STRETCH CODING ASSUMPTIONS

(Revised) 7/29/57

Preface The purpose of this write-up is to define a fundamental vocabulary and arithmetic unit operation which may be used in "experimental coding". The aim in performing such coding is to permit an evaluation of certain computer concepts which have not previously existed - at least locally. It is also possible that such coding will suggest other features and/or changes.

The primary areas for evaluation are expected to be:

(a) Geometric Indexing

(b) Second Address(SA) and its associated indexing (SAX)

(c) Pre-Post secondary operations (SAUD)

(d) Index Register format.

In addition there are some other areas for evaluation such as:

(a) Indirect addressing technique

(b) Chain indexing

(c) Programmers Tag bits (PT_1) in data words

(d) Indexed Δ , Δ_1 (index register modifier)

(e) "- to memory" type arithmetic operations.

It should not be assumed that the included vocabulary is complete. Certain types of operations have not been defined such as fixed point arithmetic operations, shifts, logical operations (AND, OR, etc.), and I/O instructions. In order that coding not be prevented due to these omissions, a "temporary vocabulary" has been included. The inclusion of the I/O operations is made so the points in the code where I/O work is assumed can be simply designated.

TABLE OF CONTENTS

STRETCH CODING ASSUMPTIONS

I. INTRODUCTION

A. Stretch Memory Organization

There are 3 types of random access memory. Their speeds and sizes are currently assumed to be:

(a) Main Memory (MM): 2.0 μs R/W cycle. Four boxes of 8192 wds. (Total of 2^{15} wds.)

(b) Fast Memory (FM): 0.5 μs R/W cycle. Two boxes of 512 wds. (Total of 2^{10} wds.)

(c) Ultra-Fast Registers (UFR): 0.2 μs R/W cycle. 16 registers which are used as index registers and high-duty rate temporary (erasable) storage.

The currently accepted memory addressing system is:

1-16: Accumulator, selectors, instruction counter and other addressable registers.

16-32: Index registers and other 0.2 μs. registers. (UFR)

33-1024: Fast Memory (0.5 μs)

1024 -32768: Main Memory (2.0 μs)

B. Arithmetic Speeds

Floating Point arithmetic speeds are assumed to be:

(a) addition or subtraction: 0.6 μs

(b) multiplication: 1.2 μs

(c) division: 1.8 μs

Other assumed times are

(a) Clear and Add (Load, etc.): 0.8 μs from MM
0.2 μs from FM
0.2 μs from UFR

(b) Store: probably .1 μs for all types of memory.

C. Some Comments

(1) Insofar as possible, one would like to assume that data blocks are stored in Main Memory, instructions in FM, and indexing quantities in UFR. Perhaps some frequently used constants might also be stored in FM.

(2) Occasionally several "quick" references are needed for the same variable stored in MM, such as the following calculation of x^2 (x in MM):

α	CA	L(x)
$\alpha+1$	FM	L(x).

This code would make inefficient use of the Arithmetic Unit since the first reference to x would tie up MM for 2 μs and the FM cannot begin until $\alpha+1$ completes its reference for x the second time. Hence 2.0 + .8 = 2.8 μs must elapse before the multiplication can start. If α, however, could store x in a faster memory, say R, and $\alpha+1$ could get its x from there, then the multiply could begin after .8 + .2 + .2 = 1.2 μs (or possibly .8 μs):

It is considerations of this type which lead to the postulation of a restricted second address and the concepts of "Pre-load", "Pre-Transmit" (or "Pre-Store"), "Post-Store", and "Preload and Poststore". By test coding we hope to determine the frequence of use and the saving of arithmetic unit time through this idea. Another open question is whether one gains much by having some indexing associated with this second address.

