S

IBM

Reference Manual

704-709-7090 Programming Package for the IBM 7030 Data Processing System

IBM Reference Manual

704-709-7090 Programming Package for the IBM 7030 Data Processing System

The program descriptions in this manual are considered preliminary and subject to future revision. Revisions will reflect changes in the programs and correction and clarification of details.

Strap-1 was produced by the Los Alamos Scientific Laboratory. The chapter describing Strap-1 was prepared by IBM, but includes many descriptions and examples supplied by Los Alamos personnel. The Simulator programs and write-up were produced by IBM.

Neither IBM nor Los Alamos guarantee the accuracy of the Strap-1 and Simulator programs. Furthermore, neither IBM nor Los Alamos assume any responsibility for errors resulting from the use of these programs.

CONTENTS

STRAP-1

STRAP-1 is a symbolic programming system for the IBM 7030 which utilizes a 704 with 32,768 word capacity for assembly. It is a planned predecessor of Strap-2, a more elaborate programming system for the 7030 which is to use the 7030 instead of the 704 for assembly. Because Strap-1 is a planned subset of Strap-2, the specifications defined here are applicable to both Strap-1 and Strap-2.

Strap-1 specifications are divided into three categories: Category 1 pertains to the Strap coding form. In this category a form is defined that conveniently allows for the expression of both machine instructions and pseudo-instructions that direct the assembler itself. Category 2 pertains to the expression of symbolic machine instructions. In this category, definitions are made covering symbolic instruction formats, the fields which make them up, and the various mnemonics, classes of symbols, and numbers that may be used in fields. Category 3 pertains to the expression of the compiler's pseudo-instructions. In this category pseudo-instruction mnemonics, formats, and addresses are defined.

STRAP CODING FORM

The coding form is directly related to the instruction card form. Both have 80 columns and are divided into 4 fields. These four fields and their respective positions are:

1 CLASS	2 NAME 9	10 STATEMENT 72	73 IDENTIFICATION 80

STRAP CODING FORM

The purpose of each field is:

1. Class (1 column): to identify the card format (binary, decimal, symbolic, etc.).
2. Name (8 columns): to identify the statement.
3. Statement (63 columns): to express a machine instruction or a pseudo-instruction.
4. Identification (8 columns): to identify the card.

Card identification (columns 73-80) is reproduced on the listing, but does not contribute any information to the assembly program for translating instructions.

EXPRESSION OF MACHINE INSTRUCTIONS

Machine instructions are written symbolically on the coding form described previously. They are usually entered one per line, according to a prescribed format that varies with the type of instruction operation. The instructions are written with fixed mnemonic operation codes.

Symbolic instructions are divided into fields (operation, address, offset, etc.) by commas. These fields may be stated within the general symbolic forms of the system, and, when so stated, constitute symbolic expressions. The order and manner in which such symbolic expressions are written in specifying the elements of any particular instruction are dictated by the symbolic instruction format; that is, a general type that provides for the expression of a whole class of particular machine instructions. Major fields may be further divided into subfields or modified by expressions contained in parentheses, such as index register specifications, secondary operations in progressive indexing, and so on.

1. A 11-0 double punch (;) is used to imply the end of a statement, so that multiple statements may be written per line. The number of instructions that may be written on one line is limited only by the number of columns available in the statement field of the card. The symbol in the name field of a card having more than one instruction in the statement field is associated with the first instruction only. The remaining instructions are treated as if they appeared on separate cards having blank name fields.

2. A single instruction cannot be continued from one card to another.

3. A comment may follow any instruction. A comment is initiated by the symbol ' (an 8-4 double punch) and is terminated either by the end of the card or by a ; . Thus, the character ; may never be used in a comment. A ' in the name field causes the whole card to be treated as a comment; it will be printed on the listing but will not otherwise affect the assembly.

SYMBOLIC INSTRUCTION FORMATS

Symbolic instructions are entered in the statement field. Within this field, variable length operation codes and address expressions are separated by commas and form subfields. A variable length modifier to either an operation or an address is enclosed in parentheses and attached to the modified subfield. Blanks have no meaning in any field except to indicate the spacing desired on the printed output listing. Blank cards are ignored. The twelve symbolic instruction formats for Strap-1 are:

	Format Type	Operation
1.	OP(dds), A_{18}(I)	Floating point
2.	OP, A_{19}(I)	Miscellaneous, unconditional branch, SIC
3.	OP, J, A_{19}(I) <u>or</u> OP, J, A_{18}(I)	Direct index arithmetic
4.	OP, J, A_{19} <u>or</u> OP, J, A_{18}	Immediate index arithmetic
5.	OP, J, B_{19}(K)	Count and branch
6.	OP, B_{19}(K)	Indicator branch
7.	OP(dds), A_{24}(I), OF_7(I')	VFL arithmetic, connect, convert
8.	OP_1(OP_2)(dds), A_{24}(I), OF_7(I')	Progressive indexing

	Format Type	Operation
9.	OP, J, A_{18}(I), A'_{18}(I')	Swap, transmit full words
10.	OP, A_{24}(I), B_{19}(K)	Branch on bit
11.	OP_1(OP_2), A_7(I), CW_{18}(I')	Input-output
12.	LVS, J, A, A', A'', A''',....	Load value with sum

Definitions of the above format symbols are:

1.	OP and OP_1	Primary instruction operation.
2.	OP_2	A secondary operation permitted only in progressive indexing and input-output.
3.	An	An "n" -bit data address.
4.	B_{19}	A 19 -bit branch address.
5.	I	A 4-bit index address where (0) signifies no indexing and (1.) to (15.) signifies indexing by the corresponding index register.
6.	K	A 1-bit index address where no modification (0) or modification by index register 1 (1.) are the only possibilities.
7.	OF_7	A 7-bit offset field.
8.	dds	Data description (see "Data Description").
9.	J	A 4-bit index address that refers to an index register as an operand. Here (0) refers to index register 0, word 16.
10.	A_7	A 7-bit input-output channel address.
11.	CW_{18}	An 18-bit control word address.
12.	LVS	Refers to one specific operation: Load Value with Sum
13.	primes	Used to distinguish otherwise identical fields in a format.

There is a general right-to-left drop-out order for all fields separated by commas. For example, a VFL instruction (format 7 above), for which the offset and its index modifier are zero, is written:

OP, A (I)

The comma is the major separator for the symbolic instruction types. If there are less than the maximum number of major symbolic fields in a given instruction expression (as indicated by the comma count), the instruction is compiled as if the missing fields contained zeros and had been added at the end of the statement. Such fields, whose contents are implied in a standard way by the omission of any explicit specification, are called null fields. A null field is always compiled as a zero, with the exception (indicated under "Data Description") of those subfields of a data description which express mode and byte size. Within a major field, a parenthesized subfield may be made null by omission. Thus, in the VFL example above, if the main index designation were to be zero but the offset and its index modifier (which in the hardware also modifies field length and byte size) were both to be one, the instruction could be written:

OP, A, 1(1.)

A major field may be null, even though other non-null fields follow it. Such is the case if nothing but the comma denoting the field termination is written. Thus, in the example just shown, if the offset and its modifier were both to be one but the principal address and its modifier were both to be zero, the instruction could be written:

OP, , 1(1.)

DATA DESCRIPTION (dds)

The small letters "dds" enclosed in parentheses in the special instruction formats stand for the data description field. This field is established by specifying:

1.	M	use mode
2.	L	field length, and
3.	BS	byte size

These three entries appear in the above order within parentheses and are separated by commas, thus: (M, L, BS).

A data description given with any of the four data entry or data reservation pseudo-operations (DD, DDI, SYN, and DR) is attached to the symbol in the name field, and is automatically invoked whenever that symbol appears in the principal address field of an instruction. When a string of symbols is added in an address field, the last symbol written is the one whose data properties control those of the instructions. When the data description is specified explicitly as a modifier to the operation code in the two machine instruction formats where it applies (VFL and floating point), it overrules any other data description derived from a symbolic address. Thus, in straightforward coding, it is unnecessary to write a data description in machine operations.

A description of the method by which a data description may be attached to the symbol that names a piece of data is given under "Data Definition."

There are seven fixed use mode designators:

1.	N	Normalized Floating Point
2.	U	Unnormalized Floating Point
3.	B	Binary
4.	BU	Binary Unsigned
5.	D	Decimal
6.	DU	Decimal Unsigned
7.	P	Properties Mode

The mnemonic "P" in the mode field of a data description has the following meaning:

(P, RIVER)

implies in an instruction that the data description associated with the symbol RIVER is to be invoked as if it had been written out explicitly. Thus, in an instruction, the dds of RIVER overrules anything implied by the symbol in the major address field. The P mode can be used only with legal machine instructions, never with a pseudo-operation.

Within a data description field, the usual right-to-left drop-out order and null field conventions hold (except, as indicated, that the mode field may not be null), so that a data description may appear in any of the following four forms:

(M)	Field length and byte size are null
(M, L)	Byte size is null
(M, , BS)	Field length is null
(M, L, BS)	

If the field length is null, a field length of 0 (effectively 64, except in the case of immediate VFL operations, where it is 24) is compiled. If the byte size is null, the compiled byte size is a function of the mode:

Mode	Standard Byte Size
D or DU	4
B	1
BU	8
N or U	Fixed format of 64 bits; field length and byte size not appropriate.

Cases can arise from programmer errors in which a data description and an operation are not mutually consistent. In this case, the operation overrules. If there is no way to obtain a data description from either the symbolic address or an explicit data description field, three cases arise:

1. The operation symbol can stand for either floating point or VFL operations (+, -, *, /). The operation is assembled as a VFL operation with data description (BU, 64, 8).
2. The operation symbol can stand for a VFL operation only (M+1). It is assigned a data description (BU, 64, 8). If VFL immediate, (BU, 24, 8) is assigned.
3. The operation can stand for a floating point operation only (-A, *NA). The operation is assembled as normalized floating point, except for E+1 and its modified forms, which are made unnormalized unless overruled.

An error mark will be printed on the listing in any of these cases. (See description of error flag "M" in Appendix D.)

MNEMONICS

A complete list of all machine mnemonics is included in Appendix A. Both operation codes and system symbols are included in the list.

A complete list of Strap-1 pseudo-operation mnemonics is given in Appendix B.

NUMBERS AND SYMBOLS

Two different number systems, in general, run through the Strap-1 language: the ordinary system of real numbers, and a bit-address numbering system. The ordinary real numbers are restricted in all non-data fields to being integers. Real numbers that are not integers may be entered as data, but they may not take part in arithmetic expressions nor may they be symbolized, so that the general forms of the language are actually limited to integers and bit addresses.

A bit address is a style of writing a machine address; it consists of a pair of integers separated by a period. The integer to the left of the period specifies a word address, and the integer to the right of the period specifies a bit address. Thus, 6.32 is the decimal equivalent of either a 19- or 24-bit binary address specifying bit 32 of storage location 6--the bit preceded by exactly six and one-half storage words. (Note that only the presence of the period distinguishes a bit address from an integer.)

Example: 505.17 = 500.337 = 0.32337

As the name "bit address" implies, the two integers are converted to and carried as 24-bit binary integers, such as are appropriate to the address fields of VFL instructions. When used in the address field of an instruction for which a shorter address is appropriate, a bit address is truncated to the correct length and inserted. The location counter contains a bit address. There is no limit on the size of the pair of integers in a bit address except that 64 x word address portion + bit address portion = $< 2^{24}$.

Thus, the address designation A(I) has two possible meanings:

1. If I is a bit address, then it designates an index word and is compiled in the so-called I field.
2. If I is an integer, then an address equal to A plus I times the field length of A is compiled.

A symbol is any sequence of six or fewer alphabetic and numeric characters conforming to the following conditions:

1. It contains only alphameric characters.
2. Its first character is specifically alphabetic.
3. It appears in the name field of an instruction, by virtue of which it is "defined" and is assigned a value that is either a 24-bit binary address or an integer, or occasionally both.

A given symbol may appear in the name field only once. The name of an ordinary machine instruction or data entry pseudo-operation is set equal to the value of the assembly program location counter at the point of its appearance in the code.

Symbols that identify storage elements in the object program are automatically assigned bit addresses that locate these storage elements. A symbol may, however, be given the value of an integer through the use of a "synonym" pseudo-operation. Thus, in general, both bit addresses and integers may be symbolized. The term "integer" is used to denote either an integral number or a symbol that takes on an integral value.

Symbols that name instructions themselves are automatically assigned data descriptions. Specifically, instruction-naming symbols are given field lengths equal to the lengths of the particular instructions named (either 32 or 64), and are defined as unsigned binary with byte size 8.

A programmer symbolized field is a field that may contain programmer symbols and/or system symbols. Of the fields shown in the instruction formats, all may contain programmer symbols except the operation field and the mode field of a data description.

Integers in programmer symbolized fields are always converted to binary. They are limited in length to the length of the field in which they are to be inserted. An integer larger than 24 bits cannot be symbolized.

Bit addresses and symbols for bit addresses are intended primarily for use in address fields of machine instructions. Integers and symbols for integers are intended primarily for use in fields for which they seem more appropriate: counts, shifts, field length, byte size, and so on.

ARITHMETIC EXPRESSIONS

Arithmetic expressions in Strap-1 may be composed of addition and/or subtraction of any combination of symbols, integers, and bit addresses. Although integers and bit addresses are generally used in different fields, algebraic addition of the two types of numbers is defined; the result is a function of the type of field into which the sum is to be inserted.

Integers add into all fields as integers; that is, the units digit adds into the low-order position of the field. Symbols for both bit addresses and integers are signed numbers. The number of additive operands in an arithmetic expression is limited only by the space available on the card.

Example: SAM- JOE+ FRED- 72.386+5,

This example, where SAM and JOE are defined as bit addresses and FRED is an integer, is, in general, a legal address. The data description of the final symbol, FRED, applies to the whole combination.

If the field for which the address is intended is signed (for example, the value field of XW or VF), the sign will be placed in the correct bit and the true value will be compiled.

If the final result is negative and the n-bit field for which it is intended is unsigned, a 2's complement is formed and inserted, except in the case of EXT (L, L') where|L| and|L'|are used. For example, in the case of the 7-bit offset field of a symbolic instruction, negative numbers may be used to describe the low-order position of the data field in relation to the left rather than the right end of the accumulator. Thus, the 128 bits of the accumulator, proceding from left to right, bear the offset addresses "127, 126,....1,0" or, alternatively, of "-1, -2,....-127, -128." The programmer is reminded that a 2's complement must be used with care on the 7030 in order not to cause the address invalid indicator to be turned on.

A positive result is inserted as a true value.

Either an integer or a bit address, or a combination of the two, may appear in any programmer symbolized field with five restrictions:

1. The "I" or "K" index field must contain at least one bit address term.
2. The entries in an array specification must not contain any bit address terms. In EXT (L, L'),(L, L') is not considered an array specification (See "Pseudo-operations that Direct the Compiler" for a discussion of DR and the specifications of arrays).
3. A period may not appear in the field of a parenthetical integer entry. A bit address appearing in such a field is treated as a 24-bit integer. For example, V+I, (.18) 4.32 is not allowed, but V+I, (.18)9 is.
4. No arithmetic may appear in the name field, which is reserved entirely for the definition of symbols. Only one symbol per statement may be defined in this manner.
5. Arithmetic expressions may not appear in the operation code part of the operation field, the mode subfield of the data description field or any entry mode field. These exceptions are reserved for designations whose meanings to the compiler are absolute and may not be symbolized.

Rules for Combining Integers and Bit Addresses

The following rules describe the method by which bit addresses and integers are truncated and added:

1. The numbers are shifted with respect to each other by the proper amount. See the following diagram.
2. The numbers are assumed to be signed 24-bit integers before the operation. Addition is algebraic.
3. The result is complemented if necessary.
4. The result is truncated if necessary.
5. The result is inserted into the correct position in the operation word.

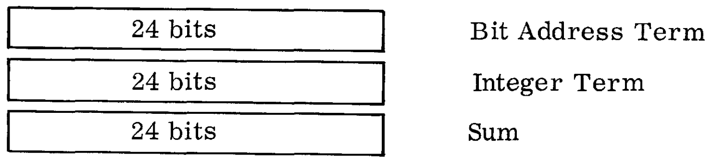
Although the diagrams show the final sum truncated to the appropriate length, the bits are not actually discarded unless they fall outside the address field of the instruction. Some operations do not use all of the space available in their address fields (transmit, input-output select), and in these cases bits may be placed in the unused portions.

An error indication is given if non-zero bits are discarded when truncation occurs (see explanation of "V" error flag in Appendix D), except in the case of index fields where a "1" bit in the fifth position from the right(in the "16" position) is discarded without error indication.

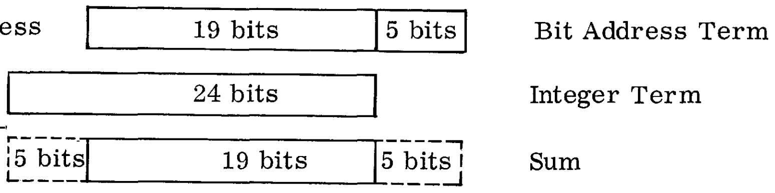
Truncation occurs for particular fields in the following manner:

1. A_{24} Bit Address

Rule: No truncation
Note: An integer in a 24-bit field counts bits

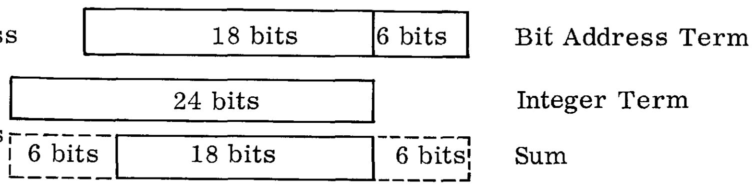


2. A_{19} Half-Word Address



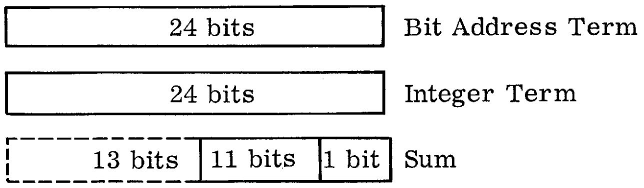
Rule: Leftmost 5 bits and rightmost 5 bits are truncated from sum
Note: An integer in a 19-bit field counts half-words

3. A_{18} Full-Word Address



Rule: Leftmost 6 and rightmost 6 bits are truncated from the sum
Note: An integer in an 18-bit field counts full words or unit address, control operation, control word address, and so on, in right I-O address.

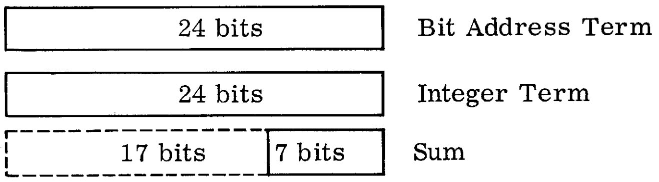
4. $A_{11\pm}$ Signed 11-Bit Address



Rule: Leftmost 13 bits are truncated from the sum. Rightmost 11 bits plus sign are placed in leftmost 12 bits of address field of shift and Add Immediate to Exponent instructions

Note: Integer counts number of bits in shift or number of bits to be added to exponent of floating point word

5. OF_7 Offset

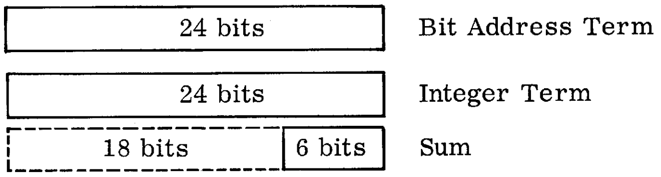


Rule: Leftmost 17 bits of sum are truncated

Note: Integers count number of bits of offset

Bit address 1.32 = .96 = integer 96

6. FL_6 Field Length

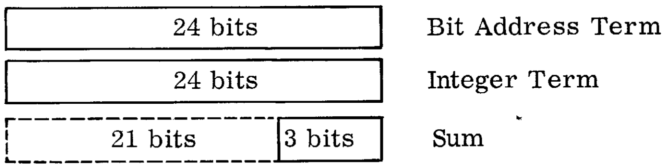


Rule: Leftmost 18 bits of sum are truncated

Note: Integers count length of field in bits

Bit address 1.0 = .64 = 0 not error marked

7. BS_3 Byte Size

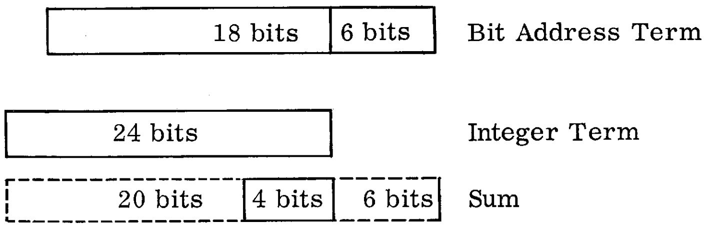


Rule: Leftmost 21 bits of sum are truncated

Note: Integers count byte size in bits

.8 = 8 = 0 not error marked

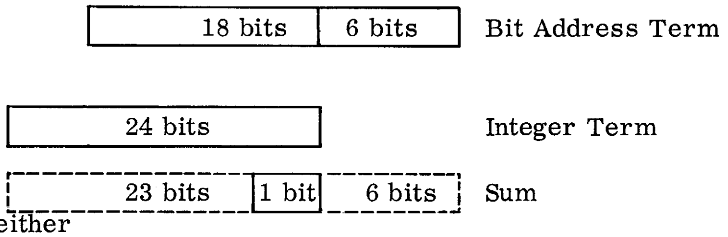
8. I, J 4-Bit Index Fields



Rule: Leftmost 20 bits and rightmost 6 bits of sum are truncated

Note: Integers represent index register number. A "1" in the bit position immediately to the left of the final sum field is discarded with no error indication.

