	xxx	OFF
IX1:	xxx	+0001012

SEARCH EQUAL (SEAE) - PSEUDO.

This instruction is a special case of the SEA operation code (refer to the search command SEA) which performs a search equal operation (C address controller assembled as UN).

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
LØØKUPSEAE	02	05	ARG#					TABLE				TABLE	+30			

There are no LABEL restrictions for this instruction.

Coding for the remaining fields and the settings of Program Reserved Memory, comparison, and OVERFLOW indicators are identical to that of the SEA instruction.

SEARCH LOW (SEAL) - PSEUDO.

This instruction is a special case of the SEA operation code (refer to the search command SEA) which performs a search low operation (C address controller assembled as SN).

The format for this instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	SEAL	0	2	0	5ARG*			TABLE				TABLE	+30			

There are no LABEL restrictions for this instruction.

Coding for the remaining fields and the settings of Program Reserved Memory, comparison, and OVERFLOW indicators are identical to that of the SEA instruction.

SEEK DISK RECORD (SEEK) - PSEUDO.

This instruction is defined for Advanced Assembler only. It initiates a search for a random-access disk record, and reads it into an I/O buffer.

The format for the SEEK instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
FINDITSEEK				DISK1											

There are no LABEL restrictions for this instruction.

A SEEK with LOCK on a shared disk file is specified by a 1 or an L in column 18 of the AF field.

The A ADDRESS field specifies the internal file-name of the disk file containing the record. Incrementing, indexing, and address controlling are not permitted.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

The SEEK function in certain cases must be performed by the MCP, if it has not been performed by the program. Explicit SEEK statements within the program will speed execution in the following cases:

a. Random-access read (input or input/output mode).

b. Random-access write to a blocked file (output or input/output mode).

An actual key number must be loaded into the key area before a SEEK is performed. To obtain greater execution speed (indeed, to prevent

a slowdown), a SEEK should follow a READ of an input file. A SEEK must not precede a write command when the file is unblocked or the write was immediately preceded by a read (input/output mode).

For further explanation of the SEEK functions, refer to the B 2500/B 3500 Master Control Program Information Manual.

DEFINE START OF OVERLAYABLE SEGMENT (SEGM) - PSEUDO.

This pseudo defines the start of an overlayable segment and declares the name of the segment.

The format for the SEGM instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
RECAP	SEGM														

The segment name is specified in the LABEL field of this segment. It must be unique within the program and conform to the syntactic standards for normal labels. The segment name is optional in Basic Assembly language.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The following notes apply to programs containing overlayable segments:

a. The statements preceding the first SEGM card are considered to be the main routine of the program. This routine is permanently resident in core.

b. Segments may be nested to a depth of 32.

c. When one segment is overlaid by another, any processing results stored in areas within the first segment are lost. The first segment is <u>not</u> copied out to disk. If and when it is called for again, a fresh copy is loaded from the original object program file.

d. If a segment relinquishes control and regains it again later without having been overlaid in the meantime, a fresh copy will not be loaded. Thus when coding overlayable segments, strict initialization discipline should be adhered to.

e. A branch-address command may be used to pass control of a segment to any of its parent segments (at any level).

f. The OVLY command may be used under the Advanced Assembler to pass control of a segment to its immediate subsegments only, one level down. Recommended OVLY coding methods for Basic Assembler language will be described in a subsequent publication.

g. A segment may make data references to itself or to any of its parent segments, but not to any other segments (including its own subsegments).

h. The sum of the lengths of the largest combination of nested segments is the core requirements for a segmented program.

i. All coding for a segment (or the main routine) should precede the SEGM card for the first of its subsegments.

j. The A-LABEL field may optionally contain a 6-digit resequence amount. This value will replace the original resequence amount specified in the SPEC card. This value is ignored if the SPEC card did not specify resequencing.

SEND DATA TO ANOTHER PROGRAM IN CORE (SEND) - PSEUDO.

This pseudo is defined for the Advanced Assembler only. It transmits data to another program, using the core-to-core transfer function in the MCP. When running under a CP version of the MCP, the CRCR option must be set, i.e., CRCR=1.

The format for the SEND instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
XTALK	SEND	0	0	RCVPRG				DATA				NØRECV			

There are no LABEL restrictions when using this instruction.

The VAR field specifies the number of characters to be transmitted (up to 9999). If blank, the length associated with the B ADDRESS label field will be used.

The A ADDRESS label field points to the 6-character alphanumeric field containing the left-justified program ID of the program which is to receive the data. This field may not be indexed. If the referenced program ID is 6 blanks, data will be transmitted to any program issuing a corresponding RECV. This is referred to as being a "global" SEND.

The B ADDRESS label field points to the alphanumeric field to be transmitted. This field may not be indexed.

The C ADDRESS field is optional and specifies a branch address to which control should transfer if the receiving program has not yet issued the corresponding RECV command. If this operand is not coded and the SEND command is issued before the RECV command in the other program, the program containing the SEND is suspended until the RECV command is issued.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

Before transmission can occur, a SEND from program A to program B must be matched by a corresponding RECV (refer to the RECV command) in program B. If the SEND is issued first, program A is suspended until program B executes the RECV command. If the declared lengths differ (SEND vs RECV), the shorter length controls the move.

SET DATA FIELD TO A ONE (SETT) - PSEUDO.

This instruction moves a numerical literal of "1" into the data field specified by the A ADDRESS.

The format for the SETT instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	SETT			BØØLEN											

There are no LABEL restrictions when using this instruction.

The A ADDRESS points to the field which is to be set to one.

The remaining fields are not used by this instruction.

This pseudo yields a MVN instruction. The AF and BF fields, and address controller are obtained from the declaration of the data field referenced by the A ADDRESS.

SGNM

DECLARE SEGMENT NUMBER (SGNM) - DECLARATIVE.

This instruction is defined for Advanced Assembler only. It provides the assembled segment-number as a constant.

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
SEGTBLSGNM				EXAMIN												

There are no LABEL restrictions when using this instruction.

The segment-name is coded in the A ADDRESS label field. This label must be defined elsewhere in a SEGM card. Incrementing, indexing, and address controlling are not permitted.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

The segment-number will be assembled as a 3-digit UN constant.

STANDARD I/O PACKAGE (SIOC) - PSEUDO.

This pseudo is defined for Basic Assembler only. It assembles a standard file-oriented input/output control package into the user's program.

The format for the SIOC instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	SIØC			SPØ											

LABEL entries are not permitted with this instruction.

To cause inclusion of ACPT/DISP capabilities in the SIOC package, this word SPO may be optionally coded left-justified in the A ADDRESS label field.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The SIOC routines permits the use of the OPEN, CLOS, READ, WRIT, and POSN macros for file handling. Refer to BCP I/O Coding Methods (section 5) for a complete description of the capabilities and limitations of the SIOC package. The Advanced Assembler treats the SIOC pseudo as a comment.

BEGIN SORT KEY DEFINITION (SKEY) - DECLARATIVE.

This instruction is defined for Advanced Assembler only. It loads a constant (12 UN), at the current value of the location counter, with specifications which are used by the MCP sort intrinsic through the sort pseudo.

The format for the SKEY instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
KEYDEFSKEYFR				00006000				USERBLØCK				ØRØP			

This instruction must directly precede the first KEYA or KEYD declaration.

This instruction must be labeled; program points are permitted.

The AF field specifies the input file close type, and the BF field specifies the output file close type. These types are the same as defined in the CLOS pseudo.

The A ADDRESS field must contain an 8-digit number, left-justified, which specifies the number of records per area to be used for sort work files. 20 areas are allocated by the SORT intrinsic.

The B ADDRESS field may contain the reserved word USERBLOCK if this option is desired. It permits redefinition of a disk file with respect to record length and blocking factor for sorting purposes. For example, a file created with 100-character records blocked 3 could be specified as containing 300-character records blocked 1.

The C ADDRESS field specifies the action to be taken if a parity error occurs. The following reserved words are used:

Word	Meaning
USE	USE records in block with parity.
DROP	DROP records in block with parity.
blank	Initiate MCP DS or DP action if parity is encountered.

The remaining fields are not used by this instruction. Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

See figure 6-2 (page 6-162A) which illustrates a sample program using the SORT pseudo.

SLST

SEARCH LOWEST (SLST) - PSEUDO.

This pseudo is a special case of the SEA operation code (refer to the search command SEA) which performs a search lowest operation (C address controller assembled as UA).

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	SLST		02	05ARG#2				TABLE				TABLE	+30		

There are no LABEL restrictions for this instruction.

Coding for the remaining fields and the settings of Program Reserved Memory, comparison, and OVERFLOW indicators are identical to that of the SEA instruction.

SET EBCDIC/USASCII MODE FLIP-FLOP (SMF) - MACHINE CODE-47.

This pseudo sets or resets the EBCDIC/USASCII mode flip-flop, thus determining what bit configurations are generated for the numeric subset and sign zone digits.

The format for this instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
ASKY	SMF	10													

There are no LABEL restrictions for this instruction.

The 10 digit of the AF field (column 18) specifies which mode is to be established. Coding is as follows:

Code	Mode
0	EBCDIC.
1	USASCII.

The units digit of the AF field must be zero.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The processor is usually set in the EBCDIC mode, but the USASCII mode is advantageous when the program is processing large amounts of data in the USASCII character set. The programmer must establish the USASCII mode within the program.

SORT FILE (SORT) - PSEUDO.

The function of this pseudo is to sort a data file using the MCP sort intrinsic and subsequently to produce an output file of sorted records. This instruction is defined for Advanced Assembler only.

The format of the SORT pseudo is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	SØRT			FILIN				FILØUT				KEYDEF			

There are no LABEL restrictions when using this instruction.

The A ADDRESS field contains the internal file name of the INPUT file.

The B ADDRESS field contains the internal file name of the OUTPUT file.

The C ADDRESS field points to the SKEY declaration. Incrementing and indexing are not permitted.

The remaining fields are not used by this instruction. Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

The user should not OPEN the specified files prior to the initiation of this instruction, as the sort intrinsic will perform the open functions as an inherent feature.

When this pseudo is included in a program, the object program is assembled with a minimum core requirement of 21,000 digits.

Figure 6-2 (page 6-162A) illustrates a sample program using the sort pseudo.

```
PROGRAM ID - SRTDEM                                B3500 ASSEMBLER  11/12/68                                  PAGE     1
   MEMORY   SEQ NO  LABEL    OP     AFBF  A-LBL   INC AIAC  B-LBL   INC BIRC  C-LBL   INC CICC  REMARKS
                             SPEC         CARD    :   : :          :   : :          :   : :   LIST CODE        1
                             IDNT         SRTDEM  :   : :
                                                      ***************************
                                                      * INPUT AND OUTPUT FILES *
                                                      ***************************
     (64)            INPUT   FILE         INPUT   :   : :    MTP   :   : :          :  5: :                          000
    (304)            INREC   RECD  100    UA      :   : :          :   : :          :   : :                          000
                             ENDF
   (1672)            OUTPUT  FILE         OUTPUT  :   : :    DSK   :100:0:20       :  5: :                          000
   (1912)            OUTREC  RECD  100    UA      :   : :          :   : :          :   : :                          000
                             ENDF
   (3152)            PRTOT   FILE         PRTOT   :   : :    PRN   :   : :          :   : :                          000
   (3392)            PRTOTR  RECD  132    UA      :   : :          :   : :          :   : :                          000
                             ENDF
                                                      *************************
                                                      * SORT KEY DEFINITIONS *
                                                      *************************
   (4088)            KEY1    SKEY F  L    000060:00 : :                :   : :    DROP  :   : :                    000
                                                      12 DIG @ 04088, 000=000060000060
   (4100)                    KEYD    4    00014   :   : :UA             :   : :                                    000
                                                      12 DIG @ 04100, 000=000400014240
   (4112)                    KEYA   10    00030   :   : :              :   : :                                    000
                                                      12 DIG @ 04112, 000=001000030010
   (4124)                    KEYA   20    00050   :   : :SA *          :   : :                                    000
                                                      12 DIG @ 04124, 000=002000050411
   (4136)            KEY2    SKEY FR      000060:00 : :                :   : :          :   : :                    000
                                                      12 DIG @ 04136, 000=000060002400
   (4148)                    KEYA   25    00005   :   : :UA *          :   : :                                    000
                                                      12 DIG @ 04148, 000=002500005211
                                                      *************************
                                                      * SORT FILE TWICE *
                                                      *************************
   (4160)                    SORT         INPUT   :   : :    OUTPUT:   : :    KEY1  :   : :                      000
                                                      34 DIG @ 04160, 000=300254270041940040885000006400167 2
   (4194)                    SORT         INPUT   :   : :    PRTOT :   : :    KEY2  :   : :                      000
                                                      34 DIG @ 04194, 000=300254270042280041365000006400315 2
   (4228)                    STOP                                                                                  000
                             FINI                     06 DIG @ 04228, 000=300194
```

Figure 6-2. Use of Sort Pseudo

START NEW LISTING PAGE (SPAC) - PSEUDO.

This pseudo causes the assembler output listing to skip to another page.

The format for the SPAC instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	SPAC															

LABEL entries are not permitted for this instruction.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

This instruction has no operands. This pseudo is ineffective if column 80 of the SPAC card specifies no heading (value 0). The SPAC card itself prints as the last line on the old page.

SPECIFY ASSEMBLER OPTIONS (SPEC) - PSEUDO.

This pseudo specifies device types for assembly input and output, and permits selection of assembly output options.

The format for this instruction is:

OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ADDRESS ± Inc.	A ADDRESS Ai	A ADDRESS Ac	B ADDRESS Label	B ADDRESS ± Inc.	B ADDRESS Bi	B ADDRESS Bc	C ADDRESS Label	C ADDRESS ± Inc.	C ADDRESS Ci	C ADDRESS Cc	F
1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8 ... 6 16
SPEC			CARD	CRF			TAPE				0010000000100				LIST-CØDE
															1

LABEL entries are not permitted for this instruction. An M in column 3 of the SEQUENCE field will cause symbolic code to be listed with every MACRO and LIBRARY call. A D in column 6 will cause the Advanced Assembler to print an exact image of the disk object program at the end of the Assembly listing.

The A ADDRESS label field specifies the symbolic input device. Coding is as follows:

Code	Definition
blank	Card reader (file-name is CARDS).
CARD	Card reader.
TAPE	Magnetic tape (file-name is SYMTIN).
TAPEB	Blocked tape (core = 55000 or more) Advanced Assembler only.
DISK	Disk file, Advanced Assembler only (file-name is SYMDIN).
DISKB	Blocked disk (core = 55000 or more) Advanced Assembler only.
PAPER	Paper tape reader.

To obtain a cross-reference listing, code CRF in the A ADDRESS increment field (Advanced Assembler only).

The B ADDRESS label field specifies the symbolic output device. Coding is as follows:

Code	Definition
blank	No symbolic output desired.
CARD	Card punch (EBCDIC).
CARDN	BCL card punch output.
TAPE	Magnetic tape.
TAPEB	Blocked tape (core = 55000 or more) Advanced Assembler only.
DISK	Disk file (Advanced Assembler only)
PAPER	Paper tape punch.

The B ADDRESS increment, index, and controller fields specify the object program output device for the Basic Assembler only (Advanced Assembler always writes the object program output to disk). The object program file-name is obtained from the IDNT card. Coding is as follows:

Code	Definition
blank	No object program output desired.
CARD	Card punch.
TAPE	Magnetic tape.
PAPER	Paper tape punch.
LIB	Add object program to existing systems tape.

Columns 40 through 45 may also be used to specify the number of lines per page desired on the symbolic listing by coding SKIPnn, where nn is the number of lines per page. If this option is used, the SPAC pseudo remains effective.

If resequencing of the symbolic output file is desired, a 6-digit unsigned starting sequence number is specified in the C ADDRESS label field. Otherwise, the field is blank. If resequencing has been called for, the increment, index, and address controller fields of the C ADDRESS specify a 6-digit sequence-increment value.

The appearance of LIST in columns 58 through 61 of the REMARKS field causes the symbolic input to be printed. The reserved word PART coded in columns 58 through 61 causes a symbolic listing of only those segments being patched to be printed.

Code	Definition
PRNT	Printer only (Advanced Assembler only).
TAPE	Backup tape only.

Also, symbolic input is printed if the SPEC card is omitted. Refer to the LIST and NOLI commands.

The appearance of CODE in columns 63 through 66 of the REMARKS field causes the assembled machine language to be printed. Refer to the CODE and NOCD commands.

SEQ in columns 68 through 70 causes sequence checking of the symbolic input file. Any occurrence of a sequence value equal to or less than the previous card's value will be flagged, and will inhibit the creation of an object program file.

If SYNTAX appears in columns 72 through 77, creation of an object program file will be inhibited.

A code of blank or 2 in column 80 yields a double spaced printed

listing with page headings. Zero (0) yields a single spaced listing without headings and overrides the SKIPnn option. A one (1) yields a single spaced listing with headings.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

This card is optional. If present, it must be the first card of the symbolic input file.

FILL AREA (SPRD) - PSEUDO.

The function of this pseudo is to fill a designated area with multiple copies of a 4-digit mask.

The format for the Fill Area instruction (SPRD) is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
FILLUP	SPRD	4040		TAPREC											

There are no LABEL restrictions when using this instruction.

The VAR field specifies a fill pattern as four 4-bit digits or undigits.

The A ADDRESS field specifies the area to be filled. This area must be word-oriented and its length must be an exact number of words (2n character or 4n digits). The area must be at least two words long. The B and C ADDRESS fields are not used for this instruction.

The OVERFLOW indicator and Program Reserved Memory are not changed.

This command is frequently used to clear an output record area to blanks. Operational examples of the Fill Area instruction (SPRD) are as follows:

Clearing Output Record Area To Blanks.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SPRD	40	40	TAPREC		

	BEFORE	AFTER
A ADDRESS	xxxxxxxxxxxx	404040404040 (blanks)
B ADDRESS	————	————
C ADDRESS	————	————
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Resetting Accumulator Blocks.

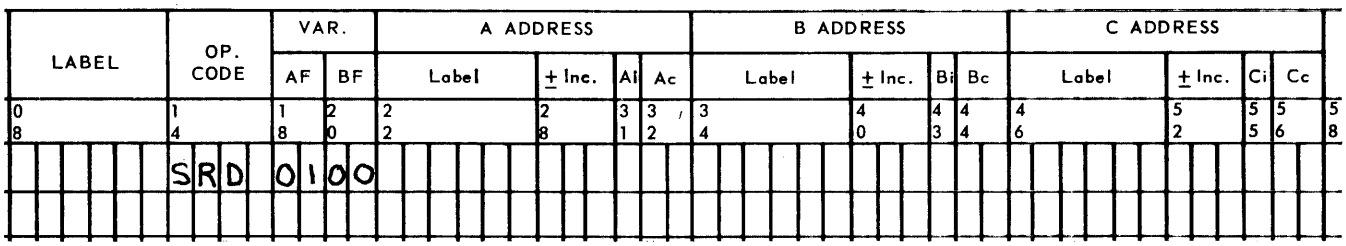
OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SPRD	00	00	ACCUMS		

	BEFORE	AFTER
A ADDRESS	xxxxxxxxxxxx	000000000000
B ADDRESS	————	————
C ADDRESS	————	————
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

SCAN RESULT DESCRIPTOR (SRD) - MACHINE CODE-91.

This instruction may not be used in normal state programs. It scans the result descriptor areas in Systems Reserved Memory to locate unserviced I/O or processor interrupt conditions.

The format for the SRD instruction is as follows:



There are no LABEL restrictions for this instruction.

The VAR field contains a 4-digit number which is interpreted as the low-order digits of an absolute machine address in Systems Reserved Memory (like the BCT command). High-order digits are assumed to equal 00.

The remaining fields are not used by this instruction.

The INTERRUPT indicator is reset at the start of execution. The OVERFLOW indicator is not changed by this instruction.

The address specified in the VAR field is assumed to point to a 16-bit result descriptor area. The first bit of this area is scanned and:

a. If it is equal to 0 (no result descriptor present), the four digits immediately following the 16-bit result descriptor area are accessed and:

   1) If they are 0000, the comparison indicator is set to EQUAL and the instruction terminates (no result descriptors found).

2) If they are not 0000, they replace the original address value specified in the variant field of the SRD command and the scan is then repeated.

b. If it is equal to 1 (result descriptor present), the address of the result descriptor area is stored into Systems Index Register One. The next bit is examined and:

1) If it is 0 (normal termination), the comparison indicator is set to HIGH.

2) If it is 1 (abnormal termination), the comparison indicator is set to LOW. The instruction then terminates.

This command has no operands.

TERMINATE PROGRAM EXECUTION (STOP) - PSEUDO.

This pseudo causes termination of execution and the performance of the standard End-of-Run procedures for the Control Program when using the MCP. With BCP, a coded halt is executed.

The format for the STOP instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
ENDALL	STØP														

There are no LABEL restrictions when using this instruction.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

This statement need not be the last executable statement in the program. It may be inserted wherever program execution should logically terminate.

STT

SET TIMER (STT) - MACHINE CODE-97.

This instruction may not be used in normal state programs. It loads a timer-limit value from a specified storage address into the second timer word in scratch pad memory.

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
MCPTIMSTT				K10000											

There are no LABEL restrictions for this instruction.

The A ADDRESS field points to a 6-digit field which is to be loaded into the timer word. The final A ADDRESS must be even. The A address controller may only be UN or IA.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

When the contents of the first timer word (automatically incremented by the hardware) reach the same value as the contents of the second timer word, a timer interrupt is generated. This interrupt is used to control and schedule program execution time.

THREE-ADDRESS SUBTRACT (SUB) - MACHINE CODE-04

This instruction subtracts the value of the A field from that of the B field and stores the result into the C field.

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	SUB	1		5				NLHØURS				ØRDHRS			

There are no LABEL restrictions when using this instruction.

The AF and BF fields specify the length of their corresponding fields. If the A and B field lengths are unequal, left-zero-fill is added to the shorter of the two in the processor until their lengths are equal. The length of the C field is assumed equal to the larger of the AF and BF values. If the number of significant digits in the difference is greater than the C field length, the subtraction is not performed.

The A ADDRESS field points to the subtrahend field, and all address controllers are valid. UN fields are assumed to be positive. Only the numeric digits of a UA field enter into the subtraction and the sign is assumed to be positive.

The B ADDRESS field points to the minuend field. The address controller conventions are the same as for the A ADDRESS.