D. Miscellaneous Assumptions

(1) Arithmetic Unit

(a) Universal Register (no distinction between accumulator and MQ). Universal Register (Accumulator) is double length, (64 + 64 = 128 bits)

(b) There are 2 types of triggers:

1. Automatically set (such as 3 (?) overflow triggers, underflow triggers, divide check, etc.)
2. Program set (such as 704 sense lights)

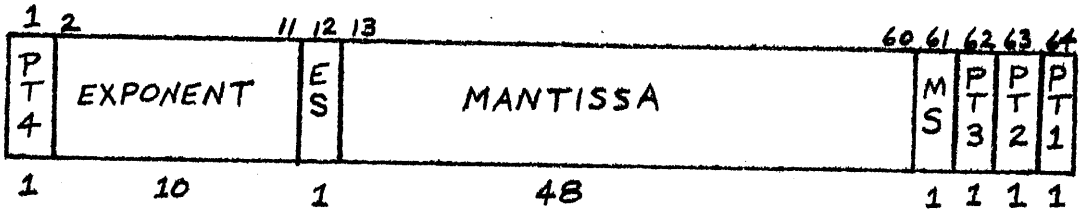
It is assumed that a 64-bit selector exists for each of the two above types. In case (1) one bit corresponds to each of the various conditions. (See VI.)

(2) Fixed-Point Definitions are not considered in this write-up except for indexing arithmetic and the logical connectives (C and CM). (a description of C and CM will be issued shortly).

(3) The machine is completely binary and no decimal operations are assumed.

virtual memory — real memory

II. FLOATING-POINT DATA FORMAT



ES: Exponent Sign

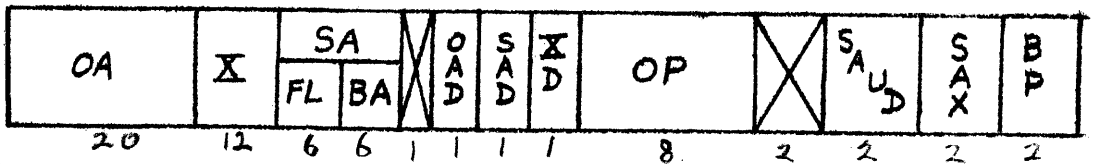
MS: Mantissa Sign

PT: Programmer's Tag Bits.

One would like to determine to what extent the four PT bits are of value. It might be that they would be of value in marking beginnings and endings of arrays, boundaries, etc.

III. INSTRUCTION FORMAT AND TYPES OF ADDRESSING

At this time only one instruction format is considered.



OA: (Operand Address). OA may be immediate, direct or indirect depending on the definition of the operation and the OAD bit.

X: (Index field). May be used directly to address a single index register in which case X contains a 12-bit address, or geometrically to address a subset of index registers 1 - 12.

SA: (Second Address). SA may be immediate, direct, or indirect depending on the definition of the operation and the SAD bit. SA in some operations is split into two fields, FL (field length) and BA (bit address).

OAD: (Operand Address Designator) (See 2 below)

0: OA used as basically defined

1: immediate → direct, or direct → indirect

SAD: (Second Address Designator) (See 3 below)

Same as OAD except it modifies SA instead of OA.

XD: (Index Designator)

0: Direct

1: Geometric

OP: (Operation)

SAUD: (Second Address Use Designator)

00: Pre-Transmit (T) or No Action if SA = 0 and SAX = 0

01: Pre-Load (L)

10: Post-Store (S)

11: Pre-Load and Post-Store (LS)

The SAUD field is not used on non-arithmetic operations even though SA is used.

SAX: (Second Address Index field) Geometric indexing only. The first bit position refers to index register 1 (in common with 1 in X) and the second bit position refers to index register 13.

BP: (Break-Point) To be defined later

Note that there are 3 unused bits.

Types of Addressing

1. Immediate, Direct, and Indirect.

The definition of each operation states (or implies) whether OA and SA is immediate or direct in its basic sense. The following statements phrased in terms of OA, OAD, and X apply as well when the corresponding substitution; SA, SAD, and SAX; is made (except in the definition of the contents of IAW)

immediate address: The address, OA, or the effective address, OA + C(X), is itself used by the operation. Examples are: Transfers, Branches, Shifts and Δ's in the SA field.

direct address: The contents of the address, C(OA), or the contents of the effective address, C(OA + C(X)), is used by the operation. The arithmetic operations are a common example.

