MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

TX-2 USERS HANDBOOK

ALEXANDER VANDERBURGH, Jr. (Ed.)

LINCOLN MANUAL NO. 45

JULY 1961

LEXINGTON   MASSACHUSETTS

Note to TX-2 Users -

The TX-2 Users Handbook will be printed in several installments - you now have the first. There will be seven chapters - they are listed below in order of (expected) appearance:

Chapter 4 - In-Out System - Sections 46 and 74 (Mag. Tape and
            Plotter) to come later.
   "   7 - Charts   ⎫
   "   6 - M4 Utility System ⎬ Summer '61
   "   5 - Lights and Buttons ⎭
   "   3 - Operation Code   → Fall '61
   "   2 - Functional Description ⎫ Who knows?
   "   1 - Introduction   ⎭

Your comments, criticisms, and (unfortunately,) corrections, are requested.

A. Vanderburgh

A. Vanderburgh

July 1961
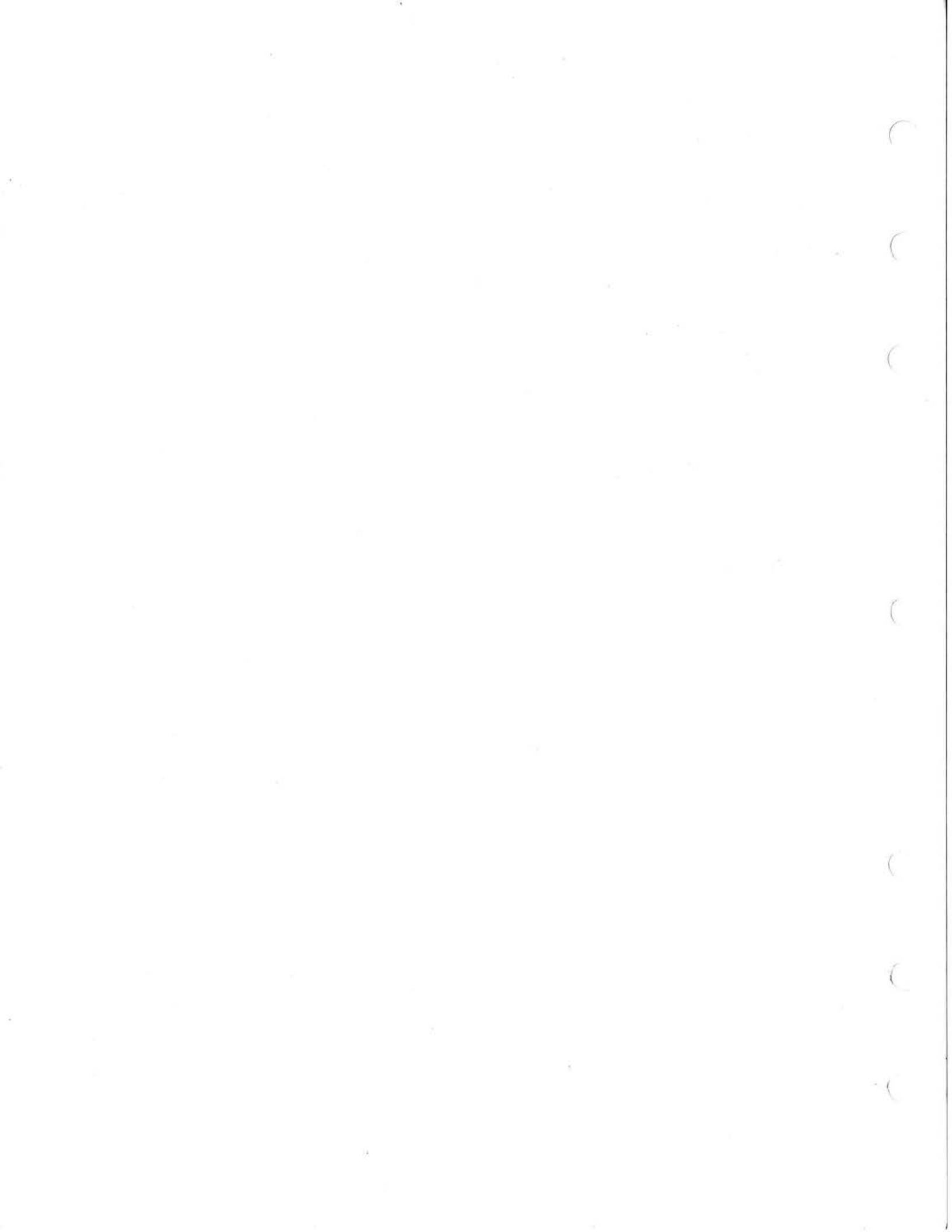
Note to TX-2 Users:

BEHOLD !!!

You now have Chapters 3, 4, 6, and 7.

Chapter 5 - Lights and Buttons - is typed and awaits final proofreading.

A draft of Chapter 1 - Introduction - is available.