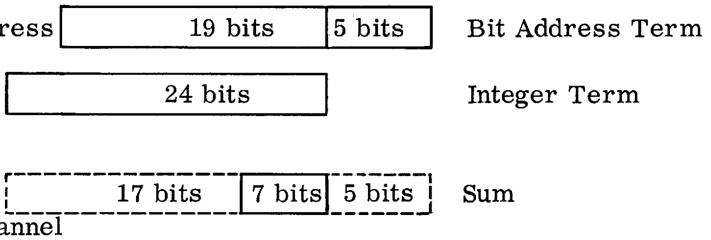
9. K Single Bit Index Field



Rule: Leftmost 23 bits and rightmost 6 bits of sum are truncated

Note: Integers specify either index register 0 or index register 1. A "1" in the bit position that corresponds to "16" in the sum is discarded with no error indication.

10. A_7 I-O Left Effective Address



Rule: Leftmost 17 and rightmost 5 bits are truncated from sum

Note: Integers specify channel address

SYSTEM SYMBOLS

System symbols are symbols whose values are fixed in the compiler. They are identified in programmer symbolized fields by the appearance of the special prefix character $ (which, as one of the non-alphameric characters, can never appear in a programmer symbol), followed by five or fewer alphabetic or numeric characters. System symbols may appear in arithmetic expressions in programmer symbolized fields where, in cases where restrictions apply, they can be considered the same as numeric entries because their values are <u>immediately</u> available to the compiler.

All system symbols that represent the addresses of special registers in storage ($AOC, the All Ones Counter) or special bits in storage ($LC, the Lost Carry indicator) are bit addresses. All others are integers or real numbers.

The appearance of the $ character alone makes for a special system symbol that provides a standardized substitute in place of a name for the current statement. That is, the character $ is a bit address which, in any particular statement where it appears, functions as if it had been defined by being written in the name field of that statement. Because it represents the value of the location counter when the instruction is encountered by the compiler (if the instruction actually compiles space in the program), the appearance of the $ as follows:

```
B, $-2.
```

means "Branch to the instruction which begins two full words before this branch instruction." In another illustration:

```
B, $+.32
```

the meaning is "Branch to the next instruction.", effectively a "no operation."

Another special use of the $ character is to prefix any operation code in this manner: $OP. This directs the compiler to suppress any error indications that arise in connection with the compilation of this statement.

The system symbols are:

1. Index Registers

$0 through $15, identical to $X0 through $X15, represent index registers 0 through 15, addresses 16.0 through 31.0 in storage. For example, $5 (or $X5) will be correctly replaced by the index field 5 if it appears in an I or J field, or by the address 21.0 if it appears in an address field.

2. Special Registers

The mnemonics for the system symbols that stand for special registers in the 7030 are listed below with the bit address and name for each.

Bit Address	Mnemonic	Name
0.0	$Z	Word number zero
1.0	$IT	Interval timer
1.28	$TC	Time clock
2.0	$IA	Interruption address
3.0	$UB	Upper boundary
3.32	$LB	Lower boundary
3.57	$BC	Boundary control
4.32	$MB	Maintenance bits
5.12	$CA	Channel address
6.0	$CPU	Other CPU
7.17	$LZC	Left zeros count
7.44	$AOC	All ones count
8.0	$L	Left half of accumulator
9.0	$R	Right half of accumulator
10.0	$SB	Sign byte
11.0	$IND	Indicator register
12.20	$MASK	Mask
13.0	$RM	Remainder register
14.0	$FT	Factor register
15.0	$TR	Transit register

3. Indicator Bits

The symbol for any indicator bit may be prefixed with a dollar sign and placed in a programmer symbolized field, where it will represent the correct bit address in word 11. Note that when the indicator symbols are inserted in the "branch on indicator" instructions, the dollar sign prefix is omitted. System symbols for indicator bits are listed in Appendix A.

4. Input-Output Addresses

Since the actual numeric addresses which are to identify particular I-O units and channels may be chosen arbitrarily, system symbols that represent integers are provided for use in addressing I-O equipment. The numeric values of members of this set of system symbols, unlike the values of all other system symbols, may vary from one installation to another in order that RDR, for example, may represent the card reader channel address independently of what that address, in any particular installation, may be. I-O system symbols are:

Symbol	Meaning
$PCH	Punch (Channel Address)
$PRT	Printer (Channel Address)
$RDR	Reader (Channel Address)
$DISK	Disk Unit (Channel Address)
	Note: The arcs of a disk may be addressed by any legal symbolic integer expression evaluated modulo 2^{12} to assure a valid arc address.
$CNSL	Console (Channel or Unit Address)
$TC1,TC2,...$TCK	Tape Channels 0, 1,...,K

If more than one punch, printer, console or any other input-output unit is attached to the machine, the same numbering system used in channel and tape addresses is adopted, where $CONSL=$CONSL0, and so on. Thus, one may have $PRT0, $PRT1, $PRT2, etc.

At each installation's option, some system symbols representing equipment not included in the particular system at hand may elicit error flags in the listings.

5. Mathematical Constants

Four mathematical constants, useful in many scientific and engineering problems, can be represented by system symbols. The four system symbols and their real number values are:

Symbol	Mathematical Constant
$E	e
$M	$\log_{10}e$
$N	$\log_e 2$
$PI	π

These four symbols can be used only in the data field of a DD statement using normalized floating-point mode. All of the system symbols in classes 1, 2, and 3 are bit addresses and are assigned standard data descriptions with mode binary unsigned, byte size 8, and a field length equal to the length of the register (or bit, in which case BS=1).

VARIABLE-IN-NUMBER FIELD FORMAT

The Load Value with Sum (LVS) instruction may be written with a variable number of address fields, each of which actually picks out a single bit position within the LVS address field itself. For an LVS order, each address field may specify one of index registers 0 through 15. These fields are evaluated exactly as if they were regular index designator fields, so that index addresses may be specified in terms of either bit addresses or integers in the usual manner.

PSEUDO-OPERATIONS

In this section are itemized a number of operation codes provided for purposes of defining data and controlling and directing the assembly process itself. Because these codes do not directly produce machine instructions in the object program, the functions which they do trigger are referred to as "pseudo-operations."

The pseudo-operations are grouped according to class. There are two main classes of pseudo-operations:

1. Those that create storage elements.
2. Those that control the assembly process.
   a. Those that define symbols by assigning values that appear in the variable fields.
   b. Those that give directions to the compiler.

The name field of all pseudo-operations that neither create storage elements nor define symbols is ignored, with the exception of CNOP (see "Pseudo-Operations that Direct the Compiler").

PSEUDO-OPERATIONS THAT CREATE STORAGE ELEMENTS

The following provide the basic means for defining and entering generalized data in the Strap-1 language:

Mnemonic	Name	Usage
1. DD	"DATA DEFINITION"	(EM) DD (dds), D, D', D'',... Where the bracketed dds is a data description prescribing the meaning of all succeeding numbers (D). The numbers D are compiled in consecutive fields and any symbol appearing in the name field of the DD statement applies to the <u>first</u> such field.

The use of the pseudo-operation DD enables the programmer to enter data into a program in a variety of forms. Among the possibilities that exist are:

a. Decimal to floating binary conversion, either normalized or unnormalized.
b. Conversion of decimal fraction to binary fraction in fixed point.
c. Integer to integer conversion from any of the radices 2 through 10 and 16 to a radix of either 2 or 10.
d. Conversion of alphabetic information to binary coded form.

In the general form illustrated above, the field symbolized by (EM) represents the entry mode, a field which supplies information about the form in which data appear on the card (see "Entry Mode in Data Definition Statements").

The data description (dds) is identical in form and content to that described under "Data Description;" that is, to the data description that may be used when writing an individual instruction (except that the "P" mode is not permitted in this or any other pseudo-operation). Thus, a data description may be given with a number at the point of definition of the number itself, or may be given at the point of reference as part of an instruction referring to the number. The relationship between these two different points of possible definition is as follows:

When the data description is given by a DD statement(or other data defining operation), the description is invoked whenever the symbol appearing in the name field of the DD statement is used in the principal address field of an instruction. The instruction mode and--in the case of a VFL order--the field length and byte size are supplied by this data description, which is logically affixed to the name of the DD statement.

Such a description set down at the point of symbol definition is overruled by a description appearing in an instruction referring to the symbol. Whenever an overruling description appears in the data description field of an instruction, the entire description which was given at the point of definition of the address symbol is overruled. Thus, the statement:

OP (BU), JOE

causes the binary and unsigned modifiers to be compiled along with an implicitly defined field length of 64 and a byte size of 8, regardless of the description occurring in the statement in which JOE appeared in the name field. Overruling is strictly local and applies only to the instruction at hand.

If symbols are used in defining either the field length or byte size subfields of a DD statement's data description, the symbols must be fully defined when the compiler encounters the DD statement on the second pass. This requirement is not imposed on the data description of an instruction because, in that instance, no assignment of storage space is dependent on the contents of the subfields.

The address fields, D, D', D", etc. are all equivalent to each other. They are compiled sequentially as separate pieces of data, each having the data description specified, but the name specified in the name field is attached only to the first piece of data. The effect produced is exactly the same as if the entry mode, operation, and data description were repeated on separate cards with only one D field per card and blank name fields. If one wishes to name the separate entries D, D', D", etc., it is necessary to punch each one on a separate card with its own name.

Programmer symbols may not appear in the main body of a D field; various letters in the main body of a D field have fixed meanings not subject to programmer control.

2. XW "INDEX WORD" XW, VALUE, COUNT, REFILL, FLAG

The location counter is rounded to the next full word. The contents of the four symbolic fields following the operation are converted and compiled in an index word format. FLAG denotes the machine field comprised of bits 25, 26, and 27. An expression in the flag field of an XW statement is therefore evaluated modulo 2^3. The octal integer 4 written in the flag field turns on the index flag in the index word being compiled.

NOTE: Bit 24 of the word format is taken to be the VALUE sign position. A negative sign is interpreted in two's complement form in the usual way for all other fields.

3. VF "VALUE FIELD" VF, VALUE

The location counter is rounded to the next half word. The contents of VALUE are compiled as a 24-bit plus sign quantity in positions 0-24 of the next half word. The location counter stands at bit 25 at the end of the operation.

4. CF "COUNT FIELD" CF, COUNT

The location counter is rounded to the next half word. The contents of the count field are compiled as an 18-bit integer in positions 0-17. The location counter stands at bit 18 at the end of the operation.

5. RF "REFILL FIELD" RF, REFILL

This pseudo-operation is the same as CF, except Refill is substituted for Count.

NOTE: The last four operations (the index word pseudo-operations) defined above are given data descriptions by the compiler, as though they had been defined by DD statements. Specifically, the index elements created by these orders have had the following data descriptions affixed automatically, and cannot be overruled in the pseudo-operation statement:

Operation	Data Description
XW	(BU)
VF	(B, 25)
CF or RF	(BU, 18)

6. CW "CONTROL WORD" CW(OP), ADDRESS, COUNT, CHAIN ADDRESS

The pseudo-operation CW employs a special symbolic format as illustrated above and defined initially under "Symbolic Instruction Formats." A set of secondary operations is provided-- whose members are expressed as parenthesized secondary operations in the manner of (OP) above--for the purpose of providing mnemonics for control word functions:

		Multiple Bit	Chain Bit	Skip Flag
CR	"Count Within Record"	0	0	0
CCR	"Chain Counts Within Record"	0	1	0
CD	"Count Disregarding Record"	1	0	0
CDSC	"Count Disregarding Record, Skip, and Chain"	1	1	0
SCR	"Skip, Count Within Record"	0	0	1
SCCR	"Skip, Chain Counts Within Record"	0	1	1
SCD	"Skip, Count, Disregarding Record"	1	0	1
SCDSC	"Skip, Count, Disregarding Record, Skip and Chain"	1	1	1

The location counter is rounded up to insure that the control word compiled will begin at a full word address. CW is assigned a data description of (BU, 64, 8).

ENTRY MODE IN DATA DEFINITION STATEMENTS

The data description field represents a kind of generalized use mode for the data, in that properties specified in this field are translated into bits and numbers that are compiled into machine instructions referring to the data. A corresponding field called the entry mode is available to specify properties which describe the source language information and its form, but which are not themselves compiled into the object program.

The entry mode may be employed in one of three ways.

Statement Entry Mode (EM) DD (dds), D, D', D'',...

An entry mode may be used to specify the properties of all data in a DD or DDI statement. When used in this fashion, it is enclosed in parentheses and appears before the DD or DDI operation code in the operation field. The mode is more general in form in its usage in connection with the data of a DD or DDI statement, as it may in this instance--but only in this instance--designate that alphabetic information is to be compiled. The two entry modes that may only appear as statement entry modes--that is, immediately before the operation code (of a DD or DDI statement)--are:

(Ax) Alphabetic Conversion (AQ) DD (BU, 60, 6), DO NOT PANIC Q

The card code characters beginning with the one after the comma which terminates the operation field are converted to IBM <u>tape</u> BCD until the character "x" is reached. The end-of-statement character is not itself compiled. (Note that <u>tape</u> BCD is different from internal 704 BCD.) Blanks occurring within the field to be converted are retained and stored correctly. The characters are counted by Strap-1 and the location counter is properly advanced.

The byte size of converted characters may range from 1 through 12 in a DD statement, or 4 through 12 in a DDI statement, and is specified by the dds. Leading zeros are inserted for each byte where BS >6; leading bits are truncated from each byte where BS <6. The byte size compiled in an operation referring to the data is set to either the specified byte size or 8, whichever is smaller.

The statement terminating character "x" may be any legal card code character except:

)
' (8-4)
; (11-0)
blank

Only one D field is allowed per statement.

(IQSx) Inquiry Station Conversion

The IQS entry mode operates in exactly the same fashion as the alphabetic entry mode, except that card code characters are converted to the 7-bit inquiry station code. Therefore, leading zeros are inserted where BS >7, and leading bits are truncated where BS <7.

Although the IQS code includes a large number of special characters, Strap-1 is limited to those which can be entered by means of IBM off-line card and tape equipment.

Statement or Field Entry Modes

Some entry modes may be used either to specify the properties of all fields of a statement or to specify the properties of a specific field or fields in a statement. Statement entry modes and field entry modes may both appear in the same statement. When contradictory properties (for instance, two different radices) are implied by the statement and field entry modes, the field entry mode overrules for the case of the particular field on hand. Entry modes may not appear in a manner that causes parentheses within parentheses.

(Fn): The entry mode (Fn) implies that the data which follow are written in the decimal radix, are to be converted to binary, and may include a decimal fraction portion that is to be converted to a binary fraction of length n bits.

The "n" symbolizes a decimal integer that specifies the number of fractional bits desired to the right of the binary point when the number or numbers which follow are converted. (Fn) appears in DD or DDI binary mode statements only.

Radix Specifications: In any programmer symbolized field not enclosed by parentheses, numerical integers and bit addresses may be written in any radix from 2 through 10, or 16. The radix is specified by enclosing the appropriate integer, written in decimal, in parentheses at some appropriate point in the subfield. (Usually, but not always, the radix specifier is the first item to appear in the subfield.) The radix applies to the entire subfield unless it is reset before reaching the end. If no radix is specified, the base 10 is assumed. If used as a statement entry mode, the radix specified applies to the entire statement unless individual fields contain their own radix specifier, in which case the field entry mode overrules the statement entry mode for that field only.

In the case of data entry, the radix specifier can be used with integers only; a decimal point or floating point notation implies a radix of 10. The entry mode radix specifies the radix in which an integer is written on the card, but says nothing about the one to which it is converted.

Some examples of the use of the radix specifier are:

1. (8)573 - 34+50 (all numbers are in octal)
2. (2) 11011011100011.111100 (bit address written in binary)
3. (5) SAM - 342 (the symbol SAM is not affected by the radix, having been previously converted to binary. The integer 342 is written in the number system of the base 5.)
4. (8)7436.(10)60+9 (the full word portion of this bit address is written in octal, whereas the bit address portion and the integer 9 are written in decimal.)
5. (2)DD(B, 16, 8), (10)-972, 111011110 (the first D field is written in decimal, the second one is in binary)

Field Entry Mode--Parenthetical Integer Entry

One entry mode in Strap-1 may never appear as a statement entry mode. By means of the parenthetical integer entry, any integer or pattern of bits may be stored in any position of an instruction or data entry field. The general format for this entry mode is:

$$(.n)\ A_{n+1}$$

The symbol .n represents the bit address of the rightmost bit of the field into which the integer is to be entered. The integer A_{n+1} is formed as an unsigned n+1-bit field and added into the addressed instruction or data field by means of a logical OR into the leftmost n+1 bits.

The parenthetical integer entry is made by means of a logical OR so that it may be combined with other fields of the statement or other parenthetical OR fields. The first bit of the statement is counted as bit 0. Although the parenthetical field may cross field lines within a statement, it may not cross statement lines. Thus, if the bit address is specified as ".n", the parenthetical expression has a field length of n+1 and is evaluated modulo 2^{n+1}. All parenthetical fields are regarded as unsigned, so that a negative number is compiled as the complement, re 2^{n+1}, of the magnitude of the number.

This entry mode cannot be used in pseudo-operations that give instructions to the compiler (SLC, END). This mode must appear in a statement that compiles space in storage. It is a modification that may be appended to any D field or to any programmer symbolized field (or in place of such a field) which is not enclosed in parentheses. Thus, FL and BS may not contain a parenthetical entry.

In the case of an instruction, the position of the entry is determined by counting the bits of the whole instruction field, no matter in which subfield the integer entry may be appended. For example, in a VFL instruction so modified, OP, A_{24}(I)(.n)A_{n+1}, OF_7 is exactly the equivalent of OP, A_{24}(I), OF_7(.n)A_{n+1}. In the case of a DD pseudo-operation, the position of the parenthetical field is determined by counting bits of the D field in which it appears; i.e., from the previous comma forward. In any case, the integer entry must follow all other information in the field or subfield in which it appears, except for another parenthetical entry.

Although one entry could be made to serve in any single instruction, it is more convenient to write several different integer entry specifications when one wishes to place numbers in various positions in an instruction. Therefore, no limit is set on the number of consecutive entries which can be written together, except as imposed by the length of the statement field on the card.

Because the parenthetical entry is not permitted to cross statement lines, ".n" must be less than or equal to 31 in a half word instruction, and less than or equal to 63 in a full word instruction.

Example: E+I, (.8) 41	The integer 41 will be converted to binary and OR'ed into the leftmost 9 bits of the E+I instruction.

Radix designators are permitted in parenthetical OR fields, separated by commas from the bit address designation, and the two may be in any order. Thus, (.32,8) or (8, .32) signifies an octal field to be terminated at bit 32.

Parenthetical expressions may contain anything that goes in a normal address field (except periods), but may not have other information such as real numbers or alphabetic characters which are permitted in a DD or DDI statement. A data description associated with a symbol appearing in a parenthetical field has no effect in this usage of the symbol. All numbers appearing in a parenthetical field are converted to an internal binary format, never to decimal or floating point.

