H. G. Kolsky

STRAP 1

An Assembly Program for Stretch

by

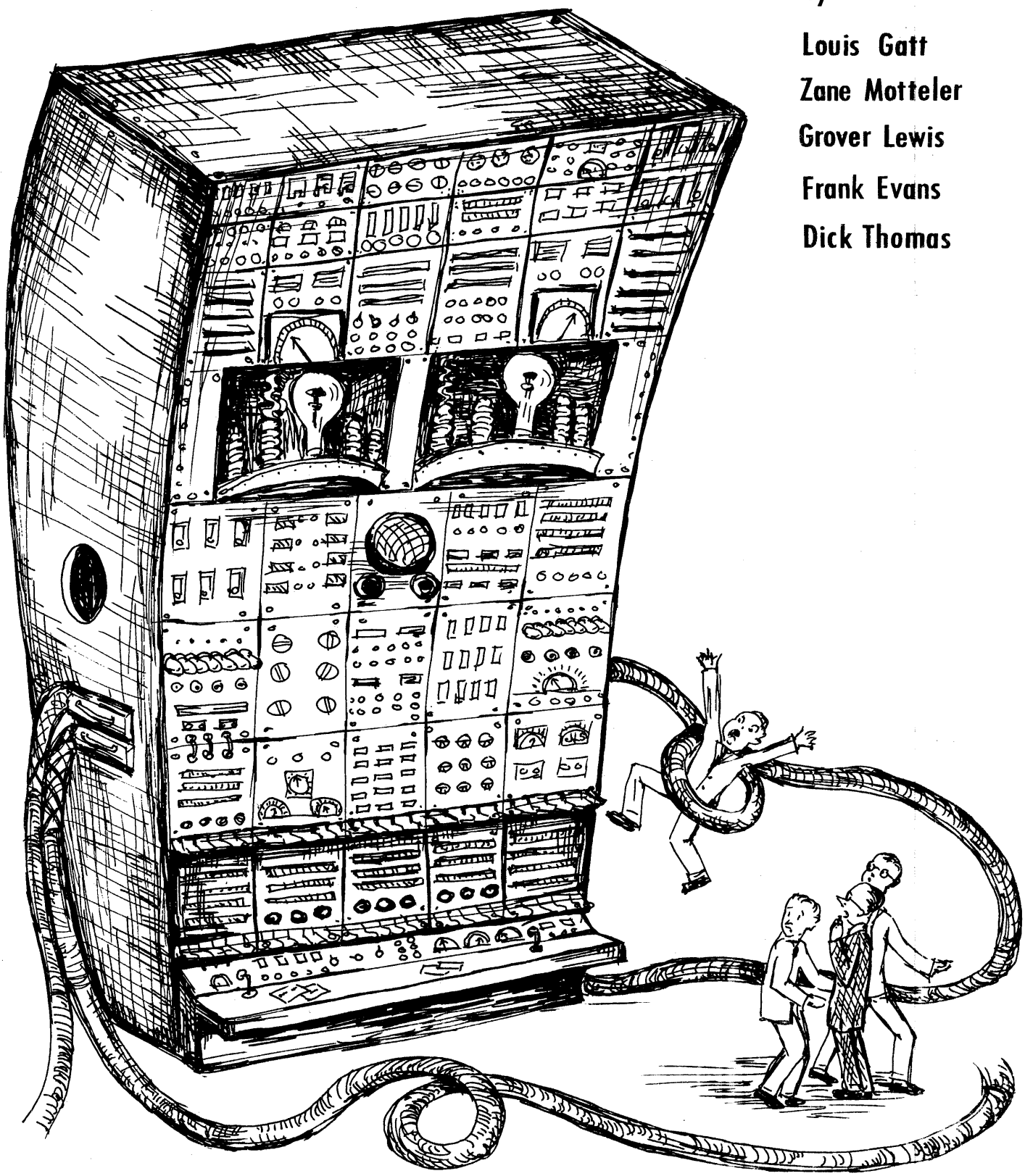
Louis Gatt

Zane Motteler

Grover Lewis

Frank Evans

Dick Thomas



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.