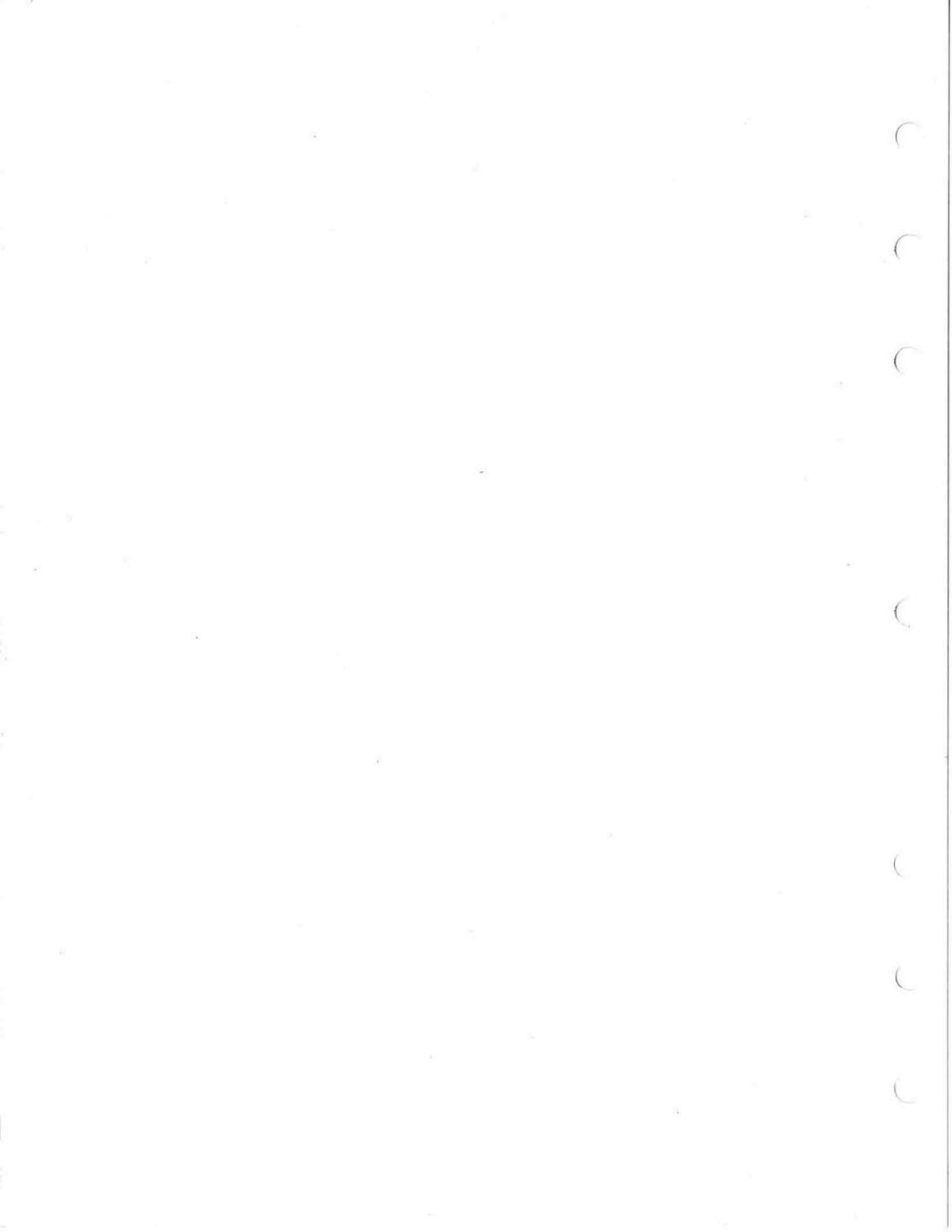Chapter 2 - Functional Description - is being written at last.

And, Chapter 4 - In-Out - is being rewritten. (Time marches on, you know.)

Please report any mistakes or criticisms quietly and discreetly, and I will try to disseminate the required word with dispatch.

Alx. Vanderburgh

A. Vanderburgh

August 1963

————— So here is Chapter 5 — Lights & Buttons —————

Many TX-2 users have asked for this chapter. Now that it is out, I hope it teaches them a lesson.
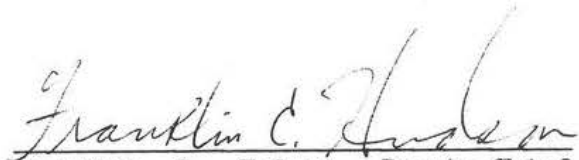
The next installment will be a re-issue of the second part of Chapter 4 (IN-OUT). In the past two years, nearly _all_ the IN-OUT units have been modified or replaced, and a few new ones have been added to the system.

Please do not stand on one foot waiting for Chapters 1 and 2. There may be some delay there.

A. Vanderburgh

This technical documentary report is approved for distribution.

Franklin C. Hudson, Deputy Chief
Air Force Lincoln Laboratory Office

November 1963

TX-2 USERS HANDBOOK

CHAPTER 3 - OPERATION CODE

TABLE OF CONTENTS

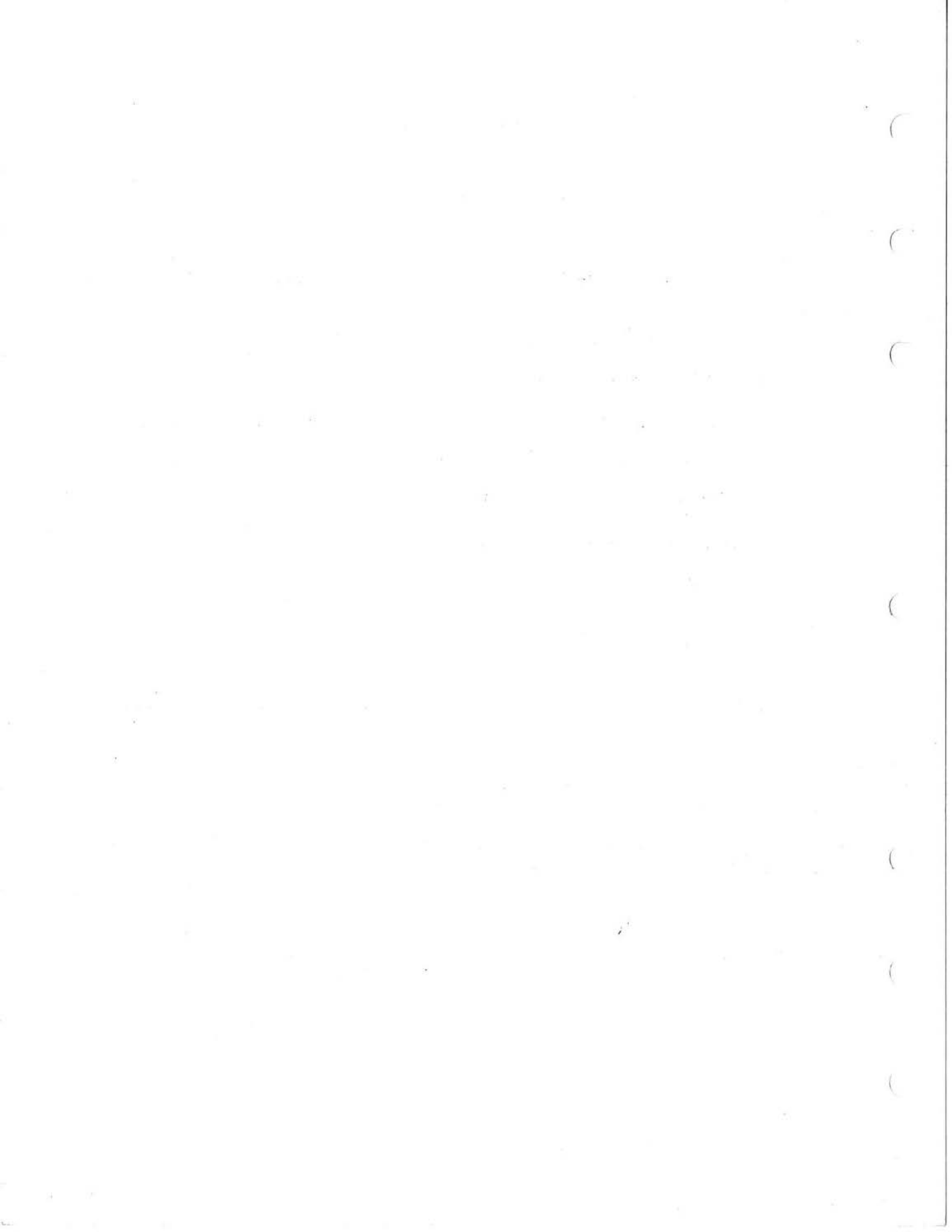|	
X_j	X Memory Register "j"
$[X_j]$	Contents of X Memory Register j
T	STUV memory address "T" (STUV memory is "S", "T", "U", and "V" memories)
T_j	$T + [X_j]$
$[T_j]$	Contents of STUV Memory Register T_j
F_α	F memory register α
$[F_\alpha]$	Contents of F memory register α
$^\alpha[T_j]$	$[T_j]$ Configured 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 cf 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 intc 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
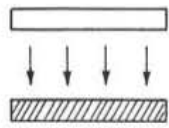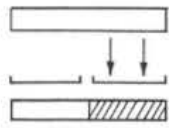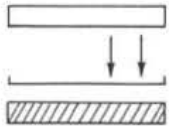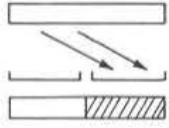
LOAD AE (24-27)

