1959

May 22, 1959

Here is a revised and amplified version of the STRAP I write-up. This will be followed (soon?) by appendices C, D, E, . . . listing error marks, system symbols, . . . and an Index.

22 May 1959 LOS ALAMOS

Kolsky

STRAP 1

An Assembly Program for Stretch

by

Louis Gatt

Zane Motteler

Grover Lewis

Frank Evans

Dick Thomas

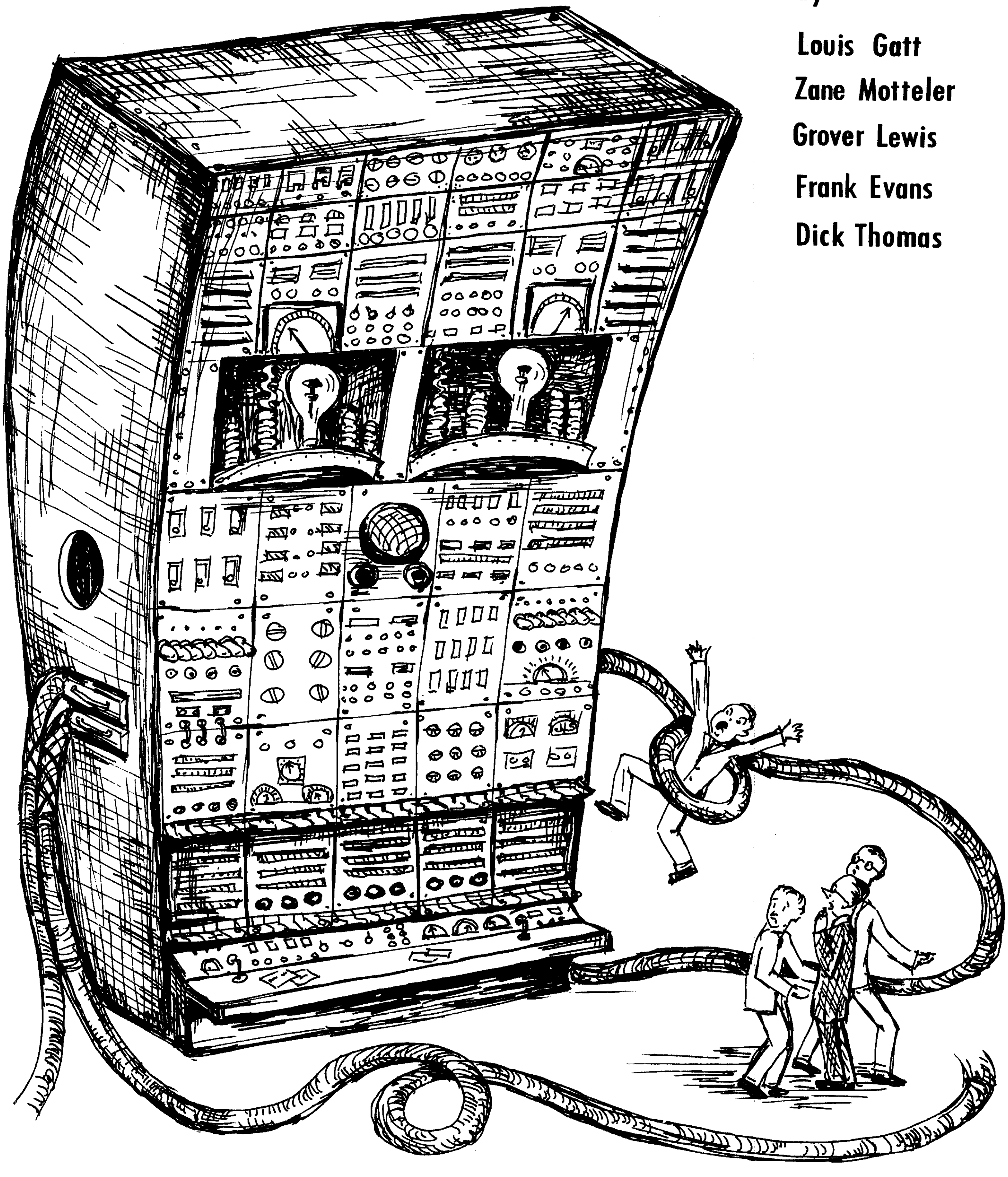


TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

Revised January 8, 1960

GENERAL

Strap 1 is a program for assembling symbolic programs for Stretch, utilizing a 32K 704. It is a predecessor to Strap 2, which will utilize the Stretch machine itself for assembly. All programs which can be assembled by Strap 1 can also be assembled by Strap 2. Appendix A contains such a program.

1. STRAP CODING FORM

The coding form and the card form are divided into 4 fields. These fields and their positions are shown below.

	1	2-9	10 72	73 - 80
Col.	Class	Name	Statement	Identification

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 by a symbol (optional)
3. Statement (63 columns) - to express a machine or pseudo-instruction. Information punched in column 72 will not appear on the listing, but is assembled as part of the statement.
4. Identification (8 columns) - to identify the card or program (does not affect assembly)

2. INSTRUCTION FORMATS

2.0. General

Machine instructions are written and punched symbolically in the statement field of the form described above. A card may contain several instructions separated by ; . (The keypunchers will be instructed to punch this symbol as 11-0 double punch.) The number of instructions which may be punched on a card is limited by the number of columns available in the statement field. The symbol in the name field of a card having more than one instruction in the statement field is associated with the first instruction. The remaining instructions are treated as if they appeared on separate cards having blank name fields. (It is not necessary to name an instruction unless it is referred to in the program.) A single instruction cannot be continued from one card to another. A comment may follow any instruction. A comment is initiated by the symbol " (an 8-4 double punch) and terminated either by the end of the card or a ; . A " in the name field causes the whole card to be treated as comment; it will be printed on the listing but will not otherwise affect assembly.

Symbolic instructions are divided into subfields (e.g., operation, address, offset, etc.) by commas. These subfields may in turn be subdivided or modified by expressions contained in parentheses, such as index register specifications, secondary operations in progressive indexing, etc. Three general classes of operations can be defined in Strap 1: legal machine operations, data-entry psuedo-operations, and instructions-to-the-compiler pseudo-operations. See Appendix A for examples. Appendix B and C contain a list of the legal machine operations.

2.1 Machine Instructions

	Format	Operation
1.	OP(dds), A_{18}(I)	Floating point
2.	OP,A_{19}(I)	Miscellaneous, unconditional branch, SIC
3.	OP, J, A_{19}(I) or OP, J, A_{18}(I)	Direct index arithmetic
4.	OP, J, A_{19} or 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(OP_2)(dds), A_{24}(I),OF_7(I')	Progressive indexing
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, IO(I), CW_{18}(I')	Input-output select
12.	LVS, J, A, A', A", A''',···	Load value with sum

2.2 Data Entry Instructions

	Format	Operation
1.	(EM)DD(dds), D, D',D",···	Data definition
2.	CW(OP_2), FWA, C, R	Input-output control word
3.	XW, V, C, R, 0-7	Index word
4.	VF, V	Value field
5.	CF, C	Count field
6.	RF, R	Refill field
7.	EXT(L, L') any legal instruction	Extract

2.3 Instructions to Compiler

	Format	Operation
1.	SYN(dds), A_{24}	Synonym
2.	DDI(dds), D	Data definition for immediate op
3.	SLC, A_{24}	Set location counter
4.	END, B_{19}	End of program
5.	DR(dds), (L, L', L",···)	Data reservation
6.	CNOP, A_{19}(I)	Conditional no-op
7.	TLB, B_{19}	Terminate loading and branch

2.4 Format Symbols Defined

1.	OP or OP_1	A fixed symbolic (hopefully mnemonic) representation of a machine operation.
2.	OP_2	A secondary operation in progressive indexing or input-output.
3.	A_n	A data address of length n bits.
4.	B_{19}	A 19-bit branch address.
5.	I	A 4-bit index address in which 0 signifies no indexing and 1 to 15 signifies indexing by the corresponding index register.
6.	K	A single bit index address in which the choice is 0--no indexing, or 1--index with register 1.
7.	J	A 4 bit index address which refers to an index register as an operand. In this case 0 refers to index 0, word 16.

8.	OF_7	Offset.
9.	IO	Input-output unit address.
10.	CW_{18}	Control word address.
11.	EM	Entry mode.
12.	D	Numerical data.
13.	FWA	First word address of words to be transferred in input-output operation.
14.	C	Count field (18 bits, unsigned).
15.	R	Refill field (18 bits, unsigned).
16.	V	Value field (25 bits, signed).
17.	L	Symbolic or numeric integer.
18.	dds	Data description.
19.	primes	Used to distinguish otherwise identical fields in a format. In transmit the data is transmitted from A to A'.

3. Data Description

In the format specifications above, the symbol dds is added as a modifier to certain operations and stands for the data description field. It is specified by:

1.	M	the use mode,
2.	FL	the field length,
3.	BS	the byte size.

These three entries appear within parentheses in the above order, thus; (M, FL, BS). A data description given with any of the four pseudo-ops, DD, DDI, SYN, or DR, applies to the symbol in the name field of the card and is automatically assumed whenever that name appears in an address field of an instruction. This data description may be overruled by writing a different data description explicitly as a modifier in the two machine instruction formats where it applies. There are seven fixed mode designators as follows:

1.	N	Normalized floating point,
2.	U	Unnormalized floating point,
3.	B	Binary signed VFL,
4.	BU	Binary unsigned VFL,
5.	D	Decimal signed VFL,
6.	DU	Decimal unsigned VFL,
7.	P	A special character designating "data properties of."

Within a data description field the byte size or field length may be omitted, but never the mode. If byte size or field length, or both, are omitted, the mode will imply the missing part of the data description as follows:

N, U	fixed format of 64 bits; field length and byte size not appropriate,	
B	FL = 64	BS = 1,
BU	FL = 64	BS = 8,
D, DU	FL = 64	BS = 4.

Note: Some pseudo-ops (e.g. DDI) imply FL $\neq$ 64. See description of individual pseudo-op for details.

A data description using P is written as follows: (P, Symbol). It means that the data properties associated with the given symbol are to apply to the instruction with which it is written. P can be used only with legal machine instructions, never with a pseudo-op.

In straightforward coding it is unnecessary to write a data description on machine operations. The data description associated with the definition of a symbol (in a data-entry or data-reservation pseudo-op) is automatically applied to the machine operation in whose address the symbol appears. If a data description is given on a machine operation, it overrules any data description derived from the symbolic address.

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

1. The operation symbol can stand for either floating point or variable field length operations (e.g., +, -, *, /). The operation is assembled as VFL with data description (BU, 64, 8).

2. The operation symbol can stand for VFL only (e.g., M+1). It is assigned a data description (BU, 64, 8).

3. The operation symbol can stand for floating point only (e.g., +A, *NA). The operation is assembled as normalized floating point, except E+I and its modified forms, which are unnormalized unless overruled.