1. STRAP CODING FORM

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

Col.	1	2-9	10 71	72 - 80
	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 (62 columns) - to express a machine or pseudo-instruction
4. Identification (9 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.

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, (OP_2)(dds), $A_{24}(I)$,$OF_7(I')$	VFL arithmetic, connect, convert
8. OP, J, $A_{18}(I)$, $A'_{18}(I)$	Swap, transmit full words
9. OP, J_5, $A_{19}(I)$, $A'_{19}(I')$	Transmit half 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, O-7	Index word
4. VF, F	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 The format symbols used above are defined as follows:

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. J = 5 bits in immediate transmits.

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 under the heading Address Arithmetic.

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.) The integer n is the number of the rightmost bit of the parenthetical field. The address 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."

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",···, L^r). This statement is interpreted as reserving space for an L x L' x L" x ··· x 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",···, q^r). The first element of the array is Symbol (0, 0, 0,···, 0) = Symbol, and the last element of the reserved space is Symbol (L-1, L'-1, L"-1,···,L^r-1). The address of an arbitrary element is computed by means of the formula: Address of [Symbol (q, q', q",···, q^r)] = Address of [Symbol (0, 0, 0,···,0)] + FL x (q+q'L+q"LL'+q'"LL'L"+ ···), 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. Bit Addresses and Integers

8.0 Definition

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 bit address sum is then truncated on the right if necessary and the result added algebraically to the integer sum. 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.)

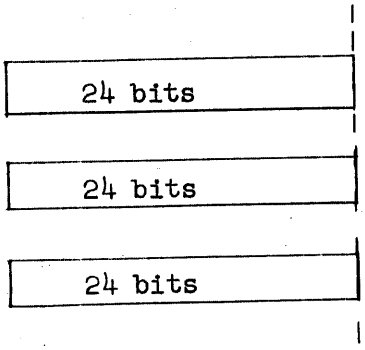
3. A period may not appear in the field of a general integer entry. A symbolic bit address appearing in such a field is treated as a 24-bit integer.

4. No arithmetic can appear in the name field, which is reserved for defining symbols.

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:



Note: Integer counts bits.

2. A_{19} Half-word address: B.A. 19 bits | 5 bits

I. 24 bits

Sum 19 bits

Note: Integer counts half words.

3. A_{18} Full-word address: B.A. 18 bits | 6 bits

I. 24 bits

Sum 18 bits

Note: Integer counts full words.

4. $A_{11\pm}$ Signed 11 bit address: B.A. 24 bits

I. 24 bits

Sum 11 bits ← 1 bit sign

5. OF_7 Offset: B.A. 24 bits

I. 24 bits

Sum 7 bits

Note: Bit address 1.32 = .96 = integer 96

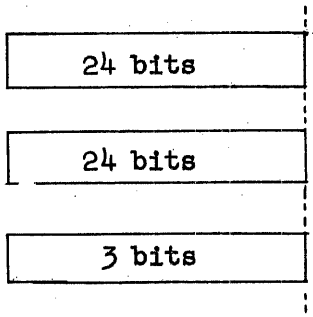
6. FL_6 Field length: B.A. 24 bits

I. 24 bits

Sum 6 bits

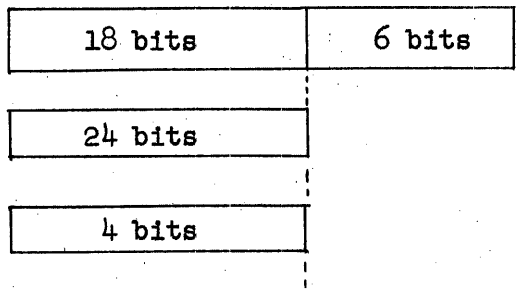
Note: 1.0 = .64 = 64 = 0 not error marked