LOAD E REGISTER (20)

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

| $^{\alpha}$LDA T$_j$ | $^{\alpha}[\text{T}_j] ==> \text{A}$ |
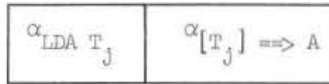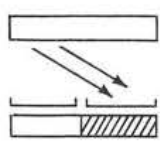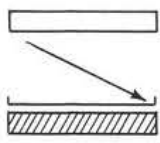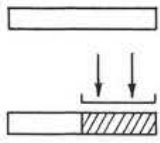
LOAD means copy into the AE from STUV memory. STUV memory is not changed. Activity, Sign Extension, and permutation are used. ALL load instructions except LDE perform the standard $[\text{T}_j] ==> \text{E}$.

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

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

August 1963

5.	^{2}LDA A	A (Before) / A (After)	$L[A] \Longrightarrow R(A)$ $[A] \Longrightarrow E$	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 $[F_{16}] = 163$	T_j / A	$q4[T_j] \Longrightarrow q1(A)$ $Sq4[T_j] \Longrightarrow q2,3,4(A)$ $[T_j] \Longrightarrow E$	The nine bit number in quarter 4 of T_j is expanded to 36 bits in A.
7.	^{1}LDA $(T_k)^*_j$	$(T_k)_j$ / A	$R[(T_k)_j] \Longrightarrow R(A)$ $[(T_k)_j] \Longrightarrow E$	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.)
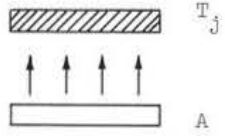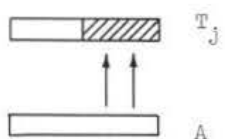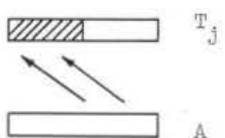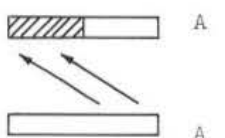
STORE AE (34-37)

STORE E (30)

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

αSTA T$_j$	α[A] ==> T$_j$
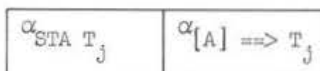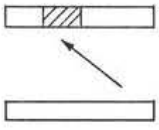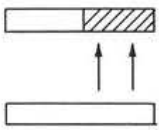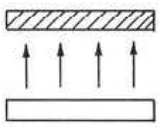
STORE is a non-destructive copy from AE to STUV memory. With a partially active configuration it becomes a _partial_ store. Subword form is meaningless - only active pathways are used. The E register is set from the memory word after the store operation (except for STE which does not change E).

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

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENT
1.	STA T$_j$	T$_j$ / A	[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$_j$	T$_j$ / A	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$_j$	T$_j$ / A	R[A] ==> L(T$_j$) **	^{12}STA would be exactly the same. $[F_{12}] = 142$
4.	^{2}STA A	A / 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.

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

August 1963   3-9

$$\begin{array}{|c|c|}
\hline
\multirow{2}{*}{$^{\alpha}\text{EXA } T_j$} & {}^{\alpha}[A] \implies T_j \\
\cline{2-2}
 & {}^{\alpha}[T_j] \implies A \\
\hline
\end{array}$$
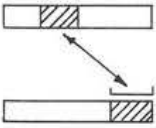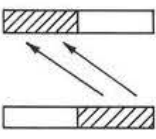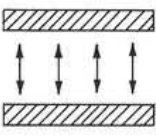
EXCHANGE A is a combination of STA and LDA. Sign extension, if any, occurs only in A and <u>after</u> the exchange of data. Subword form, Activity, and permutation are all used.

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

EXAMPLES:

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

** 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 $[F_5] = 762$	T_j A	$q3[T_j] ==> q1A$ $q1[A] ==> q3(T_j)$	**
6.	^{2}EXA A	A (Before) A (After)	$R[A] ==> L(A)$	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	$[(T_k)_j] ==> A$ $[A] ==> (T_k)_j$	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$_j$ |
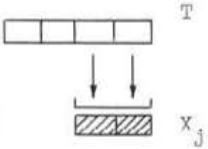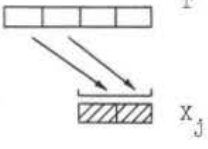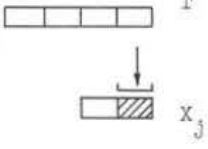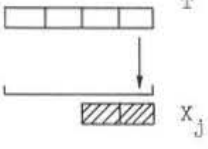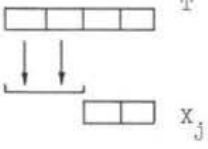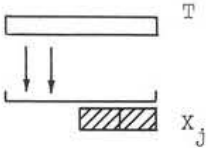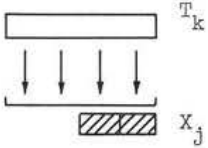
RESET is a non-destructive copy from STUV memory into X memory.

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.	^{1}RSX$_j$ T		$R[T] ==> X_j$ $[T] ==> E$	^{0}RSX would do the same.
2.	^{2}RSX$_j$ T		$L[T] ==> X_j$ $[T] ==> E$	^{12}RSX would do the same. $[F_{12}] = 142$
3.	^{3}RSX$_j$ 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).

