December 4, 2000

This is Chapter 3 of the TX-2 User's Handbook
dated August, 1963.

Help in obtaining the rest of this document,
and other material on TX-2, including software,
would be appreciated.

Al Kossow (aek@spies.com)

CHAPTER 3 - OPERATION CODE

TABLE OF CONTENTS

Page

3-1 BRIEF GUIDE TO THE ABBREVIATIONS 3-3

3-2 OP CODE DESCRIPTIONS - (For In Out, See Chapter 4.) 3-5

3-2.1 LOAD-STORE CLASS 3-5

LDA, LDB, LDC, LDD, (LDE) - LOAD - 3-6

STA, STB, STC, STD, (STE) - STORE - 3-8

EXA - EXCHANGE 3-10

3-2.2 INDEX REGISTER CLASS 3-13

RSX - Reset Index 3-14

DPX - Deposit Index 3-16

EXX - Exchange Index 3-18

AUX - Augment Index 3-20

ADX - Add Index 3-22

SKX - Skip on Index 3-24

JPX - Jump on Positive Index 3-26

JNX - Jump on Negative Index 3-26

3-2.3 JUMP-SKIP CLASS 3-29

JMP - Jump (with variations) 3-30

JPA - Jump on Positive Accumulator 3-32

JNA - Jump on Negative Accumulator 3-32

JOV - Jump on Overflow 3-32

SKM - Skip on Bit 3-34

SED - Skip if E Differs 3-36

3-2.4 SCALE, NORMALIZE CYCLE 3-37

SCA, SCB, SAB - Scale 3-38

NOA, NAB - Normalize 3-40

CYA, CYB, CAB - Cycle 3-42

Page

3-2.5 LOGIC, INSERT, COMPLEMENT/PERMUTE 3-45

ITA, UNA, DSA, ITE - Logic 3-46

INS - Insert 3-48

COM - Complement/Permute 3-50

3-2.6 CONFIGURATION MEMORY CLASS 3-53

SPF, SPG - Specify 3-54

FLF, FLG - File 3-55

3-2.7 ARITHMETIC CLASS 3-57

ADD, SUB 3-58

MUL 3-60

DIV 3-62

TLY 3-65

3-3 OPERATION CODE CHART (Wesley A. Clark) 3-67

3-3.1 NUMBER SYSTEMS 3-68

3-3.2 GLOSSARY OF TERMS 3-68

OPERATION CODE CHART 3-71

3-3.3 NOTES ON THE CODING CHART 3-73

3-4 CHAPTER 3 INDEX (Alphabetical and Numerical) 3-77

3-1 BRIEF GUIDE TO THE ABBREVIATIONS

X_j	X Memory Register "j"
$[X_j]$	Contents of X Memory Register j
T	STUV memory address "T" (STUV memory is "S", "T", "U", and "V" memories)
T_j	$T + [X_j]$
$[T_j]$	Contents of STUV Memory Register T_j
F_α	F memory register α
$[F_\alpha]$	Contents of F memory register α
$^\alpha[T_j]$	$[T_j]$ Configured as specified by α
q	Quarter
L	Left Half
R	Right Half
S	Sign of
SE	Sign Extended (i.e. "With Sign Extension")
==>	Is copied into (Goes into)

Examples:

$^\alpha[T]$ ==> A	The configured contents of STUV memory register T goes into the accumulator.
Sq3(A) ==> q4A	The sign of quarter 3 of A is copied into all of quarter 4 of the accumulator.
$[X_j]$ ==> L(T)	The contents of X memory register j goes into the left half of STUV register T.
L[T] ==> X_j	The left half of STUV register T goes into X register j.
q1$[T_j]$ ==> F_α	Quarter one of the contents of STUV memory T_j is copied into F memory register α.

The notation below is borrowed from the M4 Utility system. (See Chapter 6.)

{w}	Register Containing w
*	Deferred address
A,B,C,D,E	The AE addresses: 377604, 377605, 377606, 377607, and 377610
#	The current location - i.e. the location of the instruction being performed.

3-2 Op Code Descriptions

3-2.1 LOAD, STORE, EXCHANGE

LDA
LDB
LDC
LDD
LDE

STA
STB
STC
STD
STE

EXA

LOAD AE (24-27)

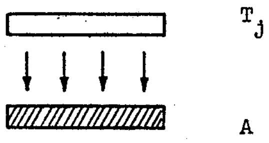
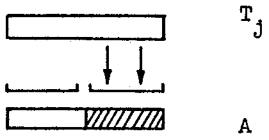
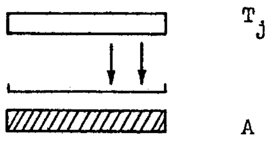
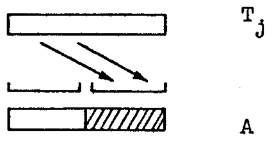
LOAD E REGISTER (20)

LDA, 24
LDB, 25
LDC, 26
LDD, 27
LDE, 20

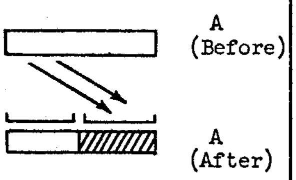
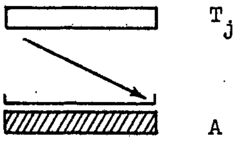
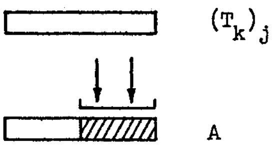
$^{\alpha}$LDA T_j	$^{\alpha}[T_j]$ ==> A

LOAD means copy into the AE from STUV memory. STUV memory is not changed. Activity, Sign Extension, and permutation are used. ALL load instructions except LDE perform the standard $[T_j]$ ==> E.

EXAMPLES: **(Standard F memory - Chart 7-2)

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENT
1.	LDA T_j		$[T_j]$ ==> A $[T_j]$ ==> E	Since all four quarters are active, subword form is immaterial. ^{20}LDA or ^{30}LDA would be equivalent.
2.	^{1}LDA T_j		$R[T_j]$ ==> R(A) $[T_j]$ ==> E	The left half of A is not changed.
3.	^{11}LDA T_j $[F_{11}]$ = 140		$R[T_j]$ ==> R(A) $SR[T_j]$ ==> L(A) $[T_j]$ ==> E	The 18 bit word from STUV is "expanded" to 36 bits through "sign extension."
4.	^{2}LDA T_j		$L[T_j]$ ==> R(A) $[T_j]$ ==> E	A "Right Half Load" - the left half of A is not affected.

**All examples apply directly to LDA, LDB, LDC, and LDD. LDE is essentially the same - only the final M to E copy is omitted.

5.	^{2}LDA A		$L[A] \Longrightarrow R(A)$ $[A] \Longrightarrow E$	The left half of A is unchanged. The right half becomes the same as the left. In a similar manner, ^{22}LDA A sets the left equal to the right. ^{12}LDA would clear the left half word through sign extension.
6.	^{16}LDA T_j $[F_{16}] = 163$		$q4[T_j] \Longrightarrow q1(A)$ $Sq4[T_j] \Longrightarrow q2,3,4(A)$ $[T_j] \Longrightarrow E$	The nine bit number in quarter 4 of T_j is expanded to 36 bits in A.
7.	^{1}LDA $(T_k)^*_j$		$R[(T_k)_j] \Longrightarrow R(A)$ $[(T_k)_j] \Longrightarrow E$	This is double indexing. $(T_k)_j \equiv T+[X_k]+[X_j]$. (It is not always faster because the defer cycle takes time also.)

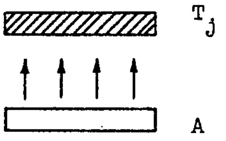
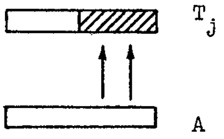
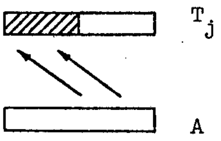
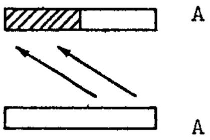
STORE AE (34-37)
STORE E (30)

STA, 34
STB, 35
STC, 36
STD, 37
STE, 30

$^{\alpha}$STA T_j	$^{\alpha}$[A] ==> T_j

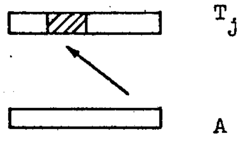
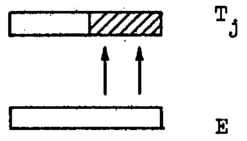
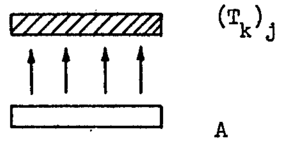
STORE is a non-destructive copy from AE to STUV memory. With a partially active configuration it becomes a partial store. Subword form is meaningless - only active pathways are used. The E register is set from the memory word after the store operation (except for STE which does not change E).

EXAMPLES: **(Standard F Memory - Chart 7-2)

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENT
1.	STA T_j		[A] ==> T_j **	T_j is set from A, A is not changed. Since all quarters are active, all are copied into T_j.
2.	^{1}STA T_j		R[A] ==> R(T_j) **	Since there is no sign extension, ^{11}STA would have the same affect. [F_{11}] = 140
3.	^{2}STA T_j		R[A] ==> L(T_j) **	^{12}STA would be exactly the same. [F_{12}] = 142
4.	^{2}STA A		R[A] ==> L(A) **	This sets the left equal to the right (as does ^{22}LDA A). Since there is no sign extension on STA, ^{12}STA would do the same. [F_{22}] = 232

** After the store operation is complete, the new content of T_j is copied into E except for the STE instruction which does not change E.

STA, 34
STB, 35
STC, 36
STD, 37
STE, 30

5.	^{5}STA T_j $[F_5]$ = 762		q1[A] ==> q3(T_j) **	Quarter 1 is copied into quarter 3 of T_j. The rest of T_j is unchanged.
6.	^{1}STE T_j (Store E)		R[E] ==> L(T_j)	Stores in the right half only - useful for setting address sections - (For example, at start of sub-routines entered via hJPQ).
7.	STA $(T_k)_j^*$		[A] ==> $(T_k)_j$ **	Double indexing - $(T_k)_j \equiv T+[X_k]+[X_j]$

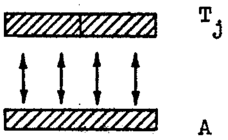
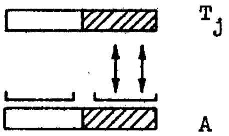
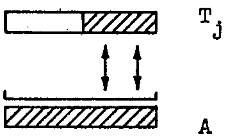
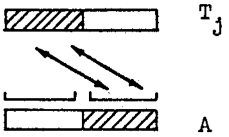
EXCHANGE A (54)

$^{\alpha}$EXA T_j	$^{\alpha}[A] \Longrightarrow T_j$
	$^{\alpha}[T_j] \Longrightarrow A$

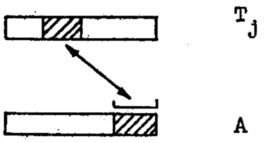
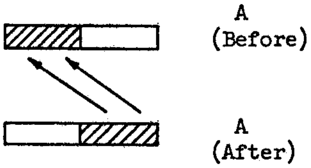
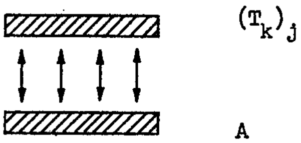
EXCHANGE A is a combination of STA and LDA. Sign extension, if any, occurs only in A and after the exchange of data. Subword form, Activity, and permutation are all used.

The E register is set equal to the STUV memory word used.

EXAMPLES:

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENT
1.	EXA T_j		$[T_j] \Longrightarrow A$ $[A] \Longrightarrow T_j$	**
2.	^{1}EXA T_j		$R[T_j] \Longrightarrow R(A)$ $R[A] \Longrightarrow R(T_j)$	**
3.	^{11}EXA T_j $[F_{11}] = 140$		$SR[T_j] \Longrightarrow L(A)$ $R[T_j] \Longrightarrow R(A)$ $R[A] \Longrightarrow R(T_j)$	Sign extension occurs in A, but not in T_j. **
4.	^{2}EXA T_j		$L[T_j] \Longrightarrow R(A)$ $R[A] \Longrightarrow L(T_j)$	**

** The two copy operations that perform an exchange take place simultaneously. Remember also that E is changed - it is set equal to the final contents of the STUV memory word.

5.	^{5}EXA T_j $[F_5]$ = 762		$q3[T_j]$ ==> q1A q1[A] ==> $q3(T_j)$	**
6.	^{2}EXA A		R[A] ==> L(A)	When "A" is used as the address section, EXA has the same affect as STA. No exchange is made, and there is no sign extension
7.	EXA $\{T_k\}^*_j$		$[(T_k)_j]$ ==> A [A] ==> $(T_k)_j$	Double indexing: $(T_k)_j = T+[X_k]+[X_j]$

3-2.2 Index Register Class

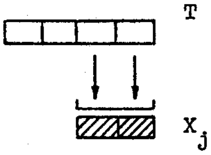
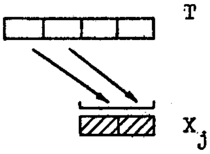
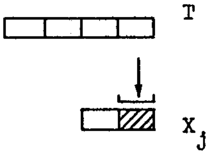
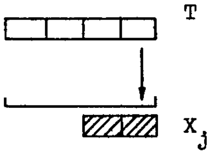
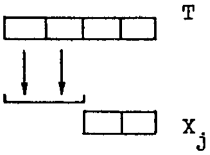
```
RSX
DPX
EXX
AUX
ADX
SKX  ------->  **REX, SEX
JPX             INX
JNX             DEX
                SXD
                SXL
                SXG
                RXF
                RDX
                RFD
```

** Supernumerary Mnemonics for SKX.

$^{\alpha}RSX_j$ T	$^{\alpha}[T] ==> X_j$

RESET is a non-destructive copy from STUV memory into X memory.
Subword form, Activity, and Permutation are used.
The E register is set equal to the STUV memory word used. (Usually "T", but see example 7.)