7. BS_3 Byte size:



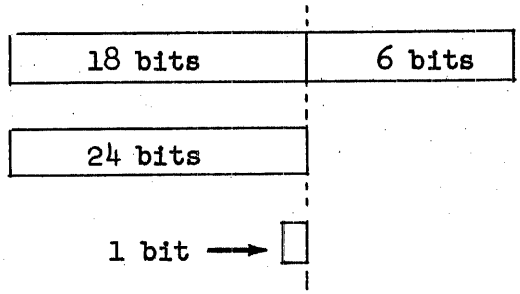
Note: .8 = 8 = 0 not error marked

8. I, J 4 bit index fields:



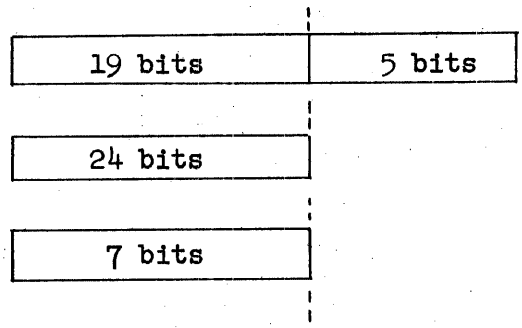
Note: A "1" in the bit position immediately to the left of the final sum field is discarded with no error indication.

9. K single bit index field:



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:



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)1101101110011.11110 (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 have to be specified with .n. If no radix is specified, the current operative one is continued; it is not

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

transfered 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. A symbol in the address field of a SYN card may not be preceded by a sign. 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. A minus sign preceding the programmer symbol is ignored. 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)

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. The index field "I" may be specified in parentheses of the end of any A field. If more than one "I" is entered they will be combined by means of a "logical or."

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. CV+I, "Count, add immediate to value"
4. CV-I, "Count, subtract immediate from value"
5. CRV+I, "Count, refill, add immediate to value
6. CRV-I, "Count, refill, subtract immediate from value"

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.

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 subfield 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

2. Other special registers (continued)

Location Word No.	Mnemonic	Name
6.0	$CPUS	CPU signal
7.17	$LZC	Left zeros count
7.14	$AOC	All ones count
8.0	$L	Left half of accumulator
9.0	$R	Right half of accumulator
10.0	$SB	Sign byte
12.21	$MASK	Mask
13.0	$RR	Remainder register
14.0	$FR	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	Insuiry station
$CNSL	Console

If more than one punch, printer, console or any other 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 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 integers only; 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 either the specified byte size or 8, whichever is smaller. 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 unnormalized 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:

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 number is converted to normalized floating binary form according to the standard Stretch 64-bit format. A four-bit sign byte is formed and stored. If none is entered by means of an "S", the sign before the number is used and the flag bits are set to zero.

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	$\log_e 2$

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.

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

This example is identical with example 2 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.

If the entry mode is omitted, two cases arise.

a. If the number is entered is 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 60 and an exponent of 48 supplied. In the second field the same thing is done except that the exponent is set to -35.

b. DD(U), 17.5

In this example 17.5 is converted to normalized floating binary and stored as such. However, instructions whose normalization bits depend on the symbol in the name field of this pseudo-op will have

them set to "unnormalized."

Note: 17 E 5 is an integer and will be recognized as such.

17 E-5 is a decimal fraction and will be normalized.

17.5 E 5 is an integer but will be treated as a fraction and normalized. 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

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

Implied field length is 64. Octal specification in last D field overrules (F7) written before DD.

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

Binary entry, overruled in 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. Binary Unsigned VFL

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. Decimal Signed VFL

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

DD(D, FL, BS), ±(R) xx···xx Sn

DD(D, FL, BS), ± xx···xxEyy Sn

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), 1110100011015S7

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	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 DDI

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 only one 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 not 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 - 34012 = -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.