The C ADDRESS field points to the field into which the difference is stored. If the field is UN, the sign of the difference is lost and the absolute value of the difference will be stored. If the field is UA, the sign of the difference is lost and the absolute value of

the difference is stored into the numeric digit positions of the C field and the zone-digit positions are set to the numeric subset zone digit.

The remaining fields are not used by this instruction.

If the difference is too large to fit into the C field, the OVERFLOW indicator is set and the comparison indicators remain unchanged; otherwise, the comparison indicators are set according to the sign of the difference (even if the size is not stored). These settings are: LOW if negative, EQUAL if zero, and HIGH if positive.

Program Reserved Memory is not changed by this instruction.

The following are operational examples of the SUB instruction.

Normal Three Address Subtract.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SUB	01	05	FIELD1(UN)	FIELD2(UN)	FIELD3(SN)

	BEFORE	AFTER
A ADDRESS	5	5
B ADDRESS	ABCDE	ABCDE
C ADDRESS	xxxxxx	+12340
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	

Three Address Subtract Causing Overflow.

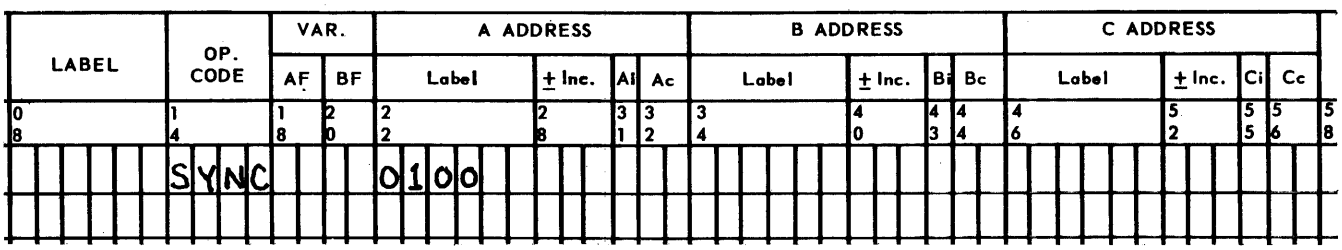
OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SUB	03	03	FIELDA(SN)	FIELDB(SN)	FIELDC(SN)

	BEFORE	AFTER
A ADDRESS	-500	-500
B ADDRESS	+525	+525
C ADDRESS	unchanged	
COMP. INDC.	unchanged	
OVERFLOW	xxx	ON

SYNCHRONIZE LOCATION COUNTER (SYNC) - PSEUDO.

This pseudo forces the Location Counter value up to the next exact multiple of a specified number unless its value is already synchronized.

The format for the SYNC instruction is:



LABEL entries are not permitted for this instruction.

Specified in the A ADDRESS label field is a 4-digit value that is left-justified and zero-filled.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

This instruction is mostly used to assure thousands-orientation for Translate Tables (see operation code TRN) and word-orientation for record-storage fields.

SCAN DELIMITER ZONE EQUAL (SZE) - MACHINE CODE-18.

This instruction scans the zone portion of each character in the B field against a list of delimiter-zones in the A field and identifies the first equal condition.

The format for the SZE instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
SCANITSZE		01		A				ALFIELD							

There are no LABEL restrictions when using this instruction.

The AF field specifies the number of delimiter-zone characters in the A field. The BF field specifies the number of characters to be examined for zone in the B field.

The A ADDRESS field points to the delimiter-zone list and the B ADDRESS field points to the character string to be scanned for the occurrence of delimiter-zones. Their address controller must be UA or IA and the numeric portion of their field characters does not enter into the operation.

The remaining fields are not used by this instruction.

The comparison indicator is set to LOW if the zone portion of the first B field character is equal to the zone portion of one of the A field delimiter-zone characters. It is set to EQUAL if the zone portion of some character in the B field (not the first) is equal to the zone portion of one of the A field delimiter-zone characters. It is set to HIGH if no zone portion of any of the B field characters is equal to the zone portion of any of the A field delimiter-zone characters. The OVERFLOW indicator is not changed by this instruction.

This instruction stores a character count (not storage position) into Program Reserved Memory locations 00038-39 according to the following rules:

a. 00 is stored if the first B field zone digit is equal to some delimiter zone.

b. The number of characters in the B field preceding the zone-equal character is stored if some non-first B field zone digit is equal to some delimiter zone.

c. The length of the B field minus one is stored if no B field zone digit is equal to any delimiter zone.

Upon completion of a SZE command, the programmer should use the LEQ command if he wishes to branch upon finding a zone.

The sequence of execution is: the first B field zone digit is compared with all the A field delimiter zones and, if a match is found, the scan is complete. If not, the next B field zone digit is compared and so forth until a match is found, or until the B field is exhausted.

The following are operational examples of the SZE instruction.

Scan Delimiter Zone Equal - First Zone Found.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SZE	02	03	DELIMS	FIELD	

	BEFORE	AFTER
A ADDRESS	AJ	AJ
B ADDRESS	KWA	KWA
C ADDRESS	———	———
COMP. INDC.	xxx	LOW
OVERFLOW	unchanged	
00038:	xx	00

Scan Delimiter Zone Equal - Zone Found Other Than First.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SZE	02	04	DELIMS	FIELD	

	BEFORE	AFTER
A ADDRESS	AJ	AJ
B ADDRESS	WKAM	WKAM
C ADDRESS	———	———
COMP. INDC.	xxx	EQUAL
OVERFLOW	unchanged	
00038:	xx	01

Scan Delimiter Zone Equal -
Zone Not Found.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SZE	02	04	DELIMS	FIELD	

	BEFORE	AFTER
A ADDRESS	1-AJ	AJ
B ADDRESS	WXYZ	WXYZ
C ADDRESS	____	____
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	
00038:	xx	03

SCAN DELIMITER ZONE UNEQUAL (SZU) - MACHINE CODE-19.

This instruction scans the zone portion of each character in the B field against a list of delimiter-zones in the A field and identifies the first unequal condition.

The format for the SZU instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	SZU	03		AJS				ALFIELD							

There are no LABEL restrictions for this instruction.

The AF field specifies the number of delimiter-zone characters in the A field. The BF field specifies the number of characters to be examined for zone in the B field.

The A ADDRESS field points to the delimiter-zone list and the B ADDRESS field points to the character string to be scanned for the non-occurrence of delimiter zones. Their address controller must be UA or IA. The numeric portion of their field characters does not enter into the operation.

The remaining fields are not used by this instruction.

The comparison indicator is set to LOW if the zone portion of the first B field character is not equal to the zone portion of any of the A field delimiter-zone characters. It is set to EQUAL if the zone portion of some character in the B field (not the first) is not equal to the zone portion of any of the A field delimiter-zone characters. It is set to HIGH if the zone portion of every B field character matches an A field delimiter-zone character.

BUMP

INCREMENT BY ONE (BUMP) - PSEUDO.

This instruction increments the contents of the A field by one.

The format for the BUMP instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	BUMP			DTAFLD											

There are no LABEL restrictions when using this instruction.

The A ADDRESS points to the field to be incremented.

The remaining fields are not used by this instruction.

This pseudo yields an increment instruction. The length of the field to be incremented, and its controller are obtained from the declaration of the data field referenced by the A ADDRESS.

This instruction stores a character count (not storage position) into Program Reserved Memory location 00038 through 00039 according to the following rules:

a. 00 is stored if the first B field zone digit is not equal to any delimiter zone.

b. The number of characters in the B field preceding the zone-unequal character is stored if some non-first B field zone digit is not equal to any delimiter zone.

c. The length of the B field minus one is stored if all the B field zone digits are equal to the delimiter zones.

Upon completion of a SZU command, the programmer should use the LEQ command if he wishes to branch if a non-delimiter zone is found.

The sequence of execution is: the first B field zone digit is compared with all the A field delimiter zones and, if a match is found, the scan is complete. If not, the next B field zone digit is compared and so forth until a match is found, or until the B field is exhausted.

The following are operational examples of the SZU instruction.

Scan Delimiter Zone Unequal - First B Zone Unequal.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SZU	01	04	DELIMS	FIELD	

	BEFORE	AFTER
A ADDRESS	A	A
B ADDRESS	JABX	JABX
C ADDRESS	———	———
COMP. INDC.	xxx	LOW
OVERFLOW	unchanged	
00038:	xx	00

Scan Delimiter Zone Unequal - Non-First B Zone Unequal.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SZU	02	04	DELIMS	FIELD	

	BEFORE	AFTER
A ADDRESS	AJ	AJ
B ADDRESS	ABXC	ABXC
C ADDRESS	———	———
COMP. INDC.	xxx	EQUAL
OVERFLOW	unchanged	
00038:	xx	02

Scan Delimiter Zone Unequal -
All B Zones Equal.

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
SZU	02	04	DELIMS	FIELD	

	BEFORE	AFTER
A ADDRESS	AJ	AJ
B ADDRESS	CDMO	CDMO
C ADDRESS	————	————
COMP. INDC.	xxx	HIGH
OVERFLOW	unchanged	
00038:	xx	03

READ SYSTEMS CLOCK (TIME) - PSEUDO.

This pseudo copies the value of the system clock at run time into a specified location.

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	TIME			NØW											

There are no LABEL restrictions for this instruction.

The A ADDRESS field receives the systems time and must be a 10-digit UN area. The systems time is expressed in milliseconds.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

TEST I/O COMPLETE (TIOC) - PSEUDO.

This pseudo is defined for Basic Assembler only. It suspends all processing until a designated I/O operation is completed.

The format for the TIOC instruction is:

LABEL	OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ADDRESS ± Inc.	A ADDRESS Af	A ADDRESS Ac	B ADDRESS Label	B ADDRESS ± Inc.	B ADDRESS Bi	B ADDRESS Bc	C ADDRESS Label	C ADDRESS ± Inc.	C ADDRESS Ci	C ADDRESS Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	TIØC			TDESC	- 4			TABNØR							

There are no LABEL restrictions for this instruction.

The A ADDRESS field points to the field which will receive the result descriptor from the I/O operation. This field is four digits to the left of the original I/O descriptor.

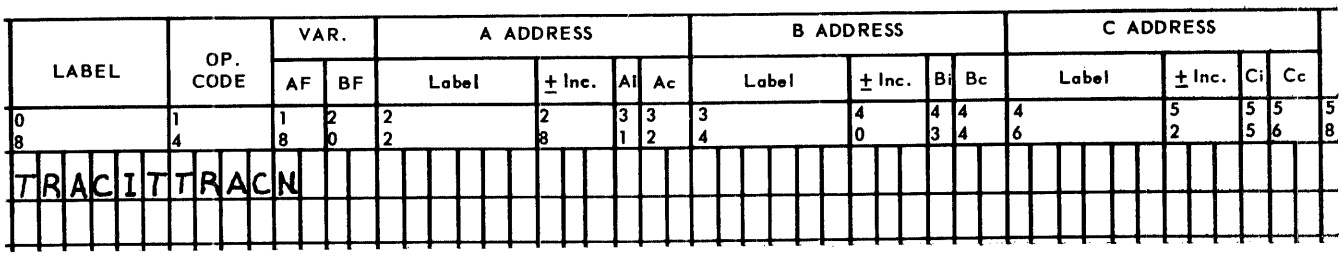
The B ADDRESS field optionally contains a branch label. Control will be transferred to this label upon completion of the I/O operation <u>if</u> the result descriptor indicates abnormal completion. Control passes to the next instruction upon completion of the I/O if the operation was normal or if the B ADDRESS is not coded.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

TRACE PROGRAM EXECUTION (TRAC) - PSEUDO.

This pseudo either sets or resets the trace option, thus producing a one-instruction-per-line trail of program execution.

The format for the TRAC instruction is:



There are no LABEL restrictions for this instruction:

The AF field specifies the options to be set or reset. Coding is as follows:

Code	Definition
blank	Cease all tracing.
C	Trace control state (MCP/BCP) only.
N	Trace normal state (program) only.
CN	Trace all machine instructions.
NC	Trace all machine instructions.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

This operation should be used sparingly since it requires considerable extra storage and is quite slow.

The output of the trace is specified in the B 2500/B 3500 Master Control Programs Information Manual under debugging aids.

TRANSLATE BY TABLE (TRN) - MACHINE CODE-15.

This instruction translates a string of digits/characters according to a specified equivalence table and stores the translated string into a specified address.

The format for the TRN instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
CØDER	TRN	0025		NAME				TRTABL				NEWNAM			

There are no LABEL restrictions when using this instruction.

The VAR field specifies a 4-digit count of digits/characters to be translated. The maximum count of 10,000 is specified as 0000.

The A ADDRESS field points to the string to be translated and the address controller is unrestricted, however, the sign of a SN field will be ignored. If the A field format is UN or SN, the applicable numeric-subset zone will be forced in over each digit before it is translated.

The B ADDRESS field points to the equivalence table to be used for translation. The final B ADDRESS must be modulo-1000, i.e., the last three digits of the final address must be 000. The SYNC command should be used when declaring the table, and the B address controller must be UA or IA.

The C ADDRESS field points to the field into which the translated characters will be stored and the address controller must be UA or IA.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

The character translation is performed by the following algorithm:

a. A low-order zero bit is appended to the 8-bit character to be translated.

b. The resultant nine bits are divided into three 3-bit groups.

c. The 3-bit groups are interpreted as three octal digits.

d. The three octal digits are combined with the B ADDRESS to produce a table-access address.

e. The character in the derived table-access address is moved to the C field.

EXAMPLE:

EBCDIC A is 11000001_2

The stages of translation for the above example are:

a. For step a above, 11000001_2 becomes 110000010.

b. For step b above, 110000010 becomes 110 000 010.

c. For step c above, 110 000 010 becomes 602_8.

d. For step d above, table access address B = 602 is generated.

e. For step e above, the character in B + 602 is moved to the C field.

Figure 6-1 shows the general layout of translate tables in core.

Those areas in figure 6-1 which are Xed out are never accessed by the TRN instruction.

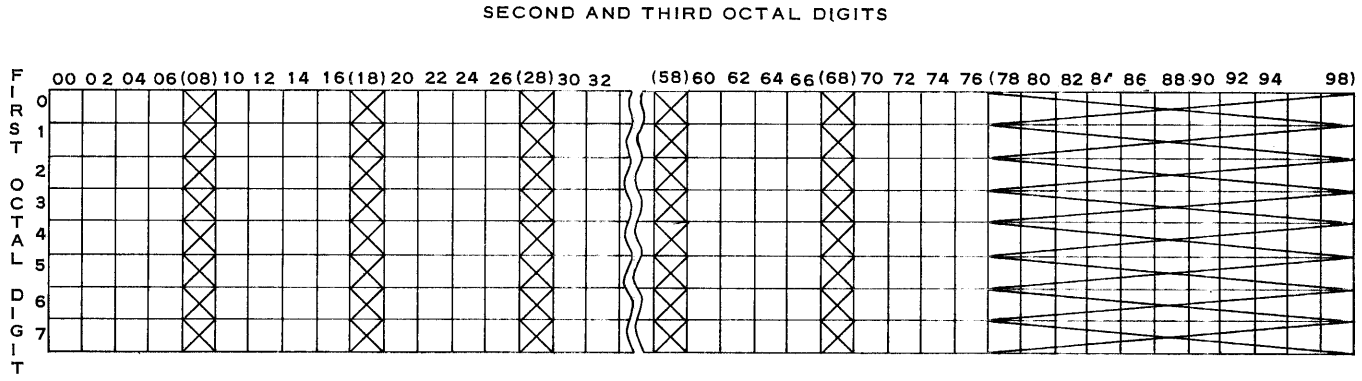


Figure 6-1. Translate Tables in Core

The following is an operational example of the TRN instruction.

Normal Translate

OP	AF	BF	A ADDRESS	B ADDRESS	C ADDRESS
TRN	00	10	NAME	ALFTAB	CHKNAM

	BEFORE	AFTER
A ADDRESS	JOHN-8-DOE	JOHN-8-DOE
B ADDRESS	table converts letters to 0, blanks to 0, all else to 1	
C ADDRESS	xxxxxxxxxx	0000010000
COMP. INDC.		unchanged
OVERFLOW		unchanged

UNLOCK RECORD (UNLK) - PSEUDO.

This pseudo causes an UNLOCK to be performed on a shared disk file.

The format for this pseudo is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	UNLK			SHARED												

The A ADDRESS label field specifies the internal file name of the shared file to which a record is to be unlocked. Incrementing, indexing, and address controlling are not permitted.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

Shared disk files must be declared random-access. Therefore a key value must be loaded into the actual key field before the UNLK statement is issued.

DECLARE USER I/O PROCEDURES (USER) - DECLARATIVE.

This declarative is defined for Advanced Assembler only. It specifies the entry addresses for user routines which handle special I/O conditions.

The format for the USER instruction is as follows.

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	USER			LABEL1				TPERR1							

LABEL entries are not permitted for this instruction.

The three ADDRESS label fields (A, B, and C), plus columns 58 through 63 in the REMARKS field point to routines for various functions depending on the peripheral device. These routines are as follows:

a. For conventional devices (printer, tape, disk):

1) A label - label handling routine.
2) B label - error handling routine.
3) C label - End-of-Page/beginning-file-limits violation.
4) REMARKS - ending-file-limits violation.

(Data Communications devices may use the A and B ADDRESS fields).

b. For a MICR sorter-reader:

1) A label - read error (memory access, unreadable, uncoded).
2) B label - amount error.*
3) C label - transit field error.*
4) REMARKS - pocket select.

* These addresses are effective only if the FILE declarative (column 21) specifies distinction between amount errors and transit errors. If both fields contain errors, the general read error routine will be selected.

Columns 58 through 63 are used for a user routine label.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

USER routines for conventional devices exit (return control to the MCP) by way of the RTRN command, while USER routines for the sorter-reader exit by way of the PCKT command.

This declarative must immediately precede the FILE declarative to which it applies.

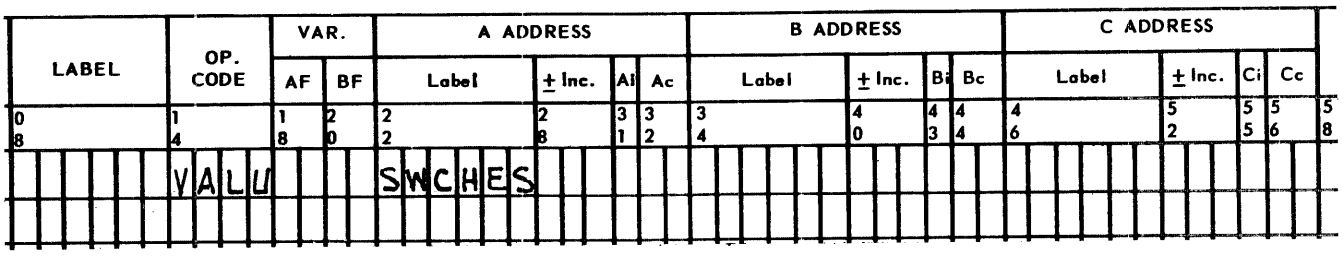
(Methods and restrictions on USER routine coding will be supplied at a future date.)

VALU

OBTAIN CARD VALUE CONTROL (VALU) - PSEUDO.

This pseudo is defined for Basic Assembler only. It receives (from the BCP) the six characters from columns 8 through 13 of the program call-out record (e.g., TASMBLER 000002).

The format for the VALU instruction is:



There are no LABEL restrictions for this instruction.

The A ADDRESS field points to a 6-character storage area into which the control information is copied.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

The control field may be used to specify bit/character switches for program options. The control card is read in the EBCDIC mode.

WRITE RECORD (WRIT) - PSEUDO.

This pseudo releases the current logical record to an output or input/output file, and specifies printer/punch control.

The format for this instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
ØUT1	WRIT	1		PRTREC				ENDPAG								

There are no LABEL restrictions for this instruction.

A WRITE with LOCK or a WRITE without unlocking on a shared disk file is specified by an L in column 18 of the AF field. When referencing a shared file, a blank in column 18 is interpreted as a WRITE with UNLOCK.

The AF field (column 19) optionally specifies space-after-print. Coding is as follows:

Code	Definition
0	Space suppress.
1	Single space.
2	Double space.

The BF field (columns 20-21) optionally specifies a channel number on the printer control tape or punch stacker selection. Coding is as follows:

	Code	Description
Printer	01-11	Skip to channel 1 through 11 (channel 12 is reserved for End-of-Page detection).

	Code	Description
Punch	00 or blank	Primary stacker.
	01	Auxiliary stacker.
	02	Error stacker.

If both spacing and skipping are specified, only the skip is performed.

The A ADDRESS label field specifies the internal file-name of the file to which a record is to be released. Incrementing, indexing, and address controlling are not permitted. For the Advanced Assembler only, a record-name associated with the file may be used instead of the file-name.

The B ADDRESS field optionally specifies an End-of-Volume branch address. If a buffer and work area access technique is being employed for this file, another WRIT should be issued at this address. Otherwise, the last print line will be lost. If the IX2 access technique is being employed, a second WRIT should not be issued, as the print line will not be lost. To avoid confusion, it is recommended that a user routine be used for End-of-Page condition rather than this B ADDRESS branch.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

A file must be opened before records can be written to it. The FILE declaration determines the function of the WRIT statement:

a. If the FILE declaration specifies the work area technique, the WRIT statement causes the current record to be moved from the work area to an output buffer area.

b. If the FILE declaration specifies the no-work-area technique, the WRIT statement causes IX2 to be incremented past the current record in the buffer area. If the records

are created elsewhere than in the output buffer area, the last command before the WRIT should move the current record to the location addressed by IX2.

If the FILE declaration is disk random-access, a key value must be loaded into the actual key field before the WRIT statement is issued.

EXECUTE CONTROL CARD FUNCTION (ZIPP) - PSEUDO.

This pseudo sends an alphanumeric test field to the MCP/BCP which handles the test as though it were a control card.

The format for the ZIPP instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
CALLUPZIPP				NXPRØG											

There are no LABEL restrictions for this instruction.

The A ADDRESS field points to an area which contains the control text. Under control of the MCP (Advanced), the control text may contain any or all of the following functions:

a. EXECUTE
b. COMPILE.....WITH
c. FILE
d. VALUE
e. CHANGE
f. REMOVE
g. DUMP
h. LOAD
i. PRIORITY
j. CORE

The control text must end with a period. If it is longer than 72 characters, it must also begin with a period. When under control of the BCP, the control text must be a 13-character field containing a standard program call from tape.