An error mark will be printed in any of these cases.

4. Strap 1 Location Counter

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 upwards 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 on full-word addresses. The SLC pseudo-operation provides a means of setting the assembly location counter to any value at any point in a code, and thus gives the programmer complete control of the location of his code. Following an SLC, the location counter is advanced in normal fashion until another SLC card resets it.

5. Symbols

A programmer symbol is any sequence of six or fewer alphabetic and numeric characters, the first of which must be specifically alphabetic. Such a symbol is defined by the programmer and may represent a machine address of not more than 24 bits plus a sign, or a signed integer of not more than 24 bits. A symbol is defined when it appears in the name field of a card. Hence 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 a code. There exist special pseudo-operations capable of defining a symbol as an address or an integer independently of the location counter.

A system symbol consists of a dollar sign followed by five or fewer alphabetic and numeric characters. System symbols represent various special registers, indicators and input-output units. Their meaning is fixed by the assembly program and is not subject to programmer control.

A programmer symbolized field is a field which may contain programmer symbols and/or system symbols. Of the fields shown in the instruction formats above all may contain programmer symbols except OP, OP_1, OP_2, EM, D, and the mode field of a data description. All others may be symbolized by the programmer subject to the rules and restrictions given below.

6. General Parenthetical Integer Entry

By means of the general integer entry any integer or arbitrary pattern of bits may be stored in any position of an instruction or data entry field. This type of entry may not be used with the pseudo-ops classified as instructions to the compiler. The format for general integer entry is: $(.n)A_{n+1}$. It is a modification which may be appended to a D field or to any programmer symbolized field (or in place of such a field) which is not enclosed by parentheses. (Thus, for example, FL and BS fields cannot contain a (.n) entry.) n is the number of the rightmost bit of the parenthetical field. The integer A_{n+1} is formed as an unsigned n+1--bit field and added to the instruction or data field by means of a logical "or" in the leftmost n+1--bits. Subfield boundaries are ignored by general integer entry. The position of the entry is determined by counting the bits of the whole instruction field no matter which subfield the integer entry may happen to be appended to. Thus, for example, in a VFL instruction so modified, OP, A_{24}(I)(.n)A_{n+1}, OF_7 is exactly equivalent to OP, A_{24}(I), OF_7(.n)A_{n+1}. In the case of a DD pseudo-op the position of the parenthetical field is determined by counting the bits of the field, D, with which it is written. In any case the general integer entry must follow all other information in the field or subfield in which it appears, except for another general integer 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 places in a field. 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 of the card. If A_{n+1} is negative, an n+1--bit 2's complement is taken. The maximum size of n is restricted by the total length of the instruction or data field, m. $0 \leqslant n < m$. For example, in a half-word instruction $0 \leqslant n \leqslant 31$; in a full-word instruction $0 \leqslant n \leqslant 63$. The radix of A_{n+1} may be specified as mentioned below under "Radix Specification." Ex.: E + I, (.8)41. The integer 41 will be entered in the left most 9 bits (8 + 1) of the E + I instruction.

7. Multidimensional Arrays

Strap 1 provides a convenient method of defining multidimensional arrays of data and of addressing individual elements of an array. All indexing, of course, must be handled explicitly by the programmer. A symbol is defined as the first element of an array of n+1 dimensions by virtue of its appearance in the name field of a data reservation statement of the following sort: DR(dds), $(L, L', L'', \cdots, L^r)$. This statement is interpreted as reserving space for an $L \times L' \times L'' \times \cdots \times L^r$ array of data fields. A number of bits equal to the field length of each element multiplied by the product of the dimensions is set aside for this array and the location counter advanced accordingly. (If the data description specifies floating point words, the correct number of full words is reserved, beginning at a full-word boundary.) In addition the number and value of the dimensions is permanently associated with the symbol so defined. Then in any address field a specific member of this array may be addressed by writing: Symbol $(q, q', q'', \cdots, q^r)$. The first element of the array is Symbol $(0, 0, 0, \cdots, 0)$ = Symbol, and the last element of the reserved space is Symbol $(L-1, L'-1, L''-1, \cdots, L^r-1)$. The address of an arbitrary element is computed by means of the formula: Address of [Symbol $(q, q', q'', \cdots, q^r)$] = Address of [Symbol $(0, 0, 0, \cdots, 0)$] + FL x $(q+q'L+q''LL'+q'''LL'L''+ \cdots)$, where FL is the field length of an element in the array. Strap 1 will handle a maximum of fifteen dimensions in this fashion. Such an array address may be used in any programmer symbolized field not in parentheses, except a general parenthetical integer entry.

8. <u>Bit Addresses and Integers</u>

8.0 <u>Definition</u>

Two kinds of numbers have been defined for use in the programmer symbolized fields of Strap statements. A bit address is a style of writing a machine address by specifying n_w, a number of full 64-bit words, and n_b, a number of bits. The format is $n_w . n_b$. The period separating the two integers distinguishes the bit address from an ordinary integer n_i, which is the second kind of number allowed to appear in address fields. As the name "bit address" implies, these numbers 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 instructions 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 numbers n_w and n_b except that $64n_w + n_b$ must be less than 2^{24}.

Example: 505.17 = 500.337 = 0.32337

Integers in programmer symbolized fields are always converted to binary. They are limited in length to the length of the field into which they are to be inserted, with the additional restriction that 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, etc.

8.1 Addition of Integers and Bit Addresses

Although it is expected that integers and bit addresses will generally be used in different fields, addition of the two types of numbers is defined, the result being a function of the type of instruction field for which the number is intended. Algebraic addition is permitted in all fields which may be symbolized by the programmer. Symbols for both bit addresses and integers are signed numbers. The number of terms which may appear in a field is limited only by the space available on the card, except for the case of SYN and DR, noted below in sections 10.0 and 11.0.

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

where SAM and JOE are defined as bit addresses and FRED is an integer, will in general be a legal address. The data description of the final symbol, FRED, will apply to the whole combination. In computing such an address, the sum of the bit addresses is obtained separately from the sum of the integers; the integer sum is then shifted left if necessary and the result added algebraically to the bit address. If the field for which the address is intended is signed, the sign will be placed in the correct bit. 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 EXT (L, L') where $|L|$ and $|L'|$ are used. A positive final result, of course, is inserted as a true figure. The programmer is reminded that a 2's complement must be used with care on Stretch in order not to get an "address invalid" indication.

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

1. The "I" or "K" index fields must contain at least <u>one</u> 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.)

3. A period may not appear in the field of a general integer entry. A symbolic bit address appearing in such a field is treated as a 24-bit integer. Ex: V+I, (.18)4.32 is not allowed, but: V+I, (.18)9 is.

4. No arithmetic can appear in the name field.

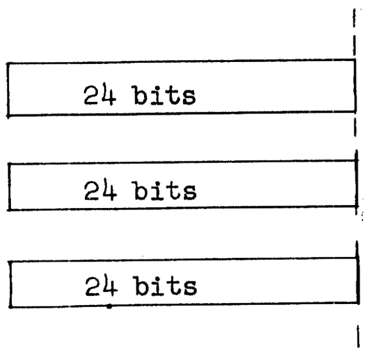
8.2 Rules for Combining Integers and Bit Addresses

The following rules describe the method by which bit addresses and integers are truncated and added. The numbers are assumed to be signed 24-bit integers before the operation. Addition is algebraic. An error indication will be given if non-zero bits are discarded, except for the "16" bit of an index field. In the diagrams below integers and bit addresses are drawn shifted with respect to each other by the proper amount. The numbers are algebraically added with the offset shown, complemented (if necessary), truncated (if necessary) to the correct final length, and inserted into the correct position in the operation word. Although the diagrams show the final sum field truncated to the appropriate length, the bits are not actually discarded unless they would fall outside the address field of the instruction. Some operations do not use all the space available in their address fields (e.g. transmit, input-output select), and in these cases bits may be placed in the unused portions by this means.

1. A_{24} Bit address: B.A.

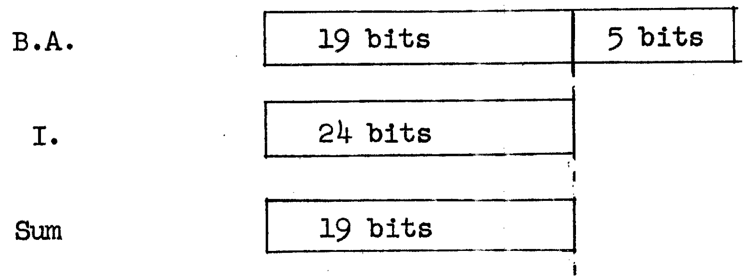
Integer: I.

Sum



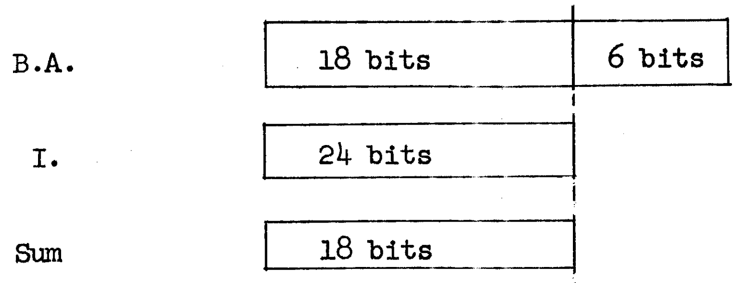
Note: Integer counts bits.

2. A_{19} Half-word address:



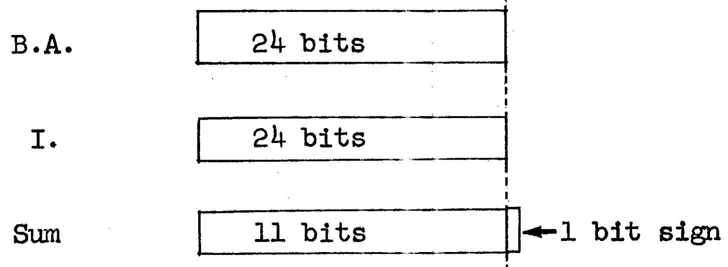
Note: Integer counts half words.

3. A_{18} Full-word address:

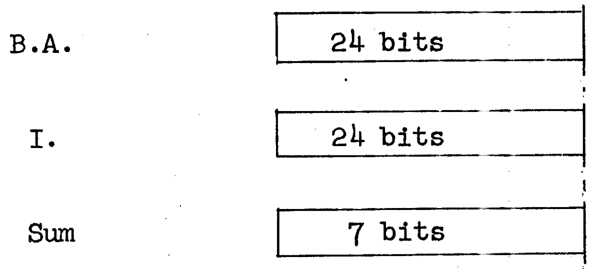


Note: Integer counts full words.