EXAMPLES: (Standard Configurations - Chart 7-2)

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED EXPLANATION	COMMENT
1.	$^1RSX_j$ T		$R[T] ==> X_j$ [T] ==>E	^{0}RSX would do the same.
2.	$^2RSX_j$ T		$L[T] ==> X_j$ [T] ==> E	^{12}RSX would do the same. $[F_{12}]$ = 142
3.	$^3RSX_j$ T		$q1[T] ==> R(X_j)$ [T] ==> E	The right half of X_j is set from T. The left nine bits are not changed.
4.	$^{13}RSX_j$ T $[F_{13}]$ = 160		$q1[T] ==> R(X_j)$ $Sq1(T) ==> L(X_j)$ [T] ==> E	Sign of quarter 1 of T is extended throughout the left half of X_j. The right half is set as above. ^{33}RSX would do the same. $[F_{33}]$ = 320
5.	$^{21}RSX_j$ T $[F_{21}]$ = 230		[T] ==> E	Nothing happens (other than changing E).

DEPOSIT INDEX (DPX, 16)

$^{\alpha}\mathrm{DPX}_j$ T	$^{\alpha}[X_j] \Rightarrow$ T

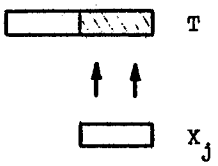
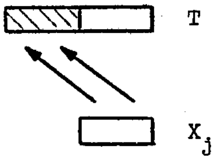
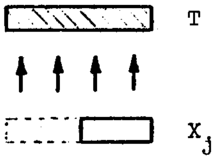
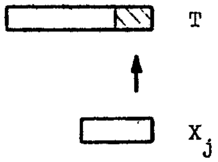
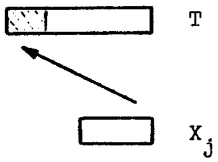
DEPOSIT is a non-destructive copy from X memory into STUV memory.

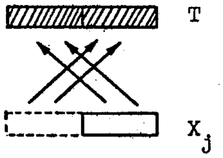
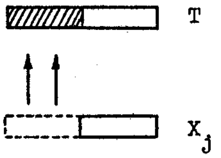
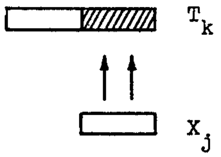
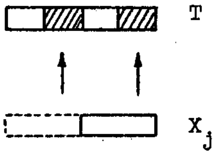
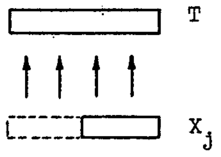
Activity and Permutation are used.

The X memory word is expanded to a full 36 bit subword by extending bit 2.9 (the X register sign bit) but only active quarters are used. (The subword form is immaterial.)

The E register is set equal to the STUV memory used. (Usually "T", but see examples 8 and 10.)

EXAMPLES: (Standard F Memory - Chart 7-2)

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED EXPLANATION	COMMENT
1.	$^{1}\mathrm{DPX}_j$ T		$[X_j] \Rightarrow R(T)$	Only the right half of T is changed.
2.	$^{2}\mathrm{DPX}_j$ T		$[X_j] \Rightarrow L(T)$	Only the left half of T is changed.
3.	DPX_j T		$[X_j] \Rightarrow R(T)$ $SX_j \Rightarrow L(T)$	All of T is used. Note that DPX_0 T (or DPX T) is a handy clear instruction. ($[X_0] \equiv +0$ and cannot be changed.)
4.	$^{3}\mathrm{DPX}_j$ T		$R[X_j] \Rightarrow q1(T)$	Only quarter 1 of T is changed.
5.	$^{16}\mathrm{DPX}_j$ T $[F_{16}] = 163$		$R[X_j] \Rightarrow q4(T)$	Only quarter 4 is changed for only one path is active.

6.	$^{17}DPX_j$ T $[F_{17}] = 202$		$SX_j \Longrightarrow R(T)$ $[X_j] \Longrightarrow L(T)$	All of T is affected.
7.	$^{21}DPX_j$ T $[F_{21}] = 230$		$SX_j \Longrightarrow L(T)$	Surprisingly enough, this does do something. (See example 5, RSX.)
8.	$^{1}DPX_j\ \{T_k\}^*$		$[X_j] \Longrightarrow T_k$ $[T_k] \Longrightarrow E$	Deposit is indexable with deferred addressing.
9.	$^{33}DPX_j$ T $[F_{33}] = 320$		$SX_j \Longrightarrow q3(T)$ $[X_j] \Longrightarrow q1(T)$	Note that bit 2.9 of X_j is used even though quarter 2 is not active.
10.	DPX 377720		$[X_j] \Longrightarrow R(E)$ $SX_j \Longrightarrow L(E)$	V memory, except the A, B, C, D, and E registers can not be changed by any instruction. Note that E is set to "what-would-have-gone-into-T."

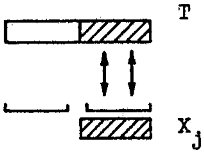
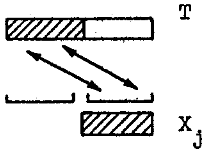
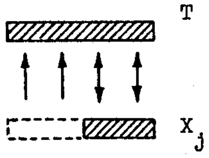
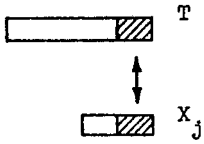
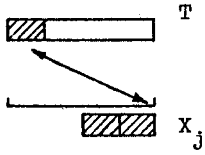
EXCHANGE INDEX (EXX, 14)

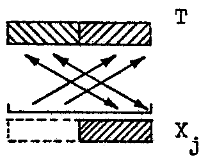
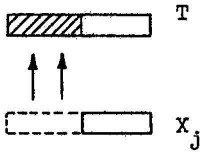
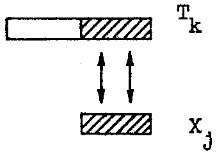
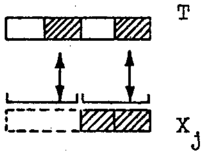
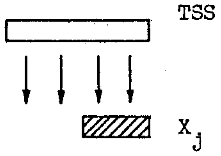
$^{\alpha}EXX_j$ T	$^{\alpha}[X_j]$ ==> T $^{\alpha}[T]$ ==> X_j

EXX is a combination of RSX and DPX. Except for sign extension, it does just what its name implies - i.e. it will interchange words between X memory and STUV memory.

Subword Form, Activity, and Permutation are used. The E register is set equal to the STUV memory word used.

EXAMPLES: (Standard F Memory - Chart 7-2.)

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED EXPLANATION	COMMENT
1.	$^{1}EXX_j$ T		R[T] ==> X_j [X_j] ==> R(T) [T] ==> E	
2.	$^{2}EXX_j$ T		L[T] ==> X_j [X_j] ==> L(T) [T] ==> E	
3.	EXX_j T		R[T] ==> X_j [X_j] ==> R(T) S(X_j) ==> L(T) [T] ==> E	Note that left half of T is cleared.
4.	$^{3}EXX_j$ T		q1[T] ==> R(X_j) R[X_j] ==> q1(T) [T] ==> E	Nine bit exchange.
5.	$^{16}EXX_j$ T [F_{16}] = 163		R[X_j] ==> q4(T) q4[T] ==> R(X_j) Sq4(T) ==> L(X_j) [T] ==> E	Sign is extended in X_j but not in T.

6.	$^{17}EXX_j$ T $[F_{17}]$ = 202		$[X_j]$ ==> $L(T)$ $L[T]$ ==> X_j $S(X_j)$ ==> $R(T)$ $[T]$ ==> E	Sign of X_j is extended into the right half of T.
7.	$^{21}EXX_j$ T $[F_{21}]$ = 230		$S(X_j)$ ==> $L(T)$ $[T]$ ==> E	Same as $^{21}DPX_j$ T.
8.	$^{1}EXX_j$ $(T_k)^*$		$R[T_k]$ ==> X_j $[X_j]$ ==> $R(T_k)$ $[T_k]$ ==> E	EXX is indexable if a deferred address is used.
9.	$^{33}EXX_j$ T $[F_{33}]$ = 320		$Sq1[T]$ ==> $L(X_j)$ $q1[T]$ ==> $R(X_j)$ $R[X_j]$ ==> $q1(T)$ $S(X_j)$ ==> $q3(T)$ $[T]$ ==> E	Note that bit 2.9 is used for sign extension (not 1.9).
10.	$^{1}EXX_j$ 377720		$R[377720]$ ==> X_j $[X_j]$ ==> $R(E)$ $L[377720]$ ==> $L(E)$	Same as $^{1}RSX_j$ 377720. (Toggle registers must be changed by hand. Note that E is set to what would have gone into T.)

AUGMENT INDEX (AUX, 10)

$^{\alpha}AUX_j$ T	$[X_j] + {}^{\alpha}[T] ==> X_j$

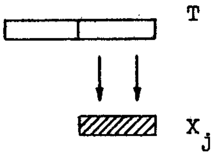
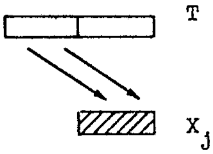
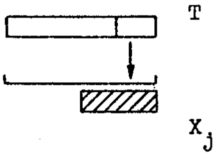
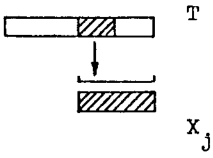
AUX forms an 18 bit ring sum in X_j. There is no overflow detection. All of X_j is affected. STUV memory is not affected.

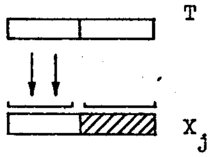
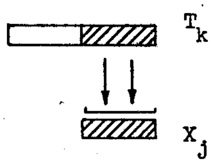
Activity and permutation are used. Sign extension applies to the operand taken from STUV memory. If quarters 1 and 2 are active, subword form is immaterial.

If one quarter of the STUV memory operand is inactive (as in standard configuration #3, for example), +0 is used for that quarter.

The E register is set equal to the STUV memory word. (This is "T" except when a deferred address is used. See example 6.)

EXAMPLES: (Standard F Memory - Chart 7-2.)

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED EXPLANATION	COMMENT
1.	$^{1}AUX_j$ T		$[X_j] + R[T] ==> X_j$ $[T] ==> E$	Standard configurations #0, 11, 20, and 30 would do the same. $[F_{11}] = 140$ $[F_{20}] = 200$ $[F_{30}] = 600$
2.	$^{2}AUX_j$ T		$[X_j] + L[T] ==> X_j$ $[T] ==> E$	Standard configuration #12 would do the same. $[F_{12}] = 142$
3.	$^{13}AUX_j$ T $[F_{13}] = 160$		$[X_j] + q1[T]_{SE} ==> X_j$ $[T] ==> E$	Standard configuration #33 would do the same (but NOT #3!) (See note on next page.) $[F_{33}] = 320$
4.	$^{\alpha}AUX_j$ T $[F_{\alpha}] = 220$		$[X_j] + q2[T]_{SE} ==> X_j$ $[T] ==> E$	This has sign extension to the right. (There is no suitable standard configuration.)

5.	$^{21}AUX_j$ T		$[X_j]$ + (+0) ==> X_j [T] ==> E	Register T is ignored, and X_j is not changed. Except for E, this instruction is innocuous.
6.	$^{1}AUX_j$ $(T_k)^*$		$[X_j]$ + R$[T_k]$ ==> X_j $[T_k]$ ==> E	Same as example 1, but indexed via a deferred address.

NOTE: E is cleared and then loaded as if by $^{\alpha}$LDE. The sum of R[E] and $[X_j]$ then goes into X_j (circuitously) and E is set equal to the STUV register used (ie.[T] or $[T_k]$ if a deferred address was used). X_j is always set. Note - If either quarter 1 or 2 is not part of an active subword, (as, for example, with standard configuration #3) one operand of the sum is not completely specified and +0 will be used as that part of the operand.

ADD INDEX (ADX, 15)

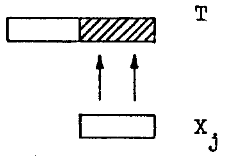
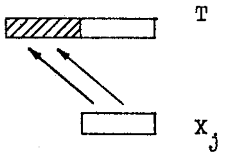
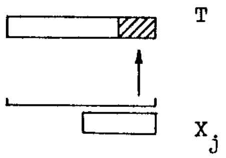
${}^{\alpha}\mathrm{ADX}_j$ T	$[X_j] + {}^{\alpha}[T]$ ==> T

ADX forms an 18 bit ring sum usually in STUV memory although only the active quarters are stored. There is no overflow detection. The operands are always 18 bit words - one from X memory the other from STUV memory. A configuration should be chosen such that the word from STUV memory has both quarters active, or is an extended 9 bit subword. If only one quarter is active, the inactive quarter of the operand is set to +0.

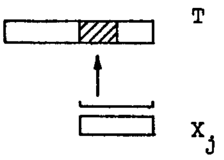
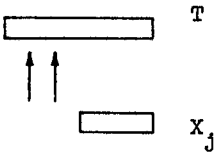
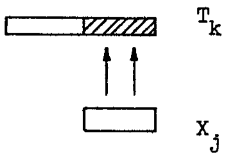
Activity and Permutation are used. Only active quarters are stored, but sign extension applies to the operand taken from STUV memory.

The E register is set equal to the STUV memory word used. (This is "T" except when a defer is involved. See example 6.)