Example:

Txxxxxxyyyyyy

where xxxxxx is the ID of the program to be loaded and executed, and yyyyyy is the (optional) value to be delivered to the ZIPPed program.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

When under control of the MCP, control returns after the ZIPP statement. When under the BCP, the program which issues the ZIPP is terminated.

SECTION 7
DATA COMMUNICATIONS OPERATIONS CODES

GENERAL

The data communications operation codes are described in alphabetical sequence, and each description is presented in a standard format. The symbolic code (i.e., pseudo) precedes the general description of the function of each pseudo. A coding example of the operation code and a description of LABEL entry restrictions and entries (if any) follow. After the LABEL entry description, each field used is defined. Any changes to the comparison and OVERFLOW indicators and Program Reserved Memory are described next.

ACCEPT FROM REMOTE SPO (ACPR) - PSEUDO.

This is a data communications pseudo and is defined for the Advanced Assembler only. Its function is to permit the entry of data from a remote SPO to an object program.

The format of the ACPR instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	ACPR	25		MSGE				REMSPØ								

There are no LABEL restrictions when using this instruction.

The AF field specifies the number of characters to be read from the Supervisory Printer. If it is left blank, the declared length associated with the A ADDRESS label is used. The maximum number of characters which can be read is 72.

The A ADDRESS field points to the alphanumeric area which is to receive the data transmitted from the remote SPO. Incrementing is permitted. Indexing and indirect addressing are not permitted. The address controller must be defined as being UA.

The B ADDRESS field points either to the alpha mnemonic of the remote SPO or to the channel and unit of the remote SPO.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

An alpha mnemonic of a remote SPO is from one to six characters. If less than six characters, it must be followed by a blank, and it must be the same as the mnemonic stated on the unit card. Similarly, the alpha channel and unit specification must be followed by a blank and is in the following format: CC/U.

This instruction causes the operating object program to halt and wait for appropriate data to be entered through the remote SPO. If the named device is not a remote or not a remote SPO, or not logged-in, the accept will be processed to the local SPO.

CANCEL DC I/O IF INACTIVE (CNCL) - PSEUDO.

This is a data communications pseudo and is defined for the Advanced Assembler only. Its function is to cancel a previously issued FILL command if data is not being received. It also cancels a previously issued ENBL command if an inquiry has not come from the device.

The format for the CNCL instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
LØKØUTCNCL				REMØT1											

There are no LABEL restrictions when using this instruction.

The A ADDRESS label field specifies the internal file-name for which the FILL/ENBL command is to be cancelled. Incrementing, indexing, and address controllers are not permitted.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

For more detailed information, refer to the B 2500/B 3500 Master Control Programs Information Manual (1031218).

DISPLAY ONTO REMOTE SPO (DISR) - PSEUDO.

This is a data communications pseudo and is defined for the Advanced Assembler only. Its function is to provide for the transmittal of data, error messages, and operator instructions from an object program to a remote SPO.

The format of the DISR instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6
	DISR30			MESSGE				ØNSPØ							

There are no LABEL restrictions when using this instruction.

The AF field specifies the number of characters to be typed out on the remote SPO. If it is blank, the length associated with the A ADDRESS label is used. The maximum number of characters which can be displayed is 72.

The A ADDRESS field points to an alphanumeric area which contains the data to be displayed on the remote SPO. Incrementing is permitted; indexing and indirect addressing are not permitted. Literals are not permitted, and the address controller must be UA.

The B ADDRESS field points to a data field containing either the alpha mnemonic name of the remote SPO or the channel and unit designation of the remote SPO.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

An alpha mnemonic of a remote SPO is from one to six characters. If less than six characters, it must be followed by a blank, and it must

be the same as the mnemonic stated on the unit card. Similarly, the alpha channel and unit specification must be followed by a blank and is in the following format: CC/U.

This instruction causes the designated data to be written on the remote SPO from the MCP/SPO queue to ensure that a program is not operationally deterred while a message is printing if the named device is not a remote, or not a remote SPO, or not logged-in. The display will be processed to the local SPO.

ENABLE DC DEVICE (ENBL) - PSEUDO.

This is a data communications pseudo and is defined for the Advanced Assembler only. It recognizes input inquiry requests from a remote device or disconnects the telephone line for dial lines and recognizes a ringing signal.

The format for the ENBL instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
HELLØ	ENBL			REMØTI				INQRYI								

There are no LABEL restrictions when using this instruction.

The A ADDRESS label field specifies the internal file-name of the device. Incrementing, indexing, and address controllers are not permitted.

The B ADDRESS label field may optionally contain an inquiry label. A WAIT statement may be used to suspend processing until the message is entered, at which time processing resumes with the statement immediately following the WAIT or with this optionally specified inquiry label.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

ENBL allows the device to establish a connection with the B 2500/ B 3500 system by depressing the inquiry key, if the device is connected on leased lines, or by dialing the systems telephone number if the device is on dialed lines.

FILL

FILL INPUT AREA FROM DC DEVICE (FILL) - PSEUDO.

This is a data communications pseudo and is defined for the Advanced Assembler only. It initiates a data communications input/output operation, and then returns to the program while the operation continues.

The format of the FILL instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
TANKUPFILLDT				SPØ1				WCRC				ENDSP1			

There are no LABEL restrictions when using this instruction.

The VAR field may contain up to four of the function modifiers, left-justified, and in any order:

Code	Description
T	Inhibit time-out.
X	Preset STX.
D	Dial.
E	Delete ETX.
V	Audio response.
P	Poll.
Q	Ignore ENQ.
R	Tone Response.

The A ADDRESS field specifies the internal file-name on which the operation is to be started. Indexing, incrementing, and address controlling are not permitted.

The type of operation to be started is specified left-justified in the B ADDRESS label field. These are operation codes in their own right and their functions are briefly explained in this manual. These codes are as follows:

Code	Definition
REED	Read from DC device.
WCRC	Write-to-control/read-to-control.
WCRT	Write-to-control/read-transparent.
WTRC	Write-transparent/read-to-control.
RITE	Write a data communications record.

The C ADDRESS field may optionally contain an action label. Control is transferred to this label if the FILL operation is completed while the program is suspended in a WAIT pseudo.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

The file declared in the A ADDRESS field must have a work area declared, otherwise, the pseudo is void. Stream mode is not permitted.

For a more detailed explanation of the data communications functions, refer to the B 2500/B 3500 Master Control Programs Information Manual (1031218).

INTA

OBTAIN I/O CHARACTER COUNT (INTA) - PSEUDO.

This is a data communications pseudo and is defined for the Advanced Assembler only. It makes the character count of the last I/O operation of a specified file available to an object program.

The format of the INTA instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
GET	INTA			INPUT				AMØUNT							

There are no LABEL restrictions when using this instruction.

The A ADDRESS field must contain the file-name.

The B ADDRESS field must contain a label referencing a 6-digit field into which the value is to be placed.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and overflow indicators are not changed by this instruction.

INTERROGATE DC RESULT DESCRIPTOR (INTR) - PSEUDO.

This is a data communications pseudo and is defined for the Advanced Assembler only. It obtains the result descriptor for a specified DC file.

The format of the INTR instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
WHAT	INTR			REMØTI				RSOSC							

There are no LABEL restrictions when using this instruction.

The A ADDRESS field specifies the internal file-name for which a result descriptor is desired.

The B ADDRESS field points to a 16-digit unsigned numeric field into which the converted result descriptor is stored. Each bit of the hardware result descriptor is converted into a digit position, 0 if the bit was off, and 1 if it was on.

The bits within the result descriptor are assigned the following meanings:

1 - operation complete.
2 - exception condition.
3 - not ready local (single-line)
(multiline if during operation).
4 - data error.
5 - abandon call retry (ACR).
6 - cancel complete.
7 - end-of-transmission (EOT).
8 - attempt to exceed maximum address.
9 - time-out.

10 - memory parity error.
11 - write error.
12 - carrier loss.
13 through 16 - unit number.
4 and 5 - data loss.
6 and 7 - break detected.

Explanation of the result descriptor bits is as follows:

a. 1 - always ON if the attempted operation was completed.

b. 2 - will be ON if any combination of three through 16 are ON. This is the test position to see if any exception exists. If this position is ON by itself, a partially complete condition exists due to the use of READ STREAM MODE and will not occur in any other situation.

c. 3 - will be ON if the single-line control or the local Data Set is not ready and the operation will be terminated. For multiline control the digit is set ON in the channel result descriptor unless it occurs during an operation, in which case it is set ON in the adapter result descriptor.

d. 4 - if a data error (message or character parity) occurs, a READ operation continues until terminated in a normal manner. The phone line is not disconnected. Attempts to exceed maximum address, time-out, or EOT can also occur.

e. 4 and 5 - if data loss (missed memory access or MLC cycle), a READ operation continues until terminated in a normal manner. Attempts to exceed maximum address, time-out or EOT can also occur. The phone line is not disconnected. A WRITE operation is terminated immediately and position 11 is set.

f. 5 - if an abandon call retry condition exists, this position will be set ON and the telephone line is disconnected.

g. 6 - if a cancel complete condition exists, this position will be set ON and CANCEL is initiated.

h. 6 and 7 - if a break is detected, these positions will be set on for a WRITE operation. The telephone line is not disconnected.

i. 7 - if the EOT exists, this position is set ON and the telephone line is disconnected.

j. 8 - if an attempt to exceed maximum address exists, a READ operation will initiate a time-out and wait for a control code denoting End-of-Message. This position will be set ON if an ETX is received before time-out. This position along with position 7 will be set ON if an EOT is received before time-out. This position and position 9 will be set ON if time-out occurs without ETX or EOT. A WRITE operation is immediately terminated and this position along with position 11 is set ON. The telephone line is disconnected in each case.

k. 9 - if time-out exists, this position is set ON and the telephone line is not disconnected. Time-out occurs on READ instructions only.

l. 10 and 11 - if a memory parity error exists, these positions are set ON and the telephone line is not disconnected. Memory parity error occurs only on a WRITE.

m. 12 - a READ operation continues until terminated in a normal manner. The phone line is not disconnected. Attempt to exceed maximum address, time-out, or EOT can also occur.

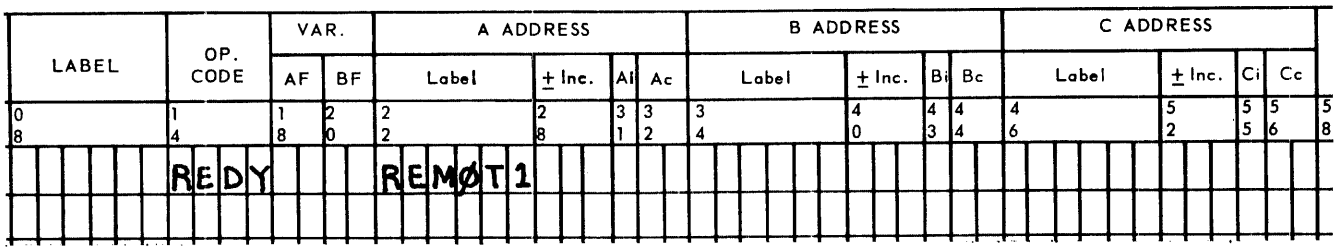
n. 13 and 16 - unit number for single line will be 0. Unit number for multiline will be assigned.

REDY

CONTINUE STREAM MODE INTO NEXT BUFFER - (REDY) - PSEUDO.

This is a data communications pseudo and is defined for the Advanced Assembler only. The stream mode operation is to continue into the next buffer of a data communications file.

The format for the REDY instruction is:



There are no LABEL restrictions for this instruction.

The A ADDRESS label field must contain the file-name in which the stream mode operation is to continue. Incrementing, indexing, and address controlling are not permitted.

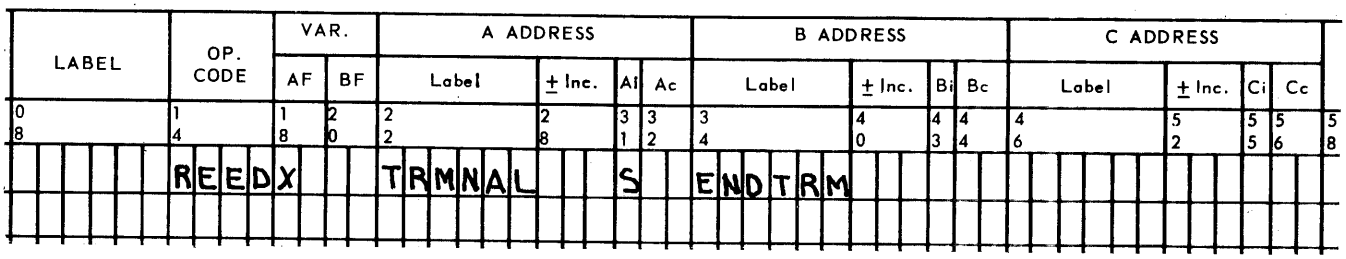
The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

For a more detailed explanation of this instruction, refer to the B 2500/B 3500 Master Control Programs Information Manual.

READ DC RECORD (REED) - PSEUDO.

This is a data communications pseudo and is defined for the Advanced Assembler only. The function of this pseudo is to load data from a remote device into ascending memory locations beginning with the location specified by the A ADDRESS field. Loading will continue until an ETX control code is detected, or until the buffer is filled.

The format of the REED instruction is:



The VAR field may contain any or all of the following function modifiers, left-justified, in any order:

Code	Function
T	Inhibit time-out.
X	Preset STX.
D	Dial.
Q	Ignore ENQ.

The A ADDRESS field specifies the internal file-name for which a record is to be read. Incrementing and address controllers are not permitted.

The A index field (column 31) may optionally contain an S to indicate a stream mode. Operation in stream mode requires a minimum record declaration of 200 digits (100 characters). The ETX control code will terminate the operation. Other indexing is not permitted.

The B ADDRESS field may optionally contain an End-of-File label.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

The program is suspended (in contrast to the FILL command) until the operation is complete.

WRITE DC RECORD (RITE) - PSEUDO.

This is a data communications pseudo and is defined for the Advanced Assembler only. The function of this pseudo is to pass data to a remote device from ascending memory locations beginning at the location specified by the A ADDRESS field and continuing until an ETX is sensed or until the end of the declared logical record.

The format for the RITE instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
.A	RITEDR			REMØT1		S		END								

There are no LABEL restrictions for this instruction.

The VAR field may contain up to four of the following functions, left-justified and in any order:

Function Modifier	Definition
X	Preset STX.
D	Dial.
E	Delete ETX.
V	Voice Response.
R	Tone Response.

The A ADDRESS field specifies the internal file-name or associated record name to be released. The A index field (column 31) may optionally contain an S to signify stream mode. Otherwise, indexing is not permitted.

The B ADDRESS field may optionally contain an At-End action label.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

CANCEL DC I/O UNCONDITIONALLY (UNCL) - PSEUDO.

This is a data communications pseudo and is defined for the Advanced Assembler only. It cancels a previous data communications instruction regardless of data flow conditions.

The format for this instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
GAG-I	TUNCL	H		DATAC1											

There are no LABEL restrictions for this instruction.

The VAR field may contain either or both of the following function modifiers, in any order:

a. B - break transmitted.
b. H - hang up (disconnect phone line).

The A ADDRESS label field specifies the internal file-name on which the I/O command is to be cancelled. Incrementing, indexing, and address controlling are not permitted.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

For a more detailed information, refer to the B 2500/B 3500 Master Control Programs Information Manual.

AWAIT INQUIRY (WAIT) - PSEUDO.

This is a data communications pseudo and is defined for Advanced Assembler only. It suspends program execution until a previously initiated FILL operation is completed, or a previously enabled device inquires, or (optional) a time-out occurs.

The format for the WAIT instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	WAIT			TENSEC											

There are no LABEL restrictions for this instruction.

The A ADDRESS field optionally points to a 5-digit UN field containing the maximum number of seconds the program is to be suspended.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

Control returns to the next instruction following the WAIT if:

a. WAIT ends with a time-out or

b. There are no FILLs pending and no devices are enabled or

c. A FILL I/O is completed, or an enabled device inquires and the corresponding FILL/ENBL command did not contain an action label.

Otherwise, control is returned to the action label specified in the FILL/ENBL command.

The programmer must assure that all action label routines that may receive control are in core when the WAIT command is executed. The MCP will not perform any implicit overlay operations. For more detailed information, refer to the B 2500/B 3500 Master Control Programs Information Manual.

WRITE TO CONTROL/READ TO CONTROL (WCRC) - PSEUDO.

This is a data communications pseudo and is defined for Advanced Assembler only. The function of this pseudo is to pass data to a remote device from memory locations and, when successfully completed, to cause data to be read from the remote and passed to appropriate memory locations.

The format for this instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
CALL	WCRC			RECØRD				ENDIT							

There are no LABEL restrictions for this instruction.

The VAR field may specify up to four of the function modifiers, left-justified, and in any order:

Code	Description
T	Inhibit time-out.
X	Preset STX.
D	Dial.
E	Delete ETX.
V	Audio response.
P	Poll.
Q	Ignore ENQ.
R	Tone response.

The A ADDRESS field specifies the internal file name or associated record name from which data is to be sent and into which data is to be received. The A index (column 31) may optionally contain an S to signify stream mode. Otherwise, indexing is not permitted.

NOTE

Stream and audio response are an illegal combination.

Data will be passed to the remote device from ascending memory locations starting with the location specified by the A ADDRESS and will continue until an ETX is detected. A READ will then be initiated on the remote device and the data will be passed to ascending memory locations beginning with the location immediately following the ETX control code which terminated the WRITE and will continue until an ETX control code from the remote device is encountered. Each portion of the message being written and read must be terminated by an ETX code.

The B ADDRESS field may optionally contain an End-of-File label.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

WRITE TO CONTROL/READ TRANSPARENT (WCRT) - PSEUDO.

This is a data communications pseudo and is defined for Advanced Assembler only. The function of Write to Control/Read Transparent is to pass data to a remote device (normally a computer) from memory locations and when successfully completed, to cause data to be read from the remote device and passed to appropriate memory locations and terminating at the end of the record-description without passing an End-of-Transmission control code.

The format for the WCRT instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	WCRTT			INRECD				ENDCTL							

There are no LABEL restrictions for this instruction.

The VAR field may contain either or both of the following function modifiers, left-justified, and in any order:

a. T - inhibit time-out.
b. D - dial.

The A ADDRESS field points to the record area which is to transmit and receive data or the file-name associated with the record. Indexing is not permitted. Data will be passed to the remote device from ascending memory locations starting at this field and will continue until a control code denoting End-of-Transmission is detected. A READ will then be initiated on the remote device, and the data word passed to ascending memory locations beginning with the location immediately following the End-of-Transmission control code which terminated the WRITE and will continue until the end of record area is filled. Control characters in the data received do not stop transmission.

The B ADDRESS field may optionally specify an End-of-File branch label.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

WRITE TRANSPARENT/READ TO CONTROL (WTRC) - PSEUDO.

This is a data communications pseudo and is defined for Advanced Assembler only. The function of this pseudo is to pass data to the remote device until the end of the record description is reached and, when successfully completed, to cause data to be passed from the remote device to memory locations starting at the end of the record-description and continuing until an ETX control code is detected. A maximum of 100 characters can be received during the READ.

The format for this instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	WTRCT			SPØ1				ENDSP1							

There are no LABEL restrictions for this instruction.

The VAR field may contain either or both of the following function modifiers, left-justified, and in any order:

a. T - inhibit time-out.
b. D - dial.

The A ADDRESS field points to the internal file-name or associated record name from which and to which data is to be transmitted. Indexing is not permitted.

The B ADDRESS field may optionally contain an End-of-File branch label.

The remaining fields are not used, nor are the Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

SECTION 8
SORTER – READER AND LISTER OPERATIONS CODES

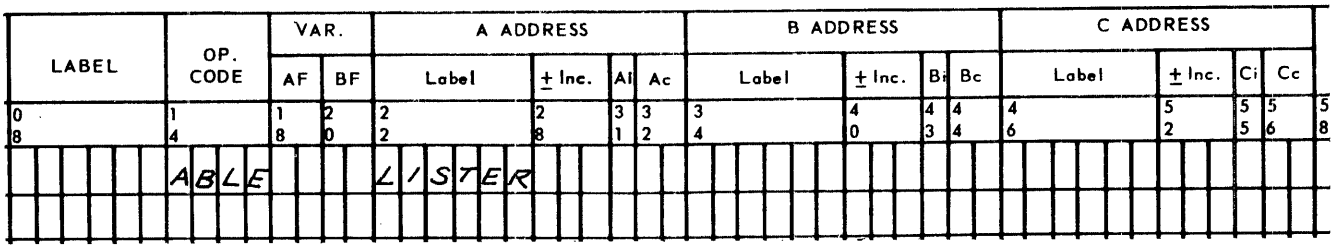
GENERAL.

The sorter-reader and lister operation codes are described in alphabetical sequence, and each description is presented in a standard format. Each instruction has a coding example followed by a description of LABEL entry restrictions and entries (if any) follow. Each field used is defined. Changes to the comparison and OVERFLOW indicators and Program Reserved Memory are described.

ENABLE LISTER (ABLE) - PSEUDO.

This is a MICR pseudo and is defined for the Advanced Assembler only. The function of this pseudo is to suspend the program until the Not Ready condition has been corrected.

The format of the ABLE instruction is as follows:



There are no LABEL restrictions when using this instruction.

The A ADDRESS field specifies the internal file-name of the sorter. The remaining fields are not used by this instruction. The file-name must have been opened before the ABLE can be executed.

Once the file has been enabled, the program is suspended until the LISTER Not Ready condition (Not Ready or End-of-Paper) has been corrected.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

ADVANCE BATCH COUNTER (CWNT) - PSEUDO.

This is a MICR pseudo and is defined for the Advanced Assembler only. It increments, by one, the batch counter in the specified sorter-reader.

The format for the CWNT instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	CWNT			SRTR1											

There are no LABEL restrictions when using this instruction. The A ADDRESS field must specify the internal file-name of the sorter-reader.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

Flow must be stopped before this command is issued.

FLOW

START SORTER-READER IN FLOW MODE (FLOW) - PSEUDO.

This pseudo physically starts a sorter-reader feed in the flow mode.

The format of the FLOW instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
AWAAY	FLØW			SRTR#1											

There are no LABEL restrictions when using this instruction.

The A ADDRESS label field specifies the internal file-name of the sorter. Incrementing, indexing, and address controlling are not permitted.