Example:

1. (.50,8)17 - JOE + (10)4203(4, .22) - 33303(.60)1030
2. (7)(.30)1265(.20)(10)138 - (6)43(.10)553

Note that the radix does not have to be specified with the .n. If no radix is specified, the current operative radix is continued; it is not reset to 10. It is understood to be 10 if no radix has been previously specified in the field to which the general parenthetical integer entry is appended.

The radices which apply in the above examples are:

Example	Number	Radix
1	17	8
1	JOE	does not apply
1	4203	10
1	33303	4
1	1030	4
2	1265	7
2	138	10
2	43	6
2	553	6

All numbers that appear within parentheses are interpreted as decimal numbers.

THE FORM OF D IN A DATA DEFINITION STATEMENT

All data fall under the category of one of the six use modes of the data description field: N, U, B, BU, D, DU. The numbers D, D', D'',... are expressed in the general form:

± XX...X.X...X

Decimal numbers are a special case; they may be written in fixed or floating point form, with or without a decimal point. The general form is:

± XX...X.X...XE±YYY

In this form E means that the number which precedes it is multiplied by 10 raised to the power which follows it. That is, 572.34E - 57 means 572.34 x 10^{-57}. Overlapping facilities for specifying an exponent "Ei" are provided in the sense that the decimal point in the number itself also indicates a decimal exponent. If no decimal point is written, the number is assumed to be an integer. Thus, parts of the general form that are not necessary for writing a number may be omitted.

a.	±XXX	integer
b.	±XXX.XX	decimal fraction
c.	±XXXE±YYY	integer times power of 10
d.	±XXX.XXE±YYY	decimal fraction times power of 10

A plus sign is understood if no sign is specified. The decimal point may be in any position in the number. The portion of the number above symbolized by X is limited in length to 15 digits; that symbolized by Y is restricted to a length of 3 digits (recall that floating point numbers in the 7030 are limited to a range of 10^{308} to 10^{-308}).

Data entries may have other quantities following them which are identified and separated from the main number by declension characters. The declension characters, which are used for the insertion of specific fields, are:

1. Sign Byte Entry--Si

The letter S is used to enter information into the sign byte of data. The letter i represents an octal integer which is evaluated and OR'ed in with any sign byte previously calculated. Thus, if either the sign of the main number or i implies a negative sign bit in the sign byte, the sign byte sign position is made negative.

The sign byte generated depends on the byte size in accordance with the following table:

Byte Size	Sign Byte	
1	S	
2	ST	
3	STU	Z = zone bit
4	STUV	S = sign bit
5	ZSTUV	T
6	ZZSTUV	U flag bits
7	ZZZSTUV	V
8	ZZZZSTUV	

In a data definition statement where byte size 1 is specified, using sign byte entry S1 yields a negative sign, whereas if byte size 4 had been specified, S10 would yield a negative sign with zero flag bits.

2. Exponent Entry--Xi

The letter "X" may be used to enter any arbitrary information into the exponent of a floating point word. The decimal integer i is compiled as the machine exponent of a floating point number. It overrules and replaces the computed exponent, which is completely eradicated by the replacement process.

Rules for Entering Data

The legal formats for entering data can be classified according to the use mode written in the data description field of the DD statement. In general, an element listed in the general format may be omitted if it is not needed to specify the data.

The data entries in a DD statement are restricted to real numbers. Bit addresses are not permitted. Integers are allowed as a special case of real numbers, but they may not be symbolized.

Floating point data are always compiled in addressable full-words; the location counter is rounded up, if necessary, to the next full-word address in order to accomplish this. This is an example of a general Strap-1 principle: a machine format that ordinarily depends in use on the fact that the 24-bit address of the lead bit ends in a string of zeros of some definite length causes the compiler to round the location counter appropriately. Thus:

1. Instructions always start at either half- or full-word bit addresses.
2. Indexing full- and half-word storage formats are forced to begin at full- and half-word addresses, respectively.

3. A floating point data block being reserved through use of a DR operation code (defined in "Pseudo-Operations That Direct the Compiler") is forced to begin at a full-word address. Moreover, when a field from an instruction format requires the truncation of the rightmost bits before compilation, a warning indication is given if significant bits are truncated (which can occur if an instruction addresses a format other than its natural one; e.g., if a floating point instruction addresses a VFL data element).

Normalized Floating Point

Format: Name | DD(N), ±xx···xx.x···xxE±yyySn

The decimal number is converted to a normalized floating binary number consisting of an 11-bit signed exponent, a 48-bit fraction, and a 4-bit sign byte. If no sign byte has been entered by means of an S, the sign preceding the number is used with the flag bits set to zero. If a different binary exponent is desired, it can be entered following an X, as follows:

Format: Name DD(N), ±xx···xx.x···xxE±yyySnXzzz

Examples:

a. DD(N), 54.73 E 4
   54.73×10^4 is converted to floating binary. The sign bit is zero (= plus), and the flag bits are zero (i, e., entire sign byte is zero).
b. DD(N), -54.73 E 4, or DD(N), 54.73 E 4 S 10
   In this case the sign bit is set to one (negative) and the flag bits are zero.
c. DD(N), -54.73 E 4 S 5
   The sign bit is one, since the number is negative, and flag bits T and V are one. U is zero.
d. DD(N), 1, 3E-5, -45.7, 12 S 17
   This example illustrates the multiple entry feature. This single DD statement compiles four 64-bit floating point words and advances the location counter accordingly.

In normalized floating point a special feature is available for use in any D field, making the entry of rational fractions and certain irrational numbers much easier. Arithmetic involving several numbers may be written using the standard Fortran symbols. Strap-1 will perform the arithmetic and compile a single normalized constant. The operations available are: addition (+), subtraction (-), multiplication (*), and division (/); only relatively simple expressions are allowed--that is, they must contain no parentheses. Multiplications and divisions are performed first (in a series of multiplications and divisions they are done in order from left to right), and then the additions and subtractions. The arithmetic is done among absolute constants, and a sign byte may be used at the end. It will be OR'ed in with the final result.

Examples:

a. DD(N), 1/3, 472*351, 4-7*5/21 S 4
   Note: Sign byte entered in last D field.

b. DD(N), 27.9/31.4/12/14 E 5, 4+3*7/5*6

The number produced in the first case is: $\dfrac{27.9}{31.4 \times 12 \times 14 \times 10^5}$

in the second: $4 + \dfrac{3 \times 7 \times 6}{5}$.

c. DD(N), 1/7 - 3/11 + 1.4321 E - 2, .12 + 1/144

As an extra convenience, certain system symbols are defined by which constants involving irrational numbers can be entered. They are:

1.	$PI	π
2.	$E	e
3.	$M	$\log_{10}e$
4.	$N	$\log_e 2$

Thus, one can enter a number such as $4\pi \times 10^{-7}$ by writing:

DD(N), 4 * $PI * 1E - 7.

Note that in Strap-1 this arithmetic feature is available with the normalized floating point mode only.

Unnormalized Floating Point

```
Format: Name | (Fn)DD(U), ± xx···x.x···xE±yyySn X±n
or             DD(U), (Fn) ± xx···xx.x···xE±yyySnX±n, (Fn)±xx···etc.
```

The number is converted to binary with the correct number of binary fractional places as specified by the (Fn) entry mode, and a correct exponent is computed and entered. This exponent is overruled and replaced by that following the X if X is used (necessary only if, for some reason, the programmer desires an incorrect exponent). The entry mode (Fn) can come before the DD, in which case it applies to all D fields of the statement, or it may form the first element of a D field, in which case it overrules one given before the DD. Either the X or the S or both may be omitted or their order may be interchanged. Omitting S has the same effect here as in the normalized case. Omitting X simply allows the correct exponent to remain as computed. Leaving out the sign, decimal point, or E is permitted as in normalized numbers.

Examples:

a. DD(U), (F21) - 343.7, (F10) 432

Two numbers are compiled. In the first, 343 is converted as an integer and .7 is converted to a 21-bit fraction. They are joined and placed in the rightmost bits of the fraction portion of the floating point word, and the correct exponent (in this case 27) and sign are supplied. In the second D field, 432 is converted to a binary integer. Because ten fractional bits are specified, but no decimal fraction is written, the ten rightmost bits of the fraction field are set to zero and the number is entered with its rightmost bit in position 50.

b. (F15)DD(U), 767.52, 767.52 X-12 S11

The (F15) applies to both D fields. In the second, the computed exponent is overruled by the specified one and the number is made negative by means of the specified sign byte.

c. (F15)DD(U), 767.52, (F20) 767.52 S11 X-12, 398
   This example is identical to example b except that in the second field the operation entry mode (F15) is overruled by a field entry mode (F20), and the order of S and X is interchanged, which makes no difference. (F15) still applies to 398, however.

If the entry mode is omitted, two cases arise:

1) If the number entered is an integer, (F0) is understood.
2) If the number entered is a decimal fraction, it is converted to a normalized floating point number, but will be used as though unnormalized.

Examples:

a. DD(U), 17, 17X-35
   In the first case 17 is converted to binary and placed in the fraction with its rightmost bit in position 60 and an exponent of 48 supplied. In the second field the same thing is done except that the exponent is set to -35.

b. DD(U), 17.5
   In this example 17.5 is converted to normalized floating binary and stored as such. However, instructions whose normalization bits depend on the symbol in the name field of this pseudo-operation will have them set to unnormalized.

Note:	17 E 5	is an integer and will be recognized as such.
	17 E-5	is a decimal fraction and will be normalized.
	17.5 E 5	is an integer but will be treated as a fraction and normalized. Thus, a normalized integer can be assigned use mode "unnormalized."

An integer greater than 2^{48} is stored as a normalized number.

Binary Signed VFL

```
Formats: (Fn)DD(B, FL, BS), ± xx···x.x···xE±yy Sn
         DD(B, FL, BS), (Fn) ±xx···x.x···xE±yy Sn
         (R)DD(B, FL, BS), ±xx···xx Sn
         DD(B, FL, BS), (R) ±xx   xx Sn
```

A data definition of binary signed data may have either (Fn) or (R) entry modes, but not both at the same time. (Fn) implies that the data following it are written in a decimal radix, whereas (R) implies that the number following it is an integer. An integer subject to a radix entry mode must be written without the aid of E because E is not defined for a radix other than 10. A decimal fraction must have a controlling (Fn) entry mode. There is no obvious way to convert to a fixed point number without specifying the binary scaling. In the data description either the field length or byte size or both may be omitted. The implied field length in this case is 64; the implied byte size is 1. The sign byte need not be specified unless the programmer desires to have flag or zone bits different from zero. Note that the sign bit position changes for a byte size less than 4. To make a number negative, specify the sign byte as:

BS = 1,	S1
BS = 2,	S2
BS = 3,	S4
BS = 4,	S10

If a number has no entry mode at all, it must be a decimal integer, but may in this case be written with the aid of the E notation.

Examples:

a. (F7)DD(B,,4), .005E3S13, -17, 143.2S11, (8) 77760, 777

Implied field length is 64. Octal specification in the fourth D field overrules (F7) written before DD, but (F7) still applies to 777 multiply an integer by positive powers of 10. If either the field length or byte size is omitted, the implied values are FL = 64, and BS = 4.

Examples:

a. DD(D), -9534812, +173E5, 18E10S13
   Field length = 64; byte size = 4. A 4-bit sign byte is formed. Decimal-to-decimal conversion.

b. (2)DD(D, 20), 111010001101S7
   Byte size = 4. Binary-to-decimal conversion.

c. DD(D, , 8), 432E3
   Field length = 64. Decimal-to-decimal conversion. Four binary zeros are inserted in the zone positions of each byte.

Decimal Unsigned VFL

```
Formats: (R)DD(DU, FL, BS), xx···xx
         DD(DU, FL, BS), (R) xx···xx
         DD(DU, FL, BS), xx···xxxEyyy
         (Az)DD(DU, FL, BS), alphabetic information to "z"
         (IQSz)DD(DU, FL, BS), alphabetic information to "z"
```

The numerical conversion is just as in decimal signed mode except for the omission of the sign byte. Alphabetic conversion is exactly as in the binary unsigned mode, except that instructions referring to these data are compiled as decimal operations. For alphabetic entry, implied field length is equal to byte size.

Examples:

a. DD(DU), 8430051, (8) 77241, 82E10
   Field length = 64; byte size = 4.
   An octal-to-decimal conversion is inserted between two decimal-to-decimal conversions.

b. (IQS3)DD(DU, , 8), PUSH PANIC BUTTON 3
   Field length = 8.

Summary of Rules for DD Statements

Entry Mode	Appropriate Use Modes
Fn	U, B, BU
R	B, BU, D, DU
A	BU, DU
IQS	BU, DU

Note: Use mode N should have no entry mode.

Special Field Entry	Appropriate Use Modes
S	N, U, B, D
X	N, U

The floating decimal notation, using E to designate multiplication by powers of 10, is appropriate to all modes although it is always restricted to a decimal radix, and in the decimal use modes, is restricted to increasing the magnitude of decimal integers.

If the field length is omitted from the dds, it will be assigned a value of 64, except in the case of alphabetic entry where it is set equal to the byte size. The maximum permissible field length for a DD statement is 64.

The following examples illustrate the use of general parenthetical integer entry with DD:

a. DD(N), 572(.59)1, 347.89E12(.63, 2)1011
   In the second case the sign byte is specified by means of (.n) entry.
b. DD(B), (F9) -35.7(.24) SAM + 4
   The address SAM + 4 is placed in the first part of the 64-bit field, followed by the converted number -35.7.
c. (8)DD(BU), 4762(.10)707(10, .20)34
   707 is written in octal, 34 in decimal.

PSEUDO-OPERATIONS THAT DEFINE SYMBOLS

Almost all pseudo-operations (excluding SLC, CNOP, etc.) define symbols in the standard manner--any symbol appearing in the name field is assigned the current value of the location counter. Grouped under the present category of pseudo-operations are those that define symbols in other than the usual manner.

1. DDI "DATA DEFINITION IMMEDIATE"

This pseudo-operation is identical to DD except for the following points:

a. Like SYN (see below) it is purely definitive in character.
b. Only one major field may follow the operation field of the statement.
c. If no field length is specified, a field length of 24 is implied.
d. If the length of a string of alphabetic characters exceeds the field length, the excessive low-order characters are lost and an error indication is given.
e. The compiled field--less than or equal to 24 bits in length--is inserted in a 24-bit field within the symbol table and left justified.
f. A general parenthetical integer entry may not be appended to the end of the data field.
g. Neither of the floating point modes can be used.

If a DDI has a field length of less than 24, the number that it defines will appear in the leftmost portion of the address of the operation when it is compiled in an immediate operation. Unused bits in the right end of the address field will be zero, but they may be loaded by means of a parenthetical integer entry in the operation itself. If the address field of an immediate operation contains arithmetic among symbols or symbols and integers, the arithmetic will be done in binary, regardless of how the symbols are defined or what the mode of the operation is. All numeric entries in such an address field are handled exactly as other addresses and are converted to binary, never to decimal. Therefore, the only way to get a decimal number into the address field of an immediate operation without writing it in the 7030 BCD code explicitly is to symbolize it and use DDI. Care should be exercised in address arithmetic among signed numbers, because the sign byte is compiled as such and does not participate in the arithmetic as a sign.

Example A:

Name	Statement
JOE	DDI (DU), 9478
SAM	DDI (DU, 12), 342
BILL	DDI (DU, 24), 12
	LI, JOE
	+I, SAM
	-I, SAM+BILL

The preceding sequence is an example of slightly unconventional coding to illustrate what is possible. JOE has a field length of 24 implied. All three symbols have a byte size of 4. The address SAM+BILL is added in binary, but because the addresses do not overlap, they produce a legal decimal number, 342012. The result is 9478 + 342 - 342012 = -332192.

Example B:

Name	Statement
ALF	DDI(B), -142
JIM	SYN(B, 24), 389
RIP	SYN(B, 24), -210
	LI, ALF
	+I, JIM
	+I, JIM+RIP

In this sequence the sum -142 + 389 + 389 -210 = 426 is obtained. Because JIM and RIP are defined by SYN cards, the address arithmetic JIM + RIP is done correctly, yielding an answer of 179. If they had been defined by DDI statements, the address arithmetic JIM + RIP would have produced a result of -599.

When compiling addresses for immediate operations, Strap-1 assumes that a symbol defined by DDI has a sign byte if one is needed. It assumes that a symbol defined in any other way does not have a sign byte and compiles one having flag and zone bits equal to zero and byte size as specified in the dds. Address arithmetic between a symbol defined by DDI and anything else is marked as a possible error (see "+" error flag in Appendix D), although it is performed as shown above.

2. SYN "SYNONYM" A | SYN (dds), Y

The operation SYN is used to define a symbol in terms of an integer, a bit address, or a combination of the two. A symbolic expression representing either an integer or a bit address may also be used, with the restriction--as with SLC--that the expression be fully defined (although this may possibly mean as late as pass 3 of Strap-1).

When one writes:

A | SYN(dds), Y

the meaning of the newly defined symbol A is that whenever A is written in the program, the effect is the same as if Y had been written. The meaning of SYN is always one of exact substitution. Thus, data properties (dds) associated with Y and its bit address or integer classification are transferred to A.

SYN statements are permitted to have their own data description field. If, however, no data description is given, the data properties of the final symbol not in parentheses are transferred to the name. If this symbol has multidimensional properties, they are also transferred to the symbol in the name field. Refer to the discussion of the DR pseudo-operation ("Pseudo-Operations that Direct the Compiler") for an explanation of the use of SYN and multidimensional arrays.

Consider the following example of the use of SYN and the data properties of the final symbol:

Name	Statement
SAM	SYN(N), 1000.0
FLAG	SYN(BU, 3, 8),.61
	(intervening code)
	L, SAM + FLAG

The Load instruction loads only the flag from the floating point word SAM preparatory to some VFL arithmetic or tests on the flag.

The difficulty of evaluating addresses on SYN cards imposes certain minor restrictions on the order in which SYN cards may occur. In general, if a SYN card address contains one or more location counter dependent symbols, both the SYN card and the instructions defining these location counter dependent symbols should occur before any SLC with an address involving the name of the SYN card. The integer portion of any symbol must be completely defined by a chain of SYN's or DDI's. The bit address portion may be completely defined by a chain of SYN's, or by a chain leading to a symbol that is defined by the location counter as a name of an instruction or data.

PSEUDO-OPERATIONS THAT GIVE DIRECTIONS TO THE COMPILER

1. SLC "SET LOCATION COUNTER" A | SLC, Y

This pseudo-operation resets the location counter to the value of the address Y. The next instruction will be compiled at this address, subject to rounding upward conventions. For example, a floating point instruction will be located at the nearest full-word address. If Y is not a full-word address, the location counter will be rounded up to the nearest one.

Y must contain a bit address expression whose value is positive. Y may be any legal symbolic expression, but it must be evaluable by the time it is encountered in pass 2 of Strap-1; thus, the restriction on SYN cards mentioned above.

An integer that appears in the variable field of an SLC instruction is added in as a 24-bit address field; i.e., as an integral number of bits, and an error warning may be given (see description of error flag "L" in Appendix D).

Any symbol A appearing in the name field is ignored.

Note: In normal operation, cards are read in sequence, and the number of bits needed for each instruction or piece of data is added to an assembly location counter in order that each instruction or data entry may be assigned an address. A principle of rounding upward is followed, guaranteeing that an instruction, value, count, or refill will begin exactly on a half-word address, and that index words, control words, and floating point data will begin only at full-word addresses. The SLC pseudo-operation provides a means of setting the assembly location counter to any value at any point in the code, and thus gives the programmer complete control over the location of his code. Following an SLC, the location counter is advanced once more in normal fashion until another SLC card resets it.

2. END "END" A | END, Y

A card with the operation code END signals the end of an assembly, and is usually included as the last card of each symbolic program deck. A branch card is then punched with the binary output deck with an address Y, so that the instruction located at Y will be the first program order executed.

The END statement also functions as an origin setting statement for the storage assignments given to all symbols that are undefined. A symbol is undefined if it never appears in the name field of any statement. All occurrences of such a symbol are flagged as possible errors. The symbol is assigned a full storage word in the block whose origin is equal to the value of the location counter when the END statement is encountered (possibly rounded up to obtain an integral full-word address), and the symbol is given a normalized floating point data description.

If an END card is missing and an end-of-file is encountered, the end of the assembly is clearly indicated and Strap-1 supplies an END card. That is, a branch card with a blank address field is punched. If the programmer wishes to punch a branch address on this same card, care must be taken to correct the check sum before attempting to load the program.

Any symbol A in the name field is ignored.