4. $A_{11\pm}$ Signed 11 bit address:

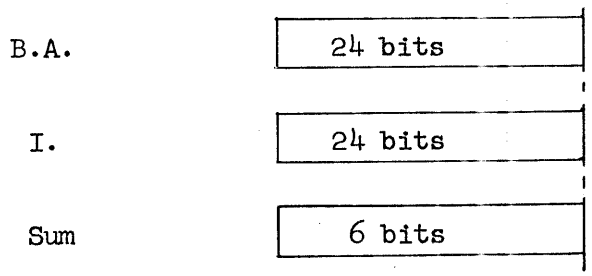


5. OF_7 Offset:



Note: Bit address 1.32 = .96 = integer 96

6. FL_6 Field length:

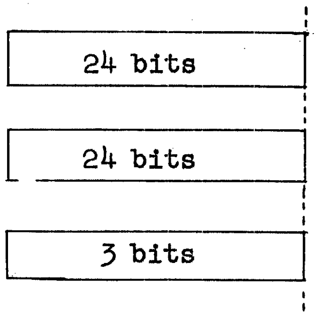


Note: 1.0 = .64 = 64 = 0 not error marked

7. BS_3 Byte size: B.A.

I.

Sum

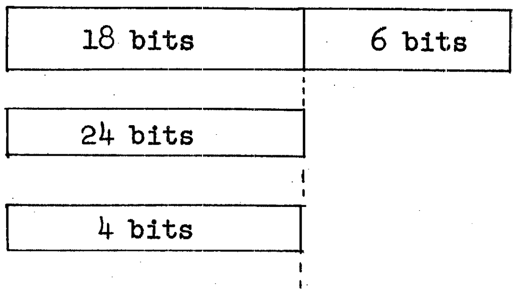


Note: .8 = 8 = 0 not error marked

8. I, J 4 bit index fields: B.A.

I.

Sum

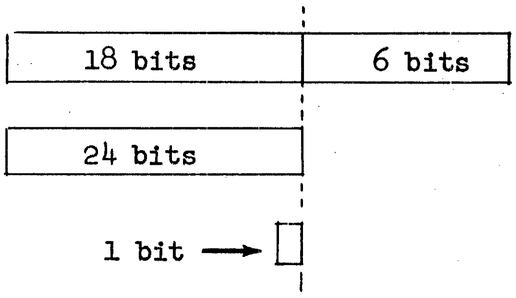


Note: A "1" in the bit position immediately to the left of the final sum field is discarded with no error indication.

9. K single bit index field: B.A.

I.

Sum

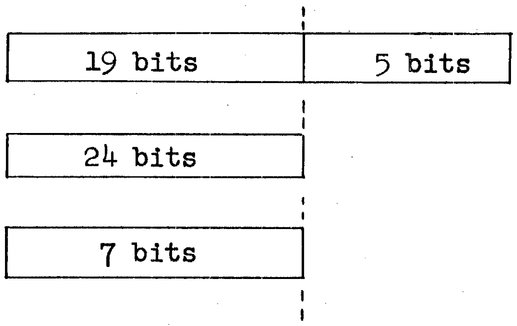


Note: A "1" in the bit position which corresponds to "16" in the sum is discarded with no error indication.

10. IO input-output address: B.A.

I.

Sum



Note: Integers count tape units, channels, etc.

9. Radix Specification

In any programmer symbolized field not enclosed by parentheses, numerical integers and bit addresses may be written in any radix from 2 to 10. The radix is specified by simply enclosing the appropriate integer (written in decimal) in parentheses at some appropriate point in the subfield. The radix applies to the entire subfield unless reset before reaching the end. If no radix base is specified, base 10 is assumed.

Some examples:

a. (8)573 - 34 + 50 (all numbers are octal)

b. (2)11011011100011.11110 (bit address written in binary)

c. (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 base 5.)

d. (8)7436.(10)60 + 9 (The full word portion of this bit address is written in octal, whereas the bit portion and the integer 9 are written in decimal.)

When writing a general parenthetical integer entry, the radix base may be specified within the same parentheses as the .n and in any order, thus, (.n, R) or (R, .n).

Examples:

a. (.50, 8)17 - JOE + (10)4203(4, .22) - 33303(.60)1030

b. (7)(.30)1265(.20)(10)138 - (6)43(.10)553

Note that the radix does not <u>have</u> <u>to</u> be specified with .n. If no radix is specified, the current operative one is continued; it is <u>not</u>

reset to 10. It will be 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 the control integers (within parentheses) are interpreted as decimal numbers.

10. Synonym

Format: Name | SYN(dds), A_{24}

The pseudo-operation SYN is used to define a symbol in terms of a bit address, an integer, or a combination of the two. The address A_{24} is evaluated and its value is attached to the symbol in the name field. The dds is attached to the name. If no data description is given, the data properties of the final symbol not in parentheses are transfered to the name. If this symbol has multidimensional properties, they are transferred to the name symbol. Specifically, one may use a SYN to define a symbol as an interior element of a multidimensional array and have the dimensional addressing properties carried along.

Example:

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

In the example the rectangular array A goes from A(0, 0) to A(9, 19); B goes from B(-5,-5) to B(4, 14), A and B using identical storage. A(0, 0) = B(-5, -5); A(1, 0) = B(-4, -5); A(1, 1) = B(-4, -4); etc. A symbol defined as a sum of bit addresses and integers will have its two parts combined at the time when it is entered into the field for which it is intended and will therefore produce the same result that would be produced if its components were explicitly written in the instruction field.

The difficulty of evaluating addresses on SYN cards imposes certain restrictions on the forms of addresses which can be allowed. In the general case (where SYN cards may be in any order) the address of a SYN may contain only one programmer symbol outside of parentheses. 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 which is defined by the location counter as a name of an instruction or of data. For a fuller discussion of SYN cards see Appendix A.

Programming note: An example of the use of SYN and the data properties of a final symbol is the following:

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.

11. Other Restrictions on Address Arithmetic

11.0 DR

Format: Name | DR(dds), (L, L', L",...)

A DR reserves space for data and specifies the dimensions of multidimensional arrays (see section on multidimensional arrays). The amount of space reserved is equal to the field length, as specified or implied in the data description, multiplied by the product of the integers, L, L', L", etc., that is, FL x L x L' x L" x ... bits. DR is error-marked if it has no data description, and normalized floating point is assumed. Each of the programmer symbolized fields, L, L', etc. may contain at most one programmer symbol. If evaluation of the complete field L produces a negative result, the absolute value will be taken.

Example:

	Name	Statement
legal	SAM	DR(B, 20), (12, K+4, L-6)
illegal	JOE	DR, (12-K, K+L, -14) "K+L is not allowed,

but 12-K and -14 are. -14 is the same as 14 in a DR.

11.1 EXT

Format: Name | EXT(L, L')OP, A

The instruction which follows the parentheses after EXT is completely formed. Then bits L to L' inclusive are extracted from it and compiled in the position in the code where the EXT occurs. The remainder of the subject instruction is discarded. The name symbol is assigned a data description of (BU, L'-L+1, 8). The fields L and L' may contain any number of symbolic integers but any bit addresses they contain either must not depend on the location counter or else must be defined by a preceding card.

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

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

11.2 SLC

Format: SLC, A_{24}

The assembly location counter is set to the value of the address of this pseudo-op. The next instruction compiled will be at this address, subject to the various rounding upwards conventions. If A_{24} contains symbols which depend on the location counter for their value, they must be defined by preceding cards. A symbol in the name field of SLC is ignored.

12. Notes on Special Operation Formats

1. LVS: "Load value with sum" Name | LVS, J, A, A', A",···

   J represents the index register whose value field will be filled. A, A', A", etc. are index-type addresses each of which causes a one to be placed in the correct position in the machine address.

2. CW: "Control word" Name CW(OP_2), FWA, C, R

   Intended for the entry of input-output control words. The location counter will be rounded to guarantee that the control word will begin on a full-word address. dds = (BU, 64, 8). The secondary operation, OP_2, provides for eight possible variations of the input-output function as follows:

			Multiple Bit	Chain Bit	Skip Flag
a.	CR:	"Count within record"	0	0	0
b.	CCR:	"Chain counts within record"	0	1	0
c.	CD:	"Count, disregarding record"	1	0	0
d.	CDSC:	"Count, disregarding record, skip, and chain"	1	1	0
e.	SCR:	"Skip, count within record"	0	0	1
f.	SCCR:	"Skip, chain counts within record"	0	1	1
g.	SCD:	"Skip, count, disregarding record"	1	0	1
h.	SCDSC:	"Skip, count, disregarding record, skip, and chain"	1	1	1

3. XW: "Index word" Name | XW, V, C, R, 0-7

The index word will begin at a full-word address.

dds = (BU, 64, 8). The integer 0-7 loads bits 25-27.

4. VF: "Value field" Name | VF, V

The value field will begin at a half-word address.

dds = (B, 25, 1)

5. CF: "Count field" Name | CF, C

The count field will begin at a half-word address.

dds = (BU, 18, 8)

6. RF: "Refill-field" Name |RF, R

The refill field will begin at a half-word address.

dds = (BU, 18, 8)

7. CNOP: "Conditional no-op" Name |CNOP, A_{19}

CNOP may or may not enter a NOP, depending on the value of the assembly location counter. This pseudo-op guarantees that the instruction following CNOP will begin at a full-word address. If a half-word NOP is required to advance the location counter to the next full word, it will be inserted.

8. Progressive indexing. OP(OP_2)(dds), A_{24}(I), OF_7(I')

The six operations which can appear in the OP_2 field in this instruction are:

1. V+I, "Add immediate to value"
2. V-I, "Subtract immediate from value"
3. V+IC, "Add immediate to value, and count"
4. V-IC, "Subtract immediate from value and count"
5. V+ICR, "Add immediate to value, count, and refill"
6. V-ICR, "Subtract immediate from value, count, and refill."

9. END: "End" END, B_{19}

An END card signifies the end of the program. Its location gives the starting point for assigning locations to undefined symbols. If it has an address, B_{19}, a transition card to B_{19} will be punched. A symbol in the name field is ignored on this pseudo-op.

10. TLB: "Terminate loading and branch" | TLB, B_{19}

When this pseudo-op is encountered a transition card is punched immediately to transfer control of the machine to the location B_{19}. The effect is the same as with an END card except that the assembly continues uninterrupted and the remainder of the program is loaded under program control. A symbol in the name field is ignored on this pseudo-op.

11. PRNS: "Print single-spaced" | PRNS

This pseudo-op causes the assembly listing to be printed single-spaced. The listing is always double-spaced unless this is given.

12. PRND: "Print double-spaced" | PRND

This pseudo-op causes the assembly listing to be printed double-spaced. It restores print conditions to normal after a PRNS.

13. PUNFUL: "Punch full cards" | PUNFUL

Full cards (80 columns) are punched, without check sum, FWA, ID, etc.

14. PUNNOR: "Punch normally" | PUNNOR

This pseudo-op restores punching to normal after a PUNFUL.

15. SKIP: "Skip paper" | SKIP, i

If i = 0 or blank, this causes the assembly listing to restore the paper immediately. If $i \neq 0$, a half-page skip will result. Note that in STRAP-2, SKIP,i will mean "skip to line number i," as opposed to "channel i" on the 704.