6.	$^{\alpha}$RSX$_j$ T $[F_\alpha]$ = 030		Sq4(T) ==> X$_j$ [T] ==> E	This time X$_j$ is cleared because of sign extension.
7.	RSX$_j$ *{T$_k$}		R[T$_k$] ==> X$_j$ [T$_k$] ==> E	With a deferred address, RSX is indexable. Note that E is set from T$_k$ this time.
8.	$^{\alpha}$RSX$_0$ T	⟶	[T] ==> E	Nothing happens because X register 0 cannot be changed. [X$_0$] = 0 permanently.

$^{\alpha}DPX_j\ T$	$^{\alpha}[X_j] \Rightarrow T$
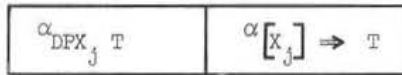
DEPOSIT is a non-destructive copy from X memory into STUV memory.
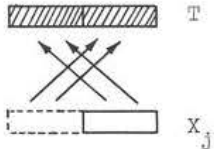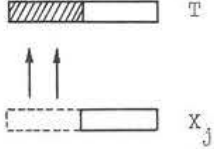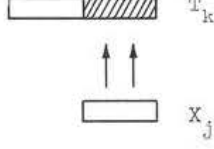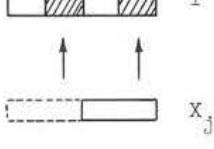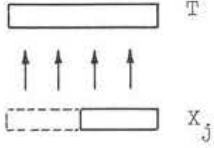
Activity and Permutation are used.

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

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

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

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED EXPLANATION	COMMENT
1.	$^1DPX_j\ T$		$[X_j] \Rightarrow R(T)$	Only the right half of T is changed.
2.	$^2DPX_j\ T$		$[X_j] \Rightarrow L(T)$	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.	$^3DPX_j\ T$		$R[X_j] \Rightarrow q1(T)$	Only quarter 1 of T is changed.
5.	$^{16}DPX_j\ T$ $[F_{16}] = 163$		$R[X_j] \Rightarrow q4(T)$	Only quarter 4 is changed for only one path is active.

6.	$^{17}DPX_j\ T$ $[F_{17}] = 202$	T X_j	$SX_j \Rightarrow R(T)$ $[X_j] \Rightarrow L(T)$	All of T is affected.
7.	$^{21}DPX_j\ T$ $[F_{21}] = 230$	T X_j	$SX_j \Rightarrow L(T)$	Surprisingly enough, this does do something. (See example 5, RSX.)
8.	$^{1}DPX_j\ \{T_k\}^*$	T_k X_j	$[X_j] \Rightarrow T_k$ $[T_k] \Rightarrow E$	Deposit is indexable with deferred addressing.
9.	$^{33}DPX_j\ T$ $[F_{33}] = 320$	T X_j	$SX_j \Rightarrow q3(T)$ $[X_j] \Rightarrow q1(T)$	Note that bit 2.9 of X_j is used even though quarter 2 is not active.
10.	DPX 377720	T X_j	$[X_j] \Rightarrow R(E)$ $SX_j \Rightarrow L(E)$	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."

$$\alpha EXX_j \ T \quad \Big| \quad \begin{array}{l} \alpha[X_j] ==> T \\[1em] \alpha[T] ==> X_j \end{array}$$
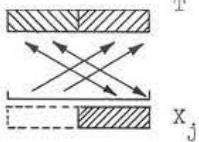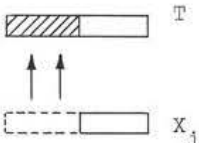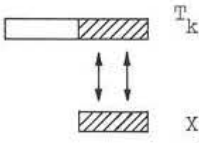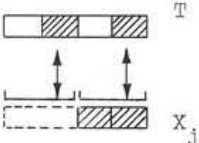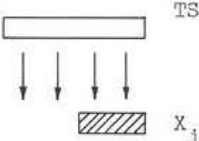
EXX is a combination of RSX and DPX. Except for sign extension, it does just what its name
implies - i.e. it will interchange words between X memory and STUV memory.

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.	$^1EXX_j \ T$		$R[T] ==> X_j$ $[X_j] ==> R(T)$ $[T] ==> E$	
2.	$^2EXX_j \ T$		$L[T] ==> X_j$ $[X_j] ==> L(T)$ $[T] ==> E$	
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.	$^3EXX_j \ T$		$q1[T] ==> R(X_j)$ $R[X_j] ==> q1(T)$ $[T] ==> E$	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.	$^{17}\text{EXX}_j$ T $[\text{F}_{17}] = 202$		$[X_j] ==> L(T)$ $L[T] ==> X_j$ $S(X_j) ==> R(T)$ $[T] ==> E$	Sign of X_j is extended into the right half of T.
7.	$^{21}\text{EXX}_j$ T $[\text{F}_{21}] = 230$		$S(X_j) ==> L(T)$ $[T] ==> E$	Same as $^{21}\text{DPX}_j$ T.
8.	$^{1}\text{EXX}_j \{T_k\}^*$		$R[T_k] ==> X_j$ $[X_j] ==> R(T_k)$ $[T_k] ==> E$	EXX is indexable if a deferred address is used.
9.	$^{33}\text{EXX}_j$ T $[\text{F}_{33}] = 320$		$Sq1[T] ==> L(X_j)$ $q1[T] ==> R(X_j)$ $R[X_j] ==> q1(T)$ $S(X_j) ==> q3(T)$ $[T] ==> E$	Note that bit 2.9 is used for sign extension (not 1.9).
10.	$^{1}\text{EXX}_j$ 377720		$R[377720] ==> X_j$ $[X_j] ==> R(E)$ $L[377720] ==> L(E)$	Same as $^{1}\text{RSX}_j$ 377720. (Toggle registers must be changed by hand. Note that E is set to what would have gone into T.)

$$\boxed{^{\alpha}\text{AUX}_j \quad \text{T} \quad \Big| \quad [\text{X}_j] + {}^{\alpha}[\text{T}] ==> \text{X}_j}$$
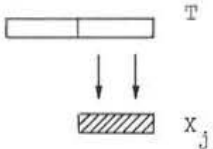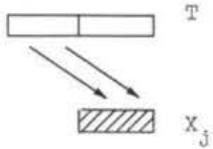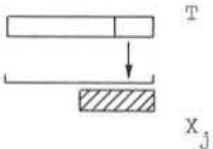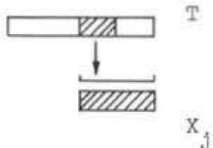
AUX forms an 18 bit ring sum in X_j. There is no overflow detection. All of X_j is affected. STUV memory is not affected.
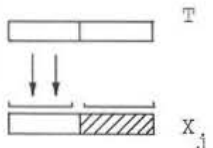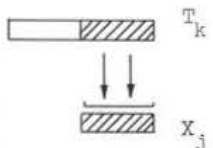
Activity and permutation are used. Sign extension applies to the operand taken from STUV memory. If quarters 1 and 2 are active, subword form is immaterial.