3. CNOP "CONDITIONAL NO OPERATION" A | CNOP,

The pseudo-operation CNOP is used to insure that the instruction or data immediately following the CNOP will be assigned a full-word address by the compiler.

When a CNOP is encountered, the location counter is immediately rounded up to the next half-word address. Then the compiler examines the location counter. If it already stands at a full-word address, the CNOP is ignored. If, however, the location counter is set to a half-word address, the machine instruction NOP is compiled. This has the effect of advancing the location counter 32 bits or one half-word to the next full-word address.

Any symbol A appearing in the name field is assigned a full-word address when the CNOP is ignored, or a half-word address when a NOP is compiled.

4. TLB "TERMINATE LOADING AND BRANCH" TLB, Y

The pseudo-operation "Terminate Loading and Branch" is similar to an END statement with one major distinction: TLB does not stop the assembly process. Therefore, TLB may be assembled at any point in a symbolic deck where a transition card is desired. The branch card thus produced will interrupt the loader when encountered in a binary deck and transfer control to the instruction at location Y. The remainder of the program is loaded under program control.

5. EXT "EXTRACT" A | EXT(I, J) STATEMENT

The "Extract" pseudo-operation has the following meaning:
First, Statement is compiled as is any legal machine instruction. Then the field beginning at I and ending at J is extracted and compiled in the position in the code where the EXT occurs. The symbol A appearing in the name field is assigned a data description (BU, J-I + 1, 8) and is attached to the quantity compiled. The terms I and J may contain any number of symbolic integers, but any bit addresses contained in the terms must not depend on the location counter, or else these bit addresses must be defined by a preceding card.

If EXT is used to specify the extraction of anything beyond the range of the single statement that follows it, up to 64 zeros will be added.

Example: EXT(18, 47) + (B, 18, 7), 73.16

First the full-word instruction + (B, 18, 7), 73.16 is formed. Then bits 18 through 47 (the first bit in the instruction is numbered 0 according to 7030 custom) are extracted and placed in the program being compiled. The dds (BU, 30, 8) is formed. The location counter is advanced 30 bits.

Note: Statement must be a legal machine instruction, not a pseudo-operation.

6. DR "DATA RESERVATION" A | DR(dds), (N)

A DR reserves space for data. The operation causes N fields of the kind described in the data description to be reserved; that is, the instruction location counter is skipped forward a quantity in bits equal to the product of N and the field length specified in the dds. Any symbol A appearing in the name field is attached to the first field reserved, as is the dds. Therefore, whenever A appears as the principal address in an instruction, this dds is invoked in the same manner as with a DD or DDI statement. Thus:

```
JOE     DR(BU, 8,8), (10)
```

reserves ten 8-bit fields (skips the location counter forward 80 bits). The dds (BU,8, 8) is attached to JOE. JOE is attached to the first 8-bit field reserved.

DR also provides a convenient method of defining multidimensional arrays of data and of addressing individual elements of arrays so defined. All indexing required for the manipulation of the array must be handled by the programmer.

The statement:

A DR(dds), (L, L', L'',..., L^r)

reserves space for an L x L' x L'' x... L^r array of data fields. The location counter is skipped forward a number of bits equal to the field length (specified in the dds) multiplied by the product of the dimensions of the array. (If the dds specifies the floating point mode, the correct number of full-words is reserved, beginning at a full-word boundary.)

Any symbol A appearing in the name field is attached to the first element of the array, and the dds is attached to the symbol in the normal fashion. Thus, in an instruction, a specific element of the array may be addressed by writing:

$$A\ (q,\ q',\ q'',\dots,q^r)$$

Note that the first element of the array has the address:

$$A\ (0,0,0,\dots,0)$$

and the last element is located at:

$$A\ (L-1,\ L'-1,\ L''-1,\dots,L^r-1)$$

The address of an arbitrary element in the array may be computed by means of the formula:

$$\text{Address of } A(q,q',q'',\dots,q^r) = \text{Address of } A(0,0,0,\dots,0)\ +FLx(q+q'L+q''LL' +q'''LL'L''+\dots)$$

where FL is the field length of an element in the array. An array address computed in this manner may be used in any programmer symbolized field not in parentheses, except a general parenthetical integer entry. The dimension of a DR statement must be evaluated by the end of pass 1. Therefore, the dimension and the field length of a DR must not be location counter dependent symbols; they may be defined by a chain of SYN's ending in an integer.

A fifteen dimensional array is the largest that can be specified in Strap-1.

L, L', L'' , etc., must be integers in symbolic or numeric form. Referring to "Number and Symbols," to apply index register I to the second element of a one dimensional array A, write:

A(1) (I)

where I must be a bit address.

SYN must be used to define a symbol as an interior element of a multidimensional array and have the dimensional addressing properties carried along. For example:

Name	Statement
A	DR(N), (10,20)
B	SYN, A(5,5)

In the above example, the rectangular array goes from A(0,0) to A(9,19); B goes from B(-5,-5) to B(4,14); A and B use identical storage. Thus, A(0,0) - B(-5, -5); A(1,0) - B(-4, -5); A(1,1) - B(-4, -4); etc.

7. DRZ "DATA RESERVATION AND SET TO ZERO" A | DRZ(dds), (N)

DRZ operation is exactly the same fashion as DR with one exception: the fields reserved are all set to zero.

8. PRNS "PRINT SINGLE-SPACED" PRNS

This pseudo-operation causes the assembly listing to be printed with single spacing. Double spacing is the normal mode unless PRNS is written.

9. PRND "PRINT DOUBLE-SPACED" PRND

This pseudo-operation restores printing to the normal double spacing condition after the use of a PRNS.

10. PUNFUL "PUNCH FULL CARDS" PUNFUL

Full cards (80 columns) are punched, without check sum, FWA, ID, and so on.

11. PUNNOR "PUNCH NORMALLY" PUNNOR

This pseudo-operation restores normal punching after the use of a PUNFUL.

12. SKIP "SKIP PAPER" SKIP, i

If i = 0, the assembly listing will restore the paper immediately. If i ≠ 0, one half-page will be skipped.

13. PUNID "PUNCH ID" PUNID, XXXXXXXX

The first 8 characters following the comma are punched in columns 73-80 of each card in the binary deck produced by the assembly program. Thus, a PUNID card should be used to identify each assembly. The X's may be any legal card code characters.

14. PRNID "PRINT ID" PRNID, COMMENT

When PRNID is encountered, the entire contents of this card are immediately printed on-line and on the output tape as well. PRNID provides a means of heading the assembly listing with such information as the problem name, programmer, and so on.

15. SEM "SUPPRESS ERROR MARKS" SEM, A, B, C

The operation code SEM, followed by a blank address field, causes all error marks detected in the statements that follow the SEM statement to be suppressed in the listing. Any particular error flag or group of flags may be suppressed by writing the letters or characters identifying the flags in the address field, separated by commas. Thus:

```
SEM,Q,T
```

suppresses error flags Q and T only. The only restriction in the use of SEM is that flags J and K, plus any flags that represent punching errors (such as ")"), can never be suppressed either by supplying an SEM statement with a blank address field or by individually addressing these flags.

16. REM "RESUME ERROR MARKS" REM, A,B,C

An REM restores normal error marking on the listing after an SEM has been used. The ability to specify individual error flags (with the same restrictions) is also available with REM. Thus, following the statement SEM,Q,T the pseudo-operation REM,Q restores error flagging involving flag Q only, while flag T remains suppressed.

17. TAIL "TAIL" TAIL, CHARACTER

The statement TAIL, X causes a block of statements to have the symbol X appended as a suffix to each symbol appearing in each statement of the "tailed" block. The block is ended by the next Tail statement or by an Untail statement. (Untail is equivalent to a Tail statement with a null field.)

In Strap-1 the symbol X may be any single alphabetic or numeric character. In Strap-2 the tail symbol may be any legal symbol.

If a basic symbol is defined two or more times in different blocks, then "within block" references may be made. References from one block to another must use the mechanism:

```
OP, JOE$X
```

where the desired JOE is defined in the block tailed by X, and is defined there only once. Also permitted is:

```
OP, JOE$
```

which references a JOE defined in an untailed block of code. An untailed block behaves exactly like a block tailed by a blank.

IBM 7030 SIMULATION SYSTEM

THE IBM 7030 Programming Package tape is available for use on the IBM 704, 709, and 7090 Data Processing Systems. This tape contains two programs: LASL Strap-1 for assemblies, and the 7030 Simulation System for program checking. Strap-1 is discussed here only as it relates to the operation of the 7030 Simulation System.

The IBM 7030 Simulation System is a binary interpretive simulator, written to assist in checking out 7030 programs before the system is available for use. It consists of:

Loader
7030 Simulator
Trace - with breakpoint facilities
Dump - with breakpoint facilities and multiple formats

Checked simulated programs will not necessarily run unchanged on the 7030, especially in the input-output area. Input-output is unusual in that almost all operations are executed, but in a limited fashion. For example, only 72 columns of an 80-column card are read on-line. No attempt is made to simulate input-output timing; therefore, input-output appears to be instantaneous since, after a select is given, the input-output is performed and the indicators are turned on before the next instruction is executed.

The restrictions of the 7030 Simulation System are:

1. The computer system to be used in simulation must have 32,768-word core storage capacity. The 7030 system simulated will have 8,192-word core storage capacity, one 256-arc disk unit, and three magnetic tape channels.
2. The program uses $(40000\text{-}77777)_8$.
3. No attempt is made to simulate timing or ECC mode.
4. Simulation of input-output does not include console, operator-initiated channel signals, IQS and certain other physical features.
5. Clocks will not function. Information in 1. 0. -1. 32 will be lost when a dump is executed (709, 7090 only).
6. No interrupts will be taken for indicators 0, 1, 2, 4, 5.
7. High density is not simulated.
8. Tape records are limited to one thousand 7030 words.
9. Backspace file will be simulated on the 704 only if the instruction is available on the particular 704 being used.

The remainder of this chapter is separated into two sections: operator's notes and programmer's notes. Programmers should read both sections carefully.

PROGRAMMER'S NOTES

This section contains information which is useful to the programmer in preparing a deck to be used by the simulator. These specifications for card formats, composition of the input deck, output formats, breakpoint dumps and traces, etc. should be thoroughly understood before any attempt is made to simulate a 7030 program. Users of the system have made many suggestions to help eliminate some of the most common errors that might occur before programmers become familiar with the system; these suggestions are included in this manual.

SETTING UP INPUT FOR STRAP-1

When preparing a symbolic deck for assembly by Strap-1, the following conventions should be obeyed:

1. A PRNID card should be included for identification of the printed output. This procedure is a courtesy to the operator.

2. A PUNID card should be included for identification of the binary cards produced by Strap-1, and for the separation of these decks. This is important!

3. Normally, the SLC address should have a decimal point. If no point is written, the address becomes a bit address. Thus, if a programmer wishes to begin his program at location 100, but writes "SLC, 100", Strap-1 interprets the address to be 100 bits or location 1.44, which is an illegal address for an SLC statement.

4. All I-O units should be symbolically addressed, since the units available will vary from installation to installation.

5. In contrast to earlier IBM Systems, the 7030 will have no Clear key on the console. Therefore, it is normally preferable to use DRZ rather than DR when the programmer wishes to reserve storage. Note that DRZ yields the same size of binary deck as DR.

6. Most programs are terminated with a BEW, A. Observe two cautions here:
   a. "A" should not be zero, or an invalid address interrupt will occur.
   b. If any indicators that are masked on have been turned on earlier in the program while in the disabled state, an interrupt will occur when the BEW is encountered.

   In any event, an interrupt table should always be supplied by the programmer even if it contains only NOP's.

STRAP-1 BINARY OUTPUT

Output cards have been designed according to the requirements of the supervisor. All cards contain a code column indicating the type of card, a 12-bit check-sum, an ID column for installation bookkeeping, and a sequence number which is used to keep the cards in the deck in order.

Origin Card

The origin card contains an origin address, a bit count, a secondary bit count with two control bits, and up to 23 half-words of data. The data are entered according to the origin address and the bit count. The secondary bit count is used to determine the number of bits to be skipped or set to zero (determined by the first control bit) before or after (determined by the second control bit) the data are entered. The card format is as follows:

Bits Assigned	Use
1.0 - 1.11	Code column (origin card 1.9, 1.10, 1.11 punches)
2.0 - 2.11	Identification column (binary)
3.0 - 3.11	Sequence number (binary)
4.0 - 4.11	Check sum

Bits Assigned	Use
5.0	A 1 bit control; 0 if skipping, 1 if setting to zero
5.1	A 1 bit control; 0 if skipping or zeroing is done before card contents are loaded, a 1 if after
5.2 - 5.11	A 10-bit count of the number of bits to be loaded from the card
6.0 - 7.11	A 24-bit address designating a new origin
8.0 - 9.11	A 24-bit address designating the number of bits to be skipped or set to zero
10.0 -10.7	Not used
10.8 -71.11	Up to 736 information bits (23 half-words)
72.0 -80.11	A 9-column field ignored by the loader; may be used for card code identification and sequencing

Flow Card

A flow card contains 25 half-words of data which are to be loaded according to the loader's location counter, i.e., in sequence with data of the previous card loaded. The card format is as follows:

Bits Assigned	Use
1.0 - 1.11	Code column (flow card 1.9, 1.11 punches)
2.0 - 2.11	Identification number (binary)
3.0 - 3.11	Sequence number (binary)
4.0 - 4.11	Check sum
5.0 - 5.3	Not presently used
5.4 -71.11	25 half-words of binary information
72.0 -80.11	A 9-column field ignored by the loader; may be used for card code identification and sequencing

Branch Card

A branch card contains an address to which the loader transfers control, i.e., resets the location counter. If no address is specified, control is transferred to the address of the first origin card.

Note that 7030 programs cannot be loaded into registers below $(40)_8$ or into locations above $(17777)_8$. Upon recognition of a branch card, the loader sets to zero that portion of 7030 memory which lies outside of the area used by the program (that is, the part defined by the upper and lower boundaries).

The format of the branch card is as follows:

Bits Assigned	Use
1.0 - 1.11	Code column (branch card--1.8, 1.9, 1.11 punches)
2.0 - 5.11	Not presently used
6.0 - 7.11	24-bit transfer address

CARD FORMATS ACCEPTED BY THE SIMULATOR LOADER

IBM 7030 programs being run on the simulator may be loaded through either the on-line card reader or tape. The loader will load five types of cards.

Standard 7030 Cards

These cards are the origin, flow and branch cards produced by the Strap-1 assembly program.

C Cards

Up to four corrections are permitted per octal-hex correction card. An origin need be specified only on the first correction card if subsequent correction cards are to be loaded sequentially. Any number of corrections from 1-4 may be loaded from a single card. Any correction may be ignored without repunching the entire card on which it appears, by punching a 1 in the column where the period appears. The loading addresses are stepped even when corrections are ignored, so that corrections following an ignored correction are loaded correctly. All four corrections on a card may be ignored, if desired.

C cards are punched in the following format:

Col 1	C	
Col 2-9	Location of the first correction to be loaded. This is punched in the form $(xxxxxx.x)_8$.	
Col 12-23	Half-word	Each half-word is in the form xxxxxx.xxbHH
Col 28-39	Half-word	where x = octal digit
Col 44-55	Half-word	b = blank
Col 60-71	Half-word	H = hex character

P Cards

Patch cards allow the programmer to introduce patches in his program. These cards will cause the instruction at location X (specified in columns 2-9) to be changed to a B,Y if it is a half-word instruction or NOP;B,Y if it is a full-word instruction. The instruction formerly at X will be loaded at the first available location at the end of the program (Y) followed by the contents of the P card. A branch back to x+.32 or x+1.0 will be inserted after the contents of the P card. P cards may be followed only by other P cards or a branch card. Note: P cards must be included when setting the boundary address registers. P cards may also be used to introduce breakpoint dumps and traces for program checking purposes. (See "Breakpoint Check-Out.") For a fuller description of this use, see the sections describing the dump and trace programs in the simulator.

The format for a P card is:

Col 1	P
Col 2-9	Location of the last instruction (must be a half-word or the first half of a full-word instruction) which is to be executed prior to the patch. The format for this location is the same as in a C card.

Col 12-23	Half-word	Format the same as in a C card
Col 28-39	Half-word	
Col 44-55	Half-word	
Col 60-71	Half-word	

T Card

A "T" punched in column 1 of a card tells the loader that the card following the T card is out of sequence and therefore the sequence counter should be reset. Thus, if two programs are put together separated by a T card, the sequence error between the last card of the first deck and the first card of the second deck will be ignored.

The T card need not immediately precede the card that will be out of sequence, as long as any cards that appear between the T card and the out-of-sequence card do not affect the sequence counter (e.g., C card, P card). Of course, a T card cannot follow P cards, since nothing but branch cards or more P cards may follow a P card.

N Card

An "N" punched in column 1 of a card tells the loader that the information punched in columns 2-72 of the same card is to be printed out under the control of SSW 1 and SSW 2. (See "Operator's Notes.")

It is strongly recommended that an N card containing identifying information be placed in front of every binary deck; all dumps and traces produced by the 7030 Simulation System will then be headed by a line of identification.

PREPARING A BINARY DECK FOR THE SIMULATOR

The input deck for the 7030 simulator should be carefully set up by the programmer to insure that the operator can run the job with the fewest complications.

The Input Deck

An N card should be the first card of every binary deck. As previously explained, the N card should be used to identify the output, and should contain at the very least the name of the programmer.

Next should come the binary cards with C and P cards, inserted in that order, before the branch card. The positioning of P cards is important (see "P Cards") and if this order is violated, a great deal of set-up time is wasted by the operator.

Information to the Operator

The timing of the simulator makes it imperative that certain information be given to the operator lest it be all but impossible for him to run any program correctly.

Some estimate of timing in minutes (assuming the average execution time of an instruction to be 10 ms) must be supplied with each binary deck to be simulated.

The address of the expected BEW instruction which is being used as the final stop must be supplied. The operator does not want the <u>location</u> of the BEW instruction.

Information on the number of breakpoint dumps and traces the programmer expects to be taken will be helpful also. In addition, 7030 tape channel and unit numbers, as well as the disk tape number, should be given to the operator in terms of the physical unit numbers as they appear in the table in "Input-Output Usage."

INPUT-OUTPUT USAGE

Seven I-O channels are provided; the assignments are as follows:

Octal	7030 Channel No.	704 Unit
0.0	0.0	Disk [Tape 10(A)]
20.0	16.0	Tapes 4 and 5 (A)
20.4	16.32	Tapes 6 and 7 (A)
21.0	17.0	Tapes 8 and 9 (B)
21.4	17.32	Unused
22.0	18.0	Reader
22.4	18.32	Printer
23.0	19.0	Punch

The instantaneous nature of the simulated I-O should be kept clearly in mind when setting up I-O interrupt handling routines.

Disk. A tape unit is used to simulate a partial disk containing 256 locatable arcs, each arc containing 1024 words for the 7030. Each arc is separated by an end-of-file on tape, so this tape must be prepared by the Write Disk program (See "Operator's Note") and retained.

Tape. On channels 16.0, 16.32 or 17.0, units 0 or 1 may be located. All control operations are accepted, but certain ones may be NOP'ed--namely, ECC, BSFL (backspace file) on the 704; UNLD (Rewind and Unload) and any other control operation which has no counterpart on the 704 or 709. HD (High Density) is accepted, but low density is both written and read. Tape 9 is used as the system output tape, so it should be used only when absolutely necessary, and the operator should be advised when it is to be used.

Reader. All types of control operations are accepted when applied to the reader. Only 72 columns of data are read from any card. The last eight columns are treated as blank input.

Printer. All types of control operations are accepted when applied to the printer, but only 72 columns are written. The other 60 columns are treated as blanks, but the end byte will terminate a line. Only single or double spacing, restoring, and space suppression are available. Any attempt to skip more than one line will be automatically treated as a double space.

Punch. As in the case of the reader, only 72 columns are available.

Console. No simulation has been attempted for the console other than the Initial Program Load Mode.

Initial Program Load Mode. IPL may be simulated at any time, either while running or when preparing to load, by depressing SSW 6 and entering the channel number in octal in the console keys (then pressing Enter MQ on the 704). Explicit directions are furnished in the Operator's Notes. Note that IPL from the card reader will treat the last one and one-half words on each card as blank.