indirect address: To perform indirect addressing it is necessary to use one or more "indirect address words", (IAW), in addition to the address and index field of the instruction itself, OA_o and X_o. The format of an indirect address word is the same as the instruction word format but only four fields are used: OA, X, OAD, XD. Occasionally several levels of indirect addressing are needed; hence the notation IAW_i, OA_i, X_i, OAD_i, and XD_i where i denotes the level of indirect addressing, i = 1,2,.... . The XD_i bit designates direct or geometric interpretation of X_i. An OAD_i bit indicates that another level of indirect addressing follows. Note that only operations defined with direct addresses can be modified for indirect addressing (and OAD_o = 1). Hence:

First level indirect addressing (i = 1; $OAD_1 = 0$)

$C(C(OA_o) + C(X_o)) \equiv C(OA_1 + C(X_o))$ is the operand used by the instruction. If there is no indexing, this definition reduces to $C(C(OA_o)) \equiv C(OA_1)$.

Second level indirect addressing (i = 2; $OAD_1 = 1$, $OAD_2 = 0$)

$$C\left\{C\left[C(OA_o) + C(X_o)\right] + C(X_1)\right\} = C\left\{C\left[OA_1 + C(X_o)\right] + C(X_1)\right\}$$
$$= C\left\{OA_2 + C(X_1)\right\}$$ is the operand used by the instruction

N^{th} level indirect addressing (i = N; $OAD_1 = \ldots = OAD_{N-1} = 1$; $OAD_N = 0$)

$C(OA_N + C(X_{N-1}))$ is the operand used by the instruction.

The definitions above are for OA indirect addressing. For SA indirect addressing, substitute SA for OA_o, SAX for X, and SAD for OAD_o. The interpretation and use of IAW_1 remains the same.

2. OAD and SAD

The three types of addressing are assumed to be ordered:

a. immediate
b. direct
c. indirect.

If OAD (SAD) is zero, then the type of addressing defined by the operation is used (immediate or direct). If OAD (SAD) is one, then the order is dropped one downward in the list above from the type defined in the operation; immediate → direct, or direct → indirect.

3. Bit and Bit Field addressing

In certain operations it seems desirable to address a particular bit or a sequence of continuous bits. For this purpose the 12-bit SA field is divided into two 6-bit fields, FL (field length) and BA (bit address). The bit address is simply the bit position in the word of the leftmost bit of the field. The field length is the number of bits to the right of BA needed to make up the contents of the bit field (BF).

Examples:

(a) If only bit 37 is desired: BA=37, FL=0
(b) If the OA field (bits 1-20) is desired: BA=1, FL=19

IV. TYPES OF INDEXING AND INDEX WORD FORMAT

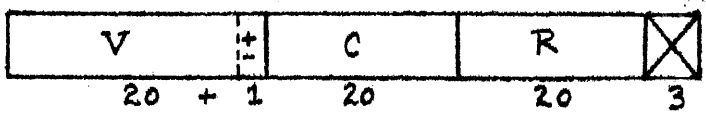
Stretch is assumed to have two general types of indexing, direct and geometric. Each instruction contains a 12-bit X field and an XD bit. The XD bit designates direct or geometric indexing.

Direct indexing: The 12-bit index field, X, contains the address of a single word to be used as an index register in forming the effective address for the instruction, OA eff. addr. Thus any word, addressable by 12 bits, can be used as an index register if necessary.

Geometric indexing: Each bit in the 12-bit index field, X, corresponds to an index register (as in the 704). Thus the formation of the effective address may depend on the sum (ΣV_1) of several index registers. ~~(See IV)~~

At present it is assumed that the second address (SA) has some limited geometric indexing indicated by the second address index field(SAX).

We assume index words of the form:



* (ON NEXT PAGE)

where:

± V (value) = the quantity used for address modification; multiple index register reference will cause ΣV_1 to modify the address.

C (count) is the control count which is counted down by 1 and tested.

R (reset address) = the address of the next word to be loaded into this register when it is reset.

A fourth quantity, Δ, or $L(\Delta)$ is carried in the SA portion of the instructions which modify and test index registers. Δ may be used to increment or decrement the value V.

The Reset address, R, is used for "chain indexing". For example, the instruction IBR (increment, branch, and reset) is executed as follows:

1. $V + \Delta \longrightarrow V$; if $C \neq 0$, $C - 1 \rightarrow C$, then
2. If $C > 0$, Transfer.