16. PUNID: "Punch ID" | PUNID,XXXXXXXX

The first 8 characters after the comma are picked up and punched in columns 73-80 of the binary deck. This card should be used to identify each assembly.

13. Miscellaneous Notes

1. Instruction data description.

   Reference to a machine instruction by another instruction requiring a data description will give a dds of (BU, 64, 8) or (BU, 32, 8) depending on whether the operation referred to occupies a full or a half word. This dds can, of course, be overruled.

2. Blanks.

   Blanks are ignored in all fields except in entering alphabetic information. They have no meaning whatever in any other field. Blank cards are ignored. An END card must be used to signify the end of the program.

3. Parentheses within Parentheses.

   In Strap 1 it is a general rule that parentheses may not appear within parentheses. Programmer symbolized fields appearing within parentheses are therefore restricted somewhat in that they must always have radix 10, may not contain array specifications, nor may they have general parenthetical integer entries appended to them.

4. Null fields.

   Certain subfields in any operation format may be omitted, and they are then said to be null fields. A right to left drop-out feature operates in assembly. If the rightmost subfield for a format is omitted it is compiled as a zero field. If the two rightmost fields are omitted they are both compiled as zero, etc. A sub-field in the interior of a format is made null by writing only the comma which ends the field thus: OP, , A. Index modifiers I and K are made null by simple omission.

5. Supression of error marks.

Error marks, except for mispunch indications, can be suppressed for any statement by prefixing the op symbol with a dollar sign. Thus $OP, A(I) will suppress error marks which would otherwise be printed in connection with compiling that operation, but only that one.

6. Name with blank statement field.

If a card contains only a name, the statement field being left completely blank or containing comments only, it is treated as a data reservation for one normalized floating point word. That is, the statement DR(N), (1) is assumed in this event by Strap 1.

7. Undefined symbols.

If a symbol appears in a programmer symbolized field, but never appears in the name field of any card, it is undefined. Undefined symbols are assumed to represent normalized floating point words and are assigned succeeding full-word locations beginning with the first one after the END instruction.

14. System Symbols

System symbols are symbols whose values are fixed in the compiler. They are identified in programmer symbolized fields by the fact that the first character of a system symbol is a dollar sign, which is a character that can never appear in a programmer symbol. Note that a dollar sign prefix in the operation field is a signal to suppress error marks and that the indicator symbols, when inserted into the "branch on indicator" instructions, do not have the dollar sign prefix. System symbols which represent special registers in memory or special bits are bit addresses; all others are integers. System symbols may appear in arithmetic expressions in programmer symbolized fields, where in cases to which restrictions apply, they can be considered in the same class as numeric entries since their values are immediately available whenever needed.

The system symbols are:

1. \$0 to \$15, identical to \$X0 to \$X15, are index registers 0 to 15, addresses 16.0 to 31.0. For example, \$5 (or \$X5) will produce the correct index field of 5 in an I-or J-field or the address 21.0 in an A-field.

2. Other special registers.

Location Word No.	Mnemonic	Name
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

Revised
12/16/59

2. Other special registers (continued)

Location Word No.	Mnemonic	Name
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
12.21	$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.

4. Location counter. Whenever the dollar sign by itself appears in a programmer symbolized field, it represents the value of the location counter at the beginning of that instruction. In effect this is the location of the instruction in which it appears if that instruction actually compiles space in the program. Example: the instruction, B, $-2. means branch to the instruction which begins two full words before. Note that B, $+.32 means branch to the next instruction, effectively no operation.

Note: All of the system symbols in classes 1, 2, 3, and 4 are bit addresses and are assigned standard data descriptions with mode binary unsigned, byte size eight, and field length depending on the length of the register.

5. Input-output addresses. Some of the system symbols for input-output addresses may have different values at different installations, since the channel to which a particular piece of equipment is connected is arbitrary. The symbols may represent either channel addresses or unit addresses, depending on the configuration of the input-output system.

System Symbol	Meaning
$PCH	Punch
$PRT	Printer
$RDR	Reader
$DISK	Disk unit
$C0, $C1,···,$Ck	Channel 0, Channel 1,···,Channel k
$T0, $T1,···,$Tk	Tape 0, Tape 1,···, Tape k
$IQS	Inquiry station
$CNSL	Console

If more than one punch, printer, console or any other input-output unit is attached to the machine, the same numbering convention used in channel and tape addresses is adopted, where $CNSL = $CNSL0, and so on. For example one may have $PRT0, $PRT1, $PRT2, etc.

15. General Data Entry

The use of the pseudo-operation DD (Data Definition) enables the programmer to enter data into a program in a variety of forms.

Among the possibilities which exist are decimal to floating binary conversion, either normalized or unnormalized, conversion of decimal fraction to binary fraction in fixed point, integer to integer conversion from any radix from 2 to 10 or 16 to a radix of either 2 or 10 and conversion of alphabetic information to binary-coded forms. The pseudo-operation DDI (Data Definition Immediate) is intended for defining data to be used in the address of immediate operations. All the features listed above, with the obvious exception of the floating point conversion, are also available with DDI. The method of use of the DD will be described first, and then the minor differences between DD and DDI will be listed.

15.1. DD

Format: Name | (EM)DD(dds), D, D', D",···.

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 only the first having a name. The effect produced is exactly the same as if the entry mode, op, and data description were repeated on separate cards with only one D-field per instruction and blank name fields. If one wishes to name the separate entries D, D', D", etc., indeed it is necessary to write each one on a separate card since the name of a DD is given the address value of the first bit of the first D-field. Programmer symbols may not appear in the main body of a D-field, but only in general parenthetical integer entry fields which are attached to the ends of D-fields. (Note: Since each D-field is essentially a separate major field, the parenthetical entry counts bits from the beginning of the D with which it is written.) In the main portion of a D-field various letters and symbols have fixed meanings not subject to programmer control.

15.2. Entry Mode

The entry mode gives information about the form in which the data appears on the card; it may also have some implications about the form to which it is converted and stored. An entry mode may appear before the DD as shown in the format. Those not concerned with entry of alphabetic information may also be used at the beginning of individual D-fields. It is not always necessary to specify the entry mode explicitly.

There are four different entry modes:

1. (R) Radix. The radix has already been explained for the case of address arithmetic. In the case of data entry it can be used with <u>integers only</u>; a decimal point or a 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.
2. (Fn) (Fn) implies that the data is written with a decimal radix and is to be converted to binary, and may include a decimal fraction portion to be converted to a binary fraction of length n bits.

   The (decimal) integer n following F specifies the number of fractional bits to be left to the right of the binary point when the number, or numbers, which follow are converted.

3. (Az) Alphabetic conversion. This entry mode must precede the DD, and only one address field "D" is allowed per statement. The Hollerith characters beginning with the one after the comma which ends the op field are converted to IBM tape BCD until the character "z" is reached. Note that tape BCD is somewhat different from internal 704 BCD. The byte size of converted characters may range from 1 through 12 in a DD, 4 through 12 in a DDI, and is specified by the dds. Leading zeroes are inserted in each byte for BS > 6, and leading bits are truncated from each byte for BS < 6. The byte size compiled in an operation referring to the data is set to **the specified byte size modulo 8.**

   The terminating character "z" itself is not included. It may be any legal Hollerith character except blank,), ;, or ' . Blanks occurring within the field to be converted are retained and correctly stored. The characters are counted by Strap 1 and the location counter properly advanced.

4. (IQSz) Inquiry station conversion. This entry mode operates exactly as (A) except that the Hollerith characters are converted to the 7-bit inquiry station code, and therefore 7 is the magic number separating truncation from addition of leading zeroes. Although the IQS code includes a large number of special characters, Strap 1 is limited to the ones which can be entered by means of IBM off-line card and tape equipment.

15.3. The Form of Decimal Numbers

Decimal numbers may be written in fixed or floating point form, with or without a decimal point. The general form is

± xxxx···xx.xx···xx E± y y y .

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 572.34×10^{-57}. Parts of the general form which are not necessary for writing a number may be omitted, thus:

a.	± xxxxxx···xx	integer
b.	± xx···xx.xx··xx	decimal fraction
c.	± xx···xx E±yyy	integer times power of 10

A plus sign is understood if omitted. The decimal point can be in any position in the number. The portion of the number symbolized above by x's is limited to 20* digits; that symbolized by y's to 3 digits (but recall that floating point numbers in Stretch are limited to a range of 10^{616} to 10^{-616}).

15.4. Insertion of Specific Fields

1. Exponent Entry: X ± n

The letter "X" may be used to enter any arbitrary exponent into a floating point word. n is a decimal integer which is converted to binary and which replaces any exponent previously calculated.

2. Sign Byte Entry: Sn

The letter "S" is used to enter a sign byte into data. n is an octal integer which is evaluated and which is "OR"ed in with any sign byte previously calculated. The sign byte generated depends on the byte size according to the following table:

* Number of digits > 20 is permitted only when the radix is < 10.

Byte Size	Sign Byte
1	S
2	ST
3	STU
4	STUV
5	ZSTUV
6	ZZSTUV
7	ZZZSTUV
8	ZZZZSTUV

where Z is a zone bit,

S is the sign bit,

T, U, V are the flag bits.

15.5. 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.

1. 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 shown below.

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 (and 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 $\frac{27.9}{31.4 \times 12 \times 14 \times 10^5}$,

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

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	$\log_e 2$
5.	∅R	$\sqrt[2]{X}$, where X is the field containing the ∅R symbol*

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

DD(N), 4 * ∅PI * 1E - 7.

It is to be especially noted that in Strap 1 this arithmetic feature is available with the normalized floating point mode only.

* The ∅R symbol may be repeated in a DD field to indicate the nth root, where n is a positive integral power of 2. If the ∅R symbol appears m times in a field, X, STRAP-1 will calculate $\sqrt[2^m]{X}$ as the final value of the field. If arithmetic symbols are included in the field, all arithmetic will be done first, then the root of the arithmetic result will be extracted as the final step in assembling the data field.

2. 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. (It is 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. Since 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 with example b except that in the second field the op 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.

a. If the number is entered as an integer, (F0) is understood

b. 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, placed in the fraction with its rightmost bit in position 59* 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), 17.0

In this example 17.0 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-op will have

* Bit positions are numbered 0-63 in any one word of memory

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.* Hence a normalized integer can be assigned use mode "unnormalized."

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

3. 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 since 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. As usual 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

* Because of the decimal point without an (Fn) entry.

for 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.

b. (2)DD(B, 16, 8) 110101S377, (10) -972, 11101110S201

Binary entry, overruled in only the second D field.

c. (F12)DD(B, 24), 1.324E3, -72.1E-4, 3.4E-4S1

Implied byte size is 1.

d. DD(B), 1489, -1272, 1491, (F13) -972.16, 13948S1, 12E5

Decimal integers except where a field entry mode is written.