EXAMPLES: (Standard F Memory - Chart 7-2)

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED EXPLANATION	COMMENT
1.	${}^{1}\mathrm{ADX}_j$ T		$[X_j] + R[T]$ ==> RT [T] ==> E	Left half of T is not changed. The sum is standard 18 bit ring sum, also called "ones complement sum."
2.	${}^{2}\mathrm{ADX}_j$ T		$[X_j] + L[T]$ ==> LT [T] ==> E	Right half of T is not changed.
3.	${}^{13}\mathrm{ADX}_j$ T $[F_{13}] = 160$		$[X_j] + ql[T]_{SE}$ ==> ql(T) [T] ==> E	This gives a 9 bit ring sum. Configuration 33 would do the same, #3 would not. See note next page. The subword length should be 18 bits.

NOTE: In example 3, the 9 bit result is an honest 9 bit ring sum only when X_j contains an extended 9 bit word. (See RSX, example 4.) ADX cannot be used to add a 9 bit word to an 18 bit word. Use AUX.

4.	$^{\alpha}ADX_j$ T $[F_{\alpha}]$ = 220		$[X_j] + q2[T]_{SE}$ ==> q2(T) [T] ==> E	Essentially the same as example 3 except that the left half of X_j is significant. $[F_{\alpha}]$ illustrated is 220. There is no suitable standard configuration.
5.	$^{21}ADX_j$ T $[F_{21}]$ = 230		[T] ==> E	"Nothing" is done here because quarters 1 and 2 are both inactive.
6.	$^{1}ADX_j$ $\{T_k\}^*$		$[X_j] + R[T_k]$ ==> RT_k $[T_k]$ ==> E	Same as example 1, but indexed via deferred indexing.

NOTE: E is cleared and then loaded as if by $^{\alpha}$LDE. The sum of R[E] and $[X_j]$ then goes into E and an $^{\alpha}$STE is performed. Inactive quarters of the STUV memory word therefore remain unchanged. If either quarter 1 or 2 is not part of an active subword (as, for example, with standard configuration #3), one operand of the sum is not fully specified and +0 is used to fill out the operand.

SKIP ON INDEX (SKX, 12)

$$\boxed{^{\alpha}SKX_j\ T}$$

SKX (or REX, or SEX) provides 32 combinations of setting, adding, comparing, skipping, flag raising, and dismissing - all relating to X memory and without changing the AE or the E register. (See examples below.)

F memory is not used. The configuration syllable specifies the desired combination. (Examples 1 - 8 show the use of bits 4.6, 5, 4 and examples 10 - 12 illustrate bits 4.8 and 4.7.)

"T", the address syllable, (or the final deferred address) is used as an <u>OPERAND</u>.

EXAMPLES:

NO.	INSTRUCTION	MNEMONIC ABBREVIATION (See Chart 7-3)	ABBREVIATED DESCRIPTION	COMMENT
1.	$^0SKX_j$ T	SKX_j T REX_j T SEX_j T (Set)	T ==> X_j	STUV memory is not used - "T" <u>is</u> the <u>operand</u>, not its location. The brackets [ ]. were left out on purpose.
2.	$^1SKX_j$ T	(Set negative)	-T ==> X_j	"Minus" T - i.e. its ones complement is used to set X_j.
3.	$^2SKX_j$ T	INX_j T (Increase)	$[X_j]$ + T ==> X_j	If the sum is zero, it will be -0 (all ones) unless $[X_j]$ was initially +0.
4.	$^3SKX_j$ T	DEX_j T (Decrease)	$[X_j]$ + (-T) ==> X_j	"-T" is added to $[X_j]$. Zero is -0. It cannot be +0.
5.	$^4SKX_j$ T	SXD_j T (Skip if X differs.)	If $[X_j] \neq$ T Skip - (i.e. #+2 ==> P)	Skip if $[X_j]$ differs from T. Note: (+0) = (-0) and if $[X_j]$ is initially (+0), it is changed to (-0).
6.	$^5SKX_j$ T	(Skip if X differs from negative.)	If $[X_j] \neq$ -T Skip - (i.e. #+2 ==> P)	Skip if $[X_j]$ differs from -T. Note: (-0) = (+0) and if $[X_j]$ is initially (-0), it is changed to +0.
7.	$^6SKX_j$ T	SXL_j T (Skip if X is less.)	If $[X_j]$ < T Skip - (i.e. #+2 ==> P)	Skip if $[X_j]$ is less than T and if $[X_j]$ -T does not overflow. (Skip range: T-377777 to T) Note: If $[X_j]$is initially (+0), it is changed to (-0).

8.	^{7}SKX$_j$ T	SXG$_j$ T (Skip if X is greater.)	If $[X_j]$ > -T Skip i.e. #+2 ==> P	Skip if $[X_j]$ is greater than -T and if $[X_j]$ + T does not overflow. (Skip range: -T to 377777-T) Note: If $[X_j]$ is initially (-0), it is changed to (+0).
9.	SKX$_j$ $(T_k)^*$	REX$_j$ $(T_k)^*$	T+$[X_k]$ ==> X_j	$[X_j]$ is set equal to T_k. e.g. a) SKX$_j$ $(0_k)^*$ ≡ set X_j from X_k. b) ^{1}SKX$_j$ $(0_j)^*$ ≡ Complement X_j.
10.	^{10}SKX$_j$ T	RXF$_j$ T (Reset and raise flag.)	T ==> X_j 1 ==> Flag$_j$	For j = 1 to 37_8, RXF is the same as ^{0}SKX for there are no flags for these numbers. Note that flag zero can be raised.
11.	^{20}SKX$_j$ T	RXD$_j$ T (Reset and Dismiss.) (See note 3)	T ==> X_j DISMISS	See Chapter 4 for the ramifications of "DISMISS." If j = the current sequence number, "T" is nearly immaterial for the subsequent change of sequence will change X_j.
12.	^{30}SKX$_j$ T	RFD$_j$ T (Reset, Raise flag, and Dismiss.)	T ==> X_j 1 ==> Flag$_j$ DISMISS	This is used to change sequence number - often in the form - ^{30}SKX$_j$ #+1. It is ignored if j = current sequence number.

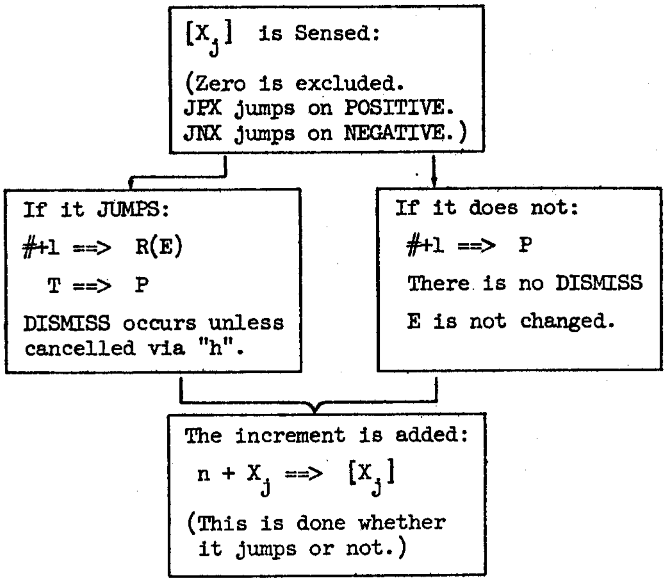
Notes:
1. "Skip" means "omit the next instruction." i.e. "Go to #+2."
2. The configuration syllable is united with the rest of the instruction. It may be given redundantly. e.g. DEX is the same as ^{3}SKX or ^{1}INX or ^{3}DEX.
3. The hold bit cancels DISMISS. (h ^{20}SKX is the same as SKX alone.)
4. RXF cannot be used as a Jump. Index register "j" is indeed set, but it will not be copied into the P register, unless a change of sequence number occurs. (See Chapter 4.)

JUMP ON POSITIVE INDEX (JPX, 06)

JUMP ON NEGATIVE INDEX (JNX, 07)

$\boxed{{}^{n}JPX_{j}\ \ T}$

JPX and JNX are "Loop-closing", "Index-sensing" jump instructions. Their operation is as follows:



$[X_j]$ is Sensed:

(Zero is excluded.
JPX jumps on POSITIVE.
JNX jumps on NEGATIVE.)

If it JUMPS:
#+1 ==> R(E)
T ==> P
DISMISS occurs unless cancelled via "h".

If it does not:
#+1 ==> P
There is no DISMISS
E is not changed.

The increment is added:
$n + X_j \Longrightarrow [X_j]$
(This is done whether it jumps or not.)

Note:
1. If the sum is zero, it is -0.
2. "n" is a signed integer: -17 to $+17_8$.
3. F Memory is not used.
4. A deferred address determines where to jump to, but not if, and the second index register is not modified.

EXAMPLES:

1. Straight Table Scan (100 register table located at "TABL.")

a.) JPX

Start → REX_j 77

Loop → LDA $TABL_j$

h^{-1} JPX_j Loop

This program scans the table "backward through the manuscript." (i.e., highest memory location first.) Note: X_j is initially set to + (n-1).

b.) JNX

Start → ${}^{1}SKX_j$ 77

Loop → LDA $(TABL + 77)_j$

h^{+1} JNX_j Loop

This program scans "forward through the manuscript." (i.e., lowest memory location first.) Note: X_j is initially set to - (n-1).

2. To scan every n^{th} table register

a)
```
START → REX_j (TL - n)
        LDA  TABL_j
        h^-n JPX_j #-1
```

b)
```
START → ^1REX_j (TL - n)
        LDA_j TABL + TL - n
        h^+n JNX_j #-1
```

These programs run for $(\frac{TL}{n})$ iterations if we assume that TL (Table Length) is an integer multiple of n. As written, they scan the first register of each block of n registers. To scan register "i" of each block, the LDA instruction could be written LDA $(TABL + i)_j$ for example "a" (JPX) and LDA $(TABL + i + TL - n)_j$ for example "b" (JNX).

3. Interlaced Table Scan

Scope flicker can be reduced by an interlaced table scan. The fact that the change in X_j is made <u>after</u> the jump decision causes a somewhat peculiar parameter configuration, but the program logic is essentially the same as above. For example, if "C" is the interlace, "TL" is the Table Length, and if "C" is <u>not</u> a factor of "TL," the program below scans the whole table with an interlace of C. (If "C" is a factor of TL, the program degenerates to example 2a.)

```
START - ^1REX_j C
        INX_j TL
        LDA (TABL + C - 1)_j
        h^-C JPX_j #-1
        JMP #-3
```

NOT SO
e.g. TL=10
C=4

X	L
	9
6	5
2	1
-2	7
4	3
0	9
6	
⋮	⋮

If C = 3, and TL = 7, the table is scanned in the following order: 6, 3, 0, 4, 1, 5, 2, 6, 3, 0, etc.

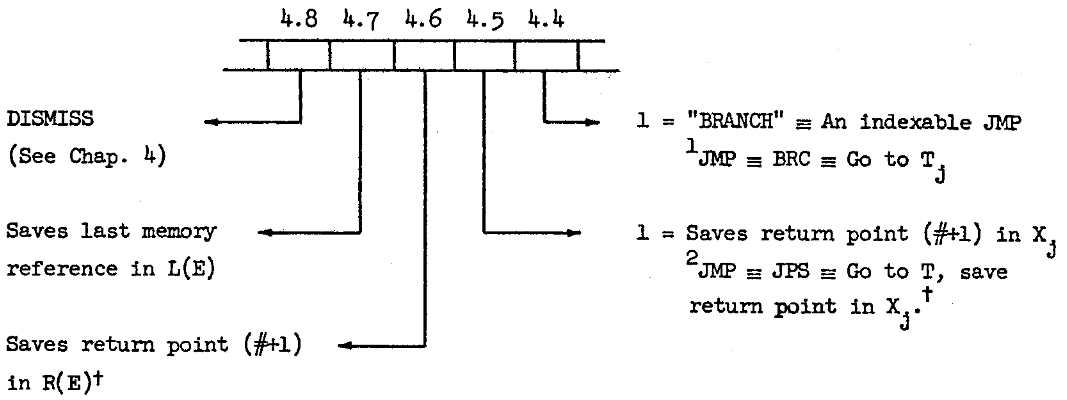
NOTE:
1. "Zero" used as an address (as above) is always +0.
2. M4 automatically puts a hold bit on JPX and JNX to cancel the automatic dismiss (see Chapter 4 and Chapter 6).
3. The address of a deferred JNX or JPX is completely determined <u>before</u> the index register is changed. Therefore a $^{-1}JPX_{a|a}$ S would jump to S_a as defined by the <u>original</u> contents of X_a - if it jumps at all.

3-2.3 JUMP SKIP CLASS

JMP
JPA
JNA
JOV
SKM
SED

$\boxed{{}^{\alpha}\text{JMP } T_j}$

JMP is an unconditional transfer of control. It means go to T (or T_j) for the next set of instructions. The configuration syllable "α" does not refer to F memory but is used directly to provide 32 variations of JMP as illustrated below:



DISMISS
(See Chap. 4)

Saves last memory
reference in L(E)

Saves return point (#+1)
in R(E)†

1 = "BRANCH" ≡ An indexable JMP
^{1}JMP ≡ BRC ≡ Go to T_j

1 = Saves return point (#+1) in X_j
^{2}JMP ≡ JPS ≡ Go to T, save return point in X_j.†

EXAMPLES: (See #10.)

NO.	INSTRUCTION	SUPERNUMERARY MNEMONIC	JUMPS TO	COMMENT
1.	^{0}JMP T_j	JMP T_j	T	X_j is ignored.
2.	^{1}JMP T_j	BRC T_j (Branch)	T_j	Indexable Jump ≡ BRANCH
3.	^{2}JMP T_j	JPS T_j (Jump and Save)	T	Jump and save return point (#+1) in the specified index register (X_j).
4.	^{3}JMP T_j	BRS T_j (Branch and Save)	T_j	Branch and save. X_j is used to evaluate the jump destination T_j and is then reset to the return point (#+1).
5.	^{4}JMP T_j	-	T	X_j is ignored, #+1 is saved in R(E)
6.	^{5}JMP T_j	^{4}BRC T_j	T_j	Return point (#+1) is saved in R(E)
7.	^{6}JMP T_j	^{4}JPS T_j	T	Return point (#+1) is saved in R(E) and also in X_j.