3. If C = 0, Control goes to next instruction and C(R) replaces the present contents of the index register.

Thus, one may have a whole series of words to be loaded into an index register, with the R's forming the connecting links.

* Inside the machine, the sign of V is assumed to follow V. Hence, the V field (20 bits) occupies the same bits as OA in an instruction.

V. VOCABULARY*

A. Arithmetic

(c(acc) means $c(acc)_{1-64}$)

1. Basic operations (assumed to be similar to 704 floating point).

A (add):

c(acc) + c(OA eff. addr.) ⟶ c(acc).

S (subtract):

c(acc) - c(OA eff. addr.) ⟶ c.(acc).

M (multiply):

c(acc) x c(OA eff. addr.) ⟶ c(acc)

D (divide):

c(acc)/c(OA eff. addr.) ⟶ c(acc), Q high, R low

L (load):

c(OA eff. addr.) ⟶ c(acc)

LL (load low accumulator):

c(OA eff. addr.) ⟶ $c(acc)_{65-128}$

ST (store):

c(acc) ⟶ c(OA eff. addr.)

STL (store low accumulator):

$c(acc)_{65-128}$ ⟶ c(OA eff. addr.)

2. Basic operations plus sign modification

AV (add value):

c(acc) + |c(OA eff. addr.)| ⟶ c(acc)

SV (subtract value):

c(acc) - |c(OA eff. addr.)| ⟶ c(acc)

MV (multiply value):

c(acc) x |c(OA eff. addr.)| ⟶ c(acc)

DV (divide by value):

c(acc)/ |c(OA eff. addr.)| ⟶ c(acc), Q high, R low

LV (load value):

|c(OA eff. addr.)| ⟶ c(acc)

LNV (load negative value):

- |c(OA eff. addr.)| ⟶ c(acc)

LN (load negative:

- c(OA eff. addr.) ⟶ c(acc)

* It should be remembered that the accumulator, instruction counter, etc. are addressable.

STV (store value):

$|c(acc)| \longrightarrow$ c(OA eff. addr.), c(acc) unchanged

STNV (store negative value):

$-|c(acc)| \longrightarrow$ c(OA eff. addr.), c(acc) unchanged

STN (store negative)

$-c(acc) \longrightarrow$ c(OA eff. addr.), c(acc) unchanged

3. Basic operations on memory (note: these operations do not destroy c(acc).)

AM (add to memory):

c(acc) + c(OA eff. addr.) $\longrightarrow$ c(OA eff. addr.)

SM (subtract to memory):

c(acc) - c(OA eff. addr.) $\longrightarrow$ c(OA eff. addr.)

MM (multiply to memory):

c(acc) x c(OA eff. addr.) $\longrightarrow$ c(OA eff. addr.)

DM (divide to memory):

c(acc)/c(OA eff. addr.) = Q $\longrightarrow$ c(OA eff. addr.), R lost

4. Basic operations plus sign modification, on memory

AVM (add value to memory):

c(acc) + $|c(OA\ eff.\ addr.)| \longrightarrow$ c(OA eff. addr.)

SVM (subtract value to memory):

c(acc) - $|c(OA\ eff.\ addr.)| \longrightarrow$ c(OA eff. addr.)

MVM (multiply value to memory):

c(acc) x $|c(OA\ eff.\ addr.)| \longrightarrow$ c(OA eff. addr.)

DVM (divide value to memory):

c(acc)/ $|c(OA\ eff.\ addr.)|$ = Q $\rightarrow$ c(OA eff. addr.) R lost

5. Exponent and sign modification

AE (add to exponent):

$c(acc)_{exp} + OA_{2-11} \longrightarrow c(acc)_{exp}$.

SE (subtract from exponent):

$c(acc)_{exp} - OA_{2-11} \longrightarrow c(acc)_{exp}$.

OA_{2-11} is immediate

CHS (change sign):

-c(OA eff. addr.) $\longrightarrow$ c(OA eff. addr.)