The B ADDRESS must contain an address to which control will pass if the sorter is in a flow stopped mode. Incrementing is allowed.

The C ADDRESS must specify a batch-ticket-routine. Incrementing is allowed.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

When a sorter-reader is started in the flow mode, logical read (SRTR) commands for each item must be executed within a time limit to prevent non-read conditions. For a more detailed explanation, refer to the B 2500/B 3500 Master Control Programs Information Manual.

TURN ON POCKET LIGHT (LGHT) - PSEUDO

This is a MICR pseudo and is defined for the Advanced Assembler only. It causes the selected pocket light on the specified sorter-reader to illuminate, thus requesting operator action.

The format of the LGHT instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
HEY	LGHT			SRTR1				K3							

There are no LABEL restrictions when using this instruction.

The A ADDRESS label field specifies the internal file-name of the sorter-reader. Incrementing, indexing, and address controlling are not permitted. The B ADDRESS points to a 2-digit unsigned numeric field containing the pocket number whose light is to be illuminated.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

Flow must be stopped and all documents pocket selected. The sorter-reader control is set to a Not Ready condition which must be cleared by the depression of the sorter-reader start button.

The next successful sorter-reader request, other than pocket light, turns off the selected pocket light.

LSTR

PRINT MTL RECORD (LSTR) - PSEUDO.

This instruction is a MICR pseudo and is defined for Advanced Assembler only. It releases a logical record to the multiple tape lister for printing.

The format of the LSTR instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	LSTR			LSTRFL				NØTRDY							

There are no LABEL restrictions when using this instruction.

The A ADDRESS label field specifies the record area containing the record to be released or the file name associated with the record. This record must be specified as being 44 characters in length. Incrementing, indexing, and address controllers are not permitted.

The B ADDRESS field must specify a Not Ready branch label to which control passes, if the lister is in a Not Ready condition, to allow for continuation or orderly suspension of the program. Indexing is not permitted.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The lister and tape designation must be loaded into the field referenced by the FILE declaration (columns 58 through 63) before this command is executed. The format for the various unit and tape designations for this field is outlined below:

Digit Positions

1	2	3	4
U	T	U	T

U = 0 - Suppress print.
U = Unit No. 1-3.
T = Tape No. 1-6.

For the master/slave/slave combinations, the first 22 characters of the record are printed on both the master tape of unit 1 and the tape designated by the first two digit positions (D1-D2). If the first digit position (D1) is zero, printing on both the master tape on unit 1 and the tape designated by the second digit position (D2) is suppressed.

For the 6-tape/6-tape combination, the second digit position (D2) must equal zero, then the first 22 characters are printed on the master tape of the unit designated by the first digit position (D1). If the first digit position (D1) is zero, printing of the master tape is suppressed.

The second 22 characters of the record are printed on the tape designated by the third and fourth digit positions (D3-D4). The printing of the tape designated by the third and fourth digit positions (D3-D4) is possible with the "18 tape lister."

For a more detailed explanation of multiple tape lister operations, refer to the B 2500/B 3500 Master Control Programs Information Manual.

EXIT FROM POCKET SELECT ROUTINE (PCKT) - PSEUDO.

This is a MICR pseudo and is defined for Advanced Assembler only. It pocket selects a MICR item, and performs the function of a RTRN.

The format of the PCKT instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
ENDPKSPCKT				MICR				PCKTAR				TØØLAT			

There are no LABEL restrictions when using this instruction.

The A ADDRESS label field must specify the internal file-name of the sorter-reader file. Incrementing, indexing, and address controllers are not permitted.

The B ADDRESS label field must point to a 4-digit unsigned numeric data field containing the following information in the format NNRV:

Code	Definition
NN	Pocket into which the item is to be selected.
R	Reserved (zero).
V	Zero (0) to continue flow, and one (1) to stop flow.

Indexing within the B field is not permitted.

The C ADDRESS field must contain a too-late-to-pocket-select branch label. If the PCKT command was issued too late, execution passes immediately to this label. Indexing is not permitted.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

For a more detailed explanation, refer to the B 2500/B 3500 Master Control Programs Information Manual.

SKIP

SKIP MTL. (SKIP) - PSEUDO.

This pseudo is defined for Advanced Assembler only. It causes the multiple tape lister to skip 2 1/2 inches of paper. Only the master tape and one detail tape will move.

The format for the SKIP instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
ENDRGNSKIP				LSTR1				NTRDY1							

There are no LABEL restrictions for this instruction.

The A ADDRESS label field specifies the internal file-name or associated record-name of the lister to be skipped. Incrementing, indexing, and address controlling are not permitted. The B ADDRESS label field must specify a Not Ready branch label to allow orderly continuation or suspension of the program. Indexing is not permitted.

The remaining fields are not used by this instruction.

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by this instruction.

The sorter must be stopped before this command is issued. Control is returned to the program after the skip has been completed.

Before this command is executed, the lister unit and the tape designation must be loaded into the field referenced by the FILE declaration (columns 58 through 63). The format of this field for the various unit and tape designations for SKIP is outlined below:

Digit Positions

1	2	3	4
U	T	U	T

U = Unit No. 1-3.
U = 0 - Suppress skip.
T = Tape No. 1-6.

For the master/slave/slave combination, a skip operation is performed on both the master tape of unit 1 and the tape designated by the first and second digit positions (D1-D2). If the first digit position (D1) is zero, then skipping on both the master tape on unit 1 and the tape designated by the second digit (D2) is suppressed.

For the 6-tape/6-tape combination the second digit position must equal zero. The skipping of the master tape on the unit designated by the first digit position (D1) is performed. If the first digit position (D1) is equal to zero, then skipping of the master tape is suppressed, an additional tape can be skipped as designated by the third and fourth digit positions (D3-D4).

If the first and third digit positions (D1-D3) are equal to zero, an error will occur.

For a more detailed explanation of multiple tape lister operations, refer to the B 2500/B 3500 Master Control Programs Information Manual.

SLEW

SLEW MTL (SLEW) - PSEUDO.

This pseudo is defined for Advanced Assembler only. It slews the paper 10 inches on the multiple tape lister.

The format for this instruction is as follows:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
ENDRUNSLEW				LISTR1				NØTRDY							

There are no LABEL restrictions for this instruction.

The A ADDRESS label field specifies the internal file-name, or an associated record-name of the lister to be slewed. Incrementing, indexing, and address controlling are not permitted.

To allow orderly continuation or suspension of the program, the B ADDRESS label field must specify a Not Ready branch label. Indexing is not permitted.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

Before this instruction is issued, the sorter must be stopped. Control returns to the program after the SLEW has been completed.

Before this instruction is executed, the lister unit and tape designation must be loaded into the field referenced by the FILE declaration (columns 58 through 63). The format of this field for the various unit and tape designations for slew is outlined below:

Digit Positions			
1	2	3	4
U	V	U	T

D1-D2: V = 0 Allow SLEW of master tape.

V = 1 Inhibit SLEW of master tape.

U = 1 SLEW all tapes, Unit 1 (V takes precedence over U).

U = 2 SLEW all tapes, Unit 2.

U = 4 SLEW all tapes, Unit 3.

U = 3 SLEW all tapes, Units 1 and 2 (For the 6/6-tape combination, only D1 = 3 can be designated).

U = 5 SLEW all tapes, Units 1 and 3.

U = 6 SLEW all tapes, Units 2 and 3.

U = 7 SLEW all tapes, Units 1, 2, and 3.

D3-D4: Suppress SLEW of designated tape (D3-D4 are used only on the 18 tape lister; otherwise D3-D4 must be zero).

U = Unit No. 1-3.
U = 0 - Do not suppress SLEW.
T = Tape No. 1-6.

SPAS

SPACE MTL ONE LINE (SPAS) - PSEUDO.

This pseudo is defined for Advanced Assembler only. It provides for single spacing on the multiple tape lister. Only the master tape and one detail tape will move.

The format for this instruction is:

LABEL	OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ADDRESS ± Inc.	A ADDRESS Ai	A ADDRESS Ac	B ADDRESS Label	B ADDRESS ± Inc.	B ADDRESS Bi	B ADDRESS Bc	C ADDRESS Label	C ADDRESS ± Inc.	C ADDRESS Ci	C ADDRESS Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
ENDBNKSPAS				LSTRFL				NØTRDY							

There are no LABEL restrictions for this instruction.

The A ADDRESS label field specifies the internal file-name or an associated record-name of the lister to be spaced. Indexing, incrementing, and address controlling are not permitted.

To allow for orderly continuation or suspension of the program, the B ADDRESS label field must specify a Not Ready branch label. Indexing is not permitted.

The remaining fields are not used, nor are Program Reserved Memory and the comparison and OVERFLOW indicators changed by this instruction.

Before this instruction is executed, the lister unit and tape designation must be loaded into the field referenced by the FILE declaration (columns 58 through 63). The format of this field for the various unit and tape designations for SPAS is outlined below:

Digit Positions			
1	2	3	4
U	T	U	T

U = Unit No. 1-3.
U = 0 - Suppress SPAS.
T = Tape No. 1-6.

For the master/slave/slave combination, a SPAS operation is performed on both the master tape of unit 1 and the tape designated by the first and second digit positions (D1-D2). If the first digit position (D1) is zero, then SPACING on both the master tape on unit 1 and the tape designated by the second digit (D2) is suppressed.

For the 6-tape/6-tape combination the second digit position must equal zero. The SPACING of the master tape on the unit designated by the first digit positon (D1) is performed. If the first digit position (D1) is equal to zero, then SPACING of the master tape is suppressed. An additional tape can be spaced as designated by the third and fourth digit positions (D3-D4).

If the first and third digit positions (D1-D3) are equal to zero, an error will occur.

For a more detailed explanation of multiple tape lister operations, refer to the B 2500/B 3500 Master Control Programs Information Manual.

SRTR

READ RECORD FROM SORTER FILE (SRTR) - PSEUDO.

This pseudo makes the next logical record from a sorter file available to the program (demand or flow mode).

The format of the SRTR instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
ROMCR	SRTR			MCRFIL				FLWSTP				BCHTKT				

There are no LABEL restrictions when using this instruction.

The A ADDRESS label field specifies the internal file-name of the sorter. Increment, indexing, and address controlling are not permitted.

The B ADDRESS label field specifies the address in the program to which control passes if the sorter is in a FLOW STOPPED condition.

The C ADDRESS label field specifies the address in the program to which control passes if the item processed by the prior SRTR command was a batch ticket (Black Band Document).

Program Reserved Memory and the comparison and OVERFLOW indicators are not changed by the instruction.

The logical record made available has already been pocket selected by the PCKT command in the USER declarative procedures.

```
PROGRAM ID = MICREX                                        B3500 ASSEMBLER  08/15/68
   MEMORY    SEQ NO   LABEL    OP      AFBF   A-LBL    INC AIAC  B-LBL    INC BIBC  C-LBL    INC CICC  REMARKS
                              SPEC    N      CARD   :    :  :            :    :  :    000010:000:0:10  LIST
              000010           IDNT           MICREX:    :  :
              000020           USER           RDERR  :    :  :    AMTERR:    :  :    RTERR  :    :  :    PKTSEL
      (64)    000030  SOR1     FILE   0 6             :    :  :    SOR    :    :  :    0      :    :  :    ENDFIL
     (304)    000040  SORREC   RECD   200    UA      :    :  :            :    :  :
     (304)    000050  OCRIT    DATA   100    UA
     (504)    000060  MICRIT   DATA   100    UA
     ******** LOC CTR ADJUST
    (1152)    000070  TAPEOT   FILE          ITMCAP:    :  :    MTP    :    :  :       2999:010:W:
    (1472)    000080  TPREC    RECD   104    UA      :    :  :            :    :  :
              000090           ENDF
    (8104)    000100  PKCELL   RECD     4    UN      :    :  :            :    :  :
    (8104)    000110  PKTID    DATA     2    UN
    (8106)    000120           DATA     1    UN
    (8107)    000130  SFLOW    DATA     1    UN
              000140           ENDR
              000150
              000160
              000170                                                 **********************
              000180                                                 * USER PROCESS PHASE *
              000190                                                 **********************
    (8108)    000200  BEGIN    OPEN   IN F   SOR1   :    :  :
    (8130)    000210           OPEN   OT     TAPEOT:    :  :
    (8152)    000220  STFLW    FLOW         SOR1   :    :  :    STFLW  :    :  :    STFLW  :    :  :
    (8186)    000230           BUN          PRCESS:    :  :
    (8194)    000240  READ     SRTR         SOR1   :    :  :    STFLW  :    :  :    BATTKT:    :  :
    (8228)    000250  PRCESS   MVA    0000   MICRIT:    :  :    TPREC  :    :  :
    (8246)    000260           MVA     4 4   PKCELL:    :  :    TPREC  :100:  :
    (8264)    000270           WRIT         TPREC  :    :  :            :    :  :
    (8294)    000280           BUN          READ   :    :  :
    (8302)    000290  BATTKT   CWNT         SOR1   :    :  :
    (8324)    000300           BUN          STFLW  :    :  :
    (8332)    000310  LATEPK   MVN     2    15     :    :  :NL  PKTID  :    :  :
    (8350)    000320           BUN          PRCESS:    :  :
    (8358)    000330  ENDFIL   DISP         JAMMSG:    :  :
    (8382)    000340           LGHT         SOR1   :    :  :    PKTID  :    :  :
    (8410)    000350           BUN          STFLW  :    :  :
    (8418)    000360  JAMMSG   CNST    18   UA                    SORTER 1 IS JAMMED
              000370
              000380
              000390                                                 ******************************
              000400                                                 * USER POCKET SELECT PHASE *
              000410                                                 ******************************
    (8454)    000420  RDERR    MVN     2    02     :    :  :NL  PKTID  :    :  :
    (8472)    000430           BUN          POCKET:    :  :
    (8480)    000440  AMTERR   MVN     2    03     :    :  :NL  PKTID  :    :  :
    (8498)    000450           BUN          POCKET:    :  :
    (8506)    000460  RTERR    MVN     2    04     :    :  :NL  PKTID  :    :  :
    (8524)    000470           BUN          POCKET:    :  :
    (8532)    000480  PKTSEL   MVN     2    01     :    :  :NL  PKTID  :    :  :
    (8550)    000490  POCKET   PCKT         SOR1   :    :  :    PKCELL:    :  :    LATEPK:    :  :
              000500           FINI
```

Figure 8-1. Example of Sorter-Reader Program

SECTION 9
FREE-FORM ASSEMBLY LANGUAGE

GENERAL.

This section describes a free-form assembly language which incorporates all the functions of the B 2500/B 3500 Assembly Language. All elements of both the Basic and Advanced Assembler languages are provided. A translator which converts free-form language to regular assembly language is provided as systems program FFT.

LANGUAGE DESCRIPTION.

The pseudo FIXD is used to signal an escape from the free-form assembly language described in this section, thus allowing insertion of statements in the regular fixed format. The pseudo FREE is used to signal return from fixed-form assembly language to free-form assembly language. The pseudo FREE must be entered in card columns 1 through 4. FIXD and FREE terminate translation of the card in which they appear. The first statement in the selected mode is assumed to start in the next card. The statement delimiter (semicolon) is optional following these pseudos.

A scale is printed on a remote 9350 following the receipt of a record containing FIXD to facilitate fixed format alignment.

DELIMITERS.

Certain special characters (delimiters) are used to locate or define fields. These delimiter characters and their function are listed below.

Character	Function
#	Locate the start of the sequence statement.
: (colon)	Delimit the statement label.
Blank	Delimit various fields. See the particular field discussion for mandatory or optional usage.

Character	Function
/	Locate the end of the AF and/or the beginning of the BF entry.
"	Enclose strings. Applies only to the following: REMK, DOCU, CNST, and literals.
( )	Locate modifiers (see modifiers).
; (semicolon)	Statement delimiter. Signals the translator to write the translated statement as output.
←	End-of-Record delimiter. Signals the translator to read another record. Automatically inserted by the translator at the end of the input buffer.
##	Error deletion signal.

NOTE

Some delimiter characters may be used in labels without conflict with their syntactic function, but delimiters should in general be avoided to prevent possible failure of the translator. The characters #, /, ", and) may be embedded in labels (for exceptions, refer to page 10-8). The characters :, blank, (, ;, and ← must not be used in labels. All delimiter characters may occur in strings without conflict.

STRINGS.

A string is a list of characters enclosed in quote marks. The quotes are not considered part of the string. A literal or constant string may contain any character including the quote mark.

Examples:

```
MVN 1/ "2" HOLD;
CNST /6 UA "#,$, "";
```

STATEMENTS.

A statement is the free-form code corresponding to a single B 2500/B 3500 symbolic assembly instruction (blank input records are ignored). A statement may begin anywhere on the input record and be continued across as many records as desired. Each statement is made up of several fields. The translator regards the following as fields:

a. Sequence number.

b. Statement label.

c. Op Code.

d. AF or BF (if both are present, then the pair constitutes a single field).

e. The A, B, and C ADDRESS.

f. ADDRESS modifiers.

g. REMARKs.

h. Strings.

i. Each word of REMK or DOCU text.

The following are restrictions for statements:

a. Every statement must be followed by the statement delimiter (semicolon).

b. A statement <u>field</u> may not be continued from one record to the next.

c. No field may begin with the error deletion signal ## unless one intends to delete the entire statement.

d. The last statement to the translator must be the FINI statement.

SEQUENCE NUMBER (OPTIONAL).

A pound sign (#) followed by one to six digits identifies the sequence number, and the pound sign is not considered part of

the sequence number. One or more spaces must follow the last character of the sequence number, and the sequence number is left-justified on the output record.

Examples:

#123456	becomes	123456
#001	becomes	001

No embedded blanks or non-numeric characters are permitted in the sequence number.

STATEMENT LABEL.

A valid label or program point followed by a colon. The colon is not considered part of the label.

Examples:

BBX33:	becomes	BBX33
.A:	becomes	.A
START:	becomes	START

No embedded colons are permitted in the statement label.

ADDRESSES.

The A, B, and C ADDRESS fields must be separated by one or more spaces or modifiers.

Examples:

```
CPA       INPUT       OUTPUT;
CPN      INPUT(UN)OUTPUT;
MPY      X Y Z COMPUTE Z=X TIMES Y;
MPY    X(UN)Y(UA)Z(UN)  COMPUTE Z=X   TIMES Y;
```

The translator contains a table indicating the maximum number of addresses associated with each Op code. Fields which do not constitute an ADDRESS are treated as remarks and placed in the REMARK field (columns 58 through 80).

MODIFIERS.

The functions of indexing, incrementing, and address controllers are regarded as modifiers of the applicable ADDRESS. Modifiers must be enclosed in parentheses and follow the A, B, or C ADDRESS to which they apply.

Examples:

```
MVA     INPUT(UA)       OUTPUT(UN);
MVR     3/40 (AL)     PRINT;
```

In the first example, the A ADDRESS modifier is UA and the B ADDRESS modifier is UN. The second example shows a blank ADDRESS modified by AL.

Spaces are not required between the ADDRESS and its modifiers.

Examples:

```
MVA     INPUT(UN)       OUTPUT;
MVN      INPUT(UN)OUTPUT;
MVA     INPUT         (   UN   )    OUTPUT    (UA);
```

Indexing is denoted by a single digit from 1 to 3.

Examples:

```
INC     FICA      PAYROL(1 UA);
EXT    BASE(3 IA);
MVA     INPUT(2)       OUTPUT(1);
```

Incrementing or decrementing must include the sign.

Examples:

```
MVA     INPUT(-23)       OUTPUT(+3);
MVN    COKE(+123)       BOTTLE(-1);
```

Modifiers may be written in any permutation. At least one space must separate modifiers from each other.

Examples:

```
MVA      INPUT     (3)     OUTPUT;
MVA     INPUT(3    UA)    OUTPUT;
INC      SUM(+2    3   UN)    TTL;
INC      SUM(UN    -12   1)   ATTL;
```

NULL MODIFIER.

Writing the left and right parenthesis alone forms the null modifier. The null modifier can be used to delimit a void ADDRESS label field, thus leaving it blank.

Examples:

```
EXT()       SKIP A-ADDRESS;
INFL   ()    SIZE;
INFL(     )     00038;
```

NOTE

To omit an optional B or C ADDRESS, an extra modifier must be coded, since one of them will be associated with the A ADDRESS entry. For example, READ FILE () () CARD READ;

OP CODE.

Any valid OP code followed by one or more blanks, or the alternate delimiters discussed below. An OP code may be delimited by a semicolon whenever the OP code by itself is a complete statement.

Examples:

```
REMK;
EXT;
ENDR;
ENDF;
```

An OP code may be delimited by a slash (/) whenever the instruction does not require an AF entry.

Examples:

```
MVA/3 INPUT OUTPUT;
CPN/1 TABLE TOP;
```

An OP code may be delimited by the null modifier whenever the AF and BF entries are not written and the A ADDRESS is to be set to blanks.

Examples:

```
INFL() SIZE;
EXT() REMARK ON EXIT;
```

NOTE

EXT, in the example above, normally allows for an A ADDRESS. Had the null modifier been omitted, the word REMARK would have appeared on the output as the A ADDRESS. The desired result, by the above coding, was to blank out the A ADDRESS and place the words REMARK ON EXIT in the REMARK field (columns 58 through 80).

AF AND BF ENTRIES.

The AF and BF entries must be separated by a slash even if only one of the entries is required or written. No slash is required if the instruction requires no AF or BF entries. One or more spaces must follow the BF field.

Examples:

FREE FORM		AF	BF
3/2	becomes	03	02
/F	becomes		F
14/	becomes	14	
2/A	becomes	02	A

The AF and BF entries are examined to determine if they contain numeric or alphanumeric data. If the contents are numeric, leading

zeros are supplied. If the contents are alphanumeric, trailing blanks are supplied. (See above example.) It may be necessary to write the slash when an AF or BF entry is not required. For example, both assemblers accept a label containing a slash such as RD/WT. Assume it is desirable to write the code as:

```
NTR     RD/WT;
```

In this case, the translator would look at the RD/WT as an AF/BF field. To avoid this conflict write:

```
NTR  /  RD/WT;
```

This allows the translator to look on the first slash as the AF/BF field (with no entries) and then process the RD/WT as an A ADDRESS field.

The restrictions for AF and BF entries are:

a. The AF entry must not contain a semicolon, an embedded blank, an End-of-Record delimiter, or a slash.

b. The BF entry must not contain a semicolon, an embedded blank, or an End-of-Record delimiter.