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

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

NOTE: E is cleared and then loaded as if by $^{\alpha}\text{LDE}$. 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.

$^{\alpha}$ADX$_j$ T	$[X_j] + {}^{\alpha}[T] \Longrightarrow T$

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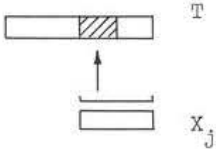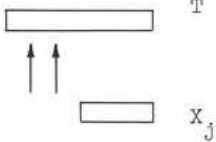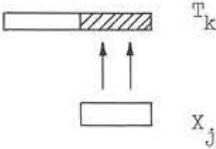
Activity and Permutation are used. Only active quarters are stored, but sign extension applies to the operand taken from STUV memory.

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}ADX$_j$ T		$[X_j] + R[T] \Longrightarrow RT$ $[T] \Longrightarrow E$	Left half of T is not changed. The sum is standard 18 bit ring sum, also called "ones complement sum."
2.	^{2}ADX$_j$ T		$[X_j] + L[T] \Longrightarrow LT$ $[T] \Longrightarrow E$	Right half of T is not changed.
3.	^{13}ADX$_j$ T $[F_{13}] = 160$		$[X_j] + q1[T]_{SE} \Longrightarrow q1(T)$ $[T] \Longrightarrow E$	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.

$$\boxed{\alpha_{\text{SKX}_j} T}$$

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 OPERAND.

EXAMPLES:

NO.	INSTRUCTION	MNEMONIC ABBREVIATION (See Chart 7-3)	ABBREVIATED DESCRIPTION	COMMENT
1.	$^0\text{SKX}_j\ T$	$\text{SKX}_j\ T$ $\text{REX}_j\ T$ $\text{SEX}_j\ T$ (Set)	$T ==> X_j$	STUV memory is not used - "T" <u>is</u> the operand, not its location. The brackets [ ] were left out on purpose.
2.	$^1\text{SKX}_j\ T$	(Set negative)	$-T ==> X_j$	"Minus" T - i.e. its ones complement is used to set X_j.
3.	$^2\text{SKX}_j\ T$	$\text{INX}_j\ T$ (Increase)	$[X_j] + T ==> X_j$	If the sum is zero, it will be -0 (all ones) unless $[X_j]$ was initially +0.
4.	$^3\text{SKX}_j\ T$	$\text{DEX}_j T$ (Decrease)	$[X_j] + (-T) ==> X_j$	"-T" is added to $[X_j]$. Zero is -0. It cannot be +0.
5.	$^4\text{SKX}_j\ T$	$\text{SXD}_j\ T$ (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.	$^5\text{SKX}_j\ T$	(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.	$^6\text{SKX}_j\ T$	$\text{SXL}_j\ T$ (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) $SKX_{j}\ \{0_{k}\}^{*} \equiv$ set X_{j} from X_{k}. b) $^{1}SKX_{j}\ \{0_{j}\}^{*} \equiv$ 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 <u>can</u> 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.)

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

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:

$\#+1 \Longrightarrow R(E)$

$T \Longrightarrow P$

DISMISS occurs unless
cancelled via "h".

If it does not:

$\#+1 \Longrightarrow P$

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.)
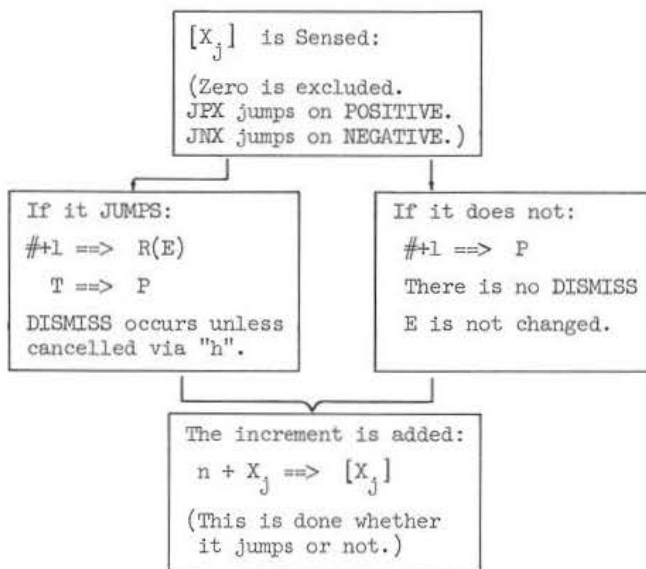
Note: 1. If the sum is zero, it is -0.
       2. "n" is a signed integer: -17 to $+ 17_{8}$.

       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   b.) JNX

Start $\rightarrow$ REX_{j} 77   Start $\rightarrow$ $^{1}SKX_{j}$ 77

Loop $\rightarrow$ LDA $TABL_{j}$ Loop $\rightarrow$ LDA $(TABL + 77)_{j}$

       h^{-1} JPX_{j} Loop h^{+1} JNX_{j} Loop

This program scans the table   This program scans "forward
"backward through the manu-   through the manuscript." (i.e.,
script." (i.e., highest   lowest memory location first.)
memory location first.) Note: Note: X_{j} is initially set to
X_{j} is initially set to $+ (n-1)$. $- (n-1)$.