4. <u>Binary Unsigned VFL</u>

Formats: (Fn)DD(BU , FL, BS), xx···x.x···xE±yy

DD(BU, FL, BS), (Fn) xx···x.x···xE±yy

(R)DD(BU, FL, BS), xx···xx

DD(BU, FL, BS), (R) xx···xx

(Az)DD(BU, FL, BS), alphabetic information to "z"

(IQSz)DD(BU, FL, BS), alphabetic information to "z"

Numerical entry is exactly the same as in binary signed data except that no sign byte is formed and if the byte size is left out of the dds, it is set to 8. Any sign or sign byte (with "S") written with mode BU is ignored. The two alphabetic modes are permitted here; they are explained in the section under "Entry Modes." Note that the alphabetic entry mode must precede the DD, that there can be only one D field per statement and that if the field length is omitted it is set equal to the byte size.

Examples:

a. (F13)DD(BU, 30), 17.2, 183, (8) 70707

b. (A*)DD(BU, 48, 6), GLORIOUS FRIDAY, THE 13TH.*

The mode and field length have no effect on the conversion and storage; they are used in compiling instructions which refer to the name of this statement. Field length 48 indicates that the programmer wants to process these characters in groups of 8.

c. (IQSS)DD(BU, 32, 8) DOG EAT DOG S

5. <u>Decimal Signed VFL</u>

Formats: (R)DD(D, FL, BS), ± xx...xxx Sn

DD(D, FL, BS), ±(R) xx···xx Sn

DD(D, FL, BS), ± xx···xxEyy Sn

(Fn) has no meaning for mode = D or DU.

The two decimal modes in DD and DDI statements represent the only cases in which Strap 1 converts numbers to an internal decimal radix. This conversion is limited a bit more in being available only from integers to integers. The radix entry mode indicates the radix in which the numbers are written on the card. Thus it is possible to write an integer in binary or octal and have it converted to decimal for machine use. If no entry mode is given, decimal to decimal is implied and the E notation can be used to 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 four-bit sign byte is formed. Decimal to decimal conversion.

b. (2)DD(D, 20), 111010001101S7

Binary to decimal conversion. BS = 4.

c. DD(D, , 8), 432E3

Decimal to decimal conversion. FL = 64. Four binary zeros are inserted in the zone positions of each byte.

6. 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 this data will be compiled as decimal operations. For alphabetic entry implied field length is equal to byte size.

Examples:

a. DD(DU), 8430051, (8) 77241, 82E10

FL = 64, BS = 4. An octal to decimal conversion is inserted between two decimal to decimal conversions.

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

FL = 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.

15.6 <u>DDI</u>

Format: Name | (EM)DDI(dds), D

DDI is used to define a symbol which is used at some other point in the program as the address of an immediate operation. It compiles no space at its location in the program, and in fact its position in the program is of no importance whatever. It may have <u>only</u> <u>one</u> D field, as shown in the format. The rules for writing the data field are the same as for DD with some obvious and relatively minor changes. Neither of the floating point modes can be used with DDI. The upper limit on field length is 24 instead of 64, and in every case where a field length of 64 is implied for a DD, a field length of 24 is implied for a DDI. A general parenthetical integer entry may <u>not</u> be appended to the end of the data field as it can in DD statements.

If a DDI has a field length of less than 24, the number which 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 will be zero, but they may be loaded by means of a general 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 were defined or what the mode of the operation itself is. All numeric entries in such an address field are handled exactly as other addresses and converted to binary, never to decimal. Therefore, the only way to get a decimal number into the address field of an immediate op, without writing it in the Stretch BCD code explicitly, is to symbolize it and use a DDI. Care should be exercised in address arithmetic among signed numbers, since the sign byte is compiled as such and does not participate in the arithmetic as a sign.

Examples:

1.

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

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

2.

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. Since 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, 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 one and compiles a sign byte 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, although it is performed as shown above.

APPENDIX A

Restrictions on Addresses in SYN, DR, and SLC

In order to finish assembling a program in a finite length of time using a finite storage, some generality has been sacrificed in the address arithmetic which can be allowed with the three pseudo-ops, SYN, DR and SLC. The underlying reason for their different treatment is that their addresses must be evaluated sooner in the program than those of other operations. Strap 1 is a three pass assembly in which the first two passes are concerned primarily with assigning values or addresses to symbols and the last with forming the machine code and revealing it to the outside world in some form of a listing and stretch column binary cards.

During pass 1 any SYN address containing only numerical entries, or numerical entries plus system symbols, can be evaluated immediately. System symbols can always be considered in the same class as integers and bit addresses since Strap-1 can evaluate them immediately. A SYN address which contains symbolic information cannot be. Strap-1 can, however, store the symbol from the name field and one symbol from the address (always the one on which the mode of the name symbol will depend if not overruled) for future reference. The same restriction applies to each of the elements of the address of a DR (each "L" in the notation of this paper). The restriction on DR addresses is really the crucial one at this point, because the DR address is completely evaluated at the end of pass 1. Therefore, each element of the DR address and the SYN or chain of SYN's defining the symbolic portion of such an address must be evaluable from a numeric part plus a single symbol. Since all of this information is stored in tables permanently and is always available to the assembly program, the order of the cards is of no importance. At the end of pass 1 an evaluation is made of all symbols defined in this simple manner, and as stated above a DR must be completely defined at this point.

During pass 2 locations are assigned to all symbols which depend on the location counter for their value, and a new attempt is made to evaluate SYN addresses not evaluated in the first pass. At this point the order of the cards can play an important role. If all of the symbols appearing in an address have appeared previously in the name field and if they in turn are defined by symbols which have appeared previously (or by the location counter) then the address can be evaluated no matter how many programmer symbols it contains or what signs they may be preceded by. If there are two or more symbols in a SYN address still not evaluated when the card is encountered in pass 2, the name symbol may never be completely evaluated and will elicit an error indication whenever it is used. If only one symbol remains not evaluated at this point then eventual success in evaluating the name symbol depends on position in the address and later evaluation. At the end of pass 2 an attempt is made to tie up all the loose ends still dangling from this particular rats' nest. If any symbols remain not evaluated after this procedure, a last try will be made when the SYN card is encountered in pass 3. But this may be too late, depending on the order of the cards.

From the preceding discussion it is clear that the address of an SLC card must be evaluable when it is encountered in pass 2. The same rules apply to it as to the address of a SYN card which can be completely evaluated at that point. However, if the address of an SLC cannot be evaluated, all is lost an no attempt is made to tidy up at the end of the pass. This last point also applies to the L and L' of EXT(L, L'). Since they are used to compute the amount to advance the location counter, they must be available when the card is encountered in pass 2.

The program on the following page is given here to give more examples of what can be assembled by STRAP-1. Each line of type represents one card of input to STRAP-1 and the comments associated with the instructions explain more what STRAP-1 does than explain what Stretch does.

```
NAME    STATEMENT

ABE     L, ZEP "Normalized fl. pt. operation since ZEP is undefined.
        +, ZEKE + 1 "Add 2nd no.; ST, TOBY; LX, $10, SAM "Many ops. per card.
BILL    LV, I, SAM - .32 "127.0 To Value field of $x9 or $9
CHICK   LVS, $5, I, I + 1.0; L(BU,3,8), TOBY + .61, 20 "Overrule dds of TOBY
DUD     SYN(N), ISH(L-12,L,20) "Array properties of ISH go to DUD
        CTI0101, MAC "MAC is an immediate address defined by a DDI
MAC     DDI(BU,3,8), (2)110 "MAC is a 3 bit number = 110_2
        BRZZ, CHICK + .32(1.0) "Index (CHICK +.32) with $x1
N       SYN, 120-M; B, ABE .32 "ABE .32 is the same as ABE + .32
        L(V+I)(D,24,6), SAM + 13.121(I-3), L-10($x3) "Progressive Indexing.
I       SYN, $x9 "I is index word $x9 with dds = (BU,64,8)
JOE     ST(V-IC)(B), ZEKE + M-4 "Progressive indexing and address arithmetic.
        LWF, EGBERT (1) "EGBERT (1) = EGBERT + .25
        LFT, PAT(20) + 12.34(I-6.0) "An exercise for the reader.
ISH     DR(U), (L,M-4,177-N) "Reserve L x (M-4) x (177-N) unnormalized words.
PAT     DR(B,25,4), (27) "PAT is a block of 25 x 27 = 675 bits.
M       SYN, L + 10
EGBERT  (F5)DD(B,25,4), 12S3, -14.6, (8)7634, 15.3
        VF, 127.0 "The location of this VF is SAM - .32
SAM     XW, 900.0, L+M-N, $ "The refill address = SAM
L       SYN, 50 "The base of the SYN chain, M = 60, N = 60
TOBY    "Blank instruction reserves one normalized word.
ZEKE    DD(N), 13, -14.5732E-101S7, 1/13 + 7/9 + $PIS5; END
```

APPENDIX B

STRAP MNEMONICS

The following list of mnemonics may be used with STRAP - 1 (and 2). A symbolic description of the mnemonic is also given to assist the programmer. The symbols used to symbolize the operations are defined for each section. One should note that the same letter ("a" and "m" for example) has a different definition for floating point and for VFL. The definition for each set should be read carefully. A more detailed description of the operation may be obtained from the IBM Stretch Manual of Operations.

A specific title to each mnemonic is not given in cases where the mnemonic is derived from the basic operation by changing the sign and absolute modifiers. In some cases no titles are given.

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

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.

Memory operands

m = bits (0 - 59) of the memory 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 memory word.

SB(m) = bits (60 - 63) of the memory word.

Fl(m) = bits (61 - 63) of the memory word.

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

$RM = Remainder operand.

Revised January 5, 1960

Floating Point operations

ADD

+	$a+m \longrightarrow a$	1. b is unchanged
-	$a-m \longrightarrow a$	2. **Fl**(a) is unchanged
+A	$a+\lvert m\rvert \longrightarrow a$	
-A	$a-\lvert m\rvert \longrightarrow a$	

TO MEMORY ADD

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

ADD TO FRACTION

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

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 un-
E+A	$e(ab)+\lvert e(m)\rvert \longrightarrow e(ab)$	normalized unless the normalized
E-A	$e(ab)-\lvert e(m)\rvert \longrightarrow e(ab)$	mode is requested by referring to normalized data or by using the dds = (N).

*ADD IMMEDIATE TO EXPONENT

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

*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. 3. The operation is not affected by
SHFA	$f(ab)\cdot 2^{\lvert M\rvert} \longrightarrow f(ab)$	the normalized modifier.
SHFNA	$f(ab)\cdot 2^{-\lvert M\rvert} \longrightarrow f(ab)$	4. The exponent is not adjusted for the shift. e(a) is unchanged.
SHFL	$f(ab)\cdot 2^{\lvert M\rvert} \longrightarrow f(ab)$	5. On a right shift, zeroes are
SHFR	$f(ab)\cdot 2^{-\lvert M\rvert} \longrightarrow f(ab)$	introduced in bit 12.