† In M4 terminology, the symbol "#" is an abbreviation for the location of the current instruction. (See Chapter 6.)

8.	^{7}JMP T_j	^{4}BRS T_j	T_j	X_j is used to determine the jump destination T_j and is then reset to the return point (#+1). The return point is saved in R(E) as well.
9.	^{10}JMP T_j	-	T	The memory location of the last data reference is saved in L(E). (i.e. the contents of the Q register)
10.	^{14}JMP T	JPQ T	T	Jump, save "p" (i.e. #+1) and "q" (location of last data reference). This is the recommended jump, for the information saved is often of use in checkout.
11.	^{15}JMP T_j	BPQ T_j	T_j	This instruction is the same as JPQ except that the jump destination is indexed.
12.	^{16}JMP T_j	JES T_j	T	Jump, save in E, and in X_j.
13.	^{20}JMP T_j	JPD T_j	T	Jump, Dismiss.
14.	^{21}JMP T_j	BRD T_j	T_j	Branch, Dismiss.
15.	^{22}JMP T_j	JDS T_j	T	Jump, Dismiss, Save in X_j.
16.	^{23}JMP T_j	BDS T_j	T_j	Branch, Dismiss, Save in X_j.

Jump and save return point (#+1) in the specified index register (X_j).

NOTE: A superscript numeral can be used redundantly on supernumerary mnemonics. For example: ^{16}JMP $\equiv$ ^{16}JES $\equiv$ JES $\equiv$ ^{2}JPQ $\equiv$ ^{14}JPS etc. (M4 "unites" them into the word.)

CONDITIONAL JUMPS

JPA (46)
JNA (47)
JOV (45)

JPA - Jump on Positive Accumulator

JNA - Jump on Negative Accumulator

JOV - Jump on Overflow

$^{\alpha}$JPA T_j
$^{\alpha}$JNA T_j
$^{\alpha}$JOV T_j

The conditional jumps go to T_j if the conditions are satisfied by any active subword. Permutation is ignored. The return point (#+1) is saved in E if the jump takes place. The accumulator and overflow flip-flops are not changed. Note that these conditional jumps are indexable.

EXAMPLES:

#1. A Four-way Switch:

```
JOV    OF        ** Goes to OF if overflow exists ($Z_4$ = 1)
JNA    N1        ** Goes to N1 if A is negative.
JPA    P1        ** Goes to P1 if A is positive.
---              ** Continues if A is zero.
```

#2. Overflow:

^{30}JOV T_j is equivalent to ^{37}JOV T_j, for both configurations specify the same active subwords. If any of the four overflow flip-flops are set to 1, control will go to T_j. The overflow indicators (Z_4,Z_3,Z_2,Z_1) are not cleared by JOV.

Active subwords use the overflow indicator associated with the sign quarter, e.g. Z_2 is associated with the right half word, Z_4 with the left half word.

#3. To Detect Minus Zero in an Index Register:

(JNX$_j$ T or JPX$_j$ T will not jump on either + or - zero.)

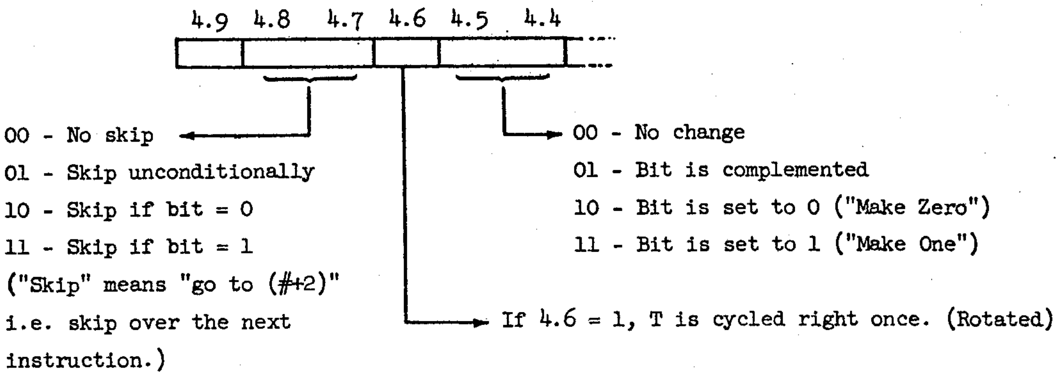
```
DPX A
$^1$DPX$_j$ A            ** (0,,-0) or (0,,+0) now in A
JPA T1               ** Goes to T1 if -0 in right half word.
                     ** Continues if +0 in both halves.
```

#4. 18 Bit Zeros Again:

```
20JPA  1P          ** One half (or both) positive - (Goes to 1P)
20JNA  1N          ** One half (or both) negative - (Goes to 1N)
  JPA  PN          ** Left (+0), Right (-0) - (Goes to PN)
  JNA  NP          ** Left (-0), Right (+0) - (Goes to NP)
                   ** Both (+0) or Both (-0) - (Continue)
```

$\boxed{{}^{c}\mathrm{SKM}_{q.b}\ \mathrm{T}}$

"Skip-on-a-bit" uses a one bit operand. It has 32 variations - some with M4 Supernumerary Mnemonics. The basic variations are as follows:

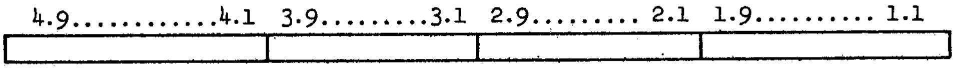


00 - No skip
01 - Skip unconditionally
10 - Skip if bit = 0
11 - Skip if bit = 1
("Skip" means "go to (#+2)"
i.e. skip over the next
instruction.)

00 - No change
01 - Bit is complemented
10 - Bit is set to 0 ("Make Zero")
11 - Bit is set to 1 ("Make One")

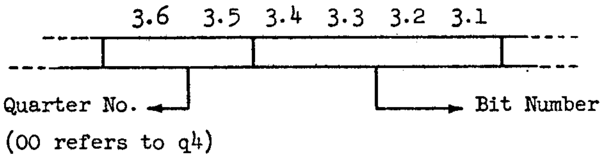
If 4.6 = 1, T is cycled right once. (Rotated)

The bit in question is identified by its quarter number and bit number as diagrammed below:



The meta bit is No. 10 (dec.). (SKM is the <u>only</u> instruction that can affect it.)
The parity bit is No. 11 (dec.).
The parity circuit is No. 12 (dec.). } These can not be changed by SKM.
(Any quarter number will do for the parity and meta bits.)

Bits and quarters are numbered from right to left and should be in subscript when used with SKM. (See chapter 6, page 6-7.) The bit designation goes in the "j bits" (3.6 - 3.1), as follows:



(When given in the form indicated above, Bit Numbers are interpreted as Decimal, e.g. 4.10 is the usual metabit designation.)

SKM is therefore non-indexable except through deferred addressing.

If a non-existent bit is selected, e.g. bit 0.0,1.0,2.0,3.0 for example, Unconditional Skips (SKU) and Rotate (CYR) will still work, but "makes" will do nothing, and conditional skips will not skip.

SUPERNUMERARY MNEMONICS (See Chart 7-3)

MKC - ^{1}SKM - Make complement
MKZ - ^{2}SKM - Make zero
MKN - ^{3}SKM - Make one

SKU - ^{10}SKM - Skip unconditionally, (go to #+2)
SUC - ^{11}SKM - Skip and complement
SUZ - ^{12}SKM - Skip and make zero
SUN - ^{13}SKM - Skip and make one

SKZ - ^{20}SKM - Skip if bit =0
SZC - ^{21}SKM - Skip on zero and complement
SZZ - ^{22}SKM - Skip on zero and make zero
SZN - ^{23}SKM - Skip on zero and make one

SKN - ^{30}SKM - Skip on one
SNC - ^{31}SKM - Skip on one and complement
SNZ - ^{32}SKM - Skip on one and make zero
SNN - ^{33}SKM - Skip on one and make one

CYR - ^{4}SKM - Cycle memory once to the right (rotate)
MCR - ^{5}SKM - Make complement and rotate
MZR - ^{6}SKM - Make zero and rotate
MNR - ^{7}SKM - Make one and rotate
SNR - ^{34}SKM - Skip on one and rotate
SZR - ^{24}SKM - Skip on zero and rotate
SUR - ^{14}SKM - Skip and rotate

NOTE: "Skip" is first, "make" next, and "rotate" last. $^{4}\mathrm{SZZ} \equiv {}^{26}\mathrm{SKM} \equiv$ Skip on zero, make zero, and then rotate.

EXAMPLES:

1. To copy a bit:

$$\left.\begin{array}{l} \mathrm{SKZ}\ Q_{2.3} \\ \mathrm{SUN}\ T_{1.1} \\ \mathrm{MKZ}\ T_{1.1} \end{array}\right\}$$

Sets bit $T_{1.1}$ equal to bit $Q_{2.3}$

2. To clear n metabits starting at T

Rex$_{\alpha}$ (n-1)
MKZ$_{4.10}|\alpha^{T}$ ** i.e. MKZ$_{4.10}\{T_{\alpha}\}$*
$^{-1}$JPX$_{\alpha}$ #-1

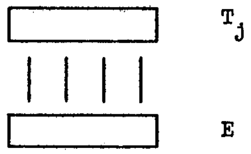
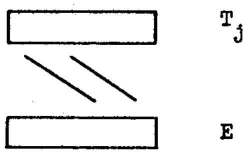
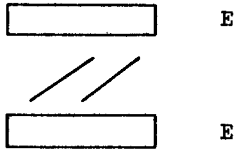
SKIP IF E DIFFERS

SED
43

$^{\alpha}$SED T_j	Only P can be changed.

SED compares all active quarters of E and T_j according to the given permutation. If any difference exists the next instruction is skipped over. No registers other than P (the central Program Counter) can be changed. (E is not changed.) Subword Form is immaterial.

EXAMPLES: (Standard F Memory - Chart 7-2.)

NO.	INSTRUCTION	DIAGRAM	COMMENT
1.	SED T_j		#+2 => P if E differs from T_j #+1 => P if they are identical
2.	^{2}SED T_j		The left half of T_j is compared to the right half of E. (^{12}SED is identical.) $[F_{12}] = 142$.
3.	^{22}SED E		The right and left halves of E are compared. ^{17}SED E, ^{2}SED E, ^{12}SED E, or ^{22}SED E would have an identical result.

3-2.4 SCALE, NORMALIZE, CYCLE

SCA
SCB
SAB
NOA
NAB
CYA
CYB
CAB

SCALE

SCA, 70
SCB, 71
SAB, 72

$^{\alpha}$SCA T_j	$^{\alpha}$[A] x $2^{^{\alpha}[T_j]}$ ==> A

"SCALE" multiplies each active subword by "a power of 2," i.e. by 2^n where n is a signed integer specified in T_j. Each active subword can be scaled a different amount. The D register is used to count the binary shifts. The details are as follows:

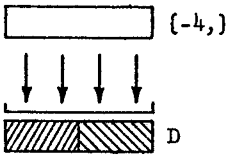
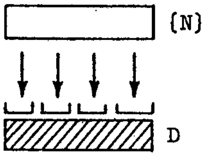
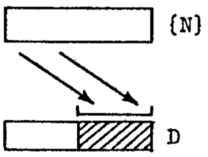
a) An $^{\alpha}$LDD T_j is performed (with permutation and sign extension as called for).

b) Each active subword (of A or AB) is scaled according to its sign quarter in D, and these sign quarters are left set to -0.

c) If an overflow exists for an active subword, the proper result is recovered by complementing the sign digit after the first shift, and the indicator is cleared. This rule is used for all operands - left (+), right (_), and zero. Overflow can not affect SCB.

Notice that SCALE amounts to shifting all the bits except the sign left or right and filling the vacant positions with copies of the sign bit (i.e. with ±0). SCALE senses overflow and corrects the sign bit if necessary. SCA and SAB always clear the overflow flip-flop - even if bits are lost off the left end. SCALE never sets the overflow flip-flop.

EXAMPLES: (SCB is illustrated to avoid overflow complications.)

NO. INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENT
1. SCB{-4,}		[B] x 2^{-4} ==> B -0 ==> q4(D) q3,2,1[T_j] ==> q321(D)	{-4,} is a M4 convention for A register with -4 in quarter 4. See Chapter 6, page 6-7 and 6-10.
2. ^{30}SCB {N} N = 2775003000$_{(8)}$		q4[B] x 2^2 ==> q4(B) q3[B] x 2^{-2} ==> q3(B) q2[B] x 2^3 ==> q2 (B) -0 ==> D	Quarter 1 of B is not changed. The sign bits are never changed. Bits may be lost off either end without any alarm.
3. ^{2}SCB {N} N = 2775003000$_{(8)}$		R[B] x 2^2 ==> R(B) -0 ==> q2(D) 775 ==> q1(D)	The left halves of B and D are not changed. Note that q4 of {N} specifies the argument of the scale operation.

Note: Scale can of course be indexed - e.g. SCA T_j where the argument comes from T_j. It is more common programming practice to use an RC word - e.g. SCA{-1,}.

4. Overflow: (SCA and SAB)

a) To "recover an overflow":

LDA {200 000 000 000}
ADD {200 000 000 000}

**Acc. will now be 400 000 000 000 (a negative number), and Z_4 (overflow bit #4) will be "1".

SCA {-3,}

**-3, ≡ 774 000 000 000. After the scale, Acc. will be 040 000 000 000 and Z_4 will be "0". Z_3,Z_2,Z_1 are not sensed nor changed.