LITERALS.

If a literal string is strictly numeric (made up only of the numbers 0 to 9), the translator automatically assigns an address controller of NL (numeric literal). If the literal is not strictly numeric, the address controller of AL (Alpha Literal) is assigned. The automatic controller assignment may be overridden by a modifier entry. Literals do not have to be written as a string unless automatic controller assignment is desired or the literal contains delimiters. Thus several variations are possible to create a literal in the A ADDRESS field.

Examples:

```
MVA 1/ """  OUTPUT;
MVA 3/ "%"!" OUTPUT;
```

```
MVN 1/ A(NL) OUTPUT;
MVN 1/ "A"(NL) OUTPUT;
MVN 1/ 2(NL) OUTPUT;
MVN 1/ "2" OUTPUT;
MVR 3/40 (AL) PRINT;
```

The following are restrictions for creating literals:

a. The AF entry must be given.
b. Delimiters, other than blank, must be in a string.

SPECIAL STRING.

A special string is the statement delimiter (;) or End-of-Record delimiter (←) enclosed in quote marks, preceded and followed by at least one blank.

Examples:

";"
"←"

Special strings must be used for carrying these delimiters as text in a REMK, DOCU, or COMMENT portion of an instruction.

REMK AND DOCU PSEUDOS.

These pseudos must be followed by one or more spaces if any text is to follow. The text begins with the first non-blank character and may continue on subsequent input records until reaching the statement delimiter (semicolon). Each word of the text is regarded as a field, and unnecessary spaces between words are ignored.

The output text is placed in the equivalent positions (columns 22 through 80). Text which exceeds these 59 character positions is truncated from the right.

The following is a restriction for the REMK and DOCU pseudos:

If the error deletion signal, statement delimiter or End-of-Record delimiter is to be carried as text, then it must be enclosed in quotes (e.g., "##").

FREE-FORM COMMENTS

It may be desirable to have certain comments appear on the free-form listing which are not to appear on the ASMBLR version. The End-of-Record delimiter implies this feature.

Examples:

```
← Starting a record with the left arrow
← allows all of these comments to
← appear on the free-form listing, yet
← not be translated or appear on the
← ASMBLR listing.
```

CNST DECLARATIVE.

Constant data must be written as a string. The class (i.e., UA, UN, SN) is treated as an A ADDRESS and not as a controller.

Examples:

```
CNST /2  UA  "AB";
CNST /12  UN  "123456789032";
CNST /3 UA BLANKS;
```

It would be incorrect to write:

```
CNST /2 "AB"(UA);
```

The length of the string is determined by the concatenation of the AF and BF entries. If the concatenation exceeds a value of 24, then only the first 24 characters will be translated.

The string size must be given in the AF/BF field.

PICT DECLARATIVE.

The PICT declarative is coded like the CNST declarative, except that the PICTURE text is <u>not</u> enclosed in quotes.

Example:

```
#020550  Z6:  PICT  /4  UA  Z(6);
```

The first address field must be coded UA. The PICTURE may not contain an embedded semicolon.

REMARKS.

REMARKs are for documentary assistance to the programmer. Any fields which remain after processing the A, B, or C ADDRESSes are regarded as remarks and are placed in the REMARK field (columns 58 through 80). REMARKs which exceed a length of 23 characters are truncated from the right. An abort signal, semicolon or End-of-Record delimiter in the text of the REMARK must be enclosed in quotes (e.g., "##").

ERROR DELETION.

Any statement may be deleted from output by the presence of two adjacent pound signs (##) at the beginning of a field. The deleted statement must be completed with a semicolon.

Examples:

```
INC  1K  ##  ;
MVA  INPUT OUTPUT  ##;
```

The delete signal may be contained within a string without causing the statement to be deleted.

Examples:

```
MVA 2/ "##" TANK  DELETE SIGNAL AS LITERAL;
CNST /2 UA "##" DELETE SIGNAL AS CONSTANT;
REMK THE DELETE SIGNAL FOR THIS PROGRAM IS "##" ;
```

FILE DECLARATIVE.

The FILE declarative is written in free-form format in a manner analogous to any other command. For the Basic Assembler language in which the maximum record size is required, the entry is written as an AF/BF entry. Modifiers are used to insert information in the required card columns other than the A, B, or C ADDRESS and the REMARK fields.

Examples:

```
(BASIC) Name:
FILE 01/00 FILEID  (MUFILE) XXXX01 (500000) S;
```

```
(ADVANCED) Name:
FILE FILEID (MUFILE) DSKS (999901) SS KEY1;
```

Entries for columns 22-27, 34-39, and 46-51 are left-justified and filled with trailing blanks to a length of six. Note that in the first example four filler characters are required for columns 34-37 and, in this case, must be zeros.

SPEC PSEUDO.

The SPEC pseudo, in fixed-form coding, must be the first input record. The translator assumes free-form format starting with the next input record.

A printed listing of the free-form coding may be obtained by punching an L in column 11 of the SPEC pseudo.

A fixed-form SYMTIN symbolic tape may be updated with free-form statements. This feature is invoked by a P in column 12 of the SPEC pseudo. The A ADDRESS of the SPEC pseudo specifies the source device for correction cards in this case.

The A ADDRESS specifies the source device for free-form statements. The following entries are permitted:

a. PAPER - paper tape reader.
b. CARDS - card reader.
c. TAPE - magnetic tape.
d. TYPE - remote B 9350 SPO (Advanced Translator only).

NOTE

When using a B 9350 SPO, the translator deletes CR and LF from the input source.

DLET PSEUDO.

The DLET pseudo is, at present, ineffective in free-form update translations.

OPEN PSEUDO.

Both Basic and Advanced Free-Form Translators expect three operands on an OPEN statement. If REMARKs are to be included and only one address is coded, null modifiers must be used. For example:

```
OPEN IN/ND file-name () () REMARKs;
```

MEMORY PSEUDO.

An Advanced Free-Form Program may inform the translator how much core is available for assembly of the translated program by means of the MEMORY pseudo. The format is:

```
MEMORY = ⟨core available for assembly, in digits⟩;
```

The MEMORY pseudo must appear on a record by itself.

Example:

```
IDNT  TSTCRD; MEMORY = 40000; AREA: CNST /5 UA "ABCDE";
```

ADVANCED FFT FILE NAMES.

For purposes of label equating, the internal file names and external file ID's for Advanced Free-Form Translator files are:

Internal File-Name	External File ID	Function
REMOTE	REMOTE	Remote SPO in
CARDS	CARDS	Cards in
SYMPIN	SYMPIN	Paper Tape in
SYMTIN	SYMTIN	Magnetic Tape in
SYMDIN	SYMDIN	Disk in
PRINT	PRINT	Print out
DFW	SYMDIN	Disk out

FREE-FORM MESSAGES.

The following output messages are embodied in the free-form language.

```
** BOJ FFT  (appears on B 9350 SPO)
SEQ# size too large
Invalid OP code
BF size too large
```

Missing quote mark - LITRL
Missing quote mark - CNST
Wrong delimiter
Size on + increment
Size on - increment
REMK exceeds 59 characters
** EOJ FFT (appears on B 9350 SPO)

```
#002 IDNT LSTCRD;
#003 CRDFIL: FILE CARD CRD SS2 CARDS IN ;
#004 CRDREC: RECD /80 UA RECORD AREA;
#005 PRTFIL: FILE PRINT PRN SS2 PRINTOUT;
#006 PRTREC: RECD 1/32 UA PRINT RECORD;
#007 ENDF;
#008 COUNT: CNST /4 UA "0000";
#020 REMK OPEN FILES, CLEAR PRINT AREA, GET READY;
#021 OPEN IN/ CRDFIL;
#022 OPEN OT/ PRTFIL;
#023 SPRD 40/40 PRTREC ;
#024 REMK MAIN READ-PRINT LOOP FOLLOWS;
#025 READ: READ CRDFIL ENDFIL;
#026 MVW /40 CRDREC PRTREC MOVE IMAGE TO PRT;
#027 INC 1/ "1" COUNT COUNT CARDS;
#028 MVA /4 COUNT PRTREC (+100);
#029 WRIT 1/ PRTREC * PRINT RECORD;
#030 BUN READ;
#031: ENDFIL: CLOS CRDFIL; #032 CLOS PRTFIL; #033 STOP; #034 FINI;
# OF ERRORS = 0000
```

Figure 9-1. Example of an Advanced Free-Form Translator Program Source Listing.

SECTION 10
MACRO FACILITY FOR ADVANCED ASSEMBLER

GENERAL.

The Advanced Assembler provides a facility for the programmer to define Macro and Library routines. These can be defined within the program itself or placed in a Macro library on disk for general use. In addition to the ability to create a library of Macro and Library routines, there are library maintenance capabilities such as patching, adding, removing and copying routines. The following paragraphs describe the syntax for writing a Macro or Library routine, the method used to call a routine, and the creation and maintenance of a Macro library on disk.

MACRO SYNTAX AND METHOD FOR CALLING A MACRO.

FEATURES OF A MACRO.

The Macro may specify assembler language statements which may or may not be assembled or may be altered depending upon conditions evaluated at assembly time. For more information on this feature, refer to the paragraph on Macro conditionals (page 10-2).

The Macro may inform the assembler to generate error messages when the rules for writing a Macro call are violated. This is accomplished through the MERR statement which is defined later.

A Macro may call other Macros.

A Macro may be user-defined, i.e., defined within a user's program, or called from a system library.

THE THREE FUNDAMENTAL COMPONENTS OF A MACRO DEFINITION.

A header statement begins the Macro definition. It is followed by model statements, and the definition is concluded with a Macro end card. These statements are fully described in the following paragraphs.

HEADER. The header card indicates the beginning of a Macro definition. The format of the header statement is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	AI	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
$NAME	MACR	MOVE	$NUMBER,	$FROM,	$TO											

A Formal Parameter is indicated by a $ followed by one to five characters. This character string must follow normal assembler label conventions. Formal Parameters in the Macro definition are replaced by corresponding Actual Parameters specified in a calling statement.

A Formal Parameter may be specified in columns 8 through 13. This is used to transfer a column 8 label on the Call Statement to a Model Statement in the Macro.

The operation code MACR specifies a Macro definition.

Columns 18 through 21 specify a 4-character name of the MACRO. Embedded blanks are not permitted.

Columns 22 through 57 specify the Formal Parameters (optional). A comma must immediately follow each Formal Parameter except the last, where the comma must be omitted. The parameters are coded in free-field format and may be continued onto additional cards as needed.

MODEL STATEMENTS. The Model Statements comprise a combination of regular assembler instructions and pseudos, plus the following Macro conditionals and pseudos.

Macro conditionals:

OP Code	Function
MEQL	Equal.
MLSS	Less.
MGTR	Greater.
MLEQ	Less than or equal.
MGEQ	Greater than or equal.
MNEQ	Not equal.
BOOL	Evaluate Boolean.

Macro pseudos:

OP Code	Function
MACR	Header.
MBUN	Unconditional branch to another Macro conditional or Macro pseudo.
MNOP	Macro NOP.
MERR	Macro specifies error message to appear in Assembly listing.
MEXT	Macro specifies that no more statements are to be generated.
SETB	Set Boolean.
CLRB	Clear Boolean.
MEND	End of definition.

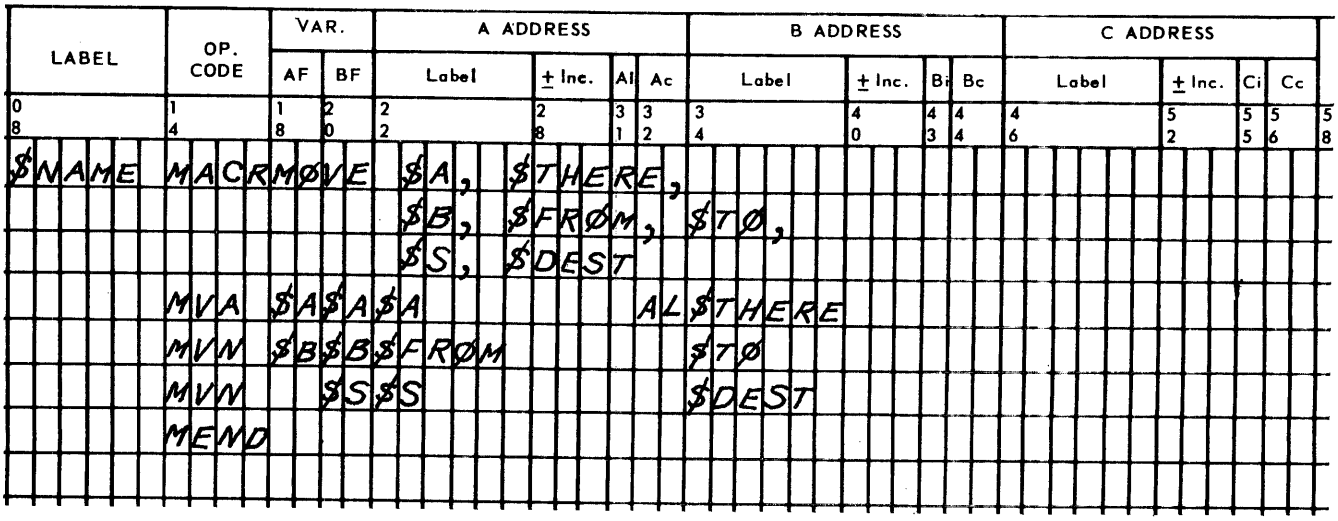
An explanation of the use of model statements follows under the heading, details for defining a Macro.

NOTE

FILE and other declaratives in Advanced Assembler cannot be generated parametrically in a Macro.

THE MEND STATEMENT. The MEND card is required as the last statement in the Macro definition.

EXAMPLE OF A MACRO DEFINITION. The format is:



CALLING A MACRO.

A Call Statement is used to reference a Macro definition. A label in column 8 is optional. The name of the Macro being referenced is used as the OP code of the calling statement. If the Macro being called is user-defined within the program and has the same name as a Macro on disk, the one defined in the program will be used.

The Actual Parameters to be passed to the Macro definition are coded in columns 22 through 57. This parameter list corresponds to the parameter list in the Macro header card with the exception that it is not necessary to make a reference to every Formal Parameter in the Macro definition. See the description of Null and Omitted Parameters (page 10-5).

Actual Parameters are coded in free-field format and may be coded onto additional cards as needed.

The parameter list may contain signed or unsigned numbers, character strings enclosed by quotation marks or program labels.

An example of the Call Statement follows along with the code that is generated as the result of the call (See Macro MOVE defined in the last example).

Call Statement:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	MØVE			"ABC",	J,		25,	X,	Y,			SØURCE,	DEST/W		

Generated Code:

LABEL	OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ADDRESS ± Inc.	A ADDRESS Ai	A ADDRESS Ac	B ADDRESS Label	B ADDRESS ± Inc.	B ADDRESS Bi	B ADDRESS Bc	C ADDRESS Label	C ADDRESS ± Inc.	C ADDRESS Ci	C ADDRESS Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	MVA	03	03	ABC			AL	J							
	MVN	25	25	X				Y							
	MVN		06	SOURCE				DESTIN							

NOTE

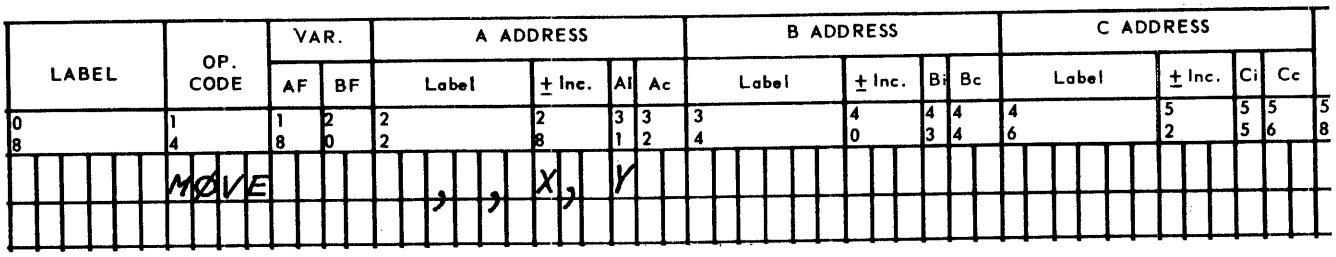
A reference to a Formal Parameter in the AF or BF field of a Macro Model Statement will yield different values dependent upon the Actual Parameter in the Call Statement. When the actual parameter is a numeric value, that numeric value is substituted as in the above example where 25 was substituted for $B. When the actual parameter is a character string, the number of characters in that string is used. As in the example, 3 was substituted for $A when the Actual Parameter was "ABC." Finally, when the Actual Parameter is a program label, the declared size of the variable to which the label refers is substituted for the Formal Parameter.

NULL AND OMITTED PARAMETERS. In the Call Statement it is not necessary to make a reference to every Formal Parameter in the Macro definition. If the parameter to be omitted in the calling statement precedes any that are to be passed to the Macro (a Null Parameter), a comma must be employed as a place-saver. This is not necessary if the omitted parameter follows the desired Actual Parameters.

NOTE

Keyed References, which are described below, may also be used when the programmer does not wish to reference all of the Formal Parameters.

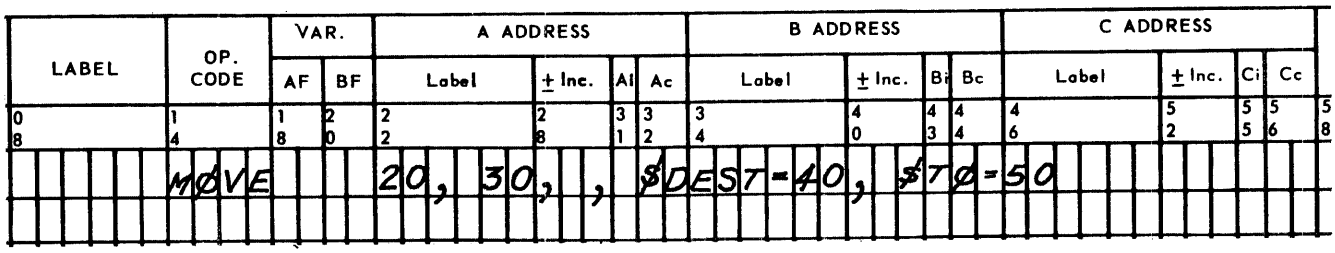
The following is an illustration of Null and Omitted Parameters where the Macro which is called was defined in a previous example.



For the above call, $A and $THERE are Null Parameters; $TO, $S, and $DEST are Omitted Parameters.

KEYED REFERENCES. In cases where the Formal Parameter list is rather long, and a limited subset of the Formal Parameters is to be referenced, Keyed References may be used. In the Call Statement, the Formal Parameter is written and equated to the Actual Parameter.

The following is an example of Keyed References, where the Macro MOVE was defined in a previous example:



In the above call, the Formal Parameters $A, $THERE, $DEST, and $TO are present; $B is null; and $FROM and $S are omitted.

In a Call, once an access is made via Keyed Reference, normal designation of Actual Parameters is not permitted.

Keyed References may be in any order.

MACROS CALLING OTHER MACROS. A call on a previously defined Macro may be embedded in a Macro definition.

The nested Call Statement may pass actual parameters as described above or by references to the Formal Parameters of the calling Macro.

The maximum level of nested Macro calls is 3-deep.

NESTED CALLS OF SYSTEM MACROS. Within a Macro definition, it is permissible to "call" a system Macro, i.e., one which has been defined and copied onto disk. To do this it is necessary to identify the system Macro for the calling routine with the SYST pseudo. This SYST card should be placed before the Header card of the Macro definition calling the system Macro.

The format of the SYST pseudo is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS				
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc	
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6	5 8
	SYST			MOVE												

The name of the referenced Macro is specified in column 22. The remaining fields are not used.

NOTE

Although system Macros may be called within a Macro definition, a Macro definition within another Macro definition is not permitted. In other words, a Macro may call other Macros, but Macro definitions may not be nested.

DETAILS FOR DEFINING A MACRO.

CNST AND DATA DECLARATIONS. If a CNST or DATA statement in a Macro definition specifies only a Formal Parameter in column 22, a complete declaration will be generated when there is a call on the Macro. The length and data format of the Actual Parameter passed in a Call Statement will be used when the CNST or DATA declaration is generated.

If there are references in fields other than one beginning in column 22 in a CNST or DATA declaration with a Macro definition, only explicit substitutions will be generated.

The following are examples of CNST declarations within a Macro definition, calls on the Macro and generated code.

Definition:

LABEL	OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ± Inc.	A Ai	A Ac	B ADDRESS Label	B ± Inc.	B Bi	B Bc	C ADDRESS Label	C ± Inc.	C Ci	C Cc
	MACRMØVE			$A,	$B,		$C								
	CNST			$A											
X	CNST	$A		$B				123456789							
Y	CNST		4UA					$C							

Call:

LABEL	OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ± Inc.	A Ai	A Ac	B ADDRESS Label	B ± Inc.	B Bi	B Bc	C ADDRESS Label	C ± Inc.	C Ci	C Cc
	MØVE			3,"UN",		"	ABCD"								

Generates:

LABEL	OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ADDRESS ± Inc.	A ADDRESS Ai	A ADDRESS Ac	B ADDRESS Label	B ADDRESS ± Inc.	B ADDRESS Bi	B ADDRESS Bc	C ADDRESS Label	C ADDRESS ± Inc.	C ADDRESS Ci	C ADDRESS Cc
	CMST		1	UM				3							
X	CMST		3	UM				123							
Y	CMST		4	UA				ABCD							

LABELS. In a Macro definition regular Assembler instructions and pseudos may reference only labels appearing on other Assembler instructions and pseudos. Similarly, Macro Conditionals and Pseudos may refer only to labels appearing on other Macro Conditionals and Pseudos.

MACRO-ASSIGNED LABELS. An "(IX)" as the last 4 characters of a label in a Model Statement causes the Assembler to provide Macro-assigned labels. The first call on the Macro will generate 0001 as the last four characters of each Macro-assigned label; the second call will generate 0002, the third 0003, and so on.

Although Assembler instructions and pseudos in a Macro definition may use normal labels and/or point labels, the above technique will avoid the generation of duplicate labels with multiple calls on a Macro within a program.

MACRO CONDITIONALS. Macro Conditionals within the Macro definition provide the capability of tailoring the generation of Assembly language statements and their sequence when the Macro is called.