BREAKPOINT CHECK-OUT

To make the simulator usable for check-out of 7030 programs, two auxiliary programs are incorporated into the system--a dump and a trace. Because of the timing problem involved in the operation of the simulator (each instruction simulated is a loop), it is difficult, if not impossible, for the operator to get any hint of improper operation of the problem program. Therefore, all check-out features are controllable by the programmer as well as the operator.

The only information available to the operator at the end of a run of a problem program is the instruction counter, which can be found in the accumulator. Thus the programmer should use the breakpoint dumping and tracing features and relieve the operator of any check-out decisions.

The Trace Program

The trace program traces the execution of each 7030 instruction simulated and prints the location of the instruction itself and its mnemonic, the contents of the effective address, and any changes that have occurred in storage locations 0-31 as a result of this instruction. Output may be obtained on tape 9(B) and on the on-line printer. Output is under the control of a sense switch. (See "Operator's Notes.")

Trace Output Format

The first line of the printed output contains:

1. The location counter. A $ will occur to the right of the location of the instruction being interrupted. An * will occur to the right of the location of any Execute instruction. In the latter case, the next instruction printed will be the instruction which is the object of the Execute instruction.
2. The instruction being traced, printed in the dump format (octal-hex).
3. The instruction mnemonic.
4. The contents of the effective address of the left address of the instruction. If the instruction in question is a full-word instruction, two full words are displayed; if a half-word instruction is being traced, only one full word is displayed. It should be stressed that this feature is available as an additional aid to the programmer. Under certain conditions, such as progressive indexing, the printed contents of the effective address may not reflect the true contents of that location.
5. The state of the machine (enable or disabled).

Subsequent lines contain:

6. The word "Panel" followed by the hex address of one or more of the first 32 locations which were changed by the instruction being traced. The hex addresses range from 00-0F, X0-XF.

7. The contents of this panel location in hex.
8. The contents of this panel location in the dump format (octal).

The indicator register (hex identification PANEL0B) and the mask register are always shown in binary; everything else is in octal.

Breakpoint Trace

Breakpoint traces may be accomplished by means of introducing a half-word, via a patch card, in the format

xxxxxx. 24 0H (H indicates any hex character may be used to specify an index register.)

The operation code 24 is recognized as an invalid operation (but indicator OP is not turned on) and tracing will start immediately and will include this invalid instruction. Tracing will continue until the location counter reaches the value specified by the effective address (using index register H) of this half-word.

Operation code 24 may also be a 64; in this case xxxxxx. 4 would be specified. At assembly time the necessary half-word may be compiled by a BE, A (.21) 20 where A is the last location to be traced. Other instructions (BD, BEW) can also be used in conjunction with the parenthetical integer entry mechanism to produce the invalid half-word operation.

If the high-order octal character of the effective address is a 4, 5, 6, or 7, the left half of the accumulator is printed in floating point format. Everything else is printed in the normal trace format.

The Dump Program

The Dump program records the contents of core storage between specified locations in mnemonic octal-hex, index word format or floating decimal on the printer and/or on tape. The dump format desired, and the limits, may be specified either by control cards or by the entry keys on the console.

Control Cards

If control cards are read from the reader, the dump program will execute dumps as specified in 9-left of each control card which has been punched in the format:

9L decrement	Starting address
9L address	Last location to be dumped
9L tag	Format of dump
tag = 0	Mnemonic, octal hex
tag = 2	Index word
tag = 4	Floating decimal

Control cards will be read and dump requests will be processed until two cards are encountered with identical information in 9L or an end-of-file occurs on the reader.

Output Formats

Every dump contains the following information:

1. The location counter plus the first 15 panel locations.
2. The seven simulated I-O channels.
3. The contents of all the index registers in index format.
4. The dumps requested, four words per line. These dumps may appear in:
   a. Octal-hex mnemonic format. A half-word is split up in the following fashion: the full word address in octal, a decimal point, the bit address in octal, a blank, bits 24-31 in two hex characters (0-9, A-F). Mnemonics may be modified (a maximum of six characters can be handled, so seven-character mnemonics are abbreviated) and are printed on a separate line below the half-word to which each refers.
   b. Floating decimal. All words are printed with exponent and fraction (fraction between 1. and 10.) in decimal with signs.
   c. Index word. All words are split into four fields: value, flags, count and refill. All fields are printed in octal.

Calling the Breakpoint Dump

A breakpoint dump may be requested by introducing a full word, via a patch card, in the format:

```
xxxxxx.xx 8H  yyyyyy.24 0H(H indicates that any hex character may be used to
                            specify an index register)
```

This statement will be interpreted as an invalid operation, as in the trace. Dumping will occur immediately in the format and between the limits specified by the effective addresses of the two half-words.

The three dump formats are available under the control of the first octal character of the second half-word: 0 gives octal-hex, 2 or 3 gives index word format, and 4, 5, 6, or 7 gives floating decimal. This character is then discarded before setting the dump limits. If an address greater than 8192 is given, 8192 is used with no error indication.

At assembly time, the appropriate illegal full-word operation may be entered by

```
SIC, A; BE, B(.21) 20
```

which will cause a dump between A and B to be executed. Other instructions may be compiled to accomplish the same end.

OPERATOR'S NOTES

The following pages are intended to be a reference for the operator using the simulation system. However, much of the information will be helpful to the programmer as well, and will guide the programmer in the preparation of decks to be run by the operator.

The material supplied to the user will include:

1. Master tape, containing LASL Strap-1 and the 7030 Simulator.
2. Binary decks of the following auxiliary programs:
   Write Disk
   Call Strap
   Column Binary Punch Simulator
   System Tape Editor

Upon receipt of the binary master tape and the auxiliary program decks the installation should:

1. Prepare an installation tape using Write Disk and System Tape Editor programs. The tape thus produced becomes the installation master.
2. Copy the installation master using the System Tape Editor again. The tape produced here should be used for the simulation work.
3. Return the original master tape to IBM.

Corrections to the simulation system (as issued by IBM) may be inserted in the System Tape Editor program and updated Installation Master and System tapes may be generated as changes are received by the user.

COPYING TAPE WITH THE SYSTEM TAPE EDITOR

The System Tape Editor program is used as a system tape copier or as a means of rewriting the 7030 Simulation System Tape while incorporating changes in the system on the new tape produced.

Making Installation Master

1. Place the master tape on unit 2(A).
2. Place a working tape on 1(A).
3. Place the system tape editor in the card reader. Press Load Cards.
4. The working tape is now the installation master. Remove the master tape on 2(A) and return to IBM. Make another copy of the installation master and use this for all simulation work.

Monthly Updating of Installation Master

1. Place the installation master on 2(A).
2. Place a working tape on 1(A).

3. Place the System Tape Editor in the card reader. Place any corrections to the simulator behind any previous corrections to the simulator and then place all simulator corrections behind the first transfer card. Place all corrections to Strap-1 behind any previous corrections to Strap-1 and then place all Strap-1 corrections behind the second transfer card.
4. Tape 1 (A) becomes the updated tape.

Stops

HPR 77776	Error in reading simulator from the master tape. Reload to try again. If error occurs repeatedly, contact IBM.
HPR 77775	(709 only) Error in writing the simulator on tape 1. Dial another tape to 1 and try again.
HPR 77774	Error in attempting to read Strap from the master tape. Reload to try again or contact IBM.
HPR 77773	(709 only) Error in writing tape 1. Dial another tape to 1 and try again.
HPR 77771	(704 only) Error in writing tape 1. Dial another tape to 1 and try again.
HPR 77770	Final stop. Tape 1 should be file-protected at this point.

ASSEMBLY USING STRAP-1

Strap-1 allows the programmer to assemble 7030 programs on the IBM 704, 709 and 7090. The minimum system required is 32K with five tapes available. (On the 709 this means two tapes on channel B and three on A.)

1. Mount the 7030 Simulation System tape on 1(A). Strap-1 is the third and fourth records on this tape.

2. Dial an intermediate tape to 2(A). An intermediate tape must always be available when using Strap-1.

3. Input
   a. Mount the symbolic input tape on 8(B). Tape 8(B) is always required when assembling on the 709.
   b. Card input: place the cards in the on-line card reader immediately behind the one-card Call-Strap program.

      On the 709, cards are read onto tape 8 until an end-of-file on the card reader is reached. Processing then proceeds as though the input were from tape.

      On the 704, cards are read onto tape 2 until an end-of-file on the reader is reached. Processing then proceeds as though input were from tape until an end card is reached. Then Strap returns to the card reader and attempts to read the next deck.

4. Output
   a. Dial a tape to 3(A). This will be the binary output tape for off-line punching, using the column binary attachment.
   b. Dial a tape to 9(B). This will be the output tape prepared for off-line printing under program control.

CHECK: System tape on 1(A), imtermediate tape on 2(A), binary card output tape on 3(A), symbolic input tape on 8(B), off-line printed output tape on 9(B), Call Strap in the on-line reader with input cards (if any) behind it.

5. Press Load Cards key.

6. If an error occurs, the sense lights tell what pass you are in when the stop occurred. If sense lights 2 and/or 3 are on, the stop occurred after completion of pass 1 and manual entry of TRA 130_8 permits processing of subsequent jobs.

7. When an end-of-file occurs on tape 8(B), Strap returns to the card reader to look for more cards. If there are no more cards the end-of-file condition (or, an immediate end-of-file on the reader if there is card input) causes Strap-1 to print on line "Strap-1 can't find any more assemblies. Press Start for more." At this point, more cards can be placed in the reader or a new tape 8(B) mounted. Then press Start to continue.

8. Write an end-of-file on tape 3(A), and on tape 9(B), if desired.

9. Tape 3(A) may now be used as input directly to the simulator. However, this practice should not normally be followed. If the column binary attachment is not available on the off-line punch, it can be simulated on-line with only 72 columns available. (See "Column Binary Punch Simulator.")

SIMULATING A 7030 PROGRAM

The 7030 Simulator requires a 32K system with at least one tape unit available.

1. Mount the 7030 Simulation System tape on 1(A). The first record on this tape is a short loader which loads the simulator. The second record is the entire simulator.

2. Input
   a. Tape, SSW3 down. Mount the column binary tape on 3(A) with an end-of-file after the last program. A straight binary board should be used when preparing the input off-line.
   b. Cards, SSW3 up. Place the cards in the on-line reader. The first card must be an origin, C, T, or N card. All P cards must be placed immediately before the branch card.
   c. The only types of cards accepted as input by the simulator loader are origin, flow, branch, N, T, C, and P cards.

3. Press the Load-Tape key to bring in the simulator and start the run.

4. No program may be loaded below $(40)_8$ or above $(17777)_8$. The simulator loader will load until a branch card is encountered, unless one of the following stops occurs. In all stops listed, the address shown is in octal as read from the address field of the storage register on the console.

HPR 2	End-of-file encountered on either card reader or tape 3 (depending on the setting of SSW 3). Press Start to read next record.

HPR 3	Tape check after two attempts to read a record from input tape. Press Start to continue loading this record.
HPR 4	Check-sum error in this record. Press Start to continue loading.
HPR 5	No origin punched in first correction card encountered. Pressing Start causes this card to be skipped.

Note that any of the following stops involving HPR 2xx means that an irreparable error has occurred in the present job; pressing Start will load the next job.

HPR 200	Illegal type of card (not origin, flow, P, C). Press Start to load next job.
HPR 201	ID mismatch. Press Start to load next job.
HPR 202	Sequence out of order. Press Start to get next job. A sequence error will often mean that the next card has a sequence number of 1. This can occur legitimately, as when a subroutine has been inserted in a problem program. A T card may be used to eliminate the stop. However, if a T card is not used, pressing Start will cause the subroutine, and not the next job, to be loaded. This procedure may be used to load several segments of a single job.
HPR 203	P card followed by a card other than a P or branch card. Press Start for next job.
HPR 204	Origin too high. Press Start for next job.
HPR 205	Origin too low. Press Start for next job.
HPR 206	Too many bits to skip or set to zero. Press Start to get next job.
HPR 207	Attempting to load above limit for the simulated system. Press Start to get next job.
HPR 210	First card in deck is a flow card. Press Start to load next job.

5. To simulate the Initial Program Load Mode:
   a. Put SSW 6 down.
   b. Enter the channel number in octal in the console keys.
   c. Press Load-Tape key.
   d. When stop at HPR 17 occurs:
      1) Press Enter MQ with machine on Manual (704 only).
      2) Reset keys. Set SSW 6 up.
      3) Press Start.

Note that with initial program loading from the card reader, the last one and one-half words will be loaded as zero.

   e. Stop at HPR 1. Machine cannot perform the initial program load function correctly. Check binary card and/or I-O unit.

6. When the simulator encounters a BEW, it will stop with HPR 77777_8 in the storage register and the contents of the instruction counter in the address field of the accumulator, offset by two bits.

Example 1:	At BEW , 2436.0_8	AC = 24360
Example 2:	At BEW , 2436.4_8	AC = 24364

If the problem program is caught in a BE,0 loop at location 20.0, the BEW stop will occur with 200_8 in the AC.

7. Dump
   a. Set sense switches:

SSW 1 up	Tape 9(B) output
SSW 1 down	Suppress tape output
SSW 2 up	Printer output
SSW 2 down	Suppress printer output
SSW 4 up	Load next program when Start is pressed after stop at HPR 30000.
SSW 4 down	Return to current program when Start is pressed after last dump request (HPR 30000).

   b. With the simulator in storage, press Load Tape.
   c. If a minus sign is entered in the MQ, the dump program will read control cards from the reader and execute dumps as specified in 9-left of each control card which has been punched in the format:

9L decrement	Starting address
9L address	Last location to be dumped
9L tag	Format of dump
tag = 0	Mnemonic, octal-hex
tag = 2	Index word
tag = 4	Floating decimal

Control cards will be read and dump requests will be processed until two cards are encountered with identical information in 9L or an end-of-file occurs on the reader. Note: A control card with a blank 9L causes a complete storage dump to be performed.

   d. If the MQ is plus, dump limits and format requests will be obtained from the entry keys on the console, again until two identical requests are made.
      At HPR 20000:
      1) 704: enter request on console keys, press Enter MQ, then Start.
      2) 709: enter request on console keys, press Start.
   e. When all dump requests have been processed (HPR 30000 in the storage register) or when a BEW stop occurs (HPR 77777 in the storage register), two paths are available when Start is pressed.
      1) SSW 4 up: press Start to load next program.
      2) SSW 4 down: press Start to re-enter the program at the location contained in the instruction counter.

When taking breakpoint dumps, the program is always re-entered when Start is pressed, regardless of the setting of SSW 4.

   f. Breakpoint dumps are also available to the programmer. (See "Programmer Notes" for description and output formats.)
   g. Stops

HPR 10001	SSW 1 and SSW 2 are set to suppress all output for the dump or trace program. Reset SSW 1 and 2 and press Start.

HPR 10002	End-of-tape sensed on output tape. Dial a new tape 9(B) and press Start.
HPR 20000	Enter first dump request--or minus sign to read cards. On 704, press Enter MQ, then Start, to enter dump request from console keys. On 709, reset console keys for next request, then press Start.
HPR 30000	If SSW 4 is up and dump request is the same as last one, press Start to load next program.

Note: A control card with a blank 9L or a zero in the console keys (press Start on the 704 only) causes a complete storage dump to be performed.

8. Trace
   a. The trace program, like the dump program, is under sense switch control. Normally, the trace will be used on-line to determine the occurrence of loops in a problem program.

SSW 1 up	Tape 9(B) output
SSW 1 down	Suppress tape output
SSW 2 up	Printer output
SSW 2 down	Suppress printer
SSW 5 up	No tracing
SSW 5 down	Trace all instructions

   b. Breakpoint tracing is also available. (See "Programmer's Notes" for description and output formats.) Breakpoint tracing is independent of the setting of SSW 5.

9. Output-trace, dump, and N card output are all under the control of the same two sense switches:

SSW 1 down	Suppress tape 9 output
SSW 2 down	Suppress printer output
SSW 1, 2 up	Normal output
SSW 1,2 down	Program stop. All output is suppressed.

Off-line output is printed using program control.
On-line output uses the SHARE 2 board in the printer.

10. Write Disk Program
    a. The Write Disk program is used to prepare a simulated disk on tape 10(A).
    b. Mount a rewound tape on 10(A).
    c. Place the Write Disk program in the card reader and ready the reader.
    d. Press Clear, then Load Cards.
    e. A stop at HPR 0, 7 is the final stop.

11. Roll-back. A roll-back feature allows interruption and savings in the running of the simulated program; later the program is reset and the simulation continues from the point at which saving occurred.
    a. Saving: Enter a 3 in the high-order octal digit in the console keys; depress SSW 6; at stop reset SSW 6; press Enter MQ (704 only); press Start. The 7030 contents will be saved on tape 2(A).

b. Restoring: Enter a 7 in the high-order octal digit of the console keys; depress SSW 6; at the stop, reset SSW 6; press Enter MQ (704 only); press Start. The 7030 program will be read in from tape 2(A) and simulation will proceed automatically.

c. Stops

HPR 50000	Error in tape reading in roll-back. Press Start to backspace and try again.
HPR 50001	Error in tape writing in roll-back (709 only). Press Start to backspace and try again.

12. To stop the simulation at any time, depress SSW 6. The user is then assured that when the simulator stops, it will have completed the simulation of the instruction being processed when SSW 6 was depressed.

13. The Column Binary Punch Simulator is used to obtain Strap output from the on-line punch at those installations not equipped with a column-binary attachment for the off-line punch.

a. At the conclusion of the Strap assembly, write an end-of-file on tape 3(A).

b. Load the column binary punch simulator in the card reader.

c. Press Clear, then Load Cards.

d. Strap column-binary cards will be punched on-line until an end-of-file is encountered on tape 3(A).

Note: Only columns 1-72 will be punched. This means that the ID columns 73-80 will be lost on normal Strap cards and one and one-half words will be lost from PUNFUL cards.

e. Stops

HPR 77767	Redundancy error on tape 3(A). Press Start to try again.
HPR 77766	End-of-file encountered on tape 3(A).

SUMMARY OF SENSE SWITCH SETTINGS

SSW 1 up	(Dump) Tape 9(B) output.
SSW 1 down	(Dump) Suppress tape output.
SSW 2 up	(Dump) Printer output.
SSW 2 down	(Dump) Suppress printer output.
SSW 3 up	(Loader) Load 7030 binary cards, corrections cards and patch cards from the on-line reader.
SSW 3 down	(Loader) Loads the same as above, but from tape 3(A).
SSW 4 up	(Dump) Load next program after the last dump request or BEW.
SSW 4 down	(Dump) Return to current program after last dump request or BEW.
SSW 5 up	(Trace) No tracing.
SSW 5 down	(Trace) Trace all instructions.
SSW 6 up	(Simulator) Normal mode.
SSW 6 down	(Simulator) Initial program load mode.

SUMMARY OF STOPS

Loader

HPR 2	End-of-file encountered on either card reader or tape 3 (depending on the setting of SSW 3). Press Start to read next record.
HPR 3	Tape check after two attempts to read a record from input tape. Press Start to continue loading this record.
HPR 4	Check sum error in this record. Press Start to continue loading.
HPR 5	No origin punched in the first correction card encountered. Pressing Start causes this card to be skipped.
HPR 200	Illegal type of card (not origin, flow, P, C). Press Start to load next job.
HPR 201	ID mismatch. Press Start to load next job.
HPR 202	Sequence out of order. Press Start to get next job.
HPR 203	P card followed by a card other than a P or Branch card. Press Start to get next job.
HPR 204	Origin too high. Press Start to get next job.
HPR 205	Origin too low. Press Start to get next job.
HPR 206	Too many bits to skip or set to zero. Press Start to get next job.
HPR 207	Attempting to load above limit for simulated machine. Press Start to get next job.
HPR 210	First card in deck is flow card. Press Start to load next job.

7030 Simulator

HPR 77777	A BEW instruction has been encountered. Press Start to continue.
HPR 17	Initial Program Load. Press Enter MQ (704 only) to enter channel number, reset keys, and SSW 6, press Start.
HPR 1	Machine cannot perform the initial program load function correctly; check binary card.
HPR 50000	Roll-back. Error in tape reading in program; press Start to backspace and try again.
HPR 50001	Roll-back. Error in tape writing (709 only); press Start to backspace and try again.

Dump

HPR 10001	SSW 1 and 2 are set to suppress all output for the dump or trace program. Reset SSW 1 and 2 and press Start.
HPR 10002	End-of-tape sensed on output tape. Dial a new tape 9(B) and press Start.
HPR 20000	Enter first dump request in console keys.
HPR 30000	Enter new dump request. If same as last request, and if SSW 4 is up, press Start to load next program.