(Any negative argument will suffice.)

b) Only active subwords are processed:

LDA {200 300 400 100}

^{30}ADD {200 300 400 300}

**Acc. will be 400 600 001 400.
**All four Z flip-flops will be "1".

^{21}SCA {774 774 774 774}

**Only L(A) is scaled. Acc. will become 040 060 001 400. Z_4 will become "0", Z_3,Z_2,Z_1 will remain "1".

^{1}SCA {774 774 774 774}

**Only R(A) is changed. Acc. becomes 040 060 700 140 and Z_2 becomes "0". Z_3 and Z_1 are still "1".

Note that Z_4,Z_3,Z_2,Z_1 are overflow indicators. They tell whether overflow has occurred. An overflow resulting from negative numbers (as in q2 above) is not treated any differently.

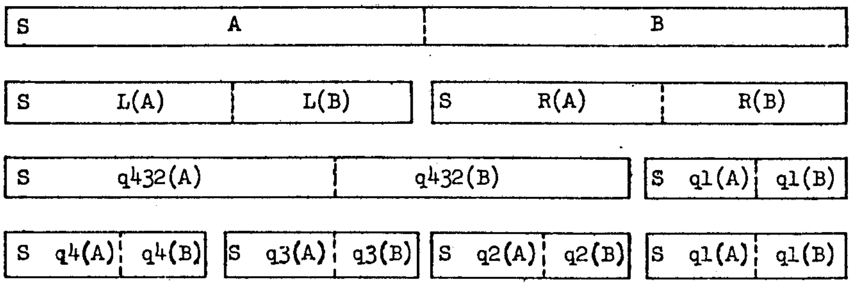
5. Subword forms for the AB register:

a) "36"

b) "18 - 18"

c) "27-9"

d) "9-9-9-9"



Note that all of B is part of the subword. There is only one sign bit in an AB subword.

NORMALIZE ACCUMULATOR NOA 64

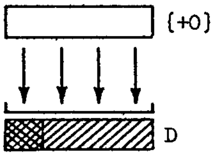
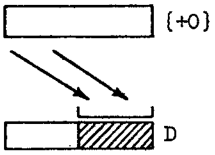
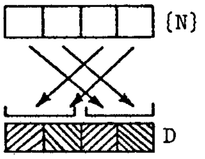
NORMALIZE AB (Extended Accumulator) NAB 66

$^{\alpha}$NOA T_j	$^{\alpha}[A] \times 2^{nz}$ ==> A $^{\alpha}[T_j]$ - nz ==> Sq(D)

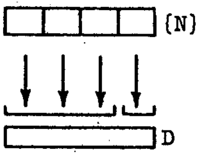
NORMALIZE scales just enough to remove leading zeros or to "recover" from OVERFLOW. It clears the active overflow indicators. The number of leading zeros (nz) is subtracted from the argument from T_j ($^{\alpha}[T_j]$) and this difference is left in the Sign Quarter of D. If an overflow condition exists at the start, "nz" is -1, the scale is one place to the right, and the sign is complemented - just as for SCA or SAB. If nz is zero, it is +0. (See Note 4 also.)

NOA and NAB start with an $^{\alpha}$LDD T_j. "nz" is subtracted from the sign quarter(s) and the rest of D is not changed. The E register becomes a copy of T_j.

EXAMPLES:†† (Assume that NO OVERFLOW exists.)

NO.	INSTRUCTION	DIAGRAM	ABBREVIATED DESCRIPTION	COMMENTS
1.	NOA{0}		$[A] \times 2^{nz}$ ==> A -nz ==> q4(D) +0 ==> q3,2,1(D)	"nz" is the number of leading "zeros" in the original contents of A. ("Zeros" can be positive zeros or negative zeros.)
2.	^{2}NOA{0}		$R[A] \times 2^{nz}$ ==> R(A) -nz ==> q2(D) +0 ==> q1(D)	The left halves of A and D are not changed. "nz" is the number of "zero" in the original contents of the right half of A. Note that the result in D is a <u>nine bit</u> numeral.
3.	^{17}NOA{N} N = a,b,,c,d $[F_{17}]$ = 202		$R[A] \times 2^{ZR}$ ==> R(A) a-ZR ==> q2(D) b ==> q1(D) $L[A] \times 2^{ZL}$ ==> L(A) c-ZL ==> q4(D) d ==> q3(D)	"ZR" and "ZL" are the leading zeros of the right and left 18 bit words of A. {N} is a register containing a,b,c, and d in quarters 4,3,2, and 1.

†† Brackets{} are used in the TX-2 M4 Assembly Program to indicate "Register Containing". See Chapter 6, page 6-10.

4.	$\alpha_{NOA}\{N\}$ N = a,b,,c,d α = 400		q432[A] x 2^{nz} ==> q432(A) a-nz ==> q4(D) b ==> q3(D) c ==> q2(D) q1[A] x 2^{nz} ==> q1(A) d-nz ==> q1(D)	With a 27,9 split, both counts will be 26 if [A] is zero. (See note on page 3-61 .)

5 - A sample program → Evaluate V = xyz

This product could have 105 significant bits (3 word lengths). One must resort to programmed arithmetic to get them all, but normalize can be used to get the 34 most significant bits. Consider the programs below.

Without Normalize:

```
LDA X
MUL Y
MUL Z
```

This program puts the 35 left bits of the 105 bit product in A and essentially worthless numerals in B. The answer in A may be too small by 1 (in the 35th place).

With Normalize:

```
LDA X
MUL Y
NAB {0}
STD T
MUL Z
SAB T
```

With normalize, the product is given in AB, to 35+nz places from the sign. (It may low by 1 in the (35+nz)th place.) "nz", the number of zeros, is in T (in negative form). nz could be as much as 69 so the last SAB may not be desired. For example, if the NAB instruction above were replaced with NAB{34.,} the answer in AB can be considered a 71 bit integer.

NOTE: 1. NOA and NAB leave E set the same as the memory register used.

2. If overflow exists, "nz" is -1 so [T_J]+1 ==> Sq(D).

3. NAB is essentially the same instruction - using the double length word (AB) instead. (See page 3-39 - "Subword forms for the AB register".)

4. Normalize is an arithmetic instruction. The sign bit is not counted. "Leading zeros" will, of course, be plus or minus zeros - i.e., the same as the sign.

CYCLE

CYA, 60
CYB, 61
CAB, 62

$\boxed{{}^{\alpha}\text{CYA } T_j}$

CYCLE logically falls in a class with LDA and STA, for it is most easily considered as a bit shifting instruction and the sign bit has no special significance. Bits shifted off one end are inserted at the other. None are lost. However, since the practical details of its use are so similar to SCALE, it is usually grouped with SCALE and NORMALIZE. The use of the memory word is the same as SCALE.

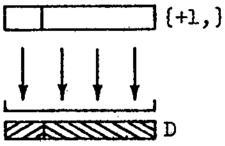
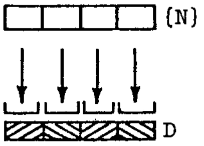
a.) An ${}^{\alpha}$LDD T_j is the first step.

b.) Each active subword is "cycled" or "rotated" according to its Sign Quarter in D and the sign quarter is left at -0. For cycle, the active subword has its ends connected - and can be considered as a ring of bits. If the number of places equals the subword length, the instruction does not change the subword. You can therefore arrive at any new position by cycling either way - the short way takes less computer time. The sign bit is handled no differently than the others and no bits are lost.

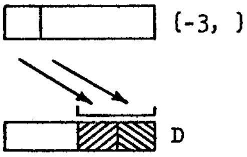
c.) Overflow is ignored.

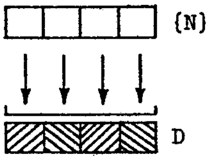
d.) The E register becomes a copy of the memory register used.

EXAMPLES: Assume [A] = 123 456 765 $432_{(8)}$ at the start

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRITPION	COMMENT
1.	CYA{+1,}		247 135 753 064 ==> A -0 ==> q4(D) +0 ==> q3,2,1(D)	One 36 bit ring cycled once to the left.
2.	^{30}CYA{N} N=1,1,,1,1		246 ==> q4(A) 135 ==> q3(A) 753 ==> q2(A) 065 ==> q1(A)	The four quarters are cycled separately i.e. four nine-bit rings, each one bit to the left.

Assume [A] = 123 456 765 $432_{(8)}$ at the start.

3. ^{2}CYA{-3, }		276 543 ==> R(A) -0 ==> R(D)	The left halves of A and D are not changed. The right half of A (a ring of 18 bits) is cycled 3 places to the right. i.e. one octal place.)

4. DPX B
CAB{+3,}
N=3,2,,5,-6
		234 567 654 320 ==> A
		000 000 000 001 ==> B
		-0 ==> q4(D)
		+2 ==> q3(D)
		+5 ==> q2(D)
		-6 ==> q1(D)
			The 72 bit ring -AB- is cycled 3 bits, i.e. one octal place to the left.

NOTES: 1. The E register becomes a copy of the memory word used.

2. CYA, CYB, CAB are indexable, and, of course, deferred addressing can also be used. (Neither of these is common. Most users use RC words.)

3. CAB uses the same word structure as SAB and NAB.

3-2.5 LOGIC, INSERT, COMPLEMENT/PERMUTE

ITA
ITE
UNA
DSA
INS
COM

BIT LOGIC INSTRUCTIONS

ITA 41
UNA 42
DSA 65
ITE 40

${}^{\alpha}$ITA T_j	${}^{\alpha}[T_j] \wedge [A] \Longrightarrow A$

For these instructions, the word is considered as a string of independent bits - each bit column is a separate entity. For ITA, UNA, and DSA, the argument ${}^{\alpha}[T_j]$, is all the active subwords - with sign extension if applicable. For these three, the E register is set, as usual, identical to the memory word used.

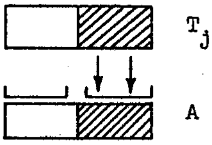
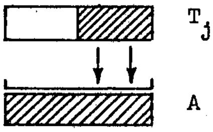
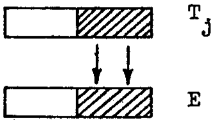
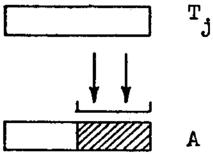
For ITE, the operand is the active quarters only. There is no sign extension. The result, of course, goes into E and there is no final E register copy from memory.

All these instructions are indexable and of course indirect addressing can be used.

Name	INTERSECT	UNITE	DISTINGUISH**
Abbreviation	ITA ITE	UNA	DSA**
Symbol	$\wedge$	$\vee$	ⓥ
Other Names	"AND"	Inclusive OR	Exclusive OR Partial Add
Logic Diagram	${}^{\alpha}[T_j]$ $\begin{array}{cc\|cc} & & 0 & 1 \\ \hline {}^{\alpha}[A] & 0 & 0 & 0 \\ & 1 & 0 & 1 \end{array}$ Note that this is the "carry" that results from addition.	${}^{\alpha}[T_j]$ $\begin{array}{cc\|cc} & & 0 & 1 \\ \hline {}^{\alpha}[A] & 0 & 0 & 1 \\ & 1 & 1 & 1 \end{array}$	${}^{\alpha}[T_j]$ $\begin{array}{cc\|cc} & & 0 & 1 \\ \hline {}^{\alpha}[A] & 0 & 0 & 1 \\ & 1 & 1 & 0 \end{array}$ Note that this is the Partial Sum.

	(ITA)	(UNA)	(DSA)
Typical Use	Masking - e.g. if T_j contains 77 ITA T; clears all of A except for the last 6 bits.	Bit Setting, or clearing to minus zero - if T_j contains 77, UNA T_j sets the last 6 bits to 1 without changing the rest.	Bit Complementing - if T_j contains 77 DSA T_j complements the last 6 bits.
Special Example F_{30}= 600 All quarters are active and independent.	^{30}SAB {-9,-9,,-9,-9} ITA B If positive, A is cleared to +0. The original [A] goes into B.	^{30}SAB {-9,-9,,-9,-9} UNA B If Negative, A is set to -0. The original [A] goes into B.	^{30}SAB{-9,-9,,-9-9} DSA B The absolute value or magnitude or each quarter goes into A The original [A] goes into B.

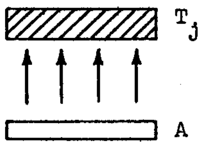
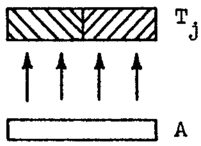
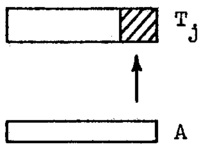
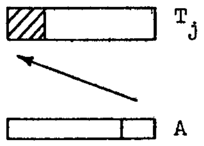
** Note: DSA affects both the C and D registers. The effect on D is equivalent to LDD T_j. The effect on C is equivalent to forming the carries and uniting them with the original contents of C. - i.e. $([A] \wedge [T_j]) \vee [C] ==> C$.

No.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENT
1	^{1}UNA T_j		$R\ [T_j] \vee R\ [A] \ => \ R\ (A)$ $[T_j] \ => \ E$	T_j is unaffected. The left half of A is also unchanged.
2	^{11}ITA T_j		$R\ [T_j] \wedge R\ [A] \ => \ R\ (A)$ $SR\ [T_j] \wedge L\ [A] \ => \ L\ (A)$ $[T_j] \ => \ E$	T_j is unaffected. Each bit of left half of A is "intersected" with bit 2.9 of T_j - Hence, if R $[T_j]$ is positive, $L(A)^j$ is cleared.
3	^{11}ITE T_j $[F_{11}]$ = 140		$R\ [T_j] \wedge R\ [A] \ => \ R\ (E)$	T_j is unaffected. L(E) is unaffected. There is no sign extension on ITE.
4	^{1}DSA T_j		$R\ [T_j] \circledvee R\ [A] \ => \ R\ (A)$ $R\ [T_j] \ => \ R\ (D)$ $[T_j] \ => \ E$ $(R[T_j] \wedge A]) \vee R[C] => R(C)$	DSA affects registers A, C, D, and E. See note above.