Macro Conditionals may examine four characteristics of the Actual Parameters: TYPE, CONTroller, SIZE, and VALUe.

The format of a Macro Conditional is:

LABEL	OP. CODE	VAR. AF	VAR. BF	A ADDRESS Label	A ADDRESS ± Inc.	A ADDRESS Ai	A ADDRESS Ac	B ADDRESS Label	B ADDRESS ± Inc.	B ADDRESS Bi	B ADDRESS Bc	C ADDRESS Label	C ADDRESS ± Inc.	C ADDRESS Ci	C ADDRESS Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	MEQL	TYPE	VALUE					$ABC				PØINTA			

The operation code in column 14 specifies the condition for which the Macro is testing.

The characteristic to be examined is indicated in columns 18 through 21. The four-letter codes are TYPE, CONT, SIZE and VALU.

The condition of the given characteristic which will be tested for true is specified in column 22. See the table below for the entries permitted with each characteristic and their meanings.

The formal parameter being referenced is written beginning in column 34.

The C ADDRESS field contains a label to which the Macro will branch to continue generation if the condition is met. This label must appear on another Macro Conditional or on a Macro Pseudo. If the condition is not met, the Macro generation continues with the next sequential statement.

The following is a table of conditional test possibilities.

Table 10-1
Conditional Test Possibilities

Characteristic of Parameter	Condition of Characteristic	Meaning of Condition	Possible Status of Condition
TYPE	VALUE	parameter is a value	MEQL
	LABEL	PARAMETER IS A LABEL	MNEQ
	NULL	PARAMETER IS NULL	
	OMIT	PARAMETER IS OMITTED	

Table 10-1 (cont)

Conditional Test Possibilities

Characteristic of Parameter	Condition of Characteristic	Meaning of Condition	Possible Status of Condition
CONT	UA	UNSIGNED ALPHA	MEQL
	UN	UNSIGNED NUMERIC	MNEQ
	SN	SIGNED NUMERIC	
SIZE	dddd (four digits)	no. of characters or digits	MEQL
			MNEQ
			MLSS
			MGTR
			MLEQ
			MGEQ
VALU	signed value	value or	MEQL
	unsigned value	character string	MNEQ
	character string	to be compared to parameter	MLSS
			MGTR
			MLEQ
			MGEQ

NOTE

In table 10-1 "VALU" cannot be tested for parameters passed by label.

BOOLEANS. The Assembler provides Boolean toggles which can be programmatically tested, set, and reset.

There are up to 99 Global toggles designated G01, G02, . . ., G99. At the beginning of an Assembly, all Global toggles are initialized to False. Any Global toggle can be set or reset by the BOOL, CLRB and SETB statements which are described below. Once set or reset, the toggle retains that status until it is changed by a subsequent BOOL, CLRB, or SETB.

There are up to 99 Local Boolean toggles designated L01, L02, . . ., L99. When a Macro is called, all the Local toggles are reset. The Formal Parameters of the Macro definition are in a one to one correspondence with the Local toggles, and these are set or reset depending on the presence or absence of Actual parameters in the Calling statement. For example, L01 is set true if the first Actual Parameter is present; it is reset False if the first Actual Parameter is null or omitted.

Boolean Conditional (BOOL).

The BOOL Conditional provides the capability to make comprehensive tests on the status of missing operands, allows communications from one Macro to another Macro, and allows for the testing of compound conditions.

The format of the BOOL Conditional is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	BOOL			FIRST	602			L01	ØR	606	;				

The A ADDRESS field optionally contains a label pointing to another Macro statement from which Macro generation will proceed if the designated Boolean expression is False. If the Boolean expression is True, Macro generation continues with the next sequential instruction. If this option is not used, Macro generation continues with the next sequential instruction regardless of the value of the Boolean expression.

The A ADDRESS increment field optionally designates a single Global or Local toggle which will be set or reset in accordance with the value of the Boolean expression.

A Boolean expression is written left-justified in the B ADDRESS and C ADDRESS fields and may extend beyond column 58 to the end of the card. Three operators are permitted in a Boolean expression. In order of precedence, these are the reserved words AND, OR, and NOT which are interchangeable with the symbols *, +, and -, respectively. The expression must be delimited by a semicolon. Parentheses may be used within a Boolean expression to change the order in which the operators are evaluated.

SETB and CLRB Pseudos.

The SETB pseudo is used to set a designated Global or Local toggle. The CLRB pseudo is used to reset a designated Global or Local toggle.

The format for the SETB and CLRB pseudos is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	SETB			601											
	CLRB			603											

The A ADDRESS field contains the toggle to be set or reset.

The remaining fields are not used by these instructions.

MERR PSEUDO. This pseudo causes the generation of a syntax error and user-defined message during assembly.

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	MERR			THE FIRST PARAMETER IS MISSING											

DEFINING AND CALLING A LIBRARY ROUTINE.

THE DEFINITION.

A Library routine is quite similar to a Macro routine with the exception that no parameters are passed to it. But because of this difference, it is much faster when called during an assembly.

Every Library routine must begin with a Header card. The format of the Header statement is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	LIBRRØUT														

The operation code is LIBR.
A 4-character name for the routine must be written in columns 18 through 21. Embedded blanks are not permitted.

The remaining fields are not used for the Header statement.

Assembler instructions and pseudos follow the Header card.

The end of a Library routine must be signaled by a LEND card. This is a statement with LEND in columns 14 through 17 and must be the last card in the definition of a Library routine.

CALLING A LIBRARY ROUTINE.

A Library routine is called with the LIBR instruction, and the associated symbolic code is copied to the assembly source input file of the calling program.

The format of the LIBR instruction is:

LABEL	OP. CODE	VAR.		A ADDRESS				B ADDRESS				C ADDRESS			
		AF	BF	Label	± Inc.	Ai	Ac	Label	± Inc.	Bi	Bc	Label	± Inc.	Ci	Cc
0 8	1 4	1 8	2 0	2 2	2 8	3 1	3 2	3 4	4 0	4 3	4 4	4 6	5 2	5 5	5 6 5 8
	LIBRRØUT														
	LIBR			RØUT											

Label entries are not permitted with this instruction.

The desired Library routine's 4-character name is coded in the VAR field or the A ADDRESS field.

The remaining fields are not used by the instruction.

See figure 10-1, page 10-16 for Macro definition and codes.

```
PROGRAM ID - DEMO                                B3500 ASSEMBLER  08/15/68                                PAGE     1
   MEMORY   SEQ NO  LABEL    OP    AFBF  A-LBL  INC AIAC  B-LBL  INC BIBC  C-LBL  INC CICC  REMARKS
                             SPEC        CARD   :   : :        :   : :        :   : :    LIST CODE              1
                             IDNT        DEMO   :   : :

                                                                    *MACRO DEFINITION*

                             MACR  "ADER"                           $A,$B,$C,$D
                             BOOL                                   ERMES1        L01 AND L02 AND L03 AND L04)
                             MNEQ  TYPE  LABEL  :   : :    $C      :   : :    ERMES2:   : :
                             MNEQ  TYPE  LABEL  :   : :    $D      :   : :    ERMES3:   : :
                             MEQL  TYPE  VALUE  :   : :    $A      :   : :    DEFNE  :   : :
                             ADD         $A     :   : :    $B      :   : :    $C     :   : :
                    NOWGO    MNOP
                             MVN       8 $C     :   : :    $D      : 30: :UA
                             MEXT
                    ERMES1   MERR                                   A PARAMETER IS MISSING
                             MEXT
                    ERMES2   MERR                                   FIELD FOR SUM IS NOT DEFINED
                             MEXT
                    ERMES3   MERR                                   PRINT AREA NOT REFERENCED BY LABEL
                             MEXT
                    DEFNE    MNOP
                    AL(IX)   CNST        $A
                             ADD         AL(IX):   : :    $B      :   : :    $C     :   : :
                             MRUN        NOWGO  :   : :
                             MEND

                                                                    *PROGRAM WITH MACRO CALLS AND 2 INTENTIONAL ERR*

      (64)                   PRINT FILE          PRNTR  :   : :    PRN     :   : :           :   : :                    000
     (304)                   PRINTR RECD   132   UA     :   : :            :   : :                                      000
                             ENDF
    (1000)                   TERM1 CNST      8   UN                  585                                                000
                                                                 08 DIG @ 01000, 000=58500000
    (1008)                   TERM2 CNST      4   UN                  244                                                000
                                                                 04 DIG @ 01008, 000=2440
    (1012)                   SUM   CNST      8   UN                                                                     000
                                                                 08 DIG @ 01012, 000=00000000
                             ADER                                  TERM1,TERM2,SUM,PRINTR
                                                                 24 DIG @ 01020, 000=020804001000001008001012
                                                                 18 DIG @ 01044, 000=110808001012200364
                             ADER                                  45,TERM2,SUM,PRINTR
                                                                 02 DIG @ 01062, 000=45
                                                                 24 DIG @ 01064, 000=020204001062001008001012
                                                                 18 DIG @ 01088, 000=110808001012200364
                             ADER                                  ,TERM2,SUM,PRINTR
**** MACRO ERROR  (USER DEFINED) ****   A PARAMETER IS MISSING
                             ADER                                  TERM1,TERM2,SUM,"AREA"
**** MACRO ERROR  (USER DEFINED) ****   PRINT AREA NOT REFERENCED BY LABEL
                             FINI
```

Figure 10-1. Example of Macro Definition with Call and Generated Code

CREATION AND MAINTENANCE OF A MACRO LIBRARY ON DISK.

FEATURES OF THE MAINTENANCE PROGRAM.

MACROS is a program which provides the capability of creating and maintaining a symbolic file of Macro and Library routines on disk. By means of a control card, the symbolic images of routines may be loaded onto a disk file. The file thus created is called MACDIR.

After the creation of MACDIR, MACROS accepts additional control cards allowing:

a. Adding additional Macro or Library routines.
b. Patching existing routines.
c. Removal of any routines.
d. Copying any routine to a selected medium.
e. Dumping MACDIR to a selected medium.

LAYOUT OF MACDIR.

MACDIR, the central file, has the following general layout:

The first segment (200-digits) is the Index Record where information about the file structure is recorded.

Segments 2 through 99 of MACDIR are reserved for a directory which contains an entry for each Macro or Library routine in the file.

Storage of the Macro/Library symbolic images follows the directory beginning in the Starting Segment. One physical segment contains one symbolic image.

INDEX RECORD. The Index Record occupies the first segment of MACDIR. It is divided into four fields with symbolic names: NAS-D, NAS-DF, TTLD, and DIREND.

NAS-D is the first field in the Index Record and contains a key pointing to the next available segment number where a new directory entry may be made.

NAS-DF, the second field, contains a key pointing to the next available segment number where a new routine may be started or an existing routine copied during patching.

TTLD records the number of entries in the directory.

DIREND, the final field, contains a key pointing to the last segment in the directory. Thus the Starting Segment will be DIREND + 1.

DIRECTORY. Each directory segment (200-digits) contains a 2-digit field giving the number of entries in the segment followed by a maximum of twelve entries. The first 6-digits of the segment are not used. The number of entries in an UN field occupying the 7th and 8th digits. The remaining 192-digits are for the entries, each being 16-digits in length.

Format of entry:

Symbolic Name	Digit Positions	Contents
NAME-D	1-8	Routine Name (4 UA)
KEY-D	9-14	Key to location in MACDIR (6 UN)
CLOS-D	15	Active = 0, Removed = 1 (1 UN)
ML-D	16	Macro = 0, Library = 1 (1 UN)

CONTROL INFORMATION.

MACROS is executed by preparing a card deck as follows:

```
?EXECUTE MACROS; DATAB CARDS
{ CONTROL CARDS
?END
```

The control cards must have a dollar sign ($) in column 1.

FILE CONTROL. File control consists of the following:

a. Loading routines.
b. Dumping routines.
c. Crunching routines.

Loading Routines.

Routines may be read from cards, magnetic tape, or disk. The Starting Segment number may be specified. The user may request a resequencing of the symbolic images with a particular increment. These options may be used alone or in conjunction with one another. The "noise words", and, from, and to, may be employed for readability.

There are three options for specifying the desired input medium:

```
$LOAD
$LOAD TAPE
$LOAD CARDS
```

In the first instance the default medium will be disk; this disk file will be labeled MACFIL.

The Starting Segment may be specified in the control card by punching SEG followed by any number greater than 2, up to the file limit. An example of this option is:

```
$LOAD TO SEG 150
```

The default starting segment is 100.

A sequence request may be indicated by SEQ followed by a 6-digit beginning number, a +, and the desired increment expressed as a 6-digit number. For example:

```
$LOAD FROM CARDS AND SEQ 00100 + 00010.
```

Dumping Routines.

Active routines in MACDIR may be written off to punch cards, magnetic tape, or a working disk file. The user may request a listing of the symbolic images as well as resequencing with a desired increment. The above options may be used alone or in conjunction with one another. The "noise words", and, from, and to, may be employed for readability.

There are three options for specifying the desired output medium:

```
$DUMP
$DUMP TO TAPE
$DUMP TO CARDS
```

In the first instance the default medium will be disk.

A listing is requested by punching LIST in the control card. For example:

```
$DUMP TO TAPE AND LIST
```

The following control card will cause the entire directory to be listed:

```
$DUMP LIST
```

A sequence request may be indicated by SEQ followed by a 6-digit beginning number, a +, and the desired increment expressed as a 6-digit number. For example:

```
$DUMP AND SEQ 00210 + 00100.
```

Crunching Routines.

After removing, patching or adding a routine, the old routine takes up valuable disk space. In effect crunching dumps the MACDIR file and reloads it after all patching, adding, removing, and copying functions are completed. The control card format for crunching is:

```
$CRUNCH
```

FILE MAINTENANCE. File maintenance consists of the following:

a. Adding routines.
b. Removing routines.
c. Patching routines.
d. Copying routines.
e. Directory dump.

Adding Routines.

A routine can be added to the existing disk file or MACDIR. It can also be written on tape or punched out on cards. The latter feature would permit a user to obtain his own personalized library file with additional routines to those in MACDIR, without adding to or altering the original disk file. This can be accomplished by first dumping MACDIR to cards or tape, then adding the desired routines to the same medium. This process leaves MACDIR unchanged.

When adding a routine to MACDIR, there are two possibilities, and they are handled in the following manner. If the routine being added already exists, it is written to MACDIR beginning at the segment indicated by NAS-DF, and the existing directory entry is used to locate the newly added version. If the routine is truly an addition, a new entry is made in the directory.

The symbolic routine to be added must be in card form. The control card format for adding a routine to MACDIR is:

```
$ADD MACR MOVE
```

Following the dollar sign is the directive ADD. Then the routine type, either MACR or LIBR is specified, followed by its 4-character name. A listing may be requested by punching LIST following the routine name. A sequence request may be indicated by SEQ followed by a 6-digit beginning number, a +, and the desired increment expressed as a 6-digit number. An example using these options is:

```
$ADD MACR MOVE LIST AND SEQ 010000 + 000001.
```

To add a routine to tape or have it punched on cards, the medium must be specified in the control card as TAPE or CARDS following the routine name. The list and sequence options may be used with this feature. An example follows:

```
$ADD LIBR FSQT TO TAPE AND LIST.
```

Removing Routines.

MACROS removes a routine by making CLOS-D with a one (1); once removed, it is permanently inaccessible. See Crunching Routines (page 10-20) for control information necessary to physically remove inactive routines from MACDIR.

The format of the control card necessary to effect the removal of a routine is:

```
$REMOVE LIBR HELP.
```

The directive REMOVE follows the dollar sign. Then the routine type, either MACR or LIBR, is specified followed by its 4-character name.

Patching Routines.

A routine may be patched, and the revised version may be optionally written in MACDIR or on tape or cards. The latter feature would permit a user to obtain an altered version of a routine in MACDIR without changing the original one in the library file.

Patches must be in card form. When patching to change a routine in MACDIR, the following occurs: The routine is merged with the card patches according to the sequence numbers. The merged version is rewritten into MACDIR beginning at the segment indicated by NAS-DF. KEY-D in the directory entry for the routine is changed to point to the new version. See Crunching Routines (page 10-20) for control information necessary to physically remove the inactive routine from MACDIR.

The control card format for patching a routine in MACDIR is:

```
$PATCH MACR MOVE.
```

Following the dollar sign is the directive PATCH. Then the routine type, either MACR or LIBR, is specified, followed by its 4-character name. A listing may be requested by punching LIST following the routine name. A sequence request may be indicated by SEQ followed

by a 6-digit beginning number, a +, and the desired increment expressed as a 6-digit number. An example using these options is:

$PATCH MACR MOVE LIST AND SEQ 010000 + 000010.

To patch a routine and write a new version on cards or tape, leaving the original unchanged in MACDIR, specify CARDS or TAPE in the control card following the routine name. The list and sequence options may be used with this feature. For example:

$PATCH LIBR HELP TO TAPE AND SEQUENCE 001000 + 000010.

Copying Routines.

Any active routine may be copied to punch cards, magnetic tape or a working disk file.

The format of the control card for copying a routine that is in MACDIR is:

$COPY MACR MOVE TO CARDS.

Following the dollar sign is the directive COPY. Then the routine type, either MACR or LIBR, is specified followed by its 4-character name. The medium is specified next: CARDS, TAPE, or BLANK. The default medium is disk. A listing may be requested by punching LIST in the control card. A sequence request may be indicated by SEQ followed by a 6-digit beginning number, a +, and the desired increment expressed as a 6-digit number. An example using all of these features is:

$COPY LIBR HELP TO CARDS LIST AND SEQ 000100 + 000010.

Directory Dump.

The user may request a listing of the directory for the file MACDIR with the following control card:

$DIRECTORY.

APPENDIX A
BASIC ASSEMBLER OBJECT PROGRAM OUTPUT

The Basic Assembler will produce an 80-column image on output. The format for a Program Parameter Card is as follows:

Columns	Code
1	6
2-9	Zeroes
10-14	Address of first instruction
15-20	Program Identification
21	0 (zero)
22-27	Address of last instruction
28-74	Zeroes
75-80	Program Identification

The format for an Instruction Format Card (Single Instruction Per Card) is as follows:

Columns	Code
1	1
2-3	Size of instruction (in characters)
4-9	Zeroes
10-14	Base relative location
15-66	Instruction (remaining portion filled with zeroes)
67-69	Segment number (first segment equals zeroes)
70-74	Card number
75-80	Program Identification

NOTE

The last card punched out will be identical to the previous card except the segment number is 999. This signals the end of the load operation to the BCP.

APPENDIX B
NOTES ON ADVANCED ASSEMBLER OPERATIONS

In a heavy multiprogrammed assembly/generation environment, special names for source program files will be a virtual necessity to avoid confusion and misidentification. The internal file names and external file ID's for Advanced Assembler files are:

Internal File-Name	External File ID	Function
CARDS	CARDS	Source of Update card input.
SYMTIN	SYMTIN	Source program magnetic tape input.
SYMTOT	SYMTIN	Updated Source magnetic tape output.
SYMTOT	SYMTIN	Updated Source punched card output.
SYMDIN	SYMDIN	Source program disk input.
SYMDOT	SYMDIN	Updated Source disk output.
PRINT	PRINT	Assembly printed output listing.

For example, to change the external ID of the updated source disk output to UPOUT, the following label equation card should be entered:

```
?  FILE SYMDOT = UPOUT.
```

The Advanced Assembler is usually called from the card reader by the following (minimal) set of control and data cards:

```
? COMPILE XXX WITH ASMBLR LIBRARY
? DATA CARDS
000100      SPEC
(source program, or updates if source input is from
 tape or disk)
? END
```

If updated source output to tape or disk is required, the SPEC card is not written to the updated file. Subsequent re-assemblies against the tape/disk file can then be run with different SPECifications.

Program generators (FFT, RPG, SRTGEN) usually write complete Assembly language source programs (including a SPEC card) to tape or disk, and then invoke the Assembler by a ZIPP operation. To inform the Assembler that there is no card input to the assembly, the following VALUE is declared in the ZIPP control field:

VALUE 0 = 2 (source input from tape, no card input), or
VALUE 0 = 3 (source input from disk, no card input)

User-written program generators may use this same convention when delivering generated symbolic programs to the Assembler for translation to machine language.

In either a ?COMPILE or ZIPPed assembly, the user may tell the Assembler to purge the source input file by the following VALUE statement:

? VALUE 0 = 100000.

This option may be used in conjunction with the "no card input" VALUE option: ? VALUE 0 = 100003 indicates that the entire source program is on disk with no updates from cards and that the input source program file is to be purged upon completion of the assembly.

To provide a combined example of all the control options explained above, assume that some program generator has just finished writing an assembly-language source program to the disk and will ZIPP the Assembler to produce a machine-language program and:

a. The name of the object program is to be EDTRUN.

b. The name of the source program file on disk is GENOUT.

c. The source program disk file is to be purged after assembly.

The control field of the generator's ZIPP to the Assembler should then read: COMPILE EDTRUN WITH ASMBLR LIBRARY FILE SYMDIN = GENOUT VALUE 0 = 100003.

Advanced Assembler can operate in 14000 digits of core, but operation is slow (overlaying, symbol tables on disk, etc.). If CORE = 30000 is declared in the COMPILE card, speed roughly triples. Speed continues to increase as more core is allocated, up to some saturation point. The larger the program, the higher the saturation point: e.g., for a 1000-card program, saturation point is near CORE = 50000.