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. 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. No sign is stored for the symbol, but a symbol for a negative quantity is all right. 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 chain of SYN's defining the symbolic portion plus a single symbol (algebraic addition, but not subtraction of a symbol, e.g., $\pm$ 5 + symbol or symbol $\pm$ 5). 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 depends on the sign which precedes it as well as its 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 and 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.

Appendices B & C

MISSING

APPENDIX D

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

A
1. No characters are given with an A or IQS entry mode in a DD statement.
2. Too many address fields were given.

B
1. The byte size of this instruction is > 8, and has been set to 8.
2. The byte size of a decimal instruction ≠ 4, but has been left at the specified value.
3. A or IQS byte size in DD statement is>12, and has been set to 8.

C
A negative field (address, index, field length, etc.) has been complemented.

This complementing takes place prior to the truncation described under "V", if any.

D
Data error in DD or DDI.

The data has been set to zero.

E
Entry mode error in DD or DDI.

Entry mode (10) is assumed.

F
1. The given field length is > 64, and has been set to 64.
2. Field length of a VFL binary multiply - or divide-type order is > 48, but has been left at specified value.

G
A "go-to" type order, i.e., a branch or END card, has a transfer address < 32 and I index 0.

H
This card has an illegal name (a non-alphanumeric character is present, the character has been ignored).

I
A branch on indicator, with indicator numerically specified, has indicator numeric > 63.

The indicator has been computed modulo 64.

J 1. There is an illegal punch on this card.

2. There is an illegal character in some field on this card. (if an illegal numeric occurs in a radix entry from R = 2 through R = 9, the field is set to zero; in all other cases the illegal punch or character is ignored)

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

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

2. An SLC address contains an integer.

M 1. The given mode is not consistent with the given operation or vice versa, e.g., M + 1 (N),... The mode, FL, and BS are ignored.

2. No mode given with DD or DR -- N is assumed.

3. No mode given with DDI -- (BU, 24, 8) is assumed.

4. An operation which could be either VFL or floating point has no overruling mode and a numeric address. VFL is assumed.

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

O 1. Offset + field length - byte size (if signed) on this op is > 128. None of the three quantities is changed.

2. Decimal offset is not divisible by 4, but is left unchanged.

P Conditional branch has K field 1.

The given K field is taken modulo 2.

Q 1. Illegal OP1 or OP2 in progressive indexing. In either case, OP2 is ignored.

2. Illegal OP3 in CW. OP3 is ignored.

3. Pseudo-op has been specified in an EXT field. The requested field has been set to 0.

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

S Illegal OP after an SIC order. (Only half-word branch orders are permitted.

The OP has, however, has been assembled as requested.

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

U VFL decimal multiply or divide has been specified. (There are no such OPs.)

The OP has been assembled as requested but it must be remembered that it acts as a LFT order.

V. 1. Overflow in an address field; the field has been truncated if necessary to fit the number of bits available.

2. A DD with entry mode A or IQS has a byte size which is too small, i.e., < 6 or < 8 respectively.

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

X 1. An address field on this card cannot be completely evaluated. E.g., due to a too-complicated chain of SYN cards, etc. The computation has been completed as far as possible.

2. A symbol specified on a PVNSYM card does not occur in the program.

Y There is a symbol on this card containing more than six characters.

Only the first six are used.

Z 1. No address or name field has been specified on a SYN or DDI card; or a SYN card is not computable by pass 3.

2. The second half word address of an Input-Output op is < 32.

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

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

The arithmetic was performed in binary unsigned.

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

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

Scripting

IBM will get in real bind

orig 9 mos, now 18 mos

orig cost – not same del. date

Bartell

– repriced – up $15K

6 mos. delay

IBM would be attacked in flying into field at LASL

To → Sen. Anderson

before December

---

407 manual ← CELahmeyer

(1401)