$^{\alpha}$INS T_j	$([A]\wedge[B])\vee([\bar{B}]\wedge[T_j]) \Rightarrow T_j$ †

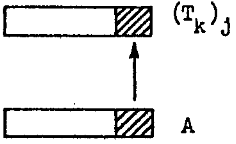
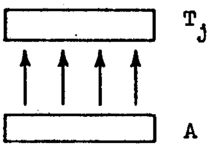
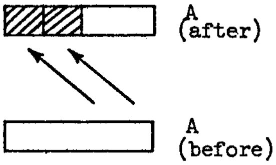
Insert is a partial STA (store accumulator) instruction — only those bits marked by a 1 in the corresponding column of B are stored in T_j. There is no sign extension, and [A] is not changed. If [B] is minus zero (all ones), INS is identical to STA. The E register is set to the final contents of the memory word used.

EXAMPLES: (Standard F Memory - Chart 7-2)

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	MASK (CONTENTS OF B)	COMMENTS**
1.	INS T_j		-0	$[A] \Rightarrow T_j$. INS is identical to STA when $[B] = -0$.
2.	INS T_j		0,,777777	$R[A] \Rightarrow T_j$. This time it looks like a ^{1}STA T_j, because of the mask.
3.	^{3}INS T_j		4,2,,3,1	Bit 1.1 of A is copied into position 1.1 of T_j. Quarters 2,3, and 4 are inactive. No other bits are changed. ^{13}INS T_j would do the same. $[F_{13}] = 160$
4.	^{6}INS T_j		4,2,,3,1	Bit 1.1 of A is copied into position 4.1 of T_j. Note that permutation has no effect on the use of B. ^{16}INS T_j is identical.

**In all cases, there is a final copy into E from the memory register used.

† "Insert" is also given by $([A] \vee [\bar{B}]) \wedge ([B] \vee [T_j])$.

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	MASK (CONTENTS OF B)	COMMENTS**
5.	^{3}INS$(T_k)_j$*		4,5,,6,7	q1[A] => q1$(T_k)_j$. ^{3}STA ••• would be equivalent.
6.	INS T_j		+0	Since [B] = +0, nothing happens.
7.	^{2}INS A		4,5,,0,7	q1[A] => q3(A). Only quarter 3 of A is changed. (Because of the mask.)

$^{\alpha}$COM T_j	$^{\alpha}\overline{[T_j]}$ => T_j T_j is permuted.

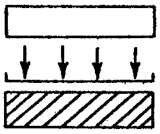
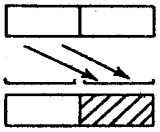
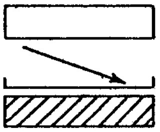
COM - Complement - performs two basic operations. The active subwords of T_j are complemented (one's complement - all ones become zeros and vice versa) (with sign extension) and all quarters are permuted whether active or not. Note that if all quarters are inactive, COM permutes all quarters of T_j without changing the data. PMT is another abbreviation - equivalent to COM.

There are 4 basic steps:

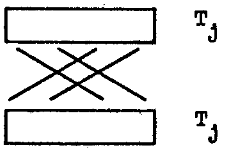
1. $[T_j]$ => E , permuted according to α.
2. Sign extension occurs in active subwords.
3. Active subwords are complemented. ($^{\alpha}\overline{[E]}$ => $^{\alpha}$E)
4. [E] => T_j straight - no permutation.

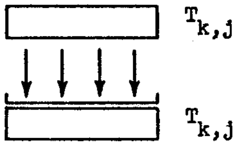
Note that, as usual, E is the same as T_j at the end.

EXAMPLES: (Standard F Memory - Chart 7-2)

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENTS
1	COM T_j	 T_j (before) T_j (after)	$\overline{[T_j]}$ => T_j $\overline{[T_j]}$ => E	All of T_j is complemented
2	^{2}COM T_j	 T_j (before) T_j (after)	$\overline{L[T_j]}$ => R(T_j) R[T_j] => L(T_j)	The halves are reversed and the right half is complemented.
3	^{16}COM T_j [F_{16}] = 163	 T_j (before) T_j (after)	$\overline{q4[T_j]}$ => q1(T_j) $\overline{Sq4[T_j]}$ => q2,3,4(Tj)	Quarters 2, 3, and 4 are set to the complemented sign extension.

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENTS

4	$^{\alpha}$COM T_j
	α = 172
	(all inactive)
			R[T_j] => L (T_j)
			L[T_j] => R (T_j)
			(Simultaneously)
				When all quarters are inactive, the data is not changed - it is merely permuted according to the given configuration.

5	COM $(T_k)^*_j$		$\overline{[T_{k,j}]}$ => $T_{k,j}$ $\overline{[T_{k,j}]}$ => E	This has double indexing. $T_{k,j}$ = T + $[X_k] + [X_j]$

Note: Since COM does not use any register other than T_j, there may be some confusion as to the meaning of "Activity". In this chapter, quarters for which arrows are drawn are active. To be consistent with other instructions, one should say that the permutation comes first, complementing second, and sign extension last. If you use the phrase "Active Subwords of T_j", the order of the first two is immaterial since both operations can be considered to take place simultaneously. In any event, sign extension uses the complemented sign.

3-2.6 CONFIGURATION MEMORY CLASS

SPF
SPG
FLF
FLG

SPECIFY FORM (SPF) SPF (21)
SPECIFY GROUP (SPG) SPG (22)

^{c}SPF T_j	q 1 [T_j] => F_c
^{c}SPG T_j	q 1 [T_j] => F_c q 2 [T_j] => F_{c+1} q 3 [T_j] => F_{c+2} q 4 [T_j] => F_{c+3}

"Specify" copies from STUV memory into F Memory. (STUV memory is not changed.) SPF sets only one F Memory word. SPG sets four. F Memory addresses are consecutive modulo 37_8 - i.e., 0, 1, 2, ..., 36_8, 37_8,0, 1, 2, etc. These instructions are indexable but not configurable. The E register is set, as usual, to the contents of the memory register used.

EXAMPLES:

NO.	INSTRUCTION	DESCRIPTION	COMMENT
1	^{o}SPF T_j	--	F_o is permanently set to +0 and can not be changed.
2	^{o}SPG T_j	q 2[T_j] => F_1 q 3[T_j] => F_2 q 4[T_j] => F_3	Same as #1.
3	^{37}SPG T_j	q 1[T_j] => F_{37} q 3[T_j] => F_1 q 4[T_j] => F_2	F_o is, of course, not changed. The F Memory address "c" is normally given in OCTAL

FILE FORM FLF 31

FILE GROUP FLG 32

^{c}FLF T_j	$[F_c] \Rightarrow q1(T_j)$
^{c}FLG T_j	$[F_c] \Rightarrow q1(T_j)$ $[F_{c+1}] \Rightarrow q2(T_j)$ $[F_{c+2}] \Rightarrow q3(T_j)$ $[F_{c+3}] \Rightarrow q4(T_j)$

"File" copies from F Memory into STUV Memory. (F Memory is not changed.) File Form (FLF) copies a single 9 bit word, File Group copies four. They are indexable, but not configurable. The F Memory Addressing is modulo 37_8– i.e. "c" = 0, 1, 2, ... 36_8, 37_8, 0, 1, 2, ... etc. The E register is set as usual, to the contents of the memory word used.

EXAMPLES:

NO.	INSTRUCTION	DESCRIPTION	COMMENT
1.	^{o}FLF T_j	$+0 \Rightarrow q1(T_j)$	F_o is permanently set to +0.
2.	^{o}FLG T_j	$0 \Rightarrow q1(T_j)$ $[F_1] \Rightarrow q2(T_j)$ $[F_2] \Rightarrow q3(T_j)$ $[F_3] \Rightarrow q4(T_j)$	– – –
3.	^{36}FLG T_j	$[F_{36}] \Rightarrow q1(T_j)$ $[F_{37}] \Rightarrow q2(T_j)$ $+0 \Rightarrow q3(T_j)$ $[F_1] \Rightarrow q4(T_j)$	The F Memory address "c" is normally given in octal.

3-2.7 ARITHMETIC CLASS

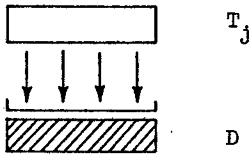
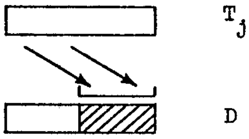
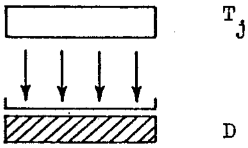
ADD
SUB
MUL
DIV
TLY (TALLY)

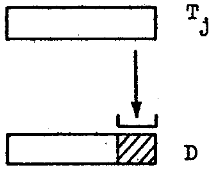
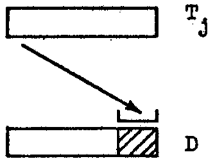
ADD (67)
SUBTRACT (77)

$^{\alpha}$ADD T_j	$^{\alpha}[A] + {}^{\alpha}[T_j] \Rightarrow {}^{\alpha}A$
$^{\alpha}$SUB T_j	$^{\alpha}[A] - {}^{\alpha}[T_j] \Rightarrow {}^{\alpha}A$

ADD and SUBTRACT are straightforward one's complement (RINGED) arithmetic instructions. The use of configuration is similar to LDA. A zero result is negative except when both arguments are zero at the start -(+0) + (+0) = +0; +0 -(-0) = +0. There are four overflow indicators--a separate indicator for each active subword. The indicator is cleared before the arithmetic is done and is set to a one for <u>either</u> type of overflow--(too negative or too positive). (With one's complement arithmetic there is a sign reversal when overflow occurs. The scale instructions take this into account.) Sign extension occurs prior to the arithmetic. The D register is set as if an $^{\alpha}$LDD T_j were done. The C register is set to the carries from each column. (In the case of subtract, "c" contains the carries from adding the complement of $[T_j]$.) The B register is unaffected. The E register is set, as usual, to the contents of the memory word used.

EXAMPLES: (Standard F Memory - Chart 7-2)

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENTS
1	ADD T_j		$[A] + [T_j] \Rightarrow A$ $[A] \wedge [T_j] \Rightarrow C$ $[T_j] \Rightarrow D$ $[T_j] \Rightarrow E$	The expression $[A] \wedge [T_j] \Rightarrow C$ is equivalent to saying the "carries" of each bit column go into the corresponding bit column of C. Z_4 is set if overflow occurs.
2	^{2}ADD T_j		$R[A] + L[T_j] \Rightarrow R(A)$ $R[A] \wedge L[T_j] \Rightarrow R(C)$ $L[T_j] \Rightarrow R(D)$ $[T_j] \Rightarrow E$	The left half of the A, C, and D registers is unchanged. Z_2 is set if overflow occurs.
3	SUB T_j		$[A] - [T_j] \Rightarrow A$ $[A] \wedge \overline{[T_j]} \Rightarrow C$ $[T_j] \Rightarrow D$ $[T_j] \Rightarrow E$	Z_4 is set if overflow occurs.

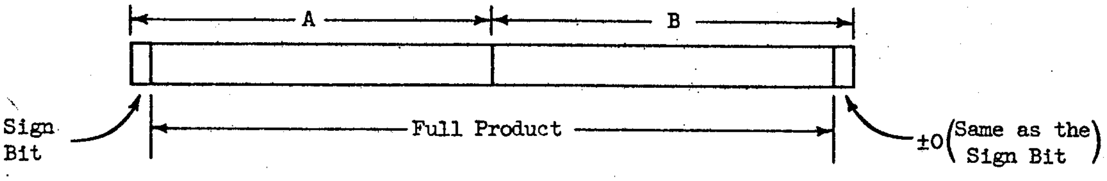
NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENTS
4.	^{3}LDA {277} ^{3}ADD {307}		606 => ql(A) 1 => Z_1 207 => ql(C) 307 => ql(D) 307 => E	{277} is the M4 representation for "a register containing $277_{(8)}$".
5.	^{6}LDA {510,0} ^{6}ADD {470,0}		201 => ql(A) 1 => Z_1 410 => ql(C) 470 => ql(D) 470,0 => E	{510,0} is the M4 representation for "A register containing $510_{(8)}$ in quarter 4, and zero in the rest of the word." See Chapter 6.

Note: The four OVERFLOW indicators are associated with the subwords by Sign Quarter Number. See table below:

SUBWORD	OVERFLOW INDICATOR
Quarter 4	Z_4
Quarter 3	Z_3
Quarter 2	Z_2
Quarter 1	Z_1
Left Half	Z_4
Right Half	Z_2
Full Word	Z_4
27 - 9	Z_4 and Z_1

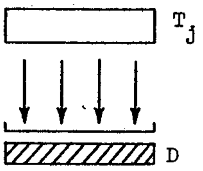
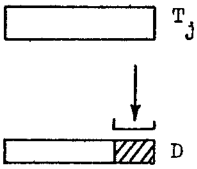
$^{\alpha}$MUL T_j	$^{\alpha}[A] \times {}^{\alpha}[T_j] \Rightarrow {}^{\alpha}(AB)$

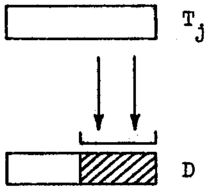
"MUL" forms the double-length, ones-complement product of [A] and $[T_j]$ and stores it in A and B. The extra bit of B -- at the extreme right -- is set equal to the sign bit of the product, i.e., to ± 0. (Bit 1.1 of B = Bit 4.9 of A after MUL.)