The following are samples of input card decks for assembly:

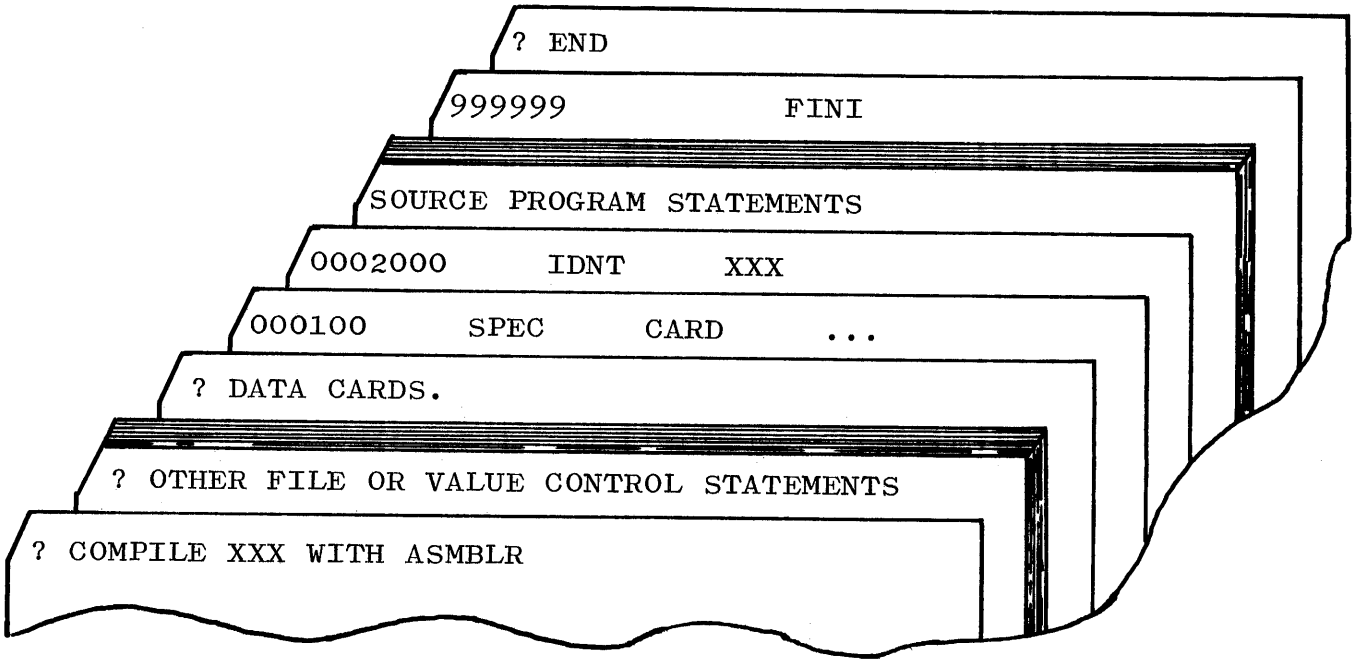


Figure C-1. Sample Advanced Assembly Card Deck

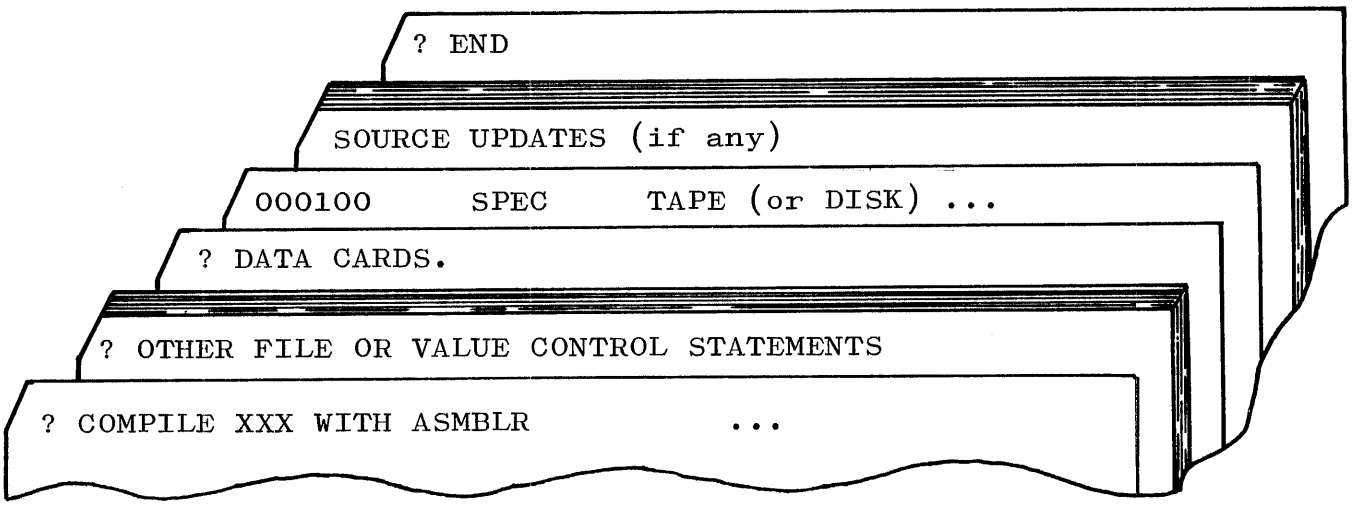


Figure C-2. Sample Advanced Update Assembly Card Deck

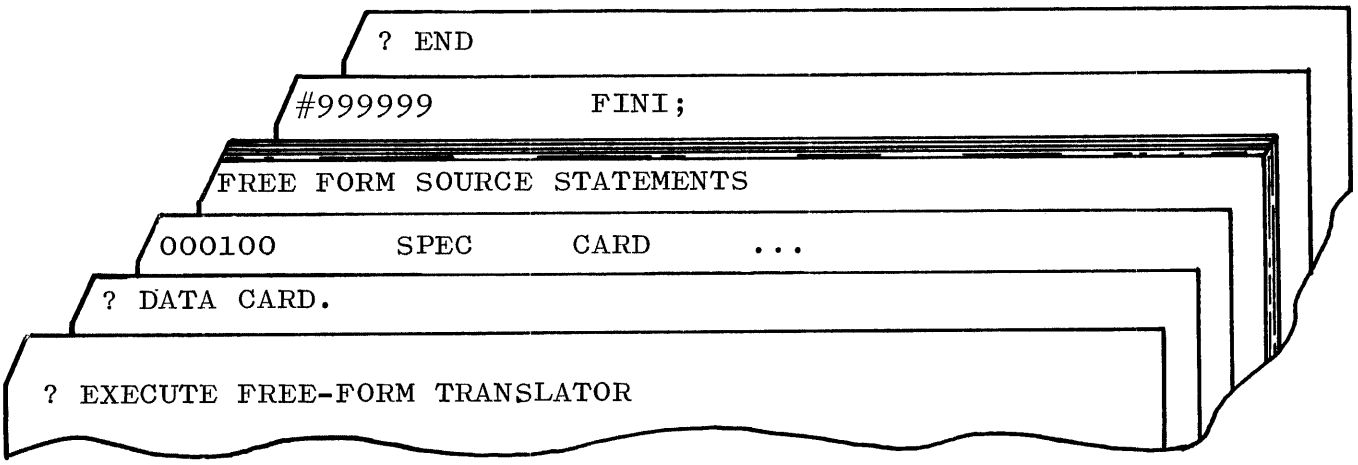


Figure C-3. Sample Advanced Free-Form Translator Card Deck

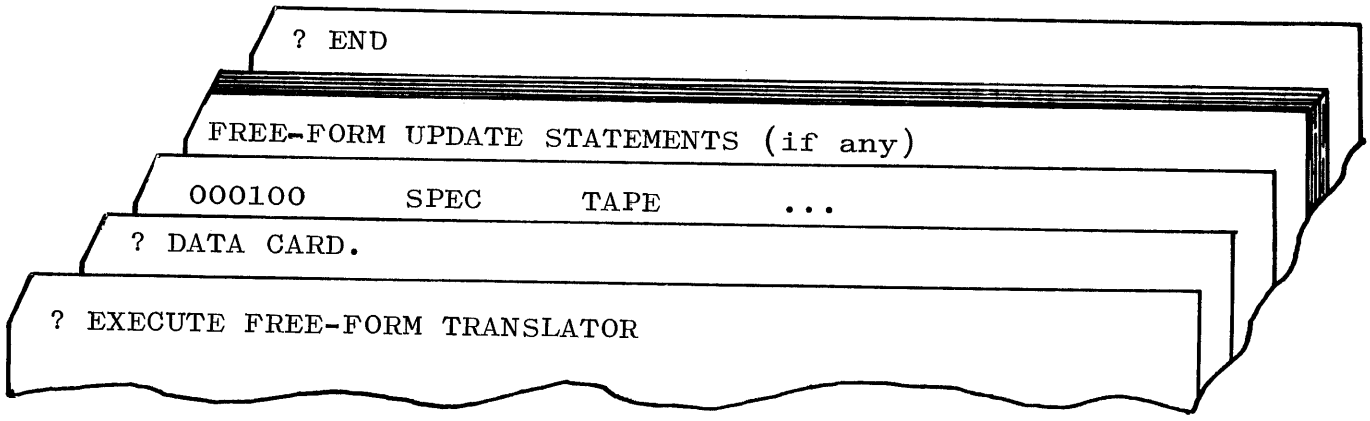


Figure C-4. Sample Advanced Update Free-Form Translator Card Deck

APPENDIX C
BCP INPUT/OUTPUT START AND RESULT DESCRIPTORS

The following chart provides the I/O Start and Result Descriptors for the Basic Control Program. An I/O Start Descriptor is an instruction to an input/output channel to perform some operation. It is formatted like a processor instruction with an operation code, variants, and one to three data addresses where needed. In the following descriptions, the letter r indicates a variant position which currently has no meaning and should be zero.

An I/O Result Descriptor is a 16-bit (4-digit) status message which a channel returns to the processor upon completion of an I/O operation. The bits within this message are numbered from 1 through 16 (from the left). Thus, RD bit 4 would be the 1 bit of the first digit, RD bit 13 would be the 8 bit of the fourth digit, etc. In all I/O Result Descriptors, bit 1 means I/O operation completed, while bit 2 means some exception condition exists.

For more information on the I/O functions, refer to the B 2500/ B 3500 Systems Reference Manual.

I/O START DESCRIPTOR						I/O RESULT DESCRIPTOR	
OP	VAR	A ADDRESS	B ADDRESS	C ADDRESS	FUNCTION	BIT	ON MEANING
					CARD PUNCH		
23	Srrr	BEGIN			Punch card BCL	3	Unit not ready
24	Srrr	BEGIN			Punch binary card	4	Punch check or memory access error
25	Srrr	BEGIN			Punch EBCDIC card	4&6	Memory parity error
35	rrrr				Input request enable	5	Not used
97	rrrr				Input request disable	7	Punch identification (special feature)
99	rrrr				Test	8-12	Not used
						13-16	Zeroes
S = stacker select (0 = normal stacker, 1 = auxiliary stacker)							
					CARD READER		
20	rrrr	BEGIN	END		Read BCL card	3	Unit not ready
21	rrrr	BEGIN	END		Read binary card	4	Memory access error
22	rrrr	BEGIN	END		Read EBCDIC card	4&6	Validity error
35	rrrr				Input request enable	5	Read check
97	rrrr				Input request disable	7-12	Not used
99	rrrr				Test	13-16	Zeroes

I/O START DESCRIPTOR						I/O RESULT DESCRIPTOR	
OP	VAR	A ADDRESS	B ADDRESS	C ADDRESS	FUNCTION	BIT	ON MEANING
					CONSOLE PRINTER		
32	rrrr	BEGIN	END		Read message	3	Unit not ready
34	rrrr	BEGIN	END		Write message	4	Memory parity error
35	rrrr				Input request enable	5-6	Not used
97	rrrr				Input request disable	7	NAK character received--error (CTRL/U on keyboard)
99	rrrr				Test	8	Attempt to exceed end address (read)
						9-12	Not used
						13-16	Zeroes
					DISK FILE		
50	UQrV	BEGIN	END	SEG#	Disk write	3	Unit not ready
51	UQrW	BEGIN	END	SEG#	Disk read	4	Memory access error
52	UQrW	BEGIN	END	SEG#	Disk check read (No data transfer)	4&6	Memory parity error (write)
						4&6	Read error (read/check)
99	UQrr				Test	5	Busy
						6	Write lockout set (write)
						7-9	Not used
						10	Memory full (read)
						11	Previously set

I/O START DESCRIPTOR						I/O RESULT DESCRIPTOR	
OP	VAR	A ADDRESS	B ADDRESS	C ADDRESS	FUNCTION	BIT	ON MEANING
					DISK FILE (cont)		
						12	Time-out (no data received during one rev)
						13-16	Electronics unit number
			U = electronics unit Q = high-order digit of disk segment number (if applicable) or electronics unit V = write protection clear (0 = remove protect, 1 = remove protect and inhibit data transfer) W = read protection test (0 = read without protect test, 1 = read and set protect, 2 = read unless protected, and 3 = read unless protected--set protect after read)			
					LINE PRINTER		
10	SUNN	BEGIN			Write line	3	Unit not ready
11	SUNN				Skip	4&5	Bit transfer error
35	rUrr				Input request enable	4&6	Memory parity error
97	rrrr				Input request disable	4&7	Print check or code parity error
99	rUrr				Test	8	Not used
						9	End-of-page (hole sensed in carr. tape channel 12)
						10-12	Not used
						13-16	Unit number (0 or 1)

I/O START DESCRIPTOR						I/O RESULT DESCRIPTOR	
OP	VAR	A ADDRESS	B ADDRESS	C ADDRESS	FUNCTION	BIT	ON MEANING
					LINE PRINTER (cont)		
			S = spacing (0 = no space, 1 = single space, 2 = double space) U = unit number for buffered printers (0 = first unit, 1 = second unit) NN = skipping (00 = no skip, 01-11 = skip to channel 01-11)				
					MAGNETIC TAPE		
01	rUrr				REWIND	3	Unit not ready
02	FUVT	BEGIN	END		Read forward	4	Memory access error (read/write/erase)
03	FUVT	BEGIN	END		Read backward	4&6	Memory parity error (write/erase)
04	FUrr	BEGIN	END		Erase forward	4&7&8	Tape parity error (read/write/space)
06	FUVr	BEGIN	END		Write	5	End-of-Tape/Beginning-of-Tape
08	FUNN				Space forward	6	Write lockout--no ring (write/test)
09	FUNN				Space backward	6	End-of-File (read/space)
99	rUrr				Test	7	Short block (read)
						7&8	Transport density setting (test)
						8	Long block (read)

I/O START DESCRIPTOR						I/O RESULT DESCRIPTOR	
OP	VAR	A ADDRESS	B ADDRESS	C ADDRESS	FUNCTION	BIT	ON MEANING
					MAGNETIC TAPE (cont)		
						9	CRC correction possible (read)
						10-12	Track in error (when bit 9 is on)
						10	Non-available option requested
						11	Unit is rewinding
						12	Instruction time-out--blank tape
						13-16	Unit number (0 through 9)
					F = density/parity (0 = 800 BPI even parity--7-track; 1 = 800 BPI odd parity; 2 = 555 BPI even parity--7-track; 3 = 555 BPI odd parity--7-track; 4 = 200 BPI even parity--7-track; 5 = 200 BPI odd parity; 7 = 1600 BPI odd parity--9-track; 8 = even parity, density set on transport; 9 = odd parity, density set on transport)		
					U = unit number (0 through 9)		
					V = options (2 = write tape mark, 4 = EBCDIC/BCL translation on 7-track transports)		
					T = select CRC correction on 9-track read (8 through undigit F selects track to be corrected)		
					NN = number of records to space over		

I/O START DESCRIPTOR						I/O RESULT DESCRIPTOR	
OP	VAR	A ADDRESS	B ADDRESS	C ADDRESS	FUNCTION	BIT	ON MEANING
					MULTIPLE TAPE LISTER		
70	UTDR	BEGIN	END		Lister print	3	Unit not ready
71	UTDR				Lister space	4&6	Memory parity error
72	UTDR				Lister skip	4&7	Print check
73	UVDR				Lister slew	5	End-of-paper
35	rrrr				Input request enable	8-12	Not used
97	rrrr				Input request disable	13-16	Zeroes
99	rrrr				Test		
			U, T, D, R, V = select units and tapes. The settings of these variants are quite complex. The user should refer to the Systems Reference Manual.				
					PAPER TAPE READER/PUNCH		
40	rrVr	BEGIN	END		Paper tape read	3	Unit not ready
41	rrrr	BEGIN	END		Forward space	4	Memory access error (read/space)
43	rrrr	BEGIN	END		Backspace	4	Memory parity error (write)
35	rrrr				Input request enable	4&6	Tape parity error (read)
47	rrrr				Rewind	5	End-of-Tape/Beginning-of-Tape (read)
48	rrVr	BEGIN	END		Paper tape write	5	Low paper (write)

I/O START DESCRIPTOR						I/O RESULT DESCRIPTOR	
OP	VAR	A ADDRESS	B ADDRESS	C ADDRESS	FUNCTION	BIT	ON MEANING
				PAPER TAPE READER/PUNCH (cont)			
97	rrrr				Input request disable	7	Incomplete (read/space/ write)
99	rrrr				Test	8-12	Block (not used)
						13-16	Zeroes
				V = code level (0 = seven-level odd parity, 1 = six-level odd parity with BCL-EBCDIC translation, 2 = eight-level no parity)			
					SORTER READER		
62	rVrC	AREA1	AREA2		Flow-mode read	3	Unit not ready
63	rVrC	AREA1	AREA2		Demand-mode read	3&5	Jam, or any sorter-not-ready stoppage
60	NNrF				Pocket select	4	Memory access error
66	rrrr				Batch counter increment	4&5&6	Too late to read
64	NNrr				Pocket light	4&7	Cannot read or uncoded document
35	rrrr				Input request enable	4&7& 10&11	Unencoded document (if start descriptor specified distinction)
97	rrrr				Input request disable		
99	rrrr				Test	4&8	Double document

I/O START DESCRIPTOR						I/O RESULT DESCRIPTOR	
OP	VAR	A ADDRESS	B ADDRESS	C ADDRESS	FUNCTION	BIT	ON MEANING
				SORTER/READER (cont)			
						4&10	Amount field error (if start descriptor specified distinction)
						4&11	Transit field error (if start descriptor specified distinction)
						5	Flow stopped
						5&7	Batch ticket
						6	Too late to pocket select
						7	EOF button depressed
						9	Not used
						12	Buffer number (note that the exception bit is not set in this result descriptor)
						13-16	Zeroes
				V = read station select code C = format and error reporting code NN = pocket number F = continue/stop flow Variant usage is complex for sorter/reader descriptors. The user should refer to the Systems Reference Manual.			

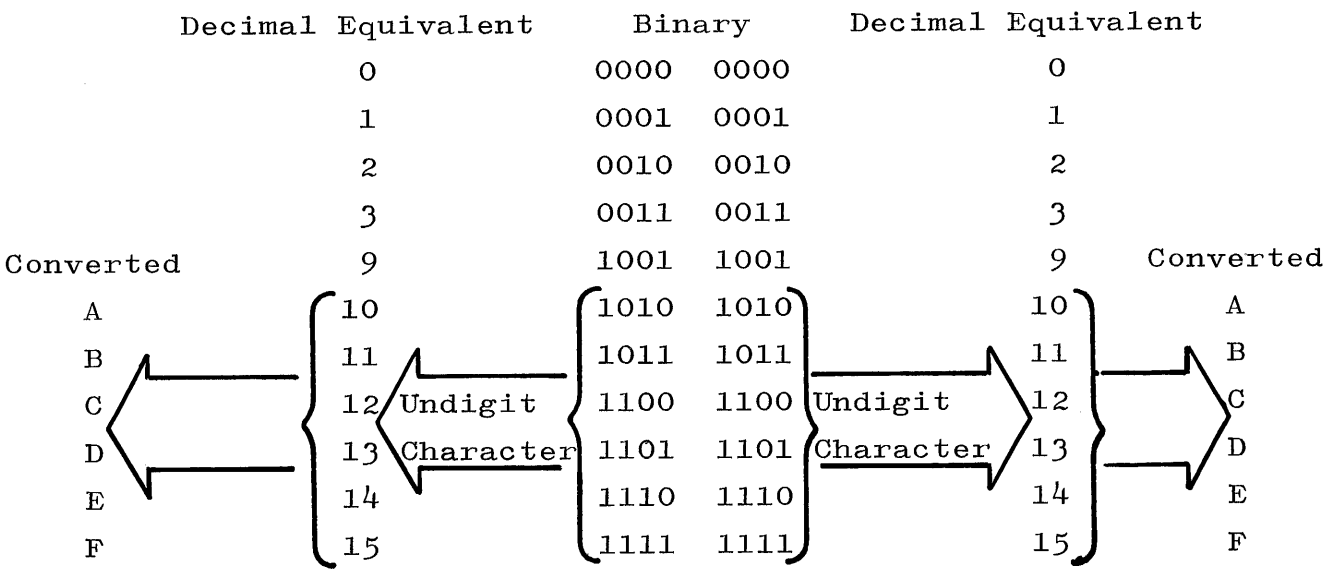
APPENDIX D
EBCDIC , USASCII , AND BCL REFERENCE TABLE

GENERAL.

This table reflects the internal EBCDIC structure in its sequential code arrangement for B 2500/B 3500 Systems, plus the USASCII and BCL magnetic tape coding structures.

The two methods of creating the two-character codes representing these structures are broken down as follows:

a. 8-bit (byte) character code:



Example:

If a memory dump reflects A3, the internal code would be 1010 0011. The highest sequential code is FF and the internal code is 1111 1111.

b. 6-bit (byte) character code:

Decimal	Binary	Decimal	Converted
0	00 0000	0	
1	01 0100	4	
2	10 1001	9	
3	11 1010	10	A
0	00 1100	12	C
2	10 1111	15	F

c. The graphics appearing in the BCL column reflect differences between EBCDIC and BCL and those pertinent ones left blank reflect that the graphics of each are the same. This method holds true for the card code column of the BCL description also.

NOTE

There are only 64 unique 6-bit BCL tape codes. Therefore, where no BCL tape code is indicated in the table, there will be no BCL graphic character.