* These operations have the format: OP(dds), A_{12} (I)

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 $R < 0$. e(a) i
+MGA	$R=\|a\|+\|m\|$	unchanged whether $R < 0$ or not.
-MGA	$R=\|a\|-\|m\|$	3. s(a) is unchanged in either case.

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 $R < 0$. e(a) is
D+MGA	$R=\|ab\|+\|m\|$	unchanged whether $R < 0$ or not.
D-MGA	$R=\|ab\|-\|m\|$	3. s(a) is unchanged in either case.

TO MEMORY ADD MAGNITUDE

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. b is unchanged.
*N	$a \cdot -m \longrightarrow a$	
*A	$a \cdot \|m\| \longrightarrow a$	
*NA	$a \cdot -\|m\| \longrightarrow a$	

DOUBLE MULTIPLY

D*	$a \cdot m \longrightarrow ab$	1. (108 - 127) of acc. 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

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

DIVIDE

/	$a/m \longrightarrow a$	1. No remainder is generated.
/N	$a/-m \longrightarrow a$	2. Quotient rounded to 48 bits.
/A	$a/\lvert m\rvert \longrightarrow a$	3. Pre-normalization of the operands is independent of the normalization modifier.
/NA	$a/-\lvert m\rvert \longrightarrow a$	4. b is unchanged.

RECIPROCAL DIVIDE

R/	$m/a \longrightarrow a$	1. Performed similar to divide.
R/N	$-m/a \longrightarrow a$	2. b is unchanged.
R/A	$\lvert m\rvert/a \longrightarrow a$	
R/NA	$-\lvert m\rvert/a \longrightarrow a$	

DOUBLE DIVIDE

D/	$ab/m \longrightarrow a+1$	1. Remainder in $RM.
D/N	$ab/-m \longrightarrow a+1$	2. $0 \longrightarrow b(61\text{-}107)$
D/A	$ab/\lvert m\rvert \longrightarrow a+1$	3. No rounding.
D/NA	$ab/-\lvert m\rvert \longrightarrow a+1$	4. SB(a) $\longrightarrow$ SB($RM).

STORE ROOT

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

LOAD

L	$m \longrightarrow a$	1. $0 \longrightarrow$ F1(a).
LN	$-m \longrightarrow a$	2. b is unchanged.
LA	$\lvert m\rvert \longrightarrow a$	
LNA	$-\lvert m\rvert \longrightarrow a$	

DOUBLE LOAD

DL	$m \longrightarrow a$	1. $0 \longrightarrow b$.
DLN	$-m \longrightarrow a$	2. $0 \longrightarrow$ F1(a).
DLA	$\lvert m\rvert \longrightarrow a$	
DLNA	$-\lvert m\rvert \longrightarrow a$	

LOAD WITH FLAG BITS

LWF	$m \longrightarrow a$	1. $Fl(m) \rightarrow Fl(a)$.
LWFN	$-m \longrightarrow a$	
LWFA	$\lvert m \rvert \longrightarrow a$	
LWFNA	$-\lvert m \rvert \longrightarrow a$	

DOUBLE LOAD WITH FLAG BITS

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

LOAD FACTOR

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

STORE

ST	$a \longrightarrow m$	1. $Fl(a) \rightarrow Fl(m)$!!
STN	$-a \longrightarrow m$	2. a is unchanged.
STA	$\lvert a \rvert \longrightarrow m$	
STNA	$-\lvert a \rvert \longrightarrow m$	

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	$-a \longrightarrow m$	
SRDA	$\lvert a \rvert \longrightarrow m$	
SRDNA	$-\lvert a \rvert \longrightarrow m$	2. $Fl(a) \rightarrow Fl(m)$.

STORE LOW ORDER

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

COMPARE

K	a:m
KN	a:-m
KA	a:|m|
KNA	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 $a > m$
2. Zero exponents of different sign are considered equal.
3. If the exponent difference is 48 the larger of the numbers is per sign and exponents regardless of fractions.

COMPARE FOR RANGE

KR	a:m
KRN	a:-m
KRA	a:|m|
KRNA	a:-|m|

1. If AH is off prior to this op, no indicators will be changed.
2. If AH is on:
   AL is unchanged.
   AE is set to one if $a < m$.
   AH is set to one if $a \geq m$.

COMPARE MAGNITUDE

KMG	|a|:m
KMGN	|a|:-m
KMGA	|a|:|m|
KMGNA	|a|:-|m|

1. Same as COMPARE, except for accumulator comparand.

COMPARE MAGNITUDE FOR RANGE

KMGR	|a|:m
KMGRN	|a|:-m
KMGRA	|a|:|m|
KMGRNA	|a|:-|m|

1. Same as COMPARE FOR RANGE, except for accumulator comparand.

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.

Memory operands

m = the memory 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;
3. length is defined by the field length in the dds;
4. sign is bit s in the sign byte.

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

The unsigned memory 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)	1. If the sign changes, bits to the right of the offset are complemented.
-	$a-m \longrightarrow a$		

TO MEMORY ADD

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

ADD TO MAGNITUDE

+MG	$R=\bar{a}+m$	(I)	1. $R \longrightarrow \bar{a}$ if $R \geq 0$. 2. $0 \longrightarrow$ entire acc. if $R < 0$. 3. s(a) is not changed by these operations.
-MG	$R=\bar{a}-m$		

TO MEMORY ADD MAGNITUDE

M+MG	$R=m+\bar{a}$		1. $R \longrightarrow m$ if $s(R) = s(m)$. 2. $0 \longrightarrow m$ if $s(R) \neq s(m)$. 3. s(m) is not changed.
M-MG	$R=m-\bar{a}$		

MULTIPLY

*	$a \cdot m \longrightarrow a_{20}$	(I)	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 $\leq$ 48 bits in binary multiply. 4. The portion of the accumulator not containing the product is set to zero.
*N	$a \cdot -m \longrightarrow a_{20}$		

MULTIPLY FACTOR AND ADD

*+	$m \cdot (¢FT)+a \longrightarrow a$	(I)	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.
*N	$-m \cdot (¢FT)+a \longrightarrow a$		

DIVIDE

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

LOAD

L	$m \longrightarrow a$	(I)	1. $0 \longrightarrow$ Fl(a). 2. The entire acc. is cleared before the load.
LN	$-m \longrightarrow a$		

LOAD WITH FLAG BITS

LWF	$m \longrightarrow a$	(I)	1. Fl(m) $\longrightarrow$ Fl(a).
LWFN	$-m \longrightarrow a$		

LOAD FACTOR

LFT	$m \longrightarrow$ \$FT	(I)	1. $0 \longrightarrow$ (61 - 63)\$FT. 2. The offset field is ignored.
LFTN	$-m \longrightarrow$ \$FT		

LOAD TRANSIT AND SET

LTRS	$m \longrightarrow$ \$TR	(I)	1. Offset $\longrightarrow$ \$AOC. 2. $11_2 \longrightarrow$ 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.
LTRSN	$-m \longrightarrow$ \$TR		

STORE

ST	$a \longrightarrow m$	1. $SB(a) \longrightarrow SB(m)$.
STN	$-a \longrightarrow m$	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 stores, 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	$m+1 \longrightarrow m$	1. The one is added to the low order byte.
M-1	$m-1 \longrightarrow m$	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 off-set 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 op, it is executed as a NOP. 2. If AH is on: AL is unchanged. AE is set to one if $a < m$ AH is set to 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 memory 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 memory has byte size (BS) less than 8 then (8-BS) lead zeroes are added to each byte from memory before the connect takes place. However, the memory operand is not changed in C xxxx or CT xxxx.

CONNECT TO ACCUMULATOR

$Cx_1x_2x_3x_4$	Result $\rightarrow$ a

CONNECT TO MEMORY

$CMx_1x_2x_3x_4$	Result $\rightarrow$ 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 and is summarized below:

Let: m = a bit from memory (may be an inserted lead 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 = " " " m = 0 and a = 1
x_3 = " " " m = 1 and a = 0
x_4 = " " " 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 one writes $CIx_1x_2x_3x_4$, $CTIx_1x_2x_3x_4$ and LFI.

ø<u>AOC</u> = All ones count register.

ø<u>LZC</u> = Left zeroes count register.

After a connective operation the two registers øAOC and øLZC contain the indicated counts of the result. Since the result may not occupy the entire accumulator, øAOC and øLZC may not give the total count of ones and left zeroes of the accumulator. However, these counts always give the correct count in CM or SF.

Convert Instructions (VFL operations)

Definitions:

a_D = accumulator in decimal, 4 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 = memory operand in binary with specified byte size and field length.

m_D = memory operand in decimal with specified byte size and field length.

$TR = 64 bit transit register with a sign byte in the rightmost 4 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	
CVN	$-a_D \longrightarrow a_{B68}$		2. The entire accumulator to the left of the offset is used.
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	$m_D \longrightarrow a_B$	(I)	1. $s(m) \rightarrow s(a)$
or	$m_B \longrightarrow a_D$		2. $0 \rightarrow Fl(a)$
			3. The entire accumulator is cleared before the load.
LCVN	$-m_D \longrightarrow a_B$	(I)	
or	$-m_B \longrightarrow a_D$		

LOAD TRANSIT CONVERTED

LTRCV	$m_D \longrightarrow \$TR_B$	(I)	
or	$m_B \longrightarrow \$TR_D$		
LTRCVN	$-m_D \longrightarrow \$TR_B$	(I)	
or	$-m_B \longrightarrow \$TR_D$		

1. The acc. and offset are ignored.
2. $0 \longrightarrow$ **Fl**($TR)
3. s(m) $\longrightarrow$ s($TR)
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 ± I, V ± IC, or V ± ICR.

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

Note: 1. The original value field of J is the effective address of op.

2. A_{24} is the immediate operand specified by J in V ± I, etc., and the value field of J is incremented by ± A_{24} according to ± I. The incrementing takes place subsequent to note 1 above.

3. J may be $XO.

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.

Memory word operands

m = bits (0 - 63) of a memory 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.

Note: For clarity the titles to the indexing and the branch operations have been omitted.

Note: The indicators: XF, XCZ, XVLZ, XVZ, and XVGZ are set by all of the direct and immediate index operations except for: 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.