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

    a) START $\rightarrow$ REX$_j$ (TL - n)   b) START $\rightarrow$ ^{1}REX$_j$ (TL - n)

             LDA TABL$_j$ LDA$_j$ TABL + TL - n

             h^{-n} JPX$_j$ #-1 h^{+n} JNX$_j$ #-1

These programs run for $\left(\frac{TL}{n}\right)$ 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 - ^{1}REX$_j$ C

                           INX$_j$ TL

                           LDA (TABL + C - 1)$_j$

                         h^{-C} JPX$_j$ #-1

                         JMP #-3

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.
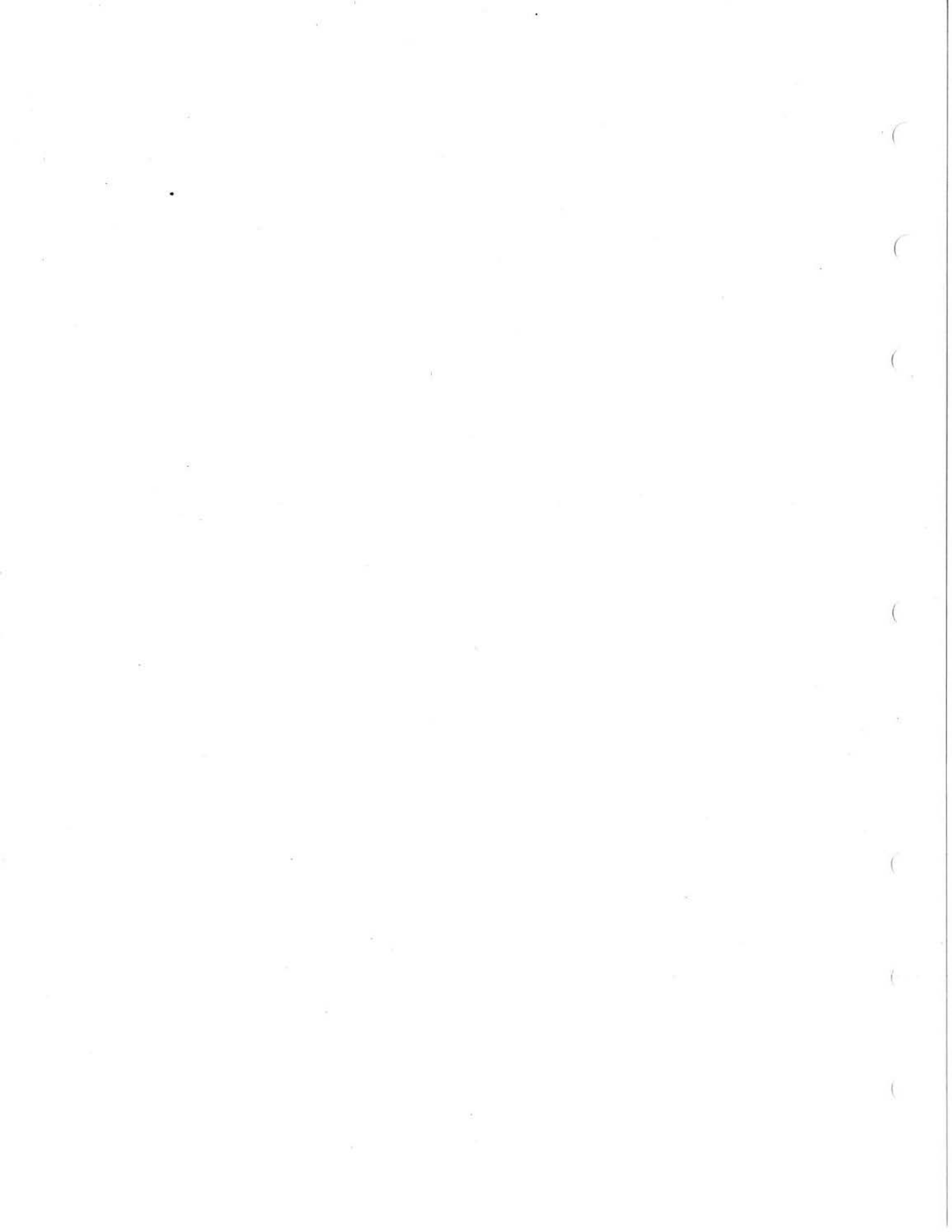
3-2.3 JUMP SKIP CLASS

JMP
JPA
JNA
JOV
SKM
SED

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

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:

4.8 4.7 4.6 4.5 4.4

DISMISS
(See Chap. 4)

Saves last memory
reference in L(E)

Saves return point (#+1)
in R(E)†

1 = "BRANCH" ≡ An indexable JMP
$^1JMP \equiv BRC \equiv$ Go to T_j

1 = Saves return point (#+1) in X_j
$^2JMP \equiv JPS \equiv$ Go to T, save
return point in X_j.†
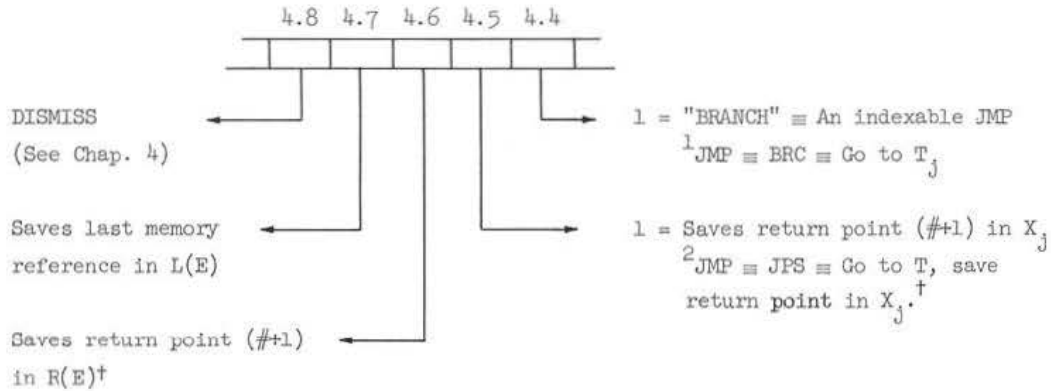
EXAMPLES: (See #10.)

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

JPA - Jump on Positive Accumulator
JNA - Jump on Negative Accumulator
JOV - Jump on Overflow

$$\boxed{\begin{array}{l} \alpha_{JPA}\ T_j \\ \alpha_{JNA}\ T_j \\ \alpha_{JOV}\ T_j \end{array}}$$
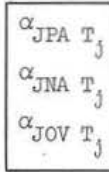
The conditional jumps go to T_j if the conditions are satisfied by <u>any active subword</u>. Permutation is ignored. The return point (#+1) is saved in E if the jump takes place. The accumulator and overflow flip-flops are not changed. Note that these conditional jumps are indexable.