Write Disk

HPR 0, 7	Final stop.

Column Binary Punch Simulator

HPR 77767	Redundancy error on tape 3(A). Press Start to try again.
HPR 77766	End-of-file encountered on tape 3(A).

System Tape Editor

HPR 77776	Error in reading simulator from the master tape. Reload to try again. If error occurs repeatedly, contact IBM.
HPR 77775	(709 only) Error in writing the simulator on tape 1. Dial another tape to 1 and try again.
HPR 77774	Error in attempting to read Strap from master tape. Reload to try again or contact IBM.
HPR 77773	(709 only) Error in writing tape 1. Dial another tape to 1 and try again.
HPR 77771	(704 only) Error in writing tape 1. Dial another tape to 1 and try again.
HPR 77770	Final stop. Tape 1 should be file-protected at this point.

APPENDIX A

STRAP-1 MNEMONICS

Assigned Strap-1 mnemonics, including both operation codes and system symbols, are listed on the following pages. The numbers in the Footnote column designate notes that follow the listing. These footnotes, in general, identify a particular class of operations that may be expanded in a standard way to produce other operations. Where footnotes specify how particular modified operation mnemonics may be constructed, these mnemonics do not appear explicitly in the listings.

The following abbreviations, used in the Type column, identify the symbolic instruction type.

V	VFL
F	Floating Point
$	System Symbol
I	Index
C	Count and Branch
M	Branches and Miscellaneous
B	Branch on Bit
T	Transmits
E	I-O Select or Control Word

Type	Mnemonic	Footnote	Name	Word No.	Bit Address
$	AD	2	Address Invalid	11	16
$	AE	2	Accumulator Equal	11	61
$	AH	2	Accumulator High	11	62
$	AL	2	Accumulator Low	11	60
$	AOC	1	All Ones Count	7	44-50
$	BC	1	Boundary Control	3	57
$	BTR	2	Binary Transit	11	39
$	CA	1	Channel Address	5	12-18
$	CBJ	2	Channel Busy Reject	11	8
$	CNSL	1	Console		
$	CPUS	1	CPU Signal	11	5
$	CPU	2	Other CPU	6	0-18
$	CS	2	Channel Signal	11	13
$	DF	2	Data Fetch	11	20
$	DISK	1	Disk		
$	DS	2	Data Store	11	19
$	DTR	2	Decimal Transit	11	40
$	E	12	e		
$	EE	2	End Exception	11	11
$	EK	2	Exchange Control Check	11	3
$	EKJ	2	Exchange Check Reject	11	6
$	EOP	2	End of Operation	11	12
$	EPGK	2	Exchange Program Check	11	9
$	EXE	2	Execute Exception	11	18
$	FT	1	Factor	14	0-63
$	IA	1	Interruption Address	2	0-17
$	IF	2	Instruction Fetch	11	21
$	IK	2	Instruction Check	11	1
$	IJ	2	Instruction Reject	11	2
$	IND	1	Indicators	11	0-63
$	IQS	1	Inquiry Station		
$	IR	2	Imaginary Root	11	25
$	IT	1	Interval Timer	1	0-18
$	L	1	Left Half of Accumulator	8	0-63
$	LB	1	Lower Boundary	3	32-49
$	LC	2	Lost Carry	11	22
$	LS	2	Lost Significance	11	26
$	LZC	1	Left Zeros Count	7	17-23
$	M	12	$\mathrm{Log}_{10}e$		
$	MASK	1	Mask	12	21-49
$	MB	1	Maintenance Bits	4	0-63
$	MK	2	Machine Check	11	0
$	MOP	2	To-Memory Operation	11	55
$	N	12	$\mathrm{Log}_e 2$		
$	NM	2	Noisy Mode	11	63
$	OP	2	Operation Invalid	11	15
$	PCH	1	Punch		
$	PF	2	Partial Field	11	23
$	PGO...PG6	2	Program Indicators	11	41-47
$	PI	12	π		
$	PRT	1	Printer		
$	PSH	2	Preparatory Shift Greater Than 48	11	27
$	R	1	Right Half of Accumulator	9	0-63
$	RDR	1	Reader		
$	RGZ	2	Result Greater Than Zero	11	58
$	RLZ	2	Result Less Than Zero	11	56
$	RM	1	Remainder	13	0-63
$	RN	2	Result Negative	11	59
$	RU	2	Remainder Underflow	11	34
$	RZ	2	Result Zero	11	57
$	SB	1	Sign Byte	10	0-7
$	TC	1	Time Clock	1	28-63
$	TCI...TCK		Tape Channels 1...K		
$	TF	2	T Flag	11	35
$	TR	1	Transit	15	0-63
$	TS	2	Time Signal	11	4
$	TX	1	Tape X (X is a numerical designation)		
$	UB	1	Upper Boundary	3	0-17
$	UF	2	U Flag	11	36
$	UK	2	Unit Check	11	10
$	UNRJ	2	Unit Not Ready Reject	11	7
$	USA	2	Unended Sequence of Addresses	11	17
$	VF	2	V Flag	11	37
$	X0	1	Index Zero	16	0-63
$	X1	1	Index One	17	0-63
$	X2	1	Index Two	18	0-63
$	X3	1	Index Three	19	0-63
$	X4	1	Index Four	20	0-63
$	X5	1	Index Five	21	0-63
$	X6	1	Index Six	22	0-63
$	X7	1	Index Seven	23	0-63
$	X8	1	Index Eight	24	0-63
$	X9	1	Index Nine	25	0-63
$	X10	1	Index Ten	26	0-63
$	X11	1	Index Eleven	27	0-63
$	X12	1	Index Twelve	28	0-63
$	X13	1	Index Thirteen	29	0-63
$	X14	1	Index Fourteen	30	0-63
$	X15	1	Index Fifteen	31	0-63
$	XCZ	2	Index Count Zero	11	48
$	XE	2	Index Equal	11	53
$	XF	2	Index Flag	11	38
$	XH	2	Index High	11	54
$	XL	2	Index Low	11	52
$	XPFN	2	Exponent Flag Negative	11	33
$	XPFP	2	Exponent Flag Positive	11	28
$	XPH	2	Exponent Range High	11	30
$	XPL	2	Exponent Range Low	11	31
$	XPO	2	Exponent Overflow	11	29
$	XPU	2	Exponent Underflow	11	32
$	XVGZ	2	Index Value Greater Than Zero	11	51
$	XVLZ	2	Index Value Less Than Zero	11	49
$	XVZ	2	Index Value Zero	11	50
$	Z	1	Word Number Zero	0	0-63
$	ZD	2	Zero Divisor	11	24

ALPHABETIC LIST OF OPERATIONS

Type	Mnemonic	Footnote	Name
V	+	3	Add
F	+	6	Add
V	+MG	3	Add to Magnitude
F	+MG	6	Add to Magnitude
V	-	3	Subtract
F	-	6	Subtract
V	-MG	3	Subtract from Magnitude
F	-MG	6	Subtract from Magnitude
V	*	4	Multiply
F	*	7	Multiply
V	*+		Multiply and Add
F	*+		Multiply and Add
F	*A +		Multiply Absolute and Add
V	*I +		Multiply Immediate and Add
V	*N +		Multiply Negative and Add
F	*N +		Multiply Negative and Add
F	*NA +		Multiply Negative Absolute and Add
V	*NI +		Multiply Negative Immediate and Add
V	/	4	Divide
F	/	7	Divide
M	B		Branch
B	BB		Branch on Bit
B	BB1		Branch on Bit and Set to One
B	BBN		Branch on Bit and Negate
B	BBZ		Branch on Bit and Zero
M	BD		Branch Disabled
M	BE		Branch Enabled
M	BEW		Branch Enabled and Wait
M	BR		Branch Relative
B	BZB		Branch on Zero Bit
B	BZB1		Branch on Zero Bit and Set to One
B	BZBN		Branch on Zero Bit and Negate
B	BZBZ		Branch on Zero Bit and Zero
V	C	10	Connect
I	C + 1		Add Immediate to Count
I	C - 1		Subtract Immediate from Count
C	CB	8	Count and Branch
C	CBR	8	Count, Branch, and Refill
C	CBZ	8	Count and Branch on Zero Count
C	CBZR	8	Count, Branch on Zero Count, and Refill
E	CCW		Copy Control Word
V	CM	10	Connect to Memory
V	CT	10	Connect for Test
E	CTL		Control
V	CV	5	Convert
F	D +	6	Add Double
F	D + MG	6	Add Double to Magnitude
F	D -	6	Subtract Double
F	D - MG	6	Subtract Double from Magnitude
V	DCV	5	Convert Double
F	DL	7	Load Double
F	DLWF	7	Load Double with Flag
F	D*	7	Multiply Double
F	D/	7	Divide Double
F	E +	6	Add to Exponent
F	E + AI		Add Absolute Immediate to Exponent
F	E + I		Add Immediate to Exponent
F	E -	6	Subtract From Exponent
F	E - AI		Subtract Absolute Immediate from Exponent
F	E - I		Subtract Immediate from Exponent
M	EX		Execute
M	EXIC		Execute Indirect and Count
F	F +	6	Add to Fraction
F	F -	6	Subtract from Fraction
V	K	4	Compare
F	K	7	Compare
I	KC		Compare Count
I	KCI		Compare Count Immediate
V	KE	4	Compare If Equal
V	KF	4	Compare Field
V	KFE	4	Compare Field If Equal
V	KFR	4	Compare Field for Range
E	KLN		Check Light On
F	KMG	7	Compare Magnitude
F	KMGR	7	Compare Magnitude for Range
V	KR	4	Compare for Range
F	KR	7	Compare for Range
I	KV		Compare Value
I	KVI		Compare Value Immediate
I	KVNI		Compare Value Negative Immediate
V	L	4	Load
F	L	7	Load
I	LC		Load Count
I	LCI		Load Count Immediate
V	LCV	4	Load Converted
V	LF		Load Field
V	LFT	4	Load Factor
F	LFT	7	Load Factor
E	LOC		Locate (same as Select Unit)
I	LR		Load Refill
I	LRI		Load Refill Immediate
I	LV		Load Value
I	LVE		Load Value Effective
I	LVI		Load Value Immediate
I	LVNI		Load Value Negative Immediate
I	LVS		Load Value with Sum
I	LX		Load Index
V	LTRCV	4	Load Transit Converted
V	LTRS	4	Load Transit and Set
V	LWF	4	Load with Flag
F	LWF	7	Load with Flag
V	M +	3	Add to Memory
F	M +	6	Add to Memory
V	M + 1		Add One to Memory
F	M + A		Add to Absolute Memory
V	M + MG	3	Add Magnitude to Memory
F	M + MG	6	Add Magnitude to Memory
V	M -		Subtract from Memory
F	M -		Subtract from Memory
V	M -1		Subtract One from Memory
F	M -A		Subtract from Absolute Memory
V	M -MG	3	Subtract Magnitude from Memory
F	M -MG	6	Subtract Magnitude from Memory
M	NOP		No Operation
M	R		Refill
M	RCZ		Refill on Count Zero
E	RD		Read
E	REL		Release
E	REW		Rewind
I	RNX		Rename
F	R/		Reciprocal Divide
I	SC		Store Count
E	SEOP	11	Suppress End of Operation
V	SF		Store Field
F	SHF	7	Shift Fraction
F	SHFL		Shift Fraction Left (same as SHFA)
F	SHFR		Shift Fraction Right (same as SHFNA)
M	SIC		Store Instruction Counter If
F	SLO	7	Store Low Order
F	SNRT	6	Store Negative Root
I	SR		Store Refill
V	SRD	5	Store Rounded
F	SRD	7	Store Rounded

Type	Mne-monic	Foot-note	Name
F	SRT	6	Store Root
V	ST	5	Store
F	ST	7	Store
E	SU		Select Unit (same as Locate)
I	SV		Store Value
I	SVA		Store Value in Address
T	SWAP		Swap
T	SWAPI		Swap Immediate
T	SWAPB		Swap Backward
T	SWAPBI		Swap Backward Immediate
I	SX		Store Index
T	T		Transmit
T	TI		Transmit Immediate
T	TB		Transmit Backward
T	TBI		Transmit Backward Immediate
I	V +		Add to Value
I	V + I	9	Add Immediate to Value
I	V + C		Add to Value and Count
I	V + CR		Add to Value, Count, and Refill
I	V + IC	9	Add Immediate to Value and Count
I	V + ICR	9	Add Immediate to Value, Count, and Refill
I	V - I	9	Subtract Immediate from Value
I	V - IC	9	Subtract Immediate from Value and Count
I	V - ICR	9	Subtract Immediate From Value, Count, and Refill
E	W		Write
E	WEF		Write End-of-File
M	Z		Store Zero

FOOTNOTES

1. This mnemonic is a system symbol. It must be prefixed by the character "$" whenever used.

2. This mnemonic is both an indicator mnemonic and a system symbol. It must be prefixed by the "$" whenever it is used as a system symbol in a symbolic field of some instruction. This mnemonic may also be used directly to express a Branch on Indicator instruction by being substituted for the letter "I" in any of the following four formats:

BI	Branch on Indicator
BIZ	Branch on Indicator and Zero
BZI	Branch on Zero Indicator
BZIZ	Branch on Zero Indicator and Zero

The mnemonics BI, BIZ, BZI, BZIZ are not in themselves legal operation codes. Any of the integers 0 through 63 may also be substituted for I if it is desired to designate an indicator numerically.

3. This operation code may be suffixed by the letter "I" to invoke immediate addressing.

4. This VFL operation code may have the following suffixes:

I	Immediate
N	Negative
NI	Negative Immediate

5. This operation code may be suffixed by the letter "N" to invoke the negative sign modifier.

6. This floating point operation code may be suffixed by the letter "A" to invoke the absolute sign modifier.

7. This floating point operation code may have the following suffixes:

N	Negative
A	Absolute
NA	Negative Absolute

8. Count and Branch operation may have the following suffixes:

+	Add one to value
-	Subtract one from value
H	Add half to value

9. This operation code may be used to indicate either an immediate indexing operation or the secondary operation of any VFL instruction.

10. This operation mnemonic specifies, potentially, 16 connect instructions. Four binary digits are written directly after the operation code to select a particular one of the 16 instructions. This operation code is also subject to Footnote 3.

11. This code may be used as a secondary operation with I-O select orders that are subject to end-of-operation interrupts.

12. These mnemonics are mathematical constants.

APPENDIX B

STRAP-1 PSEUDO-OPERATIONS

Mnemonic	Name
BS	Backspace
CCR	Chain Counts Within Record
CD	Count Disregarding Record
CDSC	Count Disregarding Record, Skip, and Chain
CF	Count Field
CNOP	Conditional No Operation
CR	Count Within Record
CRDRUN	Card Run-Out
CW	Control Word
DD	Data Definition
DDI	Data Definition Immediate
DR	Data Reservation
DRZ	Data Reservation and Set to Zero
ECC or ODDECC	ECC(and odd parity for tape)
END	End
ERG	Erase Gap
EVEN	Even Parity No ECC(tape only)
EXT	Extract
GONG	Sound Gong
HD	High Density
KLN	Check Light On
LD	Low Density
NOECC	No ECC, Even Parity (tape only)
ODDECC	Odd Parity, ECC
ODDNEC	Odd Parity, No ECC
PRND	Print Double-spaced

Mnemonic	Name
PRNID	Print ID
PRNS	Print Single-spaced
PUNFUL	Punch Full Cards
PUNID	Punch ID
PUNNOR	Punch Normally
REM	Resume Error Marks
REW	Rewind
RF	Refill Field
RLF	Reserved Light Off
RLN	Reserved Light On
SCCR	Skip, Chain Counts Within Record
SCR	Skip, Count Within Record
SCD	Skip, Count Disregarding Record
SCDSC	Skip, Count Disregarding Record, Skip and Chain
SEM	Suppress Error Marks
SKIP	Skip Paper
SLC	Set Location Counter
SP	Space
SPFL	Space File
SYN	Synonym
TAIL	Tail
TILF	Tape Indicator Light Off
TLB	Terminate Loading and Branch
UNLOAD	Unload
VF	Value Field
WEF	Write End-of-File
XW	Index Word

APPENDIX C

SYMBOLIC DESCRIPTIONS AND MNEMONICS FOR IBM 7030

The following list of mnemonics may be used with Strap-1 and Strap-2. A symbolic description of the mnemonic is given to assist the programmer. The operations symbols used are defined at the start of each section. Note that the same letter ("a" and "m" for example) has a different definition for floating point and for VFL. Carefully read the definition for each set. A more detailed description of the operation is in the IBM 7030 Reference Manual, Form A22-6530.

A specific title for each mnemonic is not given in cases where the mnemonic is derived from the basic operation by changing the sign and absolute modifiers.

In the case of VFL operations, the unsigned modifier must be implied by the data referred to or be explicitly stated in a dds.

FLOATING POINT OPERATIONS

Notation for Symbolizing the Floating Point Operations OP(dds), $A_{18}(I)$

Accumulator Operands

a	= bits (0-59) of the accumulator, and the accumulator sign, bit 4 of the sign byte register.
b	= bits (60-107) of the accumulator, and the accumulator sign.
ab	= bits (0-107) of the accumulator, and the accumulator sign.
e(a)	= bits (0-11) of a.
f(a)	= bits (12-59) of a, and s(a).
s(a)	= bit 4 of the sign byte register.
SB(a)	= bits 4-7 of the sign byte register.
Fl (a)	= bits 5-7 of the sign byte register.

R - Result

Storage Operands

m	= bits (0-59) of the storage word, and its sign, bit 60.
M	= L(m) = the effective address.
e(m)	= bits (0-11) of m.
f(m)	= bits (12-59) of m, and s(m).
s(m)	= bit 60 of the storage word.
SB(m)	= bits (60-63) of the storage word.
Fl (m)	= bits (61-63) of the storage word.

\$FT = Factor operand; SB(\$FT) = bits (60-63) of \$FT.
\$RM = Remainder operand.