Revised
12/16/59

Direct Index Arithmetic		OP, J, A_{19} (I) or OP, J, A_{18}(I) *
*LX	$m \longrightarrow J$	1. $M = A_{19}$ (I)
LV	$V(m) \longrightarrow V$	
LC	$V(m) \longrightarrow C$	2. $m = (M)$
LR	$V(m) \longrightarrow R$	3. C_2 = The count field of J after modification
*SX	$J \longrightarrow m$	
SV	$V \longrightarrow V(m)$	
SC	$C \longrightarrow V(m)$	1. $0 \longrightarrow (18 - 24)$ of m.
SR	$R \longrightarrow V(m)$	1. $0 \longrightarrow (18 - 24)$ of m.
V+	$V+V(m) \longrightarrow V$	1. There is no V - etc.
V+C	$\begin{cases} V+V(m) \longrightarrow V \\ C-1 \longrightarrow C_2 \end{cases}$	
V+CR	$\begin{cases} V+V(m) \longrightarrow V \\ C-1 \longrightarrow C_2 \\ (R) \longrightarrow (J) \text{ if } C_2 = 0 \end{cases}$	
SVA	$V \longrightarrow V(m)$	1. V is truncated to 18, 19, or 24 bits, as is appropriate for the instruction containing V(m).
LVE	$(M)^n \longrightarrow V$	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	$\begin{cases} J \longrightarrow (R(\$XO)) \\ m \longrightarrow J \\ M \longrightarrow R(\$XO) \end{cases}$	1. used for saving and restoring index registers.
LVS	(special format): LVS, J, $A^1, A^2, \ldots, A^n$	
	$\sum_{i=1}^{n} V(A^i) \longrightarrow V(J)$	1. The sum may include any subset of the index words. 2. No indexing of the address field is allowed.

* For LX, SX, and RNX, the format is: OP, J, A_{18} (I)

Revised
12/16/59

Immediate Index Arithmetic   OP, J, A_{19} or OP, J, A_{18}*

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	$-A \longrightarrow V$	1. (19 - 24) of V are set to 0.
LVI	$A \longrightarrow V$	1. (19 - 24) of V are set to 0.
* LCI	$A \longrightarrow C$	
* LRI	$A \longrightarrow R$	
V+I	$V+A \longrightarrow V$	1. (19 - 24) of V are unchanged.
V-I	$V-A \longrightarrow V$	1. (19 - 24) of V are unchanged.
V+IC	$\begin{cases} V+A \longrightarrow V \\ C-1 \longrightarrow C \end{cases}$	1. " " " " "
V-IC	$\begin{cases} V-A \longrightarrow V \\ C-1 \longrightarrow C \end{cases}$	1. " " " " "
V+ICR	$\begin{cases} V+A \longrightarrow V \\ C-1 \longrightarrow C_2 \\ (R) \longrightarrow (J) \text{ if } C_2 = 0 \end{cases}$	1. " " " " "
V-ICR	$\begin{cases} V-A \longrightarrow V \\ C-1 \longrightarrow C_2 \\ (R) \longrightarrow (J) \text{ if } C_2 = 0 \end{cases}$	1. (19 - 24) of V are unchanged.
*C+I	$C+A \longrightarrow C_2$	
*C-I	$C-A \longrightarrow C_2$	
KVI	(0 - 18) of V:A	1. (19 - 24) of V are compared with zeroes.
KVNI	(0 - 18) of V:A	1. (19 - 23) of V are compared with zeroes. and (24) of V is compared with 1(minus).
*KCI	C:A	

* For LCI, LRI, C+I, C-I, and KCI, the format is: OP, J, A_{18}

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

CB	$C_1 - 1 \longrightarrow C_2$	
	$IC_1 + 0.32 \rightarrow IC$	if $C_2 = 0$
	$M \longrightarrow IC$	if $C_2 \neq 0$
CBR	$C_1 - 1 \longrightarrow C_2$	
	$IC_1 + 0.32 \rightarrow IC$ and $(R) \rightarrow (J)$	if $C_2 = 0$
	$M \longrightarrow IC$	if $C_2 \neq 0$
CBZ	$C_1 - 1 \longrightarrow C_2$	
	$IC_1 + 0.32 \rightarrow IC$	if $C_2 \neq 0$
	$M \longrightarrow IC$	if $C_2 = 0$
CBRZ	$C_1 - 1 \longrightarrow C_2$	
	$IC_1 + 0.32$ 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 above 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 above 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	$\left\{\begin{array}{l} M \longrightarrow IC \\ M + IC_1 + 0.32 \longrightarrow IC \end{array}\right.$
BR	
BE	$\left\{\begin{array}{l} \text{Enable} \\ M \longrightarrow IC \end{array}\right.$
BD	$\left\{\begin{array}{l} \text{Disable} \\ M \longrightarrow IC \end{array}\right.$
BEW	$\left\{\begin{array}{l} \text{Enable} \\ \text{Wait} \\ M \longrightarrow IC \end{array}\right.$
NOP	$IC_1 + 0.32 \longrightarrow IC$

1. The unconditional branch orders are the only branch orders which allow a 4 bit index field, I. The conditional branch orders 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 $m_1 = 1$
BZB	$IC_1 + 0.32 \longrightarrow IC$ if $m_1 = 1$
	$M_2 \longrightarrow IC$ if $m_1 = 0$

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.

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

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

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

Note: (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

MNEMONIC	NAME	BIT ADDRESS
	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
XPO	Exponent Overflow ($EXP \geq 2^{11}$)	11.28
XPH	Exponent High ($2^{10} \leq EXP < 2^{11}$)	11.29
XPM	Exponent Medium ($2^{8} \leq EXP < 2^{10}$)	11.30
XPL	Exponent Low ($2^{5} \leq EXP < 2^{8}$)	11.31
XPN	Exponent High Negative ($-2^{11} < EXP \leq -2^{10}$)	11.32
XPU	Exponent Underflow ($EXP \leq -2^{11}$)	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

MNEMONIC	NAME	BIT ADDRESS
	PROGRAMMER INDICATORS	
PG0 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, $A_{18}(I)$, $A'_{18}(I')$

Note: (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 ≤ 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 $(M_1) \longrightarrow (M_2)$
$(M_1+1) \xrightarrow[\text{etc.}]{} (M_2+1)$

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 $(M_1) \longrightarrow (M_2)$
$(M_1+1) \xrightarrow[\text{etc.}]{} (M_2+1)$

TRANSMIT BACKWARDS

TB $(M_1) \longrightarrow (M_2)$
$(M_1-1) \xrightarrow[\text{etc.}]{} (M_2-1)$

1. Both blocks are referred to in a backwards direction.

TRANSMIT BACKWARDS IMMEDIATE

TBI $(M_1) \longrightarrow (M_2)$
$(M_1-1) \xrightarrow[\text{etc.}]{} (M_2-1)$

SWAP FORWARD

SWAP $(M_1) \longleftrightarrow (M_2)$
$(M_1+1) \xleftrightarrow[\text{etc.}]{} (M_2+1)$

SWAP FORWARD IMMEDIATE

SWAPI $(M_1) \longleftrightarrow (M_2)$
$(M_1+1) \xleftrightarrow[\text{etc.}]{} (M_2+1)$

SWAP BACKWARDS

SWAPB $(M_1) \longleftrightarrow (M_2)$
$(M_1-1) \xleftrightarrow[\text{etc.}]{} (M_2-1)$

SWAP BACKWARDS IMMEDIATE

SWAPBI $(M_1) \longleftrightarrow (M_2)$
$(M_1-1) \xleftrightarrow[\text{etc.}]{} (M_2-1)$

Miscellaneous Operations: **OP, A_{18}(I) or 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. $\left\{\begin{matrix}\text{SIC}\\\text{NOP}\end{matrix}\right.$ 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	(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 $\longrightarrow$	$(M)^1$ (M)	1. The instruction whose address is located in M is executed.
STORE ZERO			
Z	0 $\longrightarrow$	(M)	1. Full word of zeros.

* NOTE: If the OP is SIC or EX, the format is OP, A_{19}(I). i.e. these two Operations have a 19 bit address field.

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 disc specified by $A_7(I)$. LOC=SU
READ RD	$A_7(I)$ represents a channel address; a reading operation is initiated for this channel (or for a unit attached to this channel, if more than one, which has been readied by a LOC instruction). $A_{18}(I')$ is the address of a control word (see below).
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 exceptional conditions are encountered (viz. unit check and end exception).

Input-Output Instructions: OP, $A_7(I)$, $A_{18}(I')$

LOCATE LOC OR 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 using this channel; or (2) an arc on the disc specified by $A_7(I)$.
READ RD	$A_7(I)$ represents a channel address; a reading operation is initiated for this channel (or for a unit attached to this channel, if more than one, which has been selected by a LOC instruction). $A_{18}(I')$ is the address of a control word
WRITE W	Initiates a writing operation. Analogous to RD except 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 $A_{18}(I')$. In the case of high-speed disc units, $A_7(I)$ must be 0 or 1 according to whether the control word is associated with reading or writing. If the disc is actually engaged in reading or writing, however, CCW is ineffective and the Channel Busy Reject indicator is turned on.
LOC(SEOP)	Same as LOC, SU, RD, W, REL, CTL except that SEOP bit in the control word is set 1; thus program interruption on completion of an operation is suppressed, provided no exceptional conditions (viz. unit check and end exception) are encountered.
SU(SEOP)	
RD(SEOP)	
W(SEOP)	
REL(SEOP)	
CTL(SEOP)	

CONTROL
CTL

Initiates performance of certain functions at the channel indicated by $A_7(I)$, or at the last unit there selected by a LOC instruction. These functions depend on the value of $A_{18}(I')$, indicated in the following tables:

UNIT	$A_{18}(I')$ (Octal)	FUNCTION
General I/Ø unit (standard for $A_{18}(I')$)	016	RESERVED Light Off
	017	RESERVED light on
	057	ECC mode
	116	CHECK light on
	157	No-ECC mode
Card Reader		Standard (see above)
Card Punch		Standard, plus
	056	Card run-out (feeds one card)
Printer		Standard, except that 057 and 157 are not allowed
Console		Standard, except that 057 and 157 are not allowed; plus
	177	Sound gong
Disc		No control functions allowed.
Tapes		Non-standard functions.
	016	Turn off TAPE INDICATOR light (Erase end of tape condition)
	036	High density mode (556 bits/inch)
	037	Low density mode (200 bits/inch)
	056	Erase long gap (three inches)
	057	Odd parity, ECC mode
	076	Space block (record)
	077	Space file
	117	Write tape mark (EOF Mark)
	136	Rewind
	137	Rewind and unload
	156	Even parity, no-ECC mode
	157	Odd parity, no-ECC mode
	176	Backspace block (record)
	177	Backspace file

Since the above control codes are difficult to remember, STRAP-1 provides the following CONTROL pseudo-ops:

RLF	Reserved light off
RLN	Reserved light on
ECC	ECC mode
KLN	CHECK light on
NOECC	NO-ECC mode
CRDRUN	Card run-out
GØNG	Sound gong

ERETC	Erase end of tape condition
or	or
TILF	Tape indicator light off
HD	High density mode
LD	Low density mode
ERG	Erase long gap
ØDDECC	Odd parity, ECC mode
ØDDNEC	Odd parity, no ECC mode
SP	Space block
SPFL	Space file
WEF	Write end-of-file mark
REW	Rewind
UNLØAD	Rewind and unload
EVEN	Even parity, no-ECC mode
BS	Backspace
BSFL	Backspace file

By using these CONTROL psuedo-ops, only the channel address, A_7(I) need be specified, thus:

```
|ECC, $PCH
```

In the case of a tape unit, the pseudo-op will always apply to the last unit referred to in a LOC instruction:

```
|LØC, $TC1, 4
|REW, $TC1
```

This code will rewind tape 4 on tape channel 1.