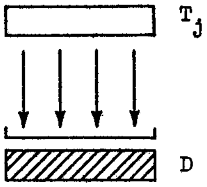


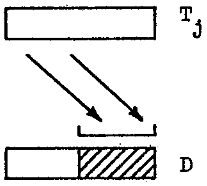
The use of configuration is similar to LDA and the relevant overflow indicator (corresponding to the active sign quarter) is cleared. No overflow can be generated. The active subwords of C are cleared to +0 and D is set as if an $^{\alpha}$LDD T_j had been done. The E register is, as usual, set to the contents of the memory word used.

EXAMPLES: (Standard F Memory - Chart 7-2.)

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENTS
1	MUL T_j		$[A] \times [T_j] \Rightarrow AB$ $\pm 0 \Rightarrow$ bit 1·1(B) $+0 \Rightarrow Z_4$ $+0 \Rightarrow C$ $[T_j] \Rightarrow D$ $[T_j] \Rightarrow E$	"AB" is the double length register diagrammed above. It is also used with SAB, CAB, and DIV. Bit 1·1 of B is set to ± 0 -- depending on the sign of the product.
2	^{3}LDA (5) ^{3}MUL (4)		000 => q 1 (A) 050 => q 1 (B) 000 => q 1 (C) 004 => q 1 (D) 0 => Z_1	With standard configuration 3, q1[AB] is an 18-bit register composed of quarter 1 of A and quarter 1 of B. The other quarters are not changed

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENTS
3.	^{1}LDA {-3} ^{1}MUL {-4}		+ 0 ==> R(A) 000030 ==> R(B) + 0 ==> R(C) - 4 ==> R(D) - 4 ==> E 0 ==> Z_2	The left half words are not changed.

4.	LDA { 3}
	MUL {-400}
			- 0 ==> A
			- 3000 ==> B
			+ 0 ==> C
			- 400 ==> D
			- 400 ==> E
			0 ==> Z_4	

5.	^{2}LDA {3 ,, 0} ^{2}MUL {4 ,, 0}		+ 0 ==> R(A) 000030 ==> R(B) + 0 ==> R(C) + 4 ==> R(D) (+4,,0) ==> E 0 ==> Z_2	Only the <u>right</u> half words are changed.

Note: When a 27-9 subword form is used, the Arithmetic Step Counter is set for the 27-bit word, if it is active. This results in too many steps for the 9-bit word if it is active also. (This is true for MUL, DIV, NOA, NAB, and TLY.) Normal use of this subword form is for floating numbers of the form $N = x \cdot 2^y$ (27 bits for "x," 9 for "y"). Since different operations are performed on the two syllables, both subwords will not be active at the same time.

$^{\alpha}\text{DIV}\ T_j$	$^{\alpha}[AB] \div {}^{\alpha}[T_j]$ ==> A
	Remainder ==> B

DIVIDE considers the contents of AB (except for the lowest order bit of B) as the numerator and the contents of T_j as the denominator. (Note that it is compatible with MUL.) Configuration is similar to ADD, LDA, etc. The Quotient is stored in A with the appropriate algebraic sign. The remainder is stored in B with the same sign as the original numerator. (The sign of the remainder is at the left, as usual.) (SAB (+n) will bring strange bits into A for the remainder (in B) is not an extension of the quotient.)

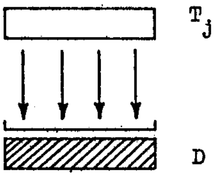
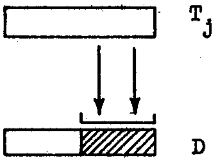
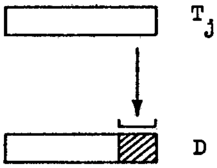
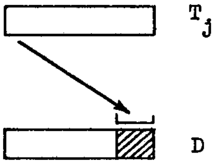
$$\frac{[AB]}{[T_j]} \equiv Q + \frac{R}{[T_j]} \qquad \begin{array}{l} Q \Longrightarrow A \\ R \Longrightarrow B \end{array}$$

The relevant overflow indicator is cleared at the outset and an overflow will be generated if $|\,[A]\,|$ exceeds or equals $|\,[T_j]\,|$.

Note:

1. If $|[A]| < 2 \cdot |[T_j]|$ overflow, if any, is guaranteed recoverable via SCA (-n) . SAB (-n) will also recover the correct answer, but it will destroy the remainder.
2. If both [AB] and $[T_j]$ are normalized (as per NAB and NOA), the condition above is met, and any overflow is recoverable.
3. On overflow, the sign of A is always the reverse of the proper algebraic sign.
4. If overflow is not recoverable, both [A] and [B] are useless.
5. $\frac{N}{+0} = \bar{N}$, and Overflow is set. (This is true for any N.)
6. $\frac{N}{-0} = N$, and Overflow is set. (Also true for any N.)
7. Divide clears C (as if by $^{\alpha}$LDC (0)) and sets D (as if by $^{\alpha}$LDD T_j).
8. The contents of the memory register go into E, as usual.
9. See also note on page 3-61.

EXAMPLES: (Standard F Memory - Chart 7-2.)

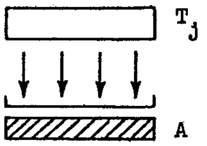
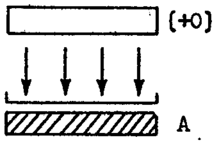
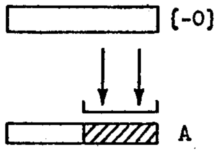
NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENTS
1.	DIV T_j		$[AB] \div [T_j]$ => A Remainder => B + 0 => C $[T_j]$ => D $[T_j]$ => E	Overflow, if any, sets Z_4.
2.	^{1}DIV T_j		$R[AB] \div R[T_j]$ => R(A) Remainder => R(B) + 0 => R(C) $R[T_j]$ => R(D) $[T_j]$ => E	Overflow sets Z_2. The left half of the arithmetic unit is unchanged.
3.	^{3}LDA (000) ^{3}LDB (052) ^{3}DIV (5)		004 => ql(A) 001 => ql(B) 000 => ql(C) 005 => ql(D) 005 => E 0 => Z_1	The numerator is actually half of 000052 since the lowest order bit of B is not part of it. In decimal, we have 21 ÷ 5 or 4 with a remainder of +1.
4.	^{6} LDA (-0,) ^{6} LDB (725,) ^{6} DIV (-5,)		+4 => ql(A) -1 => ql(B) + 0 => ql(C) -5 => ql(D) (-5,) => E 0 => Z_1	Note that [A] is <u>minus</u> zero. The numerator is therefore -21 (decimal). If [A] were +0, the numerator would be $\frac{+725}{2}(8)$ or 234 (decimal).

$^{\alpha}$TLY T_j	$^{\alpha}[T_j]$ ==> A count of ones + $[S_qD]$ ==> SqD

TLY (TALLY) loads A (as does LDA). Then the count of ones is added to the sign quarter of D. The rest of D is not affected. The sign digit is counted also if it is a "one".

The E register is set, as usual, to $[T_j]$.

EXAMPLES:

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENTS
1.	TLY T_j		$[T_j]$ => A n+q4[D] => q4D $[T_j]$ => E	"n" is the number of ones in $[T_j]$. The addition is regular 9-bit ring addition with no overflow detection.
2.	TLY (+0)		+ 0 => A + 0 => E	The D register is not changed
3.	^{1}TLY (-0)		- 0 => R(A) 18+q2[D] => q2D - 0 => E	The left half of A is not changed. Only the sign quarter (No. 2) of D is affected.

Note: When a 27-9 subword form is used, the Arithmetic Step Counter is set for the 27-bit word, if it is active. This results in too many steps for the 9-bit word if it is active also. (This is true for MUL, DIV, NOA, NAB, and TLY.) Normal use of this subword form is for floating numbers of the form $N = x \cdot 2^y$. (27 bits for "x," 9 for "y"). Since different operations are performed on the two syllables, both subwords will not be active at the same time.

3-3 OPERATION CODE CHART (Wesley A. Clark).

3-3.1 Number Systems

Let S be a binary number of length λ

3 number ranges are commonly used:

1) <u>Positive Integers</u> (e.g., r, P, Q)

$$0 \leq S \leq 2^{\lambda} - 1$$

2) <u>Signed Integers</u> (e.g., X_j)

$$-(2^{\lambda-1} - 1) \leq S \leq +(2^{\lambda-1} - 1)$$

3) <u>Signed Fractions</u> (e.g., A in <u>MUL</u>, <u>DIV</u>)

$$-(1 - 2^{-(\lambda-1)}) \leq S \leq +(1 - 2^{-(\lambda-1)})$$

<u>Negative number</u> represented by "<u>Ones Complement</u>" of corresponding positive number. $S \begin{matrix}1 \to 0 \\ 0 \to 1\end{matrix} \bar{S}$ (complement of S).

Two representations of number zero $\left.\begin{matrix}0 = 00 \ldots 0 \\ \bar{0} = 11 \ldots 1\end{matrix}\right\}$ λ bits in length

<u>Reduction Modulo</u> μ

For positive integer S $0 \leq S < 2\mu$

$$S \bmod \mu = \begin{cases} S & \text{if } S < \mu \\ S - \mu & \text{if } S \geq \mu \end{cases}$$

Example: 6 mod 7 = 6, 8 mod 7 = 1

3-3.2 Glossary of Terms

h	Hold bit
c	Configuration
i	Instruction
j	Index
r	effective address
W_r	memory operand
$W_r{}^*$	Permuted Memory Operand
W_{rj}	Memory operand (indexed)
$W_{rj}{}^*$	Permuted Indexed Memory Operand
$r, r \oplus X_j$	Operand addresses
D'	Leftmost (sign) quarter of D
$(W_{rj}{}^*)'$	Leftmost (sign) quarter of permuted indexed memory operand
G_c	Group c

			EXAMPLE 1	EXAMPLE 2
S, T	λ-bit binary numbers	S	010 011 101	111 011 010
$\bar{S}$	Complement of S (sign bit complemented)	$\bar{S}$	101 100 010	000 100 101
< S >	Inversion of S	< S >	110 011 101	011 011 010
RS	Positive (counterclockwise; left) unit rotation of S	RS	100 111 010	110 110 101
$R^{-1}S$	Negative (clockwise; right) unit rotation of S	$R^{-1}S$	101 001 110	011 101 101
2 x S	Unit positive scaling of S (S scaled up by one)	2 x S	000 111 010	110 110 101
2^{-1}x S	Unit negative scaling of S (S scaled down by one) (scaling is rotation without change of sign bit)	2^{-1} x S	001 001 110	111 101 101
n(S)	Normalizer of S (S signed fraction) $\frac{1}{2} \leq \mid 2^{n(S)} \times S \mid < 1$ Note: $n(0) = n(\bar{0}) = \lambda -1$. (Used as 9-bit number.)	n(S)	0	2
τ(S)	Tally of S (number of ones in S) (used as 9-bit number.)	τ(S)	5	6
		T	011 010 011	011 010 011
S ∧ T	S and T } for each bit b, b=1, 2, ... , λ	S ∧ T	010 010 001	011 010 010
S ∨ T	S or T	S ∨ T	011 011 111	111 011 011
S ⊗ T	S or T but not both	S ⊗ T	001 001 110	100 001 001
S ⊕ T	λ-bit binary ring sum of S and T Note: $S \oplus \bar{S} \equiv \bar{0}$	⊕	101 110 000	010 101 110
S ⊖ T	λ-bit binary ring difference = $(\overline{\bar{S} \oplus T})$	S ⊖ T	111 001 001	100 000 111

Enclosed expression applies to <u>each active quarter of operand</u> ◹

Enclosed expression applies to <u>each active subword of operand</u> ◥

A blank box indicates that no change is made.

INSTRUCTION EXECUTION TABLE. Instruction = $\{h, \omega, i, j, \lambda\}$ (ENTRIES ARE FINAL VALUES IN TERMS OF INITIAL VALUES) (BLANKS INDICATE NO CHANGE)