Add

+	$a+m \longrightarrow a$	1. ~~0 → (60-63)~~b:
-	$a-m \longrightarrow a$	2. ~~The rest of~~ b is unchanged.
+ A	$a+\|m\| \longrightarrow a$	3. Fl (a) is unchanged.
- A	$a-\|m\| \longrightarrow a$	

Add to Memory

M+	$m+a \longrightarrow m$	1. Fl (m) remain unchanged.
M-	$m-a \longrightarrow m$	2. The entire accumulator and SB(a) remain unchanged.
M+ A	$\|m\| +a \longrightarrow m$	
M- A	$\|m\| -a \longrightarrow m$	

Add to Fraction

F+	$f(ab)+f(m) \longrightarrow f(ab)$	1. e(m) is ignored; the add is performed with e(a) on both operands.
F-	$f(ab)-f(m) \longrightarrow f(ab)$	2. The normalized mode operates in the same way as in D+.
F+ A	$f(ab)+\|f(m)\| \longrightarrow f(ab)$	
F - A	$f(ab)-\|f(m)\| \longrightarrow f(ab)$	

Add to Exponent

E+	$e(ab)+e(m) \longrightarrow e(ab)$	1. f(m) is ignored.
E-	$e(ab)-e(m) \longrightarrow e(ab)$	2. Strap-1 will assemble as unnormalized unless the normalized mode is requested by referring to normalized data or by using the dds = (N).
E+ A	$e(ab)+\|e(m)\| \longrightarrow e(ab)$	
E-A	$e(ab)-\|e(m)\| \longrightarrow e(ab)$	

Add Immediate to Exponent

E+I	$e(ab)+e(M) \longrightarrow e(ab)$	1. The unnormalized mode is given unless overruled by dds = (N).
E -I	$e(ab)-e(M) \longrightarrow e(ab)$	
E+ AI	$e(ab)+\|e(M)\| \longrightarrow e(ab)$	
E -AI	$e(ab)-\|e(M)\| \longrightarrow e(ab)$	

Shift Fraction

SHF	$f(ab)\cdot 2^{M} \longrightarrow f(ab)$	1. Left shift if bit 11 of M = 0.
SHFN	$f(ab)\cdot 2^{-M} \longrightarrow f(ab)$	2. Right shift if bit 11 of M = 1.
SHFA	$f(ab)\cdot 2^{\|M\|} \longrightarrow f(ab)$	3. The operation is not affected by the normalized modifier.
SHFNA	$f(ab)\cdot 2^{-\|M\|} \longrightarrow f(ab)$	4. The exponent is not adjusted for the shift. e(a) is unchanged.
SHFL	$f(ab)\cdot 2^{\|M\|} \longrightarrow f(ab)$	5. On a right shift, zeroes are introduced in bit 12.
SHFR	$f(ab)\cdot 2^{-\|M\|} \longrightarrow f(ab)$	

Double Add

D+	$ab+m \longrightarrow ab$	1. PSH indicator goes on if the exponent difference exceeds 48.
D-	$ab-m \longrightarrow ab$	
D+ A	$ab+\|m\| \longrightarrow ab$	
D- A	$ab-\|m\| \longrightarrow ab$	

Add to Magnitude

+ MG	$R=\|a\|+m$	1. $R \longrightarrow a$ if $R \geq -0$.
- MG	$R=\|a\|-m$	2. $0 \longrightarrow f(a)$ if $R < 0$ and e(a) is unchanged.
+ MGA	$R=\|a\|+\|m\|$	3. s(a) is unchanged in either case.
- MGA	$R=\|a\|-\|m\|$	

Double Add to Magnitude

D+ MG	$R=\|ab\|+m$	1. $R \longrightarrow ab$ if $R \geq +0$.
D- MG	$R=\|ab\|-m$	2. $0 \longrightarrow f(ab)$ if $R < 0$ and e(a) is unchanged.
D+ MGA	$R=\|ab\|+\|m\|$	3. s(a) is unchanged in either case.
D- MGA	$R=\|ab\|-\|m\|$	

Add Magnitude to Memory

M+ MG	$R=m+\|a\|$	1. $R \longrightarrow m$ if s(R)=s(m).
M- MG	$R=m-\|a\|$	2. $0 \longrightarrow f(m)$ if $s(R) \neq s(m)$.
M+ MGA	$R=\|m\|+\|a\|$	3. s(m) is unchanged in either case.
M- MGA	$R=\|m\|-\|a\|$	

Multiply

*	$a\cdot m \longrightarrow a$	1. ~~0 → (60-63)b~~.
*N	$a\cdot -m \longrightarrow a$	2. ~~The rest of~~ b is unchanged.
*A	$a\cdot \|m\| \longrightarrow a$	
*NA	$a\cdot -\|m\| \longrightarrow a$	

Double Multiply

D*	$a\cdot m \longrightarrow ab$	1. (108-127) of accumulator are unchanged.
D*N	$a\cdot -m \longrightarrow ab$	
D*A	$a\cdot \|m\| \longrightarrow ab$	
D*NA	$a\cdot -\|m\| \longrightarrow ab$	

Multiply Factor and Add

*+	m · ($FT)+ ab $\longrightarrow$ ab	1. The contents of $FT remain unchanged.
*N+	-m · ($FT)+ ab $\longrightarrow$ ab	[illegible]
*A+	|m| · ($FT)+ ab $\longrightarrow$ ab	
*NA+	-|m| · ($FT)+ ab $\longrightarrow$ ab	

Divide

/	a/m $\longrightarrow$ a	1. No remainder is generated.
/N	a/-m $\longrightarrow$ a	2. Quotient ~~rounded~~ to 48 bits. [illegible]
/A	a/|m| $\longrightarrow$ a	3. Pre-normalization of the operands is ~~independent~~ of the normalization modifier.
/NA	a/-|m| $\longrightarrow$ a	4. ~~0 $\longrightarrow$ (60-63)b; the rest of~~ b is unchanged.

Reciprocal Divide

R/	m/a $\longrightarrow$ a	1. Performed similarly to divide.
R/N	-m/a $\longrightarrow$ a	2. ~~0 $\longrightarrow$ (60-63)b; the rest of~~ b is unchanged.
R/A	|m| /a $\longrightarrow$ a	
R/NA	-|m| /a $\longrightarrow$ a	

Double Divide

D/	ab/m $\longrightarrow$ ab	1. Remainder in $RM.
D/N	ab/-m $\longrightarrow$ ab	2. 0 $\longrightarrow$ b.
D/A	ab/ |m| $\longrightarrow$ ab	3. No rounding.
D/NA	ab/-|m| $\longrightarrow$ ab	4. ~~SB(a)~~ $\longrightarrow$ SB($RM). $\longrightarrow$ 0 S(a) $\longrightarrow$ [illegible]

Store Root

SRT	$\|\sqrt{a}\|$ $\longrightarrow$ m	1. ab and SB(a) are unchanged.
SNRT	$-\sqrt{a}$ $\longrightarrow$ m	2. SB(a) $\longrightarrow$ SB(m)
SRTA	$\sqrt{\|a\|}$ $\longrightarrow$ m	
SNRTA	$-\sqrt{\|a\|}$ $\longrightarrow$ m	

Load

L	m $\longrightarrow$ a	1. 0 $\longrightarrow$ Fl (a).
LN	-m $\longrightarrow$ a	~~2. 0 $\longrightarrow$ (60-63)b.~~
LA	|m| $\longrightarrow$ a	3. ~~The rest of~~ b is unchanged.
LNA	-|m| $\longrightarrow$ a	

Double Load

DL	m $\longrightarrow$ a	1. 0 $\longrightarrow$ b.
DLN	-m $\longrightarrow$ a	2. 0 $\longrightarrow$ Fl (a).
DLA	|m| $\longrightarrow$ a	
DLNA	-|m| $\longrightarrow$ a	

Load with Flag Bits

LWF	m $\longrightarrow$ a	1. Fl (m) $\longrightarrow$ Fl (a).
LWFN	-m $\longrightarrow$ a	~~2. 0 $\longrightarrow$ (60-63)b.~~
LWFA	|m| $\longrightarrow$ a	3. b is [illegible]
LWFNA	-|m| $\longrightarrow$ a	

Double Load with Flag Bits

DLWF	m $\longrightarrow$ a	1. 0 $\longrightarrow$ b.
DLWFN	-m $\longrightarrow$ a	2. Fl (m) $\longrightarrow$ Fl (a).
DLWFA	|m| $\longrightarrow$ a	
DLWFNA	-|m| $\longrightarrow$ a	

Load Factor

LFT	m $\longrightarrow$ $FT	1. ab and SB(a) are not changed.
LFTN	-m $\longrightarrow$ $FT	2. s(m) $\longrightarrow$ (60)$FT.
LFTA	|m| $\longrightarrow$ $FT	3. 0 $\longrightarrow$(61-63)$FT.
LFTNA	-|m| $\longrightarrow$ $FT	

Store

ST	a $\longrightarrow$ m	1. Fl (a) $\longrightarrow$ Fl (m).
STN	-a $\longrightarrow$ m	2. a is unchanged.
STA	|a| $\longrightarrow$ m	[illegible]
STNA	-|a| $\longrightarrow$ m	

Store Rounded

SRD	a $\longrightarrow$ m	1. A one is added in bit (60)b prior to the store: a and (60)b are unchanged.
SRDN	-a $\longrightarrow$ m	2. Fl (a) $\longrightarrow$ Fl (m).
SRDA	|a| $\longrightarrow$ m	
SRDNA	-|a| $\longrightarrow$ m	

Store Low Order

SLO	b $\longrightarrow$ f(m)	1. e(a) - 48 $\longrightarrow$ e(m).
SLON	-b $\longrightarrow$ f(m)	2. Fl (a) $\longrightarrow$ Fl (m).
SLOA	|b| $\longrightarrow$ f(m)	3. e(a) is unchanged.
SLONA	-|b| $\longrightarrow$ f(m)	

Compare

K	a:m	1. Indicators AL, AE, and AH are set as follows: AL is set to one if a < m; AE is set to one if a = m; AH is set to one if a > m
KN	a:-m	2. Zero exponents of different sign are considered equal.
KA	a:|m|	3. If the exponent difference is 48 the larger of the numbers is per sign and exponents regardless of fractions.
KNA	a:- |m|	

Compare for Range

KR	a:m	1. If AH is off prior to this op, no indicators will be changed.
KRN	a:-m	2. If AH is on: AL is unchanged. AE is set to one if a < m. AH is set to one if a ≥ m.
KRA	a:|m|	
KRNA	a:-|m|	

Compare Magnitude

KMG	|a|:m	1. Same as Compare, except for accumulator comparand.
KMGN	|a|:-m	
KMGA	|a|:|m|	
KMGNA	|a|:-|m|	

Compare Magnitude for Range

KMGR	a :m	1. Same as Compare for Range, except for accumulator comparand.
KMGRN	a :-m	
KMGRA	a :|m|	
KMGRNA	a :-|m|	

VARIABLE FIELD LENGTH OPERATIONS

Notation for Symbolizing the Variable Field Length Operations OP(dds), A_{24}(I), OF_7(I')

Accumulator Operands

a = the accumulator operand whose:

1. Low order bit is defined by the offset;
2. Byte size is four for decimal arithmetic, eight for binary arithmetic;
3. Length includes all bits in the accumulator to the left of the offset;

4. Sign is indicated by bit four of the sign byte register.

$\bar{a}$ = the accumulator operand, a, but without sign.

a_{20} = the accumulator operand, a, with offset = 20.

Fl(a) - bits 5-7 of S.B. Reg.
Z(a) - bits 1-3 of S.B. Reg.

Storage Operands

m = the storage operand whose:
1. High-order bit is defined by the bit address;
2. Byte size may be any number from one to eight, but is assumed to be four in the instruction lists below; (MUST BE 8 FOR BINARY)
3. Length is defined by the field length in the dds;
4. Sign is bit s in the sign byte.

$\bar{m}$ = the storage operand in which all bytes are processed as data; a positive sign is assumed.

The unsigned storage operand is designated by the dds.
Bits 7.17 and 7.18 are the leftmost two bits of $LZC.
$FT = Factor Operand; s($FT) = bit 60; FL($FT) = bits (61-63).
$TR = 64-bit Transit Register.

Integer Operations

Operations which can have an immediate operand are followed by (I), except for *+.

Add

+	$a+m \longrightarrow a$	(I)
-	$a-m \longrightarrow a$	

1. If the sign changes, bits to the right of the offset are complemented.
2. Fl(a) & Z(a) unchanged

Add To Memory

M+	$m+a \longrightarrow m$	
M-	$m-a \longrightarrow m$	

Add to Magnitude

+MG	$R=\bar{a}+m$	(I)
-MG	$R=\bar{a}-m$	

1. $R \rightarrow \bar{a}$ if $R \geq 0$.
2. 0→entire accumulator if $R < 0$.
3. s(a) is not changed by these operations.

Add Magnitude To Memory

M+MG	$R=m+\bar{a}$	
M-MG	$R=m-\bar{a}$	

1. R→m if s(R) = s(m).
2. 0→m if s(R) ≠ s(m).
3. s(m) is not changed.

Multiply

*	$a \cdot m \longrightarrow a_{20}$	(I)
*N	$a \cdot -m \longrightarrow a_{20}$	

1. Multiplication takes place only if mode = B or BU.
2. The decimal mode gives LTRS and 00_2 to bits 7.17 and 7.18.
3. The length of a or m must be ≤ 48 bits in binary multiply.
4. The portion of the accumulator not containing the product is set to zero.

Multiply Factor and Add

*+	$m \cdot (\$FT)+a \rightarrow a$	(I)
*N+	$-m \cdot (\$FT)+a \rightarrow a$	

1. Write: *I+ and *NI+ for an immediate operand.
2. Multiplication takes place only if mode = B or BU.
3. Decimal mode gives LTRS and 10_2 to bits 7.17 and 7.18.

Divide

/	$a/m \longrightarrow a$	(I)
/N	$a/-m \longrightarrow a$	

1. Divide takes place only in the binary mode.
2. Decimal divide gives LTRS and 01_2 in bits 7.17 and 7.18.
3. The remainder is placed in $RM. The remainder sign, (60) $RM, is the same as the original s(a). Fl ($RM) = 0.
4. Bits to the right of the offset are cleared.

Load

L	$m \longrightarrow a$	(I)
LN	$-m \longrightarrow a$	

1. 0→Fl(a).
2. The entire accumulator is cleared before the load.

Load with Flag Bits

LWF	$m \longrightarrow a$	(I)
LWFN	$-m \longrightarrow a$	

1. Fl(m)→Fl(a).

Load Factor

LFT	$m \longrightarrow \$FT$	(I)
LFTN	$-m \longrightarrow \$FT$	

1. 0→(61 - 63) $FT.
2. The offset field is ignored.

Load Transit and Set

LTRS	$m \longrightarrow \$TR$	(I)
LTRSN	$-m \longrightarrow \$TR$	

1. Offset→$AOC.
2. 11_2→bits 7.17 and 7.18.
3. Indicator $BTR = 1 and $DTR = 0 if mode is B or BU.
   Indicator $DTR = 1 and $BTR = 0 if mode is D or DU.

Store

ST	$a \longrightarrow m$	
STN	$-a \longrightarrow m$	

1. SB(a)→SB(m).
2. If the byte size is greater than four:
   Binary: zone bits of the sign byte register are stored in SB(m).
   Decimal: zone bits of the sign byte register are stored in each byte of m.

Store Rounded

SRD
SRDN

These operations are the same as the corresponding Store operations, except for:

a. Binary: a one is added one bit to the right of the offset, prior to the store.
b. Decimal: 0101 is added one byte to the right of the offset, prior to the store.
c. The accumulator is unchanged, even if rounding occurs.

Add One to Memory

M+1	$m+1 \longrightarrow m$	
M-1	$m-1 \longrightarrow m$	

1. The one is added to the low order byte.
2. The offset field is ignored.

Compare

K	a:m	(I) 1. The Compare operations set the AL, AE, and AH indicators. AL is set to one if: $a < m$ AE is set to one if: $a = m$ AH is set to one if: $a > m$ 2. All bits to the left of the offset in the accumulator participate in the compare.
KN	a:-m	

Compare for Range

KR	a:m	(I) 1. If the AH indicator is off prior to the operation, it is executed as a NOP. 2. If AH is on: AL is unchanged. AE is set to one if $a < m$ AH is ~~[illegible]~~ one if $a \geq m$
KRN	a:-m	

Compare If Equal

KE	a:m	(I) 1. If the AE indicator is off, no changes will occur. 2. If the AE indicator is on, the indicators are set as in Compare, K.
KEN	a:-m	

Compare Field

KF	$\bar{a}$:m	(I) 1. The indicators are set as in Compare. 2. The length of the accumulator comparand is the same as the length of the storage comparand. 3. The matching bits of both operands are compared.
KFN	$\bar{a}$:-m	

Compare Field for Range

KFR	$\bar{a}$:m	(I) 1. The accumulator comparand is the same as in Compare Field, KF. 2. The indicators are set as in Compare Range, KR.
KFRN	$\bar{a}$:-m	

Compare Field If Equal

KFE	$\bar{a}$:m	(I) 1. The accumulator comparand is the same as in Compare Field, KF. 2. The indicators are set as in Compare If Equal, KE.
KFEN	$\bar{a}$:-m	

Logical Connectives OP(dds), A_{24} (I), OF_7 (I')

Note: If the operand from storage has a byte size (BS) less than eight, then eight minus BS (8 - BS) leading zeros are added to each byte from storage before the connect takes place. However, the storage operand is not changed in Cxxxx or CTxxxx.

Connect to Accumulator

$Cx_1x_2x_3x_4$ Result $\longrightarrow$ a

Connect to Memory

$CMx_1x_2x_3x_4$ Result $\longrightarrow$ m

Connect for Test

$CTx_1x_2x_3x_4$ Result is not stored.

$x_1x_2x_3x_4$ is a four-bit binary configuration to describe the type of connective; it is summarized:

Let: m = a bit from storage (may be an inserted leading zero if the byte size is less than 8).
a = a bit from the accumulator corresponding to m. The accumulator byte size always = 8.
x_1 = desired result if m = 0 and a = 0
x_2 = desired result if m = 0 and a = 1
x_3 = desired result if m = 1 and a = 0
x_4 = desired result if m = 1 and a = 1

Example: C1010 (BU, 64, 4), 0 will complement the entire 128-bit accumulator.

Pseudo-Connectives

LF	(Load Field)	LF = C0011
SF	(Store Field)	SF = CM0101

Immediate Connects

To indicate immediate addressing, write: $CIx_1x_2x_3x_4$, $CTIx_1x_2x_3x_4$, and LFI.

\$AOC = All ones count register.
\$LZC = Left zeros count register.

After a connective operation the two registers, \$AOC and \$LZC contain the indicated counts of the result. Because the result may not occupy the entire accumulator, \$AOC and \$LZC may not give the total count of ones and left zeros of the accumulator. However, these counts always give the correct count in CM or SF.

Convert Instructions

Definitions:
a_D = accumulator in decimal, four-bit bytes with specified offset.
a_B = accumulator in binary with specified offset.
a_{B20} = accumulator in binary with offset = 20.
a_{B68} = accumulator in binary with offset = 68.
m_B = storage operand in binary with specified byte size and field length.
m_D = storage operand in decimal with specified byte size and field length.
\$TR = 64-bit transit register with a sign byte in the rightmost four bits.

Note: The conversion goes: from decimal to binary if the mode given is decimal; from binary to decimal if the given mode is binary.

Convert

CV	$a_D \longrightarrow a_{B68}$	if mode = D or DU	1. In binary a field of 48 bits is used.
or	$a_{B68} \longrightarrow a_D$	if mode = B or BU	2. The entire accumulator to the left of the offset is used.
CVN	$-a_D \longrightarrow a_{B68}$		
or	$-a_{B68} \longrightarrow a_D$		

Double Convert

DCV	$a_D \longrightarrow a_{B20}$	1. In binary, a field of 96 bits is used.
or	$a_{B20} \longrightarrow a_D$	2. The entire accumulator to the left of the offset is used.
DCVN	$-a_D \longrightarrow a_{B20}$	
or	$-a_{B20} \longrightarrow a_D$	

Load Converted

LCV	m_D	a_B	(I)	1. $s(m) \rightarrow s(a)$
or	m_B	a_D		2. $0 \rightarrow Fl(a)$
LCVN	$-m_D$	a_B	(I)	3. The entire accumulator is cleared before the load.
or	$-m_B$	a_D		

Load Transit Converted

LTRCV	m_D	$\$TR_B$ (I)	1. The accumulator and offset are ignored.
or	m_B	$\$TR_D$	2. $0 \rightarrow Fl(\$TR)$
LTRCVN	$-m_D$	$\$TR_B$ (I)	3. $s(m) \rightarrow s(\$TR)$
or	$-m_B$	$\$TR_D$	4. The entire $TR is cleared before the load.

Progressive Indexing

Any VFL or Connective operation (when not immediate) may have a second operation enclosed in parentheses. The second operation may be $V \pm I$, $V \pm IC$, or $V \pm ICR$.

Format: $OP(OP_2)(dds)$, A_{24} (J), OF_7 (I')

Notes: 1. The original value field of J is the effective address of operation.
2. A_{24} is the immediate operand specified by J in $V \pm I$, and so on, and the value field of J is incremented by $\pm A_{24}$ according to $\pm I$. The incrementing takes place subsequent to note 1.
3. J may be $XO.

INDEXING OPERATIONS

Notation for symbolizing the Indexing Operations

Index Word Operands

J = bits (0 - 63) of the index word.
V = bits (0 - 24) of J.
C = bits (28 - 45) of J.
R = bits (46 - 63) of J.

Storage Word Operands

m = bits (0 - 63) of a storage word.
V(m) = bits (0 - 24) of m if the second operand is V. (sign of V is in bit 24)
V(m) = bits (0 - 17) of m if the second operand is C or R.

Immediate Operands

m = bits (0 - 18) of the effective address if the second operand is V.
m = bits (0 - 17) of the effective address if the second operand is C or R.

Notes: 1. For clarity, the titles to the indexing and the branch operations have been omitted.
2. The indicators XF, XCZ, XVLZ, XVZ, and XVGZ are set by all of the direct and immediate index operations except KV, KC, KVI, KVNI, and KCI. These indicators are set before the refill (if any) takes place.
KV, KC, . . . , KCI set the index compare indicators XL, XE, and XH.