System Symbols for Channel Assignments:

In order that a coder need not know the specific numeric addresses of his installation, the following system symbols may be used:

$DISC* or $DISK*	Channel containing a disc unit. If an installation has more than one disc unit, these will be designated $DISC, $DISC 1, $DISC 2, etc.
$TC1* $TC2* $TC3*	Tape channels 1, 2, and 3. More may be added at the option of a particular installation. On each channel, tapes may have addresses 0 through 7.
$PRT	Printer
$RDR	Card reader
$PCH	Card punch
$CNSL	Console (including console typewriter)

In a multi-unit installation, these system symbols will be expanded to include $PRT1, $PRT2, or $RDR1, $RDR2, etc.

*The instruction LOC or SU must be used before reading or writing on this channel.

APPENDIX C

(Index to Mnemonics in Appendix B)

Note: + is alphabetically listed as "add"

- is alphabetically listed as "subtract"

* is alphabetically listed as "multiply"

/ is alphabetically listed as "divide"

1 is alphabetically listed as "one"

Mnemonic		Page
AD	Address invalid (indicator)	75
+	Add (floating point op.)	55
	(integer arith. op.)	61
+A	Add absolute	55
AE	Accumulator equal (indicator)	76
AH	Accumulator high (indicator)	76
AL	Accumulator low (indicator)	76
+MG	Add to magnitude (floating point op.)	56
	(integer arith. op.)	61
+MGA	Add (absolute) to magnitude	56
B	Branch (unconditional)	73
B	Branch on bit	73
BBN	Branch on bit and negate	73
BB1	Branch on bit and set to one	73
BBZ	Branch on bit and set to zero	73
BD	Branch disabled (unconditional)	73
BE	Branch enabled (unconditional)	73
BEW	Branch enabled and wait (unconditional)	73
BIND	Branch on indicator	74
BR	Branch relative (unconditional)	73
BTR	Binary transit (indicator)	75
BZB	Branch on zero bit	73
BZBN	Branch on zero bit and negate	73
BZB1	Branch on zero bit and set to one	73
BZBZ	Branch on zero bit and set to zero	73
BZIND	Branch on zero indicator	74
$Cx_1x_2x_3x_4$	Connect to accumulator	65
C+I	Add immediate to count	71
CB	Count and branch	72
CBJ	Channel busy reject (indicator)	74
CBR	Count, branch, and refill	72
CBRZ	Count, branch on zero count, and refill	72
CBZ	Count and branch on zero count	72
CCW	Copy control word	79
$CIx_1x_2x_3x_4$	Connect immediate to accumulator	66
$CMx_1x_2x_3x_4$	Connect to memory	65
CPU	Other central processing unit (indicator)	74
CS	Channel signal (indicator)	74
C-I	Subtract immediate from count	71
$CTx_1x_2x_3x_4$	Connect for test	65
$CTIx_1x_2x_3x_4$	Connect immediate for test	66
CTL,CTLSEOP	Control	79,80
CV	Convert	67
CVN	Convert negative	67

Mnemonic	Page
D+ Double add	56
D+A Double add absolute	56
D+MG Double add to magnitude	56
D+MGA Double add (absolute) to magnitude	56
DCV Double convert	67
DCVN Double convert negative	67
D/,D/A,D/N,D/NA Double divide	57
/ Divide (floating point op.)	57
(integer arith. op.)	62
/A Divide absolute	57
/N Divide negative (floating point op.)	57
(integer arith. op.)	62
/NA Divide negative absolute	57
DF Data Fetch (indicator)	75
DL,DLA,DLN,DLNA Double load	57
DLWF,DLWFA,DLWFN,DLWFNA Double load with flag bits	58
D*,D*A,D*N,D*NA Double multiply	56
DS Data store (indicator)	75
D- Double subtract	56
D-A Double subtract absolute	56
D-MG Double add to magnitude	56
D-MGA Double add to magnitude absolute	56
DTR Decimal transit (indicator)	75
E+ Add to exponent	55
E+A Add to exponent absolute	55
E+AI Add immediate to exponent absolute	55
E+I Add immediate to exponent	55
EE End exception (indicator)	74
EK Exchange control check (indicator)	74
EKJ Exchange check reject (indicator)	74
EOP End of operation (indicator)	74
EPGK Exchange program check (indicator)	74
ERG Erase gap	81
E- Subtract from exponent	55
E-A Subtract from exponent absolute	55
E-AI Subtract immediate from exponent absolute	55
E-I Subtract immediate from exponent	55
EVEN Even parity	81
EX Execute	78
EXE Execute exception (indicator)	75
EXIC Execute indirect and count	78

Mnemonics		Page
F+	Add to fraction	55
F+A	Add to fraction absolute	55
F-	Subtract from fraction	55
F-A	Subtract from fraction absolute	55
GONG	Sound gong	81
HD	High density	81
IF	Instruction fetch (indicator)	75
IJ	Instruction reject (indicator)	74
IK	Instruction check (indicator)	74
IR	Imaginary root (indicator)	75
K	Compare (floating point op.)	59
	(integer arith. op.)	64
KA	Compare absolute	59
KC	Compare count	70
KCI	Compare count immediate	71
KE	Compare if equal	64
KEN	Compare if equal negative	64
KF	Compare field	64
KFE	Compare field if equal	64
KFEN	Compare field if equal negative	64
KFN	Compare field negative	64
KFR	Compare field for range	64
KFRN	Compare field for range negative	64
KLN	Check light on	81
KMG	Compare magnitude	59
KMGA	Compare magnitude absolute	59
KMGN	Compare magnitude negative	59
KMGNA	Compare magnitude negative absolute	59
KMGR	Compare magnitude for range	59
KMGRA	Compare magnitude for range absolute	59
KMGRN	Compare magnitude for range negative	59
KMGRNA	Compare magnitude for range negative absolute	59
KN	Compare negative (floating point op.)	59
	(integer arith. op.)	64
KNA	Compare negative absolute	59
KR	Compare for range (floating point op.)	59
	(integer arith. op.)	64
KRA	Compare for range absolute	59
KRN	Compare for range negative (floating point op.)	59
	(integer arith. op.)	64
KRNA	Compare for range negative absolute	59

Mnemonics		Page
KV	Compare value	70
KVI	Compare value immediate	71
KVNI	Compare value negative immediate	71
L	Load (floating point op.)	57
	(integer arith. op.)	62
LA	Load absolute	57
LC	Last carry (indicator)	75
LC	Load count	70
LCI	Load count immediate	71
LCV	Load converted	67
LCVN	Load converted negative	67
LD	Low density	81
LF	Load field	65
LFT	Load factor (floating point op.)	58
	(integer arith. op.)	62
LFTA	Load factor absolute	58
LFTN	Load factor negative (floating point op.)	58
	(integer arith. op.)	62
LFTNA	Load factor negative absolute	58
LN	Load negative (floating point op.)	57
	(integer arith. op.)	62
LNA	Load negative absolute	57
LOC,LOCSEOP	Locate	79
LR	Load refill	70
LRI	Load refill immediate	71
LTRCV	Load transmit converted	68
LTRCVN	Load transmit converted negative	68
LTRS	Load transmit and set	62
LTRSN	Load transmit and set negative	62
LV	Load value	70
LVE	Load value effective	70
LVI	Load value immediate	71
LVS	Load value with sum	70
LWF	Load with flag bits (floating point op.)	58
	(integer arith. op.)	62
LWFA	Load with flag bits absolute	58
LWFN	Load with flag bits negative (fl. pt. op.)	58
	(integer arith. op.)	62
LWFNA	Load with flag bits negative absolute	58
LX	Load index	70
M+	To memory add (floating point op.)	55
	(integer arith. op.)	61
M+A	To memory add absolute	55

Mnemonics		Page
M+1	Add one to memory	63
M+MG	To memory add magnitude (floating point op.)	56
	(integer arith. op.)	61
M+MGA	To memory add magnitude absolute	56
MK	Machine check (indicator)	74
MOP	To memory operation (indicator)	76
M-	From memory subtract (floating point op.)	55
	(integer arith. op.)	61
M-A	From memory (absolute) subtract	55
M-1	Subtract 1 from memory	63
M-MG	From memory subtract magnitude (floating point op.)	56
	(integer arith. op.)	61
M-MGA	From memory (absolute) subtract magnitude	56
*	Multiply (floating point op.)	56
	(integer arith. op.)	61
*A	Multiply absolute	56
*A+	Multiply factor (absolute) and add	56
*+	Multiply factor and add (floating point op.)	56
	(integer arith. op.)	61
*N	Multiply negative (floating point op.)	56
	(integer arith. op.)	61
*N+	Multiply factor (negative) and add	56
*NA	Multiply negative absolute	56
*NA+	Multiply factor (negative absolute) and add	56
NM	Noisy mode (indicator)	76
ODD	Odd parity	81
OP	Operation invalid (indicator)	75
PF	Partial field (indicator)	75
PG 0-6	Programmer indicators	76
PSH	Preparatory shift > 48 (indicator)	75
R	Refill	78
RCZ	Refill if count is zero	78
RD,RDSEOP	Read	79
R/,R/A,R/N,R/NA	Reciprocal divide	57
REL,RELSEOP	Release	79
RGZ	Result greater than 0 (indicator)	76
RLZ	Result less than 0 (indicator)	76
RN	Result negative (indicator)	76
RNX	Rename	70
RU	Remainder underflow (indicator)	75
RZ	Result zero (indicator)	76

Mnemonics		Page
UF	U Flag (indicator)	75
UK	Unit check (indicator)	74
UNRJ	Unit not ready reject (indicator)	74
USA	Unended sequence of addresses (indicator)	75
V+	Add to value	70
V+C	Add to value and count	70
V+RC	Add to value, count, and refill	70
V±I	Add to or subtract from value, immediate	68,71
V±IC	Add to or subtract from value and count, immediate	68,71
V±ICR	Add to or subtract from value, count, and refill, immediate	68,71
VF	V Flag (indicator)	75
W,WSEOP	Write	79
XCZ	Index count zero (indicator)	76
XE	Index equal (indicator)	76
XF	Index flag (indicator)	75
XH	Index high (indicator)	76
XL	Index low (indicator)	76
XPH	Exponent high (indicator)	75
XPL	Exponent low (indicator)	75
XPM	Exponent medium (indicator)	75
XPN	Exponent high negative (indicator)	75
XPO	Exponent overflow (indicator)	75
XPU	Exponent underflow (indicator)	75
XVGZ	Index value greater than zero (indicator)	76
XVLZ	Index value less than zero (indicator)	76
XVZ	Index value zero (indicator)	76
Z	Store zero	78
ZD	Zero divisor (indicator)	75

INDEX TO GENERAL STRAP 1 WRITEUP

102638953