MP (make positive):

$|c(\text{OA eff. addr.})| \longrightarrow c(\text{OA eff. addr.})$

MN (make negative):

$-|c(\text{OA eff. addr.})| \longrightarrow c(\text{OA eff. addr.})$

6. Use of SAUD (Pre-Post)

(May be used with all operations defined under V.A.1 thru V.A.5)

The following definitions and rules for determining the meaning of instructions whose Op consists of a primary operation and a pre-post operation are currently assumed. In some cases, redundancies occur. Let ⊛ denote the primary operation (such as A, AM, AE, etc.)

blank:	no action	:	⊛
T:	pre-transmit	:	C(OA eff. addr.) → C(SA eff. addr.), ⊛
L:	pre-load	:	C(SA eff. addr.) → C(acc), ⊛
S:	post-store	:	⊛, C(acc) ⟶ C(SA eff. addr.)
LS:	pre-load and post-store	:	C(SA eff. addr.) → C(acc), ⊛, C(acc) ⟶ C(SA eff. addr.)

Rule: To determine the meaning of an operation, substitute the definition of the primary operation in one of the above definitions.

B. Indexing operations*

1. Operations to modify an index word directly or geometrically.

TI (transfer and increment):

$V + \Delta \longrightarrow V$; transfer to OA.

TD (transfer and decrement):

$V - \Delta \longrightarrow V$; transfer to OA.

TS (transfer and set):

$c(\text{SA eff. addr.}) \longrightarrow c(\text{X-reg})$; transfer to OA.

TR (transfer and reset):

$c(R) \longrightarrow c(\text{X-reg})$; transfer to OA.

TIR (transfer, increment and reset):

$V+\Delta \longrightarrow V$; if $c = 0$, reset; transfer to OA

TDR (transfer, decrement and reset):

$V - \Delta \longrightarrow V$; if $c = 0$, reset; transfer to OA.

} reset applies to only those index registers with c=0.

* Transfer is used for "unconditional transfer".
Branch is used for "conditional transfer".

2. Operations to test an index word. Δ in SA (immediate)

BCG (Branch if C greater)

If $C > \Delta$, transfer to OA; otherwise proceed.

BCE (Branch if C equal)

If $C = \Delta$, transfer to OA; otherwise proceed.

BCL (Branch if C less)

If $C < \Delta$, transfer to OA; otherwise proceed.

BVG (Branch if V greater)

If $V > \Delta$, transfer to OA; otherwise proceed.

BVE (Branch if V equal)

If $V = \Delta$, transfer to OA; otherwise proceed.

BVL (Branch if V less)

If $V < \Delta$, transfer to OA; otherwise proceed.

BVGN (Branch if V greater than negative Δ)

If $V > -\Delta$, transfer to OA; otherwise proceed.

BVEN (Branch if V equal to negative Δ)

If $V = -\Delta$, transfer to OA; otherwise proceed.

BVLN (Branch if V less than negative Δ)

If $V < -\Delta$, transfer to OA; otherwise proceed.

To any of the above mnemonic codes the suffix R may be added to achieve the reset operation when the condition is satisfied. For example:

BCGR (Branch if C greater and reset)

If $C > \Delta$, transfer to OA and reset; otherwise proceed.

3. Operations to modify and test an index word.

IB (Increment and Branch) immediate Δ in SA

$V + \Delta \rightarrow V$; if $C \neq 0$, $C-1 \rightarrow C$; then

- If $C = 0$, proceed
- If $C \neq 0$, transfer to OA

DB (Decrement and Branch) immediate Δ in SA

$V - \Delta \rightarrow V$; if $C \neq 0$, $C - 1 \rightarrow C$; then

- If $C = 0$, proceed
- If $C \neq 0$, transfer to OA

Note: If SAD = 1 in IB and DB, then $V \pm \Delta \rightarrow V$ is replaced by $V \pm C(SA + C(SAX))_V \rightarrow V$.