EXAMPLES:

#1. A Four-way Switch:

```
JOV    OF          ** Goes to OF if overflow exists (Z₄ = 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 <u>any</u> 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 <u>not</u> cleared by JOV.

Active subwords use the overflow indicator associated with the <u>sign</u> 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
¹DPX_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:

    20 JPA 1P   ** One half (or both) positive - (Goes to 1P)

    20 JNA 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)

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

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

4.9 4.8   4.7 4.6 4.5   4.4

00 - No skip   00 - No change
01 - Skip unconditionally   01 - Bit is complemented
10 - Skip if bit = 0   10 - Bit is set to 0 ("Make Zero")
11 - Skip if bit = 1   11 - Bit is set to 1 ("Make One")
("Skip" means "go to (#+2)"
i.e. skip over the next   If 4.6 = 1, T is cycled right once. (Rotated)
instruction.)
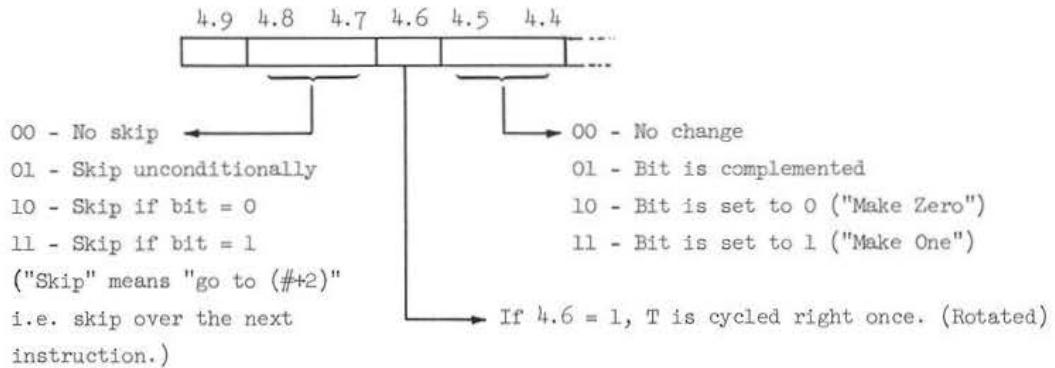
The bit in question is identified by its quarter number and bit number as diagrammed below:

4.9...........4.1 3.9.........3.1 2.9......... 2.1 1.9.......... 1.1
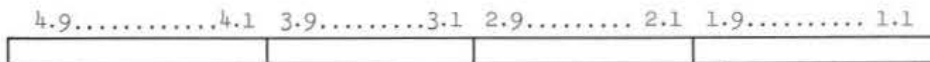
The meta bit is No. 10 (dec.). (SKM is the only instruction that can affect it.)
The parity bit is No. 11 (dec.).
                                    These can not   be changed by SKM.
The parity circuit is No. 12 (dec.).
(Any quarter number will do for the parity and meta bits.)

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:

3.6   3.5 3.4 3.3 3.2 3.1

Quarter No.   Bit Number   (When given in the form indicated above,
(00 refers to q4)   Bit Numbers are interpreted as Decimal,
                                             e.g. 4.10 is the usual metabit designation.)
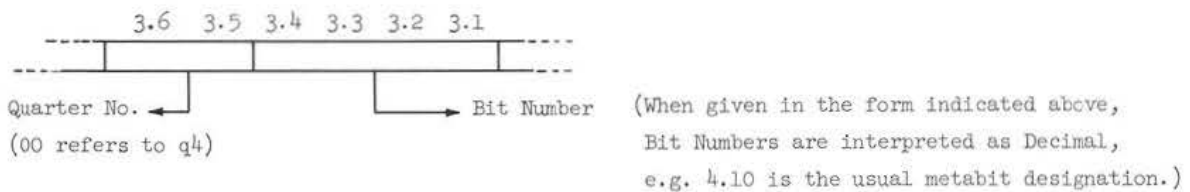
SKM is therefore non-indexable except through deferred addressing.

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}SZZ $\equiv$ ^{26}SKM $\equiv$ Skip on zero, make zero, and then rotate.

EXAMPLES:

1. To copy a bit:

$\left.\begin{array}{l}\text{SKZ } Q_{2.3} \\ \text{SUN } T_{1.1} \\ \text{MKZ } T_{1.1}\end{array}\right\}$ Sets bit $T_{1.1}$
   equal to
   bit $Q_{2.3}$

2. To clear n metabits starting at T

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

| αSED 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

$^{\alpha}\text{SCA T}_j$	$^{\alpha}[A] \times 2^{\alpha[T_j]} ==> A$

"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 $^{\alpha}\text{LDD T}_j$ is performed (with permutation and sign extension as called for).

b) Each active subword (of A or AB) is scaled according to its <u>sign quarter in D</u>, 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 <u>not</u> 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 $\pm$0). SCALE senses overflow and corrects the sign bit if necessary. SCA and SAB always <u>clear</u> the overflow flip-flop - even if bits are lost off the left end. SCALE <u>never</u> sets the overflow flip-flop.

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

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENT
1.	SCB{-4,}	{-4,} D	$[B] \times 2^{-4} ==> B$ $-0 ==> q4(D)$ $q3,2,1[T_j] ==> q321(D)$	{-4,} is a M4 convention for A register with -4 in quarter 4. See Chapter 6, page 6-7 and 6-10.
2.	$^{30}\text{SCB (N)}$ N = 2775003000(8)	(N) D	$q4[B] \times 2^2 ==> q4(B)$ $q3[B] \times 2^{-2} ==> q3(B)$ $q2[B] \times 2^3 ==> q2(B)$ $-0 ==> D$	Quarter 1 of B is not changed. The sign bits are never changed. Bits may be lost off either end without any alarm.
3.	$^{2}\text{SCB (N)}$ N = 2775003000(8)	(N) D	$R[B] \times 2^2 ==> R(B)$ $-0 ==> q2(D)$ $775 ==> q1(D)$	The <u>left</u> halves of 3 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 <u>overflow indicators</u>. They tell whether overflow has occurred. An overflow resulting from negative numbers (as in q2 above) is <u>not</u> treated any differently.

5. Subword forms for the AB register:

a) "36"

S	A	B

b) "18 - 18"

S	L(A)	L(B)	S	R(A)	R(B)

c) "27-9"

S	q432(A)	q432(B)	S	q1(A)	q1(B)

d) "9-9-9-9"

S	q4(A)	q4(B)	S	q3(A)	q3(B)	S	q2(A)	q2(B)	S	q1(A)	q1(B)