TIME (OVER BASIC MEM CYCLE)	i	ABBR.	NAME	CONDITION	P ①	Q	X_j	W_λ		$W_{\lambda j} = W_{\lambda \oplus X_j}$		Z(A)	A	B	C	D		E
								W_λ	W_λ^*	$W_{\lambda j}$	$W_{\lambda j}^*$					D'	D	
0	21	SPF	SPECIFY FORM ②		P+1	$\lambda \oplus X_j$												④
	22	SPG	SPECIFY GROUP ②															
1.2	31	FLF	FILE FORM							F_λ ②								
2.8	32	FLG	FILE GROUP							G_λ								
0	24	LDA	LOAD A		P+1	$\lambda \oplus X_j$							$W_{\lambda j}^*$					④
	25	LDB	LOAD B											$W_{\lambda j}^*$				
	26	LDC	LOAD C												$W_{\lambda j}^*$			
	27	LDD	LOAD D														$W_{\lambda j}^*$	
1.2	34	STA	STORE A								A							
	35	STB	STORE B								B							
	36	STC	STORE C								C							
	37	STD	STORE D								D							
	54	EXA	EXCHANGE A	③							A		$W_{\lambda j}^*$					
2.4	55	INS	INSERT								$(B \wedge A)$ $\vee (\bar{B} \wedge W_{\lambda j}^*)$							
0	41	ITA	INTERSECT A										$A \wedge W_{\lambda j}^*$					
	42	UNA	UNITE A										$A \vee W_{\lambda j}^*$					
	65	DSA	DISTINGUISH A										$A \circledast W_{\lambda j}^*$		$(A \wedge W_{\lambda j}^*) \vee C$		$W_{\lambda j}^*$	
	67	ADD	ADD									⑤	$A \oplus W_{\lambda j}^*$		$A \wedge W_{\lambda j}^*$			
	77	SUB	SUBTRACT										$A \ominus W_{\lambda j}^*$		$A \wedge \bar{W}_{\lambda j}^*$			
15 11 8 5	76	MUL	MULTIPLY ⑦									o	$A = W_{\lambda j}^*$		0			
73 55 37 16	75	DIV	DIVIDE ⑦	⑧								⑥	$(AB/W_{\lambda j}^*)_{quot.}$	$(AB/W_{\lambda j}^*)_{rem.}$				
$(W_{\lambda j}^*)'/3$	60	CYA	CYCLE A										$\mathcal{R}^{(W_{\lambda j}^*)'} \cdot A$			$\bar{0}$	⑧ $W_{\lambda j}^*$	
	62	CAB	CYCLE AB										$\mathcal{R}^{(W_{\lambda j}^*)'} \cdot AB$					
$(W_{\lambda j}^*)' < 9 \Rightarrow 0$	61	CYB	CYCLE B											$\mathcal{R}^{(W_{\lambda j}^*)'} \cdot B$				
$n/3$	70	SCA	SCALE A	Z(A)=0								0	$2^{(W_{\lambda j}^*)'} \cdot A$					
				Z(A)=1 $(W_{\lambda j}^*)'>0$									$2^{(W_{\lambda j}^*)'} \cdot (A)$					
				Z(A)=1 $(W_{\lambda j}^*)'<0$									$2^{(W_{\lambda j}^*)'+1} \cdot (2^{-1} \cdot A)$					
$n<9 \Rightarrow 0$	72	SAB	SCALE AB	Z(A)=0									$2^{(W_{\lambda j}^*)'} \cdot AB$					
				Z(A)=1 $(W_{\lambda j}^*)'>0$									$2^{(W_{\lambda j}^*)'} \cdot (AB)$					
				Z(A)=1 $(W_{\lambda j}^*)'<0$									$2^{(W_{\lambda j}^*)'+1} \cdot (2^{-1} \cdot AB)$					
	71	SCB	SCALE B											$2^{(W_{\lambda j}^*)'} \cdot B$				
$n/3$	64	NOA	NORMALIZE A	Z(A)=0								0	$2^{m(A)} \cdot A$			$(W_{\lambda j}^*)' \ominus m(A)$		
			⑦	Z(A)=1									$(2^{-1} \cdot A)$			$(W_{\lambda j}^*)' \oplus 1$		
	66	NAB	NORMALIZE AB	Z(A)=0									$2^{m(AB)} \cdot AB$			$(W_{\lambda j}^*)' \ominus m(AB)$		
			⑦	Z(A)=1									$(2^{-1} \cdot AB)$			$(W_{\lambda j}^*)' \oplus 1$		
12 9 6 2	79	TLY	TALLY ⑦										$W_{\lambda j}^*$			$D' \oplus c(W_{\lambda j}^*)$		
3.6	12	SKX (REX)	SKIP ON INDEX	$c<4$	P+1		0 λ 1 $-\lambda$ 2 $X_j \oplus \lambda$ 3 $X_j \ominus \lambda$ 4-7 ⑰											④
					P+2													
2.0	56	PMT	PERMUTE	$\alpha_\lambda = 17$	P+1	$\lambda \oplus X_j$				$W_{\lambda j}^*$								④
		COM	COMPLEMENT	⑨					⑨ –	$\bar{W}_{\lambda j}^*$								
0	20	LDE	LOAD E															$W_{\lambda j}^*$
1.2	30	STE	STORE E								E							
0	40	ITE	INTERSECT E															$E \wedge W_{\lambda j}^*$
	43	SED	SKIP IF E DIFFERS	ALL $E = W_{\lambda j}^*$														
				ANY $E \neq W_{\lambda j}^*$	P+2													
2.0	17	SKM	SKIP-MARK	[illegible]	[illegible]	λ		[illegible]										[illegible]
-			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0	11	RSX	RESET X		P+1	λ	(W_λ^*)											④
1.2	16	DPX	DEPOSIT X						$S_0(X_j)X_j$									
1.2	14	EXX	EXCHANGE X				(W_λ^*)		$S_0(X_j)X_j$									
.8	10	AUX	AUGMENT X				$X_j \oplus (W_\lambda^*)_1$ ⑫		⑪									
3.2	15	ADX	ADD X						$X_j \oplus (W_\lambda^*)_0$ ⑫									
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3.2	06	JPX	JUMP ON POSITIVE X	$X_j<0$	P+1		⑬ $\pm \oplus X_j$											(E_{11}): (E_{21}):
				$X_j>0$	λ													P+1
	07	JNX	JUMP ON NEGATIVE X	$X_j<0$														
				$X_j>0$	P+1													
1.6	46	JPA	JUMP ON POSITIVE A	NO $A>0$	P+1													
				ANY $A>0$	$\lambda \oplus X_j$													P+1
	47	JNA	JUMP ON NEGATIVE A	ANY $A<0$														
				NO $A<0$	P													
	44	JOV	JUMP ON OVERFLOW	NO Z(A)=1														
				ANY Z(A)=1	$\lambda \oplus X_j$													P+1
1.2	05	JMP	JUMP	c even	λ		⑭											⑭ : ⑭
		BRC	BRANCH	c odd	$\lambda \oplus X_j$													
1.6	57	TSD	TRANSFER DATA	UNIT NOT READY	P													
				UNIT READY	P+1	$\lambda \oplus X_j$				⑮	⑮							④

3-3.3 Notes on the coding chart

1. In all expressions P + 1, P + 2, sums are reduced modulo 2^{18}. $(777777 + 1) \bmod 2^{18} = 0$.

2. For SPF and FLF only quarter one of W_{rj} is used. SPG and FLG use all four quarters. F memory addressing is counted modulo 37_8 (e.g., 36, 37, 0, 1 ...)

3. If $r \oplus X_j = 377604$ (address of A reg.) then EXA has same effect as STA.

4. Final value of W_Q ==> $(Q = r, r \oplus X_j)$.

5. ADD, SUB overflow conditions:

   If $A \oplus W = A + W$ Then 0 ==> Z(A)

   If $A \oplus W \neq A + W$ Then 1 ==> Z(A)

$$\begin{aligned} Z(A_{43}) \equiv Z(A_{42}) \equiv Z(A_{41}) \equiv Z(A_4) &= Z_4 \\ Z(A_3) &= Z_3 \\ Z(A_{21}) \equiv Z(A_2) &= Z_2 \\ Z(A_1) &= Z_1 \end{aligned}$$

6. DIV Conditions:

CONDITIONS		Z(A)	A	B
$\lvert W_{rj}^*\rvert \neq 0$	$\lvert W_{rj}^*\rvert > \lvert AB\rvert$	0	QUOT	REM
	$\lvert W_{rj}^*\rvert \leq \lvert AB\rvert$	1	JUNK	JUNK
$\lvert W_{rj}^*\rvert = 0$	$\lvert AB\rvert = 0$	1	$\bar{A}$	
	$\lvert AB\rvert \neq 0$		$\bar{A} \circledvee W_{rj}^*$	R^{-1} B

   Sign of normal remainder = sign of dividend (AB).

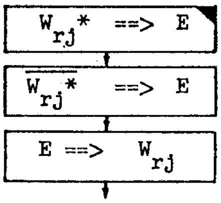
   JUNK is recoverable if $|A| < 2\,|W_{rj}^*|$.

7. Exceptions in MUL, DIV, NOA, NAB, TLY:

   Expressions listed are not correct for quarter (subword) 1 of A, B, and D' if a 27, 9 subword is chosen, and if quarter 1 is active.

8. CYCLE, SCALE, and NORMALIZE instructions begin, in effect, with LDD.

9. PMT, COM consist of 3 consecutive steps:



W_{rj}^* ==> E

$\overline{W_{rj}^*}$ ==> E

E ==> W_{rj}

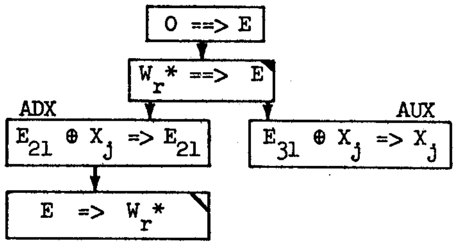
10. SKM variations:

j						$M_{r.q.b}$: selected bit $M_{r.q.10(dec)} = m_r$ $M_{r.q.11(dec)} = p_r$ $M_{r.q.12(dec)} =$ parity (M_r)
q mod 4		b				
3.6	3.5	3.4	3.3	3.2	3.1	
q = quarter; b = bit						

FUNCTION	CONDITIONS					$M_{r.q.b}$	ACTIONS (SKIP, Then MAKE, Then CYCLE)
	c						
	4.8	4.7	4.6	4.5	4.4		
	0	0	-	-	-	-	P + 1 ==> P
SKIP	0	1	-	-	-	-	P + 2 ==> P
SKIP on ZERO	1	0	-	-	-	0	P + 2 ==> P
						1	P + 1 ==> P
SKIP on ONE	1	1	-	-	-	0	P + 1 ==> P
						1	P + 2 ==> P
-	-	-	-	0	0	-	-
COMPLEMENT	-	-	-	0	1	-	$\overline{M}_{r.q.b}$ ==> $M_{r.q.b}$
MAKE ZERO	-	-	-	1	0	-	0 ==> $M_{r.q.b}$
MAKE ONE	-	-	-	1	1	-	1 ==> $M_{r.q.b}$
-	-	-	0	-	-	-	-
CYCLE	-	-	1	-	-	-	$R^{-1} W_r$ ==> W_r

11. $S_G(X_j)$ is 18-bit number 00 ... 0 or 11 ... 1 according as sign bit of X_j is 0 or 1.

12. ADX , AUX consist of sequence of steps:

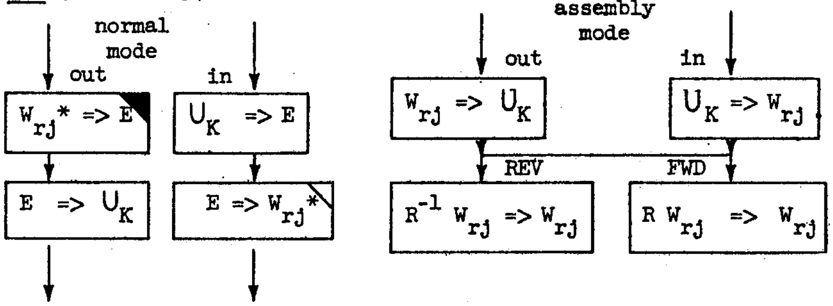


13. $\underline{c}$ is 18-bit signed integer expansion of c. (0 ≤ c ≤ 37 ; -17 ≤ $\underline{c}$ ≤ + 17)

14. JMP, BRC variations:

FUNCTION	c					ACTION
	d 4.8	4.7	4.6	4.5	4.4	
JUMP	-	-	-	-	0	$r \Longrightarrow P$
BRANCH	-	-	-	-	1	$r \oplus X_j \Longrightarrow P$
-	-	-	-	0	-	-
SAVE	-	-	-	1	-	$P + 1 \Longrightarrow X_j$
-	-	-	0	-	-	-
P + 1 => E	-	-	1	-	-	$P + 1 \Longrightarrow E_{21}$
-	-	0	-	-	-	-
Q ==> E	-	1	-	-	-	$Q \Longrightarrow E_{43}$
-	0	-	-	-	-	-
DISMISS	1	-	-	-	-	if $h = 0$, $0 \Rightarrow \phi_L$

15. TSD (Unit Ready)



INDEX

NUMERICAL ORDER			ALPHABETICAL ORDER		
CODE NO.	OPERATION	PAGE	OPERATION	CODE NO.	PAGE
4	IOS	4-7	ADD	67	3-58
5	JMP	3-30	ADX	15	3-22
6	JPX	3-26	AUX	10	3-20
7	JNX	3-26	COM	56	3-50
10	AUX	3-20	CAB	62	3-42
11	RSX	3-14	CYA	60	3-42
12	SKX	3-24	CYB	61	3-42
14	EXX	3-18	DIV	75	3-62
15	ADX	3-22	DPX	16	3-16
16	DPX	3-16	DSA	65	3-46
17	SKM	3-34	EXA	54	3-10
20	LDE	3-7	EXX	14	3-18
21	SPF	3-54	FLF	31	3-55
22	SPG	3-54	FLG	32	3-55
24	LDA	3-6	INS	55	3-48
25	LDB	3-6	IOS	4	4-7
26	LDC	3-6	ITA	41	3-46
27	LDD	3-6	ITE	40	3-46
30	STE	3-8	JMP	5	3-30
31	FLF	3-55	JNA	47	3-32
32	FLG	3-55	JNX	7	3-26
34	STA	3-8	JOV	45	3-32
35	STB	3-8	JPA	46	3-32
36	STC	3-8	JPX	6	3-26
37	STD	3-8	LDA	24	3-6
40	ITE	3-46	LDB	25	3-6
41	ITA	3-46	LDC	26	3-6
42	UNA	3-46	LDD	27	3-6
43	SED	3-36	LDE	20	3-6
45	JOV	3-32	MUL	76	3-60
46	JPA	3-32	NAB	66	3-40
47	JNA	3-32	NOA	64	3-40
54	EXA	3-10	RSX	11	3-14
55	INS	3-48	SAB	72	3-38
56	COM	3-50	SCA	70	3-38
57	TSD	4-9	SCB	71	3-38
60	CYA	3-42	SED	43	3-36
61	CYB	3-42	SKM	17	3-34
62	CAB	3-42	SKX	12	3-24
64	NOA	3-40	SPF	21	3-54
65	DSA	3-46	SPG	22	3-54
66	NAB	3-40	STA	34	3-8
67	ADD	3-58	STB	35	3-8
70	SCA	3-38	STC	36	3-8
71	SCB	3-38	STD	37	3-8
72	SAB	3-38	STE	30	3-8
74	TLY	3-65	SUB	77	3-58
75	DIV	3-62	TSD	57	4-9
76	MUL	3-60	TLY	74	3-65
77	SUB	3-58	UNA	42	3-46