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
00	NULL	12-0-9-8-1	80			
01	SOH	12-9-1	81			
02	STX	12-9-2	82			
03	ETX	12-9-3	83			
04		12-9-4	84			
05	HT	12-9-5	85			
06		12-9-6	86			
07	DEL	12-9-7	87			
08		12-9-8	88			
09		12-9-8-1	89			
0A		12-9-8-2	8A			
0B	VT	12-9-8-3	8B			
0C	FF	12-9-8-4	8C			
0D	CR	12-9-8-5	8D			
0E	SO	12-9-8-6	8E			
0F	SI	12-9-8-7	8F			

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
10	DLE	12-11-9-8-1	90			
11	DC1	11-9-1	91			
12	DC2	11-9-2	92			
13	DC3	11-9-3	93			
14		11-9-4	94			
15	NL	11-9-5	95			
16	BS	11-9-6	96			
17		11-9-7	97			
18	CAN	11-9-8	98			
19	EM	11-9-8-1	99			
1A		11-9-8-2	9A			
1B		11-9-8-3	9B			
1C	FS	11-9-8-4	9C			
1D	GS	11-9-8-5	9D			
1E	RS	11-9-8-6	9E			
1F	US	11-9-8-7	9F			
20		11-0-9-8-1				
21		0-9-1				
22		0-9-2				
23		0-9-3				
24		0-9-4				
25	LF	0-9-5				
26	ETB	0-9-6				
27	ESC	0-9-7				
28		0-9-8				
29		0-9-8-1				
2A		0-9-8-2				
2B		0-9-8-3				
2C		0-9-8-4				
2D	ENQ	0-9-8-5				
2E	ACK	0-9-8-6				
2F	BEL	0-9-8-7				
30		12-11-0-9-8-1				
31		9-1				
32	SYN	9-2				
33		9-3				
34		9-4				
35		9-5				
36		9-6				
37	EOT	9-7				
38		9-8				
39		9-8-1				
3A		9-8-2				
3B		9-8-3				
3C	DC4	9-8-4				

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
3D	NAK	9-8-5				
3E		9-8-6				
3F	SUB	9-8-7				
40	SPACE		A0	10		
41		12-0-9-1				
42		12-0-9-2				
43		12-0-9-3				
44		12-0-9-4				
45		12-0-9-5				
46		12-0-9-6				
47		12-0-9-7				
48		12-0-9-8				
49		12-8-1				
4A	[	12-8-2	DB	3C		12-8-4
4B	.	12-8-3	AE	3B		
4C	<	12-8-4	BC	3E		12-8-6
4D	(	12-8-5	A8	3D		
4E	+	12-8-6	AB	3A		12-0
4F	|	12-8-7	DE	3F	←	
50	&	12	A6	30		
51		12-11-9-1				
52		12-11-9-2				
53		12-11-9-3				
54		12-11-9-4				
55		12-11-9-5				
56		12-11-9-6				
57		12-11-9-7				
58		12-11-9-8				
59		11-8-1				
5A	]	11-8-2	DD	1E		0-8-6
5B	$	11-8-3	A4	2B		
5C	*	11-8-4	AA	2C		
5D	)	11-8-5	A9	2D		
5E	;	11-8-6	BB	2E		
5F	¬	11-8-7	DC	2F	≤	
60	-	11	AD	20		
61	/	0-1		11		
62		11-0-9-2				
63		11-0-9-3				
64		11-0-9-4				
65		11-0-9-5				
66		11-0-9-6				
67		11-0-9-7				
68		11-0-9-8				
69		0-8-1				

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
6A		12-11				
6B	,	0-8-3	AC	1B		
6C	%	0-8-4	A5	1C		
6D	Underscore	0-8-5	DF	1A	≠	0-8-2
6E	>	0-8-6	BE	0E		8-6
6F	?	0-8-7	BF	00		
70		12-11-0				
71		12-11-0-9-1				
72		12-11-0-9-2				
73		12-11-0-9-3				
74		12-11-0-9-4				
75		12-11-0-9-5				
76		12-11-0-9-6				
77		12-11-0-9-7				
78		12-11-0-9-8				
79		8-1				
7A	:	8-2	BA	0N		8-5
7B	#	8-3	A3	0B		
7C	@	8-4	C0	0C		
7D	'	8-5	A7	0F	≥	8-7
7E	=	8-6	BD	1D		0-8-5
7F	"	8-7	A2	1F		0-8-7
80		12-0-8-1				
81	a	12-0-1				
82	b	12-0-2				
83	c	12-0-3				
84	d	12-0-4				
85	e	12-0-5				
86	f	12-0-6				
87	g	12-0-7				
88	h	12-0-8				
89	i	12-0-9				
8A		12-0-8-2				
8B		12-0-8-3				
8C		12-0-8-4				
8D		12-0-8-5				
8E		12-0-8-6				
8F		12-0-8-7				
90		12-11-8-1				
91	j	12-11-1				
92	k	12-11-2				
93	l	12-11-3				
94	m	12-11-4				
95	n	12-11-5				
96	o	12-11-6				

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
97	p	12-11-7				
98	q	12-11-8				
99	r	12-11-9				
9A		12-11-8-2				
9B		12-11-8-3				
9C		12-11-8-4				
9D		12-11-8-5				
9E		12-11-8-6				
9F		12-11-8-7				
A0		11-0-8-1				
A1		11-0-1				
A2	s	11-0-2				
A3	t	11-0-3				
A4	u	11-0-4				
A5	v	11-0-5				
A6	w	11-0-6				
A7	x	11-0-7				
A8	y	11-0-8				
A9	z	11-0-9				
AA		11-0-8-2				
AB		11-0-8-3				
AC		11-0-8-4				
AD		11-0-8-5				
AE		11-0-8-6				
AF		11-0-8-7				
B0		12-11-0-8-1				
B1		12-11-0-1				
B2		12-11-0-2				
B3		12-11-0-3				
B4		12-11-0-4				
B5		12-11-0-5				
B6		12-11-0-6				
B7		12-11-0-7				
B8		12-11-0-8				
B9		12-11-0-9				
BA		12-11-0-8-2				
BB		12-11-0-8-3				
BC		12-11-0-8-4				
BD		12-11-0-8-5				
BE		12-11-0-8-6				
BF		12-11-0-8-7				
C0	(+)PZ	12-0				
C1	A	12-1	C1	31		
C2	B	12-2	C2	32		
C3	C	12-3	C3	33		

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
C4	D	12-4	C4	34		
C5	E	12-5	C5	35		
C6	F	12-6	C6	36		
C7	G	12-7	C7	37		
C8	H	12-8	C8	38		
C9	I	12-9	C9	39		
CA		12-0-9-8-2				
CB		12-0-9-8-3				
CC		12-0-9-8-4				
CD		12-0-9-8-5				
CE		12-0-9-8-6				
CF		12-0-9-8-7				
D0	(!)MZ	11-0	A1	2A	x	11-0
D1	J	11-1	CA	21		
D2	K	11-2	CB	22		
D3	L	11-3	CC	23		
D4	M	11-4	CD	24		
D5	N	11-5	CE	25		
D6	O	11-6	CF	26		
D7	P	11-7	D0	27		
D8	Q	11-8	D1	28		
D9	R	11-9	D2	29		
DA		12-11-9-8-2				
DB		12-11-9-8-3				
DC		12-11-9-8-4				
DD		12-11-9-8-5				
DE		12-11-9-8-6				
DF		12-11-9-8-7				
E0		0-8-2				
E1		11-0-9-1				
E2	S	0-2	D3	12		
E3	T	0-3	D4	13		
E4	U	0-4	D5	14		
E5	V	0-5	D6	15		
E6	W	0-6	D7	16		
E7	X	0-7	D8	17		
E8	Y	0-8	D9	18		
E9	Z	0-9	DA	19		
EA		11-0-9-8-2				
EB		11-0-9-8-3				
EC		11-0-9-8-4				
ED		11-0-9-8-5				
EE		11-0-9-8-6				
EF		11-0-9-8-7				

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
F0	0	0	B0	0A		
F1	1	1	B1	01		
F2	2	2	B2	02		
F3	3	3	B3	03		
F4	4	4	B4	04		
F5	5	5	B5	05		
F6	6	6	B6	06		
F7	7	7	B7	07		
F8	8	8	B8	08		
F9	9	9	B9	09		
FA		12-11-0-9-8-2				
FB		12-11-0-9-8-3				
FC		12-11-0-9-8-4				
FD		12-11-0-9-8-5				
FE		12-11-0-9-8-6				
FF		12-11-0-9-8-7				

APPENDIX E
BASIC ASSEMBLER AND ADVANCED ASSEMBLER ERROR MESSAGES

The following are the error messages for the Basic Assembler as they will appear on the printed listing:

XXXXXXXXXX ILLEGAL LABEL. (special characters or blanks)
XXXXXXXXXX DUPLICATE LABEL.
XXXXXXXXXX ILLEGAL OP-CODE.
XXXXXXXXXX A-LABEL.
XXXXXXXXXX A-CNT. (A address controller)
XXXXXXXXXX A-INDEX.
XXXXXXXXXX A-INCR.
XXXXXXXXXX LITSIZ. (literal size too large)
XXXXXXXXXX LIT-ERR.
XXXXXXXXXX B-LABEL.
XXXXXXXXXX B-CNT.
XXXXXXXXXX B-INDEX.
XXXXXXXXXX B-INCR.
XXXXXXXXXX C-LABEL
XXXXXXXXXX C-CNT.
XXXXXXXXXX C-INDEX.
XXXXXXXXXX C-INCR.
XXXXXXXXXX ILLEG-RECD-SZ. (illegal record size)
XXXXXXXXXX ILLEGAL AF.
XXXXXXXXXX ILLEGAL BF.
XXXXXXXXXX SEGM NESTED > 10.
XXXXXXXXXX SEQ ERR.
XXXXXXXXXX LOGICAL RECD SIZE.

The following are the error messages for the Advanced Assembler, as they will appear on the printed listing:

XXXXXXXXXX DUPLICATE LABEL CL 8.
XXXXXXXXXX INVALID HARDWARE TYPE.
XXXXXXXXXX THIS FILE SHOULD NOT BE DECLARED RANDOM.

XXXXXXXXXX REC/AREA FIELD SHOULD BE BLANK.
XXXXXXXXXX REC/AREA FIELD INVALID.
XXXXXXXXXX MISSING REC/AREA FIELD.
XXXXXXXXXX NO OF AREAS SHOULD BE BLANK
XXXXXXXXXX INVALID NO OF AREAS.
XXXXXXXXXX NO OF AREAS SHOULD NOT BE BLANK.
XXXXXXXXXX LABEL INFORMATION FIELD INVALID.
XXXXXXXXXX THIS TYPE FILE CANNOT INDICATE EBCDIC.
XXXXXXXXXX INVALID RECORDING MODE FIELD.
XXXXXXXXXX INVALID ALTERNATE AREAS FIELD.
XXXXXXXXXX TOO MANY ALTERNATE AREAS REQUESTED.
XXXXXXXXXX INVALID RETENTION FACTOR.
XXXXXXXXXX INVALID REC/BLOCK.
XXXXXXXXXX COL 58 LAB UNDEF.
XXXXXXXXXX ILLEG BUFFER ACCESS.
XXXXXXXXXX INVALID BLOCKING TECHNIQUE.
XXXXXXXXXX THIS TYPE FILE MAY NOT HAVE VAR LENGTH RECORDS.
XXXXXXXXXX INVALID MAX BLOCK SIZE.
XXXXXXXXXX SEQUENCE ERROR.
XXXXXXXXXX A-LABEL UNDEFINED.
XXXXXXXXXX A-LABEL MUST BE A FILE.
XXXXXXXXXX SEEK OP NOT APPLICABLE TO THIS FILE.
XXXXXXXXXX ILLEGAL OVERLAY REQUEST.
XXXXXXXXXX CNST MAY NOT APPEAR AS PART OF A RECD.
XXXXXXXXXX INVALID INCREMENT.
XXXXXXXXXX ILLEGAL LIBR CALL.
XXXXXXXXXX REQUESTED LIBR MACRO NOT ON LIBRARY TAPE.
XXXXXXXXXX C-LABEL UNDEFINED.
XXXXXXXXXX FILE KEY UNDEFINED.
XXXXXXXXXX B-LABEL UNDEFINED.
XXXXXXXXXX A-LABEL IS AN ILLEGAL REF OR IS UNDEFINED.
XXXXXXXXXX ILLEGAL A-CONTROLLER.
XXXXXXXXXX ILLEGAL A-INDEX.
XXXXXXXXXX DECLARED SIZE MUST BE MOD-4.

XXXXXXXXXX MISSING AF SIZE.
XXXXXXXXXX B-LABEL IS AN ILLEGAL REF OR IS UNDEFINED.
XXXXXXXXXX ILLEGAL B-CONTROLLER.
XXXXXXXXXX ILLEGAL B-INDEX.
XXXXXXXXXX MISSING BF SIZE.
XXXXXXXXXX LITERAL SIZE TOO LARGE.
XXXXXXXXXX MISSING SIGN IN COL 22.
XXXXXXXXXX C-LABEL IS AN ILLEGAL REFERENCE OR IS UNDEFINED.
XXXXXXXXXX ILLEGAL C-CONTROLLER.
XXXXXXXXXX ILLEGAL C-INDEX.
XXXXXXXXXX ILLEGAL SYNC STATEMENT.
XXXXXXXXXX ILLEGAL ALOC STATEMENT.
XXXXXXXXXX LOC MUST REMAIN WITHIN CURRENT SEGMENT.
XXXXXXXXXX FORWARD REF LOCN ILLEGAL.
XXXXXXXXXX LOC MUST REMAIN WITHIN CURRENT SEGMENT.
XXXXXXXXXX MISSING ENDF.
XXXXXXXXXX ILLEGAL RLOC.
XXXXXXXXXX LOC MUST REMAIN WITHIN CURRENT SEGMENT.
XXXXXXXXXX DECLARATION ERROR.
XXXXXXXXXX UNDEFINED OP-CODE.
XXXXXXXXXX ILLEGAL LABEL COL 8.
XXXXXXXXXX MISSING LABEL COL 8.
XXXXXXXXXX ILLEGAL RECD SIZE.
XXXXXXXXXX RECD SIZE OMITTED.
XXXXXXXXXX RECORD SIZE MUST BE MOD-4.
XXXXXXXXXX INVALID PH # LEGTH COL 65.
XXXXXXXXXX NEED COL 8 LABEL.
XXXXXXXXXX UNDEFINED OPEN.
XXXXXXXXXX AF-BF MUST BE NUMERIC.
XXXXXXXXXX DUPLICATE LABEL.
XXXXXXXXXX LOC OF A-LABEL MUST BE MOD-4.
XXXXXXXXXX SIZE OF A-LABEL MUST BE MOD-4.
XXXXXXXXXX UNLABELED DISC FILE NOT ALLOWED.
XXXXXXXXXX ILLEGAL AF-BF.
XXXXXXXXXX NEED LABEL COL 58.

XXXXXXXXXX NEED LABEL COL 8.
XXXXXXXXXX INVALID SEGM LABEL.
XXXXXXXXXX DUPLICATE LABEL.
XXXXXXXXXX MISSING SEGM DECLARATION.
XXXXXXXXXX A-ADDR HIGHER THAN 38.
XXXXXXXXXX A-ADDRESS MUST BE MOD-2.
XXXXXXXXXX B-ADDRESS HIGHER THAN 38.
XXXXXXXXXX B-ADDRESS MUST BE MOD-2.
XXXXXXXXXX PROGRAM HAS SEGMENT TOO LARGE FOR 1 DISK AREA.
XXXXXXXXXX INVALID PICTURE.
XXXXXXXXXX MISSING LITERAL SIZE.
XXXXXXXXXX LOCN ILLEGAL WITHIN A RECD.
XXXXXXXXXX COL 20 DOES NOT COMPUTE.
XXXXXXXXXX ILLEGAL MCP IN COL 11 OF FINAL SPEC.

APPENDIX F
ADVANCED ASSEMBLER PROGRAM LISTING

```
PROGRAM  ID - LSTCRD                                B3500 ASSEMBLER  10/01/67                                    PAGE     1
   MEMORY    SEQ NO   LABEL   OP     AFBF  A-LBL   INC AIAC  B-LBL   INC BIBC  C-LBL   INC CICC  REMARKS
                              SPEC R       DISC   :CRF: :          :   : :   001000:000:1:00  LIST CODE                1
             001000           IDNT         LSTCRD:   : :
      (64)   001100   CRDFIL  FILE         CARD   :   : :    CRD   :   : :    SS2   :   : :    CARDS IN                      000
     (384)   001200   CRDREC  RECD    80    UA     :   : :          :   : :                     RECORD AREA                   000
    (1208)   001300   PRTFIL  FILE         PRINT  :   : :    PRN   :   : :    SS2   :   : :    PRINTOUT                      000
    (1528)   001400   PRTREC  RECD  0132    UA     :   : :          :   : :                     PRINT RECORD                  000
             001500           ENDF
    (2768)   001600   COUNT   CNST    04    UA               0000                                                            000
                                                 08 DIG @ 02768, 000=F0F0F0F0
             001700           REMK                    OPEN FILES, CLEAR PRINT AREA, GET READY
    (2776)   001800           OPEN IN      CRDFIL:   : :                                                                   000
                                                 22 DIG @ 02776, 000=3001342700279800006400
    (2798)   001900           OPEN OT      PRTFIL:   : :                                                                   000
                                                 22 DIG @ 02798, 000=3001342700282000120810
    (2820)   002000           SPRD 4040    PRTREC:   : :                                                                   000
                                                 36 DIG @ 02820, 000=11A4044040280015281200650015280015 32
             002100           REMK                    MAIN READ-PRINT LOOP FOLLOWS
    (2856)   002200   READ    READ         CRDFIL:   : :    ENDFIL:   : :                                                  000
                                                 26 DIG @ 02856, 000=3001142700288200006400 2974
    (2882)   002300           MVW     40   CRDREC:   : :    PRTREC:   : :                     MOVE IMAGE TO PRT            000
                                                 18 DIG @ 02882, 000=120040200384201528
    (2900)   002400           INC  01      1      :   : :NL  COUNT :   : :                     COUNT CARDS                  000
                                                 18 DIG @ 02900, 000=01A104100000202768
    (2918)   002500           MVA     04   COUNT  :   : :    PRTREC:100: :                                                  000
                                                 18 DIG @ 02918, 000=100404202768201728
    (2936)   002600           WRIT 01      PRTREC:   : :    *     :   : :                     PRINT RECORD                 000
                                                 30 DIG @ 02936, 000=300234270029660012080029360100
    (2966)   002700           BUN          READ   :   : :                                                                  000
                                                 08 DIG @ 02966, 000=27002856
    (2974)   002800   ENDFIL  CLOS         CRDFIL:   : :                                                                   000
                                                 21 DIG @ 02974, 000=30015427002996000 0640
    (2996)   002900           CLOS         PRTFIL:   : :                                                                   000
                                                 21 DIG @ 02996, 000=30015427003018001 2080
    (3018)   003000           STOP                                                                                          000
                                                 06 DIG @ 03018, 000=300194
             003100           FINI
```

APPENDIX G
ADVANCED ASSEMBLER SYMBOLIC TABLE LISTING

```
BASE              @     0     [       ]  UNREF
CRDFIL (FILE)     @    64     [001100]  001100  001800  002200  002800
COUNT      4 UA   @  2768     [001600]  002400  002500
CRDREC    80 UA   @   384     [001200]  002300
ENDFIL (PROG)     @  2974     [002800]  002200
IX1        7 SN   @     8     [       ]  UNREF
IX2        7 SN   @    16     [       ]  UNREF
IX3        7 SN   @    24     [       ]  UNREF
PRTREC   132 UA   @  1528     [001400]  002000  002300  002500  002600
PRTFIL (FILE)     @  1208     [001300]  001300  001900  002900
READ   (PROG)     @  2856     [002200]  002700
```

```
NO  SYNTAX ERRORS.
ASSEMBLY TIME   21 SEC
TOTAL RECORDS -     21
REGULAR LABELS -    11
POINT LABELS -      0
CORE REQUIRED FOR THIS PROGRAM -   4000 DIGITS
DATE ASSEMBLED 08/21/67  10:04
```

APPENDIX H
ASSEMBLY OPERATION CODE LISTING

CONTROL ASSEMBLY INPUT/OUTPUT.

PSEUDO OPERATIONS WHICH AFFECT THE OBJECT PROGRAM.

CONTROL LOCATION COUNTER.

SEGMENTATION CONTROL.

MISCELLANEOUS.

** Differences between Advanced and Basic Assembler constructs and usage.
*A Only legal in Advanced Assembler.
*B Only legal in Basic Assembler.

MISCELLANEOUS (cont)

DATA DIVISION DECLARATIVES.

FILES AND RECORDS.

WORKING STORAGE.

EXECUTABLE INSTRUCTIONS AND PSEUDO INSTRUCTIONS.

DATA MOVEMENT AND CONVERSION.

** Differences between Advanced and Basic Assembler constructs and usage.
*A Only legal in Advanced Assembler.
*B Only legal in Basic Assembler.

DATA MOVEMENT AND CONVERSION (cont)

*A Only legal in Advanced Assembler.

BIT TESTING AND MANIPULATION.

BRANCHING.

SEGMENT/SUBROUTINE LINKAGE.

PRIVILEGED INSTRUCTIONS.

** Differences between Advanced and Basic Assembler constructs and usage.
*A Only legal in Advanced Assembler.
*B Only legal in Basic Assembler.

NORMAL INPUT/OUTPUT.

CONTROL PROGRAM COMMUNICATION.

** Differences between Advanced and Basic Assembler constructs and usage.
*A Only legal in Advanced Assembler.
*B Only legal in Basic Assembler.

BASIC CONTROL PROGRAM INPUT/OUTPUT.

DATA COMMUNICATIONS INPUT/OUTPUT.

SORTER-READER AND LISTER OPERATION CODES.

** Differences between Advanced and Basic Assembler constructs and usage.
*A Only legal in Advanced Assembler.
*B Only legal in Basic Assembler.

SORTER-READER AND LISTER OPERATION CODES (cont)

MACRO CONDITIONALS AND PSEUDOS.

LIBRARY PSEUDOS.

** Differences between Advanced and Basic Assembler constructs and usage.
*A Only legal in Advanced Assembler.
*B Only legal in Basic Assembler.

BURROUGHS CORPORATION
DATA PROCESSING PUBLICATIONS
REMARKS FORM

TITLE: B 2500 and B 3500 Systems
Assemblers Reference Manual

FORM: 1034949
DATE: 4-69

CHECK TYPE OF SUGGESTION:

☐ ADDITION ☐ DELETION ☐ REVISION ☐ ERROR

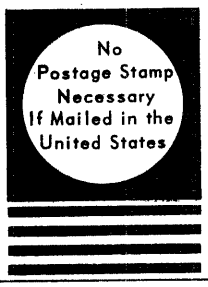
GENERAL COMMENTS AND/OR SUGGESTIONS FOR IMPROVEMENT OF PUBLICATION:

FROM: NAME _____ DATE _____
TITLE _____
COMPANY _____
ADDRESS _____
_____

cut along dotted line

STAPLE

FOLD DOWN SECOND FOLD DOWN



BUSINESS REPLY MAIL
First Class Permit No. 817, Detroit, Mich. 48232

Burroughs Corporation
6071 Second Avenue
Detroit, Michigan 48232

attn: Sales Technical Services
Systems Documentation

FOLD UP FIRST FOLD UP


Wherever There's
Business There's
Burroughs