ICB (Increment by C and Branch)

$V + C(SA + C(SAX))_C \longrightarrow V$; if $C \neq 0$, $C - 1 \rightarrow C$; then

{ If C = 0, proceed
{ If C ≠ 0, transfer to OA

DCB (Decrement by C and Branch)

same as ICB except V modification is $V - C(SA + C(SAX))_C \longrightarrow V$

Note: If SAD = 1 in ICB and DCB, then the V modification is $V \pm C(C(SA)_{OA} + C(SAX))_C \longrightarrow V$

INB (Increment and no Branch)
DNB (Decrement and no Branch)
ICNB (Increment by C and no Branch)
DCNB (Decrement by C and no Branch)

} same as above except that the proceed and transfer conditions are interchanged

Note: To any of the above eight operations may be added the reset operation by adding the suffix R to the mnemonic code. On the first four operations above, the reset occurs on the proceed branch. On the last four, reset occurs on the transfer branch.

C. Control

TSL (Transfer and set location)

$C(\text{location counter}) \longrightarrow C(SA)_{OA} = C(SA)_{1-20}$

Transfer to (OA eff. addr.)

Note that this is also an "ordinary" indexable transfer when SA = 0.

BB* (Branch on bits)

If C(BF) ≠ 0, transfer to (OA eff. addr.); otherwise proceed.

machine set indicators

BBZ*(Branch on Zero bits)

If C(BF) = 0, transfer to (OA eff. addr.); otherwise proceed.

BBC* (Branch on bits and clear)

If C(BF) ≠ 0, transfer to (OA eff. addr.), $0 \rightarrow C(BF)$; otherwise proceed.

BBZC* (Branch on zero bits, clear on proceed)

If C(BF) = 0, transfer to (OA eff. addr.); otherwise proceed, $0 \rightarrow C(BF)$.

*These operations apply only to the "implied" machine-set indicator (See VI)

D. Miscellaneous Operations

MST (masked store) No pre-post.

C(acc) $\longrightarrow$ C(OA eff. addr.) masked by C(SA eff. addr.)

ML (masked load) No pre-post.

C(OA eff. addr.) $\longrightarrow$ C(acc) masked by C(SA eff. addr.)

MLC (masked load with clear). No pre-post

0 $\rightarrow$ acc, C(OA eff. addr.) $\longrightarrow$ C(acc) masked by C(SA eff. addr.)

SWAP (interchange two wds. in mem.)

C(OA eff. addr.) $\longrightarrow$ C(SA eff. addr.) while C(SA eff. addr.) $\longrightarrow$ C(OA eff. addr.). C(acc) not disturbed.

Possibly pre-post might be useful (except SWAP,T) but none are assumed at this time.

SEA (store effective address)

$OA + C(X_i)_V \longrightarrow C(SA)_{1-21}$ geometric

OA is immediate and used as a 20 bit number whose sign is +; $C(X_i)_V$ means the ΣV_i from the geometrically indicated index register or a single V if direct. SA is assumed to refer to an index register.

VI. INDICATORS

A. Automatically set indicators

1. We assume twenty-seven indicators, to be set by the machine, as follows:

Indicator #	Condition indicated by bit being ON
1.	Accumulator zero
2.	Accumulator greater than zero
3.	Accumulator less than zero
4.	Programmer tag #1
5.	Programmer tag #2
6.	Programmer tag #3
7.	Programmer tag #4
8.	Improper divisor
9.	Fixed point overflow
10.	Fl. Pt. Overflow (a bit occurs in exponent bit 7)
11.	Fl. Pt. Overflow (a bit occurs in exponent bit 10)
12.	Fl. Pt. Overflow (overflow beyond the range of the exp.)
13.	Fl. Pt. Underflow (bit occurs in exp. bit 7, sign minus)
14.	Fl. Pt. Underflow (bit in exp. bit 10)
15.	Fl. Pt. Underflow (bit occurs beyond range of exp.)
16.	Exponent zero
17.	Exponent greater than zero
18.	Exponent less than zero
19.	Index quantity "C" less than zero (or attempte
20.	Invalid operation
21.	Invalid address
22.	Break point bit #1
23.	Break point bit #2
24.	Control error
25.	Information error
26.	Memory error
27.	|V| overflow

2. Indicator #24, 25, 26 indicate internal malfunctions in the machine.
3. Operations are postulated to test the status of these indicators in either a destructive or non-destructive fashion, where this applies, at the option of the programmer.
4. The indicator number can be used in the SA field when test programming to denote the condition being tested. For example, "BB, A, # 1" is equivalent to the 704 "TZ,A".
5. The address to which control is transferred is given by OA + C(X).*

* The notation, OA + C(X), is equivalent to the previously used notation, OA eff. addr.