Direct Index Arithmetic OP, J, A_{19} (I)

LX	$m \rightarrow J$	1. $M = A_{19}$ (I)
LV	$V(m) \rightarrow V$	2. $m = (M)$
LC	$V(m) \rightarrow C$	3. C_2 = The count field of J after modification
LR	$V(m) \rightarrow R$	
SX	$J \rightarrow m$	
SV	$V \rightarrow V(m)$	
SC	$C \rightarrow V(m)$	1. $0 \rightarrow (18 - 24)$ of m.
SR	$R \rightarrow V(m)$	1. $0 \rightarrow (18 - 24)$ of m.
V+	$V+V(m) \rightarrow V$	1. There is no V - etc.
V+C	$\left\{\begin{array}{l} V+V(m) \rightarrow V \\ C-1 \rightarrow C_2 \end{array}\right.$	
V+CR	$\left\{\begin{array}{l} V+V(m) \rightarrow V \\ C-1 \rightarrow C_2 \\ (R) \rightarrow (J) \text{ if } C_2 = 0 \end{array}\right.$	
SVA	$V \rightarrow V(m)$	1. V is truncated to 18, 19, or 24 bits, as is appropriate for the instruction containing V(m).
LVE	$(M)^n \rightarrow V$	1. (M) means contents of M $(M)^1$ " " " (M) . . . $(M)^n$ " " " $(M)^{n-1}$
KV	$V:V(m)$	1. Indicators: XL, XE, XH are set by KV and KC. This setting is the only output of KV and KC.
KC	$C:V(m)$	
RNX	$\left\{\begin{array}{l} J \rightarrow (R(\$XO)) \\ m \rightarrow J \\ M \rightarrow R(\$XO) \end{array}\right.$	1. Used for saving and restoring index registers.
LVS	(special format):	LVS, J, $A^1, A^2, \ldots, A^n$
	$\sum_{i=i}^{n} V(A^i) \rightarrow V(J)$	1. The sum may include any subset of the index words. 2. No indexing of the address field is allowed.

Immediate Index Arithmetic OP, J, A_{19}

Notes: 1. None of the immediate index instructions allow for indexing of the address. A_{19} is the effective address and is represented by A below.
2. The output of KVI, KVNI, and KCI is the setting of indicators XL, XE, and XH.

LVNI	$-A \rightarrow V$	1. (19 - 24) of V are set to 0.
LVI	$A \rightarrow V$	1. (19 - 24) of V are set to 0.
LCI	$A \rightarrow C$	
LRI	$A \rightarrow R$	
V+I	$V+A \rightarrow V$	1. (19 - 24) of V are unchanged.
V-I	$V-A \rightarrow V$	1. (19 - 24) of V are unchanged.
V+IC	$\left\{\begin{array}{l} V+A \rightarrow V \\ C-1 \rightarrow C \end{array}\right.$	1. (19 - 24) of V are unchanged.
V-IC	$\left\{\begin{array}{l} V-A \rightarrow V \\ C-1 \rightarrow C \end{array}\right.$	1. (19 - 24) of V are unchanged.
V+ICR	$\left\{\begin{array}{l} V+A \rightarrow V \\ C-1 \rightarrow C_2 \\ (R) \rightarrow (J) \text{ if } C_2 = 0 \end{array}\right.$	1. (19 - 24) of V are unchanged.
V-ICR	$\left\{\begin{array}{l} V-A \rightarrow V \\ C-1 \rightarrow C_2 \\ (R) \rightarrow (J) \text{ if } C_2 = 0 \end{array}\right.$	1. (19 - 24) of V are unchanged.
C+I	$C+A \rightarrow C_2$	
C-I	$C-A \rightarrow C_2$	
KVI	(0 - 18) of V:A	1. (19 - 24) of V are compared with zeros.
KVNI	(0 - 18) of V:A	1. (19 - 23) of V are compared with zeros and (24) of V is compared with 1 (minus).
KCI	C:A	

Count and Branch Operations OP, J, B_{19} (K)

CB $C_1 - 1 \longrightarrow C_2$
$IC_1 + 0.32 \longrightarrow IC$ if $C_2 = 0$
$M \longrightarrow IC$ if $C_2 \neq 0$

CBR $C_1 - 1 \longrightarrow C_2$
$IC_1 + 0.32 \longrightarrow IC$ and $(R) \longrightarrow (J)$ if $C_2 = 0$
$M \longrightarrow IC$ if $C_2 \neq 0$

CBZ $C_1 - 1 \longrightarrow C_2$
$IC_1 + 0.32 \longrightarrow IC$ if $C_2 \neq 0$
$M \longrightarrow IC$ if $C_2 = 0$

CBRZ $C_1 - 1 \longrightarrow C_2$
$IC_1 + 0.32 \longrightarrow IC$ if $C_2 \neq 0$
$M \longrightarrow IC$ and $(R) \longrightarrow (J)$ if $C_2 = 0$

1. K may be only 0 or 1.
2. M=the effective address of B_{19} (K).
3. IC_1 is the value of the instruction counter where the CB instruction is located.
4. C_1 and C_2 are the count field of J before and after the count portion of the instruction, respectively.

Note: In addition to the stated functions, the value field of J may be modified by placing + , - , or H after the above mnemonics. The modification of V takes place regardless of C_2 and before the refill (if any).

Example: In addition to the given functions of CB, we have:

CB	leave V alone
CB+	$V + 1.0 \longrightarrow V$
CB-	$V - 1.0 \longrightarrow V$
CBH	$V + 0.32 \longrightarrow V$

Unconditional Branch Operations: OP, A_{19} (I)

B $\left\{ M \longrightarrow IC \right.$
BR $\left. M + IC_1 + 0.32 \longrightarrow IC \right.$

BE $\left\{ \begin{array}{l} \text{Enable} \\ M \end{array} \right. \longrightarrow IC$

BD $\left\{ \begin{array}{l} \text{Disable} \\ M \end{array} \right. \longrightarrow IC$

BEW $\left\{ \begin{array}{l} \text{Enable} \\ \text{Wait} \\ M \end{array} \right. \longrightarrow IC$

NOP $IC_1 + 0.32 \longrightarrow IC$

1. The unconditional branch instructions are the only branch instructions which allow a 4 bit index field, I. The conditional branch instructions may have only a 1-bit index field, K.
2. IC_1 is the value of the instruction counter where the instruction is located (i.e., the leftmost bit of the instruction).

Branch on Bit Operations: OP, A_{24} (I), B_{19} (K)

BB $IC_1 + 0.32 \longrightarrow IC$ if $m_1 = 0$
$M_2 \longrightarrow IC$ if $m_1 = 1$

1. $m_1 = (A_{24}(I))$, the bit being tested.
2. $M_2 = B_{19}(K)$, the branch address.
3. K=0 or 1; I=0 -15.

BZB $IC_1 + 0.32 \longrightarrow IC$ if $m_1 = 1$
$M_2 \longrightarrow IC$ if $m_1 = 0$

Note: The BB and BZB may have a suffix, Z, 1, or N, which, respectively, will set m_1 to zero or to one, or negate it. This function is independent of the success of the branch. For example, the following branch on bit instructions are permissible and perform the stated functions as well as:

BB	BZB	leave m_1 alone
BBZ	BZBZ	$0 \longrightarrow m_1$
BB1	BZB1	$1 \longrightarrow m_1$
BBN	BZBN	$-m_1 \longrightarrow m_1$

Branch on Indicator Operations BIND, B_{19} (K)

BIND $IC_1 + 0.32 \longrightarrow IC$ if ind. = 0
$M \longrightarrow IC$ if ind. = 1

1. The indicators may not be set to 1 or negated with a BIND operation.

BZIND $IC_1 + 0.32 \longrightarrow IC$ if ind. = 1
$M \longrightarrow IC$ if ind. = 0

Notes: 1. The letters "IND" in BIND are replaced by the appropriate indicator mnemonics as shown in note 2 below.
2. The above operations can have a suffix, Z, which will cause the indicator being tested to be set to zero independently of the success of the branch. For example, BZXPOZ will set indicator XPO to zero arbitrarily. We may have: BXPO; BZXPO; BXPOZ; and BZXPOZ. The following list includes all of the indicator mnemonics which may be used in BIND, B_{19}(K), and their bit addresses.

Mnemonic	Name	Bit Address
	EQUIPMENT CHECK	
MK	Machine Check	11.0
IK	Instruction Check	11.1
IJ	Instruction Reject	11.2
EK	Exchange Control Check	11.3
	ATTENTION REQUEST	
TS	Time Signal	11.4
CPU	Other CPU	11.5
	INPUT-OUTPUT REJECTS	
EKJ	Exchange Check Reject	11.6
UNRJ	Unit Not Ready Reject	11.7
CBJ	Channel Busy Reject	11.8
	INPUT-OUTPUT STATUS	
EPGK	Exchange Program Check	11.9
UK	Unit Check	11.10
EE	End Exception	11.11
EOP	End of Operation	11.12
CS	Channel Signal	11.13
	(not available)	11.14
	INSTRUCTION EXCEPTION	
OP	Operation Invalid	11.15
AD	Address Invalid	11.16
USA	Unended Sequence of Addresses	11.17
EXE	Execute Exception	11.18
DS	Data Store	11.19
DF	Data Fetch	11.20
IF	Instruction Fetch	11.21
	RESULT EXCEPTION	
LC	Lost Carry	11.22
PF	Partial Field	11.23
ZD	Zero Divisor	11.24
	RESULT EXCEPTION-FLOATING POINT	
IR	Imaginary Root	11.25
LS	Lost Significance	11.26
PSH	Preparatory Shift Greater than 48	11.27
XPFP	Exponent Flag Positive	11.28
XPO	Exponent Overflow	11.29
XPH	Exponent High	11.30
XPL	Exponent Range Low	11.31
XPU	Exponent Underflow	11.32
XPFN	Exponent Flag Negative	11.33
RU	Remainder Underflow	11.34

	FLAGGING	
TF	T Flag	11.35
UF	U Flag	11.36
VF	V Flag	11.37
XF	Index Flag	11.38
	TRANSIT OPERATIONS	
BTR	Binary Transit	11.39
DTR	Decimal Transit	11.40
	PROGRAMMER INDICATORS	
PGO or PG		11.41
PG1		11.42
PG2		11.43
PG3		11.44
PG4		11.45
PG5		11.46
PG6		11.47
	INDEX RESULT	
XCZ	Index Count Zero	11.48
XVLZ	Index Value Less than Zero	11.49
XVZ	Index Value Zero	11.50
XVGZ	Index Value Greater Than Zero	11.51
XL	Index Low	11.52
XE	Index Equal	11.53
XH	Index High	11.54
	ARITHMETIC RESULT	
MOP	To-Memory Operation	11.55
RLZ	Result Less than Zero	11.56
RZ	Result Zero	11.57
RGZ	Result Greater than Zero	11.58
RN	Result Negative	11.59
AL	Accumulator Low	11.60
AE	Accumulator Equal	11.61
AH	Accumulator High	11.62
	MODE	
NM	Noisy Mode	11.63

TRANSMIT OPERATIONS: OP, J, A_{18}(I), A'_{18}(I')

Notes: 1. Full words are transmitted in all Transmit and Swap instructions.
2. In the immediate operations, J is the count of the number of full words transmitted. J must be $\leq$ 16. If J = 0, 16 words are transmitted.
3. In the others (the direct transmission) the count field of J has the number of full words to be transmitted.

Transmit Forward

T	$(M_1) \longrightarrow (M_2)$ $(M_1+1) \longrightarrow (M_2+1)$ etc.	1. M_1 is the effective address of A_{18} (I) 2. M_2 is the effective address of A'_{18} (I')

Transmit Forward Immediate

TI	$(M_1) \longrightarrow (M_2)$ $(M_1+1) \longrightarrow (M_2+1)$ etc.

Transmit Backward

TB	$(M_1) \longrightarrow (M_2)$ $(M_1-1) \longrightarrow (M_2-1)$ etc.	1. Both blocks are referred to in a backward direction.

Transmit Backward Immediate

TBI	$(M_1 \longrightarrow (M_2)$ $(M_1-1) \longrightarrow (M_2-1)$ etc.

Swap Forward

SWAP	$(M_1) \longleftrightarrow (M_2)$ $(M_1+1) \longleftrightarrow (M_2+1)$ etc.

Swap Forward Immediate

SWAPI	$(M_1) \longleftrightarrow (M_2)$ $(M_1+1) \longleftrightarrow (M_2+1)$ etc.

Swap Backward

SWAPB	$(M_1) \longleftrightarrow (M_2)$ $(M_1-1) \longleftrightarrow (M_2-1)$ etc.

Swap Backward Immediate

SWAPBI	$(M_1) \longleftrightarrow (M_2)$ $(M_1-1) \longleftrightarrow (M_2-1)$ etc.

MISCELLANEOUS OPERATIONS: OP, A_{19}(I)

Store Instruction Counter If

SIC	IC_1+1.0 $\longrightarrow$ (0-18) of A_{19}(I) if the following half word branch instruction is executed.	1. SIC; NOP will not store the IC.

Refill

R	$(R_M) \longrightarrow (M)$	1. R_M=refill field of word M

Refill If Count Is Zero

RCZ	$(R_M) \longrightarrow (M)$ if C field of M = 0

Execute

EX	Execute $\longrightarrow$ (M)	1. The instruction located at M is executed. 2. Control then goes to the instruction following EX.

Execute Indirect and Count

EXIC	Execute $(M)^1$ $(M) + 1 \longrightarrow (M)$	1. The instruction whose address is located in M is executed.

Store Zero

Z	$0 \longrightarrow (M)$	1. Full word of zeros.

INPUT-OUTPUT INSTRUCTIONS: OP, A_7(I), A_{18}(I')

Locate

LOC

Select Unit

SU

A_7 (I) represents a channel address; A_{18}(I') represents:
1. The address of one of several units attached to channel A_7(I); in this case LOC or SU must be given before a RD or W addressing this channel;
2. An address on the disk specified by A_7(I). LOC=SU.

Read

RD	$A_7(I)$ represents a channel address; a reading operation in initiated for this channel (or for a unit attached to this channel if more than one unit is available and has been readied by a LOC instruction). $A_{18}(I')$ is the address of a control word.

Write

W	Initiates a writing operation. Analogous to RD except that the skip flag of the control word is ignored.

Release

REL	Immediately terminates any operation in progress at the unit specified in $A_7(I)$, the channel address, or in the last unit at $A_7(I)$ selected by a LOC instruction, if $A_7(I)$ consists of more than one unit.

Copy Control Word

CCW	The current control word corresponding to the addressed channel $A_7(I)$ is sent to $A_{18}(I')$.

LOCSEOP RDSEOP WSEOP RELSEOP CTLSEOP SUSEOP	Same as LOC, SU, RD, W, REL, CTL except the SEOP bit in control word is set to 1; thus, program interruption on completion of an operation is suppressed, provided no exception conditions, such as unit check and end exception, are encountered.

Control

CTL	Initiates performance of certain functions at the channel indicated by $A_7(I)$, or at the last unit selected by a LOC instruction. The functions are indicated:

General I-O Unit (Standard for $A_{18}(I')$)

$A_{18}(I')$ =	0	Reserved light off
	1	Reserved light on
	2	Read-write check light on
	4	ECC mode
	5	No ECC mode

Card Reader and Card Punch
Standard, except $A_{18}(I') = 2$ also causes a card to be offset in the stacker.

Tape Units
Standard, but in addition:

$A_{18}(I')$ =	4	ECC mode, odd parity
	5	No ECC mode, odd parity
	6	No ECC mode, even parity
	7	Rewind tape
	8	Space block (record)
	9	Backspace block (record)
	10	Space file
	11	Backspace file
	12	Write tape mark (E. O. F. mark)
	13	Erase long gap
	14	High-density mode (556 bits/inch)
	15	Low-density mode (200 bits/inch)

Inquiry Station, Printer, Console
Standard, except codes 4 and 5 are missing. On console, $A_{18}(I') = 3$ causes the gong to sound.

APPENDIX D

ERROR MARKS

Strap-1 prints error marks in the rightmost columns of the assembly listing, showing actual or probable coder and/or machine errors. These error marks, the 26 letters of the alphabet and some special characters (given at the end of the list), are explained below:

A. 1. No characters are given with an A or IQS entry mode in a DD statement.
2. Too many address fields were given.
3. Symbol with XM has less than 2 dimensions.

B. 1. Byte size of this instruction is >8, and has been assembled modulo 8.
2. Byte size of a decimal instruction $\neq 4$, but has been left at the specified value modulo 8.
3. A or IQS byte size in DD statement is >12, and has been assembled modulo 8.

C. Negative field (address, index, field length, and so on) has been complemented. (This complementing takes place prior to the truncation described under V, is any.)

D. Data error in DD or DDI. (The data have been set to zero.)

E. Entry mode error in DD or DDI. (Entry mode (10) is assumed.)

F. 1. Given field length is > 64, and has been assembled modulo 64.
2. Field length of a VFL binary-multiply or binary-divide type instruction is>48. It has been left at the specified value modulo 64.

G. A "go-to" type instruction, such as a branch or end card, has a transfer address <32 and I index 0.

H. Card has an illegal name (a non-alphameric character is present; the character has been ignored).

I. A branch on indicator, with indicator numerically specified, has indicator number >63. (The indicator has been computed modulo 64.

J. 1. Illegal punch on this card:
2. Illegal character in some field on this card (if an illegal numeric occurs in a data field of radix 2 through 9, the field is set to zero; in all other cases, the illegal character is printed as an error mark just to the right of any other error marks pertaining to the field and is ignored in computing the value of the field.)

K. RTT error has been detected in reading tape 2 on this card.

L. 1. The location counter is out of range as a result of this instruction. If <32, it has not been changed; if > 777777.77_8, it has been taken modulo 1000000.00_8.
2. An SLC address contains an integer.

M. 1. Given mode is not consistent with the given operation or vice-versa, e.g., M + 1 (N),... The mode, FL, and BS are ignored.
2. No mode given with DD or DR--N is assumed.
3. No mode given with DDI--(BU, 24, 8) is assumed.
4. An operation which could be either VFL or floating point has no overruling mode and a numeric address. VFL is assumed.

N. Error in integer i.e., (, n) entry.

O. 1. Offeset + field length - byte size (if signed) on this operation is >128. None of the three quantities is changed.
2. Decimal offset is not divisible by 4, but is left unchanged.

P. Conditional branch has K field > 1. (The given K field is taken modulo 2.)

Q. 1. Illegal OP1 or OP2 in progressive indexing. In either case, OP2 is ignored.
2. Illegal OP3 in CW. OP3 is ignored.
3. Pseudo-operation has been specified in an EXT field. The requested field has been set to 0.

R. Radix is out of range. R = 10 is used.

S. Illegal operation after an SIC order. (Only half-word branch orders are permitted.) (The operation has, however, been assembled as requested.)

T. An index is given with LVS or an immediate index operation, which are non-indexable.

U. VFL decimal multiply or divide has been specified. (There are no such operations.) (The operation has been assembled as requested, but it acts as an LFT order.)

V. 1. Overflow in an address field; the field has been truncated if necessary to fit the number of bits available.
2. A DD with entry mode A or IQS has a byte size which is too small, that is, < 6 or <8 respectively.

W. The operation specified here is nonexistent. The full-word instruction SIC, $15; BE, 0 has been inserted here.

X. 1. An address field on this card cannot be completely evaluated (due to a too-complicated chain of SYN cards, and so on). The computation has been completed as far as possible.
2. A symbol specified on a PUNSYN card does not occur in the program.

Y. A symbol on this card contains more than six characters. (Only the first six are used.)

Z. 1. No address or name field has been specified on a SYN or DDI card, or a SYN card is not computable by pass 3.
2. Null J field on index operation.
3. Second half word address of an Input-Output op is <32.
4. DR has null address. No space is reserved; LCTR may be rounded however.

) No right parenthesis in a field; 2 or more left parentheses before first right parenthesis.

+ Arithmetic has been used in the address field of a VFL immediate operation (not allowed); or, a symbol defined by a DDI has been used in an arithmetic expression in a VFL immediate address field. (The arithmetic was performed in binary, unsigned.)

6 A symbol on this card has six characters and cannot be tailed.

* An undefined symbol has occurred on the card.

OTHER Characters other than the error marks listed above may be printed in the rightmost columns of the assembly listing as an indication that these characters were used in fields where they are not legal. For example, if the letter "G" were used in a hexadecimal data field, "G" would appear as an error mark at the right of the listing.

Note: The print-out of all except certain non-coder error marks (namely, K and J) can be suppressed by prefixing the OP field with the symbol $. If duplicate error marks appear, more than one error of the same type has been made. Individual error marks A, B, C,... may be suppressed by the use of the instruction SEM, A, B, C,.... All may be suppressed by SEM alone. Any of the suppressed error marks may be restored by the instruction REM which works in a similar manner. Again, J, K, and illegal punches cannot be suppressed by SEM.

International Business Machines Corporation
Data Processing Division 112 East Post Road White Plains N Y

Printed in U.S.A. C22-6531