B. Programmer set indicators

* A vocabulary to handle the programmer set indicators is not included, but is being studied.

VII. CODING FORM

The present coding form has the following format:

SEQ	OPN	A	X	SA	COMMENTS

It is suggested that the coder adjust this form to the current assumptions by making the following additions:

SEQ	OPN		A		X		SA		COMMENTS
	OP	SAUD	OA	OAD	X	XD	SA	SAD	SAX

In coding, it would be helpful in analyzing codes if the following conventions were used in the fields.

OP: letters corresponding to operation

SAUD: blank: no action

T: Pre-Transmit

L: Pre-Load

S: Post-Store

LS: Pre-Load and Post-Store

OA: number $\ll 2^{20}$ or alphabetic letter to represent data word.

OAD: blank or 1

X: number(s) in range 1 through 12 or subscript letter(s) if geometric, or single 12 bit number or letter if direct. Should be consistent with XD.

XD: blank for direct
G for geometric

SA: number $< 2^{12}$ to be used as address or Δ for indexing, or alphabetic letter to represent data word.

SAD: blank or 1

SAX: blank: no indexing of SA
1: index register 1 to be used
13: index register 13 to be used (If desired, letters (subscripts) may be used instead of 1 and 13)

BP: not defined at present.

VIII. "TEMPORARY VOCABULARY"

A. Shifts

AL: (Accumulator left)
$C(acc)_{1-128}$ shifted left OA + C(X) places; overflow indicator possibility.

HAL: (High acc. left)
$C(acc)_{1-64}$ shifted left OA + C(X) places; overflow indicator possibility.

LAL: (low acc. left)
$C(acc)_{65-128}$ left; $C(acc)_{1-64}$ unchanged.

AR: (acc. right)
$C(acc)_{1-128}$ right.

HAR: (high acc right)
$C(acc)_{1-64}$ right; $C(acc)_{65-128}$ unchanged.

LAR: (low acc right)
$C(acc)_{65-128}$ right.

B. Logical operations (same as 704)

NA: (AND to accumulator)
RA: (OR to accumulator)
NS: (AND to storage)
RS: (OR to storage)

C. I/O operations

RD: (read)
WR: (write)

} OA field can designate unit and SA field can designate which record, etc.

D. Fixed-Point Arithmetic

No fixed-point arithmetic operations are included.

SUBROUTINE

MATRIX MULTIPLY ($A \times B = C$) A, B, C STORED ROW-WISE

β CODE : CALLING SEQUENCE

α CODE : SUBROUTINE

SEQ	OPN OP	SAUD	A OA	OAD	X X	XD	SA SA	SAD	SAX	COMMENTS
β	TSL		α				13			
β+1			A_0							WHERE A_0 = FWA OF THE a_{ij} MATRIX
β+2			B_0							" B_0 = " " " b_{jk} "
β+3			C_0	V	C		R			" C_0 = " " " c_{ik} "
β+4			0		I		β+4			INDEX WORDS; THE QUANTITIES INDICATED ARE
β+5			0		K		β+5			ASSUMED TO BE IN THE V, C, R FIELDS (NOT IN
β+6			0		J		β+6			THE A, X, SA FIELDS SHOWN).
β+7	CONTROL RETURNS HERE									
α	L	S	1		13		T_0			
α+1	L	S	2		13		T_1			
α+2	L	S	3		13		T_2			
α+3	TS		α+4		4		4		13	
α+4	TS		α+5		2,3	G	5		13	
α+5	TS		α+6		1,5	G	6		13	
α+6	L	S	L(0)				Σ			
α+7	L		T_0	1	1,4	G				
α+8	M		T_1	1	3,5	G				
α+9	A	S	Σ				Σ			
α+10	ICBR		α+11		5		5		13	
α+11	IBR		α+7		1		1			
α+12	ST		T_2	1	2,3	G				STORE C_{ik}
α+13	IBR		α+6		3		1			
α+14	ICBR		α+15		2		5		13	
α+15	ICBR		α+6		4		6		13	
α+16	TSL		7		13		0			

7/29/57

H. K.

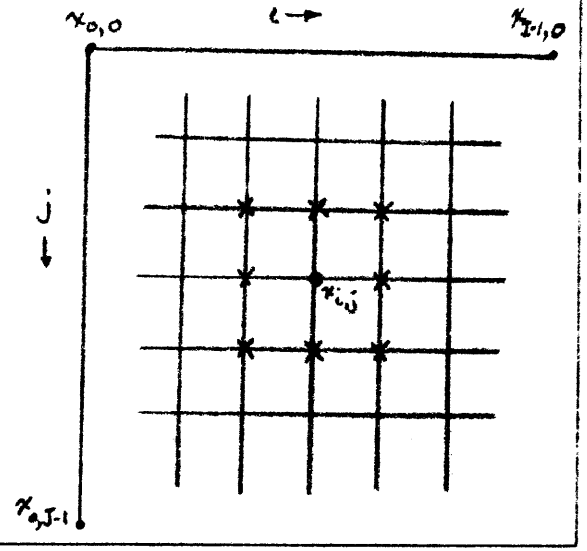
8-NEIGHBOR SUM (EXAMPLE OF "CHAIN-INDEXING")

8/2/57

SEQ	OPN (OP, SAUD)	A (OA, OAD)	X (X, XD)	SA (SA, SAD, SAX)	COMMENTS
α	TS	α+1	j	J_0	
α+1	TS	α+2	i	I_0	
α+2	TS	α+3	r	R_0	
α+3	L	L(0)			
α+4	A (replenish while Loop 2)	$L(x_{0,0})$	i,j,r G		
α+5	IBR	α+4	r	1	
α+6	BVGN	α+4	r	I	
α+7	ST	$L(x_{0,0})$	i,j		
α+8	IBR	α+3	i	1	
α+9	IBR	α+3	j	I	

	V	C	R
$C(J_0)$ =	-I	J-2	J_0
$C(I_0)$ =	1	I-2	I_0
$C(R_0)$ =	-(I+1)	3	R_1
$C(R_1)$ =	-1	1	R_2
$C(R_2)$ =	+1	1	R_3
$C(R_3)$ =	I-1	3	R_0

FOR EACH INTERIOR POINT $x_{i,j}$ ($1 \le i \le I-2$, $1 \le j \le J-2$), FORM AND STORE A NEW $x_{i,j}$ WHICH IS THE SUM OF ITS 8 NEIGHBORS.



SUBROUTINE USING "IBM" INDIRECT ADDRESSING, $C(OA_N + C(X_N))$

MATRIX MULTIPLY (A×B=C) A, B, C STORED ROW-WISE

β CODE: CALLING SEQUENCE
α CODE: SUBROUTINE

SEQ	OPN OP	SAUD	A OA	CAD	X X	XD	SA SA	SAD	SAX	COMMENTS
β	TSL		α				13			
β+1			A_0	0	1,4	G				
β+2			B_0	0	3,5	G				
β+3			C_0	0	2,3	G				
β+4			V 0		C I		R β+4			INDEX WORDS (SEE p. 21)
β+5			V 0		C K		R β+5			
β+6			V 0		C J		R β+6			
β+7	CONTROL RETURNS		HERE							
α	TS		α+1		4		4		13	
α+1	TS		α+2		2,3	G	5		13	
α+2	TS		α+3		1,5	G	6		13	
α+3	L	S	L(0)				Σ			
α+4	M	L	2	1	13		1	1	13	
α+5	A	S	Σ				Σ			
α+6	ICBR		α+7		5		5		13	
α+7	IBR		α+4		1		1			
α+8	ST		3	1	13					
α+9	IBR		α+3		3		1			
α+10	ICBR		α+11		2		5		13	
α+11	ICBR		α+3		4		6		13	
α+12	TSL		7		13					

NOTE: THE PRESERVING AND RESTORING OF INDEX REGISTERS 1 THROUGH 5 HAS NOT BEEN INCLUDED IN THE SUBROUTINE. TO DO THIS WOULD ADD 5 INSTRUCTIONS + 1 CONSTANT. (OR 4+2)

8/8/57