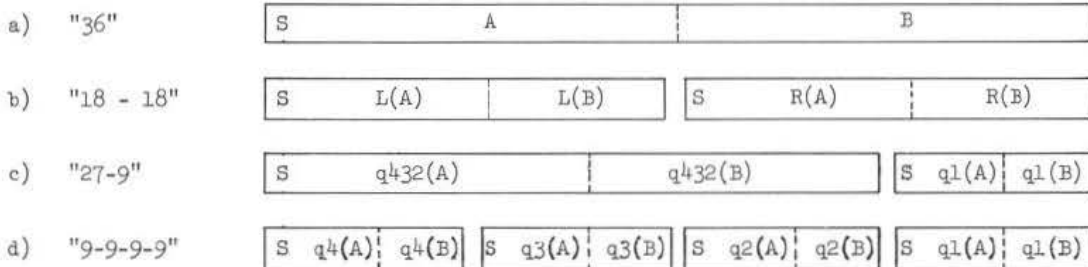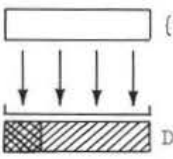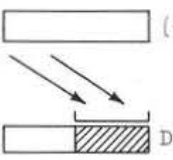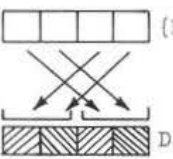
Note that all of B is part of the subword. There is only one sign bit in an AB subword.

$$^\alpha\text{NOA } T_j \quad \left| \begin{array}{l} ^\alpha[A] \times 2^{nz} ==> A \\ \\ ^\alpha[T_j] - nz ==> Sq(D) \end{array} \right.$$
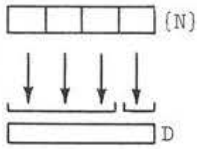
NORMALIZE scales just enough to remove leading zeros or to "recover" from OVERFLOW. It clears the active overflow indicators. The number of leading zeros (nz) is subtracted from the argument from T_j ($^\alpha[T_j]$) and this difference is left in the Sign Quarter of D. If an overflow condition exists 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 nine bit 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.

August 1963

4.	$\alpha_{NOA(N)}$ N = a,b,,c,d α = 400	▯▯▯▯ (N) ↓ ↓ ↓ ↓ ▭▭ D	$q432[A] \times 2^{nz} ==> q432(A)$ $a-nz ==> q4(D)$ $b ==> q3(D)$ $c ==> q2(D)$ $q1[A] \times 2^{nz} ==> q1(A)$ $d-nz ==> q1(D)$	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.

$$\boxed{^{\alpha}\text{CYA T}_{j}}$$

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.
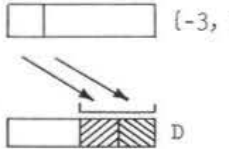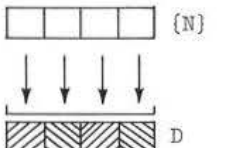
a.) An $^{\alpha}$LDD T$_{j}$ is the first step.

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,)	(+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	(N)	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, }	{-3, } D	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	{N} D	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

$$\boxed{\alpha_{\text{ITA } T_j} \quad \Big| \quad {}^{\alpha}[T_j] \ \wedge \ [A] \ ==> A}$$

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 <u>subwords</u> - with sign extension if applicable. For these three, the E register is set, as usual, identical to the memory word used.
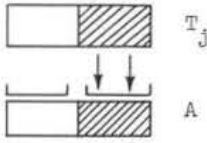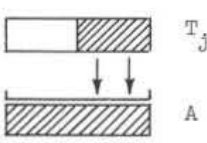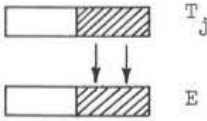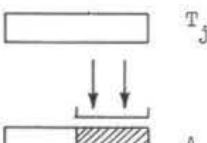
For ITE, the operand is the active <u>quarters</u> only. There is no sign extension. The result, of course, goes into E and there is no final E register copy from memory.

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

Name	INTERSECT	UNITE	DISTINGUISH**
Abbreviation	ITA ITE	UNA	DSA**
Symbol	$\wedge$	$\vee$	$\circledv$
Other Names	"AND"	Inclusive OR	Exclusive OR Partial Add
Logic Diagram	${}^{\alpha}[T_j]$ $\begin{array}{c c} & 0 \quad 1 \\ \hline {}^{\alpha}[A] \ \begin{matrix} 0 \\ 1 \end{matrix} & \begin{matrix} 0 \quad 0 \\ 0 \quad 1 \end{matrix} \end{array}$ Note that this is the "carry" that results from addition.	${}^{\alpha}[T_j]$ $\begin{array}{c c} & 0 \quad 1 \\ \hline {}^{\alpha}[A] \ \begin{matrix} 0 \\ 1 \end{matrix} & \begin{matrix} 0 \quad 1 \\ 1 \quad 1 \end{matrix} \end{array}$	${}^{\alpha}[T_j]$ $\begin{array}{c c} & 0 \quad 1 \\ \hline {}^{\alpha}[A] \ \begin{matrix} 0 \\ 1 \end{matrix} & \begin{matrix} 0 \quad 1 \\ 1 \quad 0 \end{matrix} \end{array}$ 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.
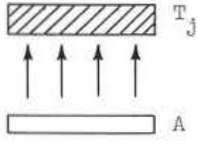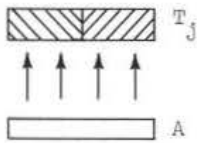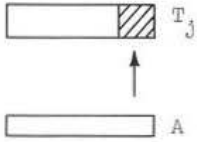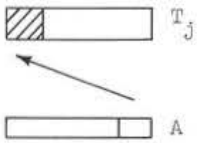
** Note: DSA affects both the C and D registers. The effect on D is equivalent to LDD T_j. The effect on C is equivalent to forming the carries and uniting them with the original contents of C. - i.e. $([A] \wedge [T_j]) \vee [C] ==> C$.

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, $L(A)$ 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		$R[T_j] \oslash R[A] => R(A)$ $R[T_j] => R(D)$ $[T_j] => E$ $(R[T_j] \wedge A]) \vee R[C] => R(C)$	DSA affects registers A, C, D, and E. See note above.

$^\alpha$INS T_j	$([A]\wedge[B]) \vee ([\overline{B}]\wedge[T_j]) \Rightarrow T_j$ |†
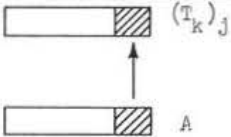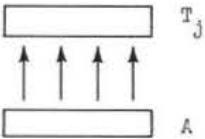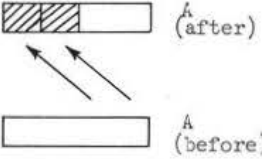
Insert is a <u>partial</u> STA (store accumulator) instruction — only those bits marked by a 1 in the corresponding column of B are stored in T_j. There is no sign extension, and [A] is not changed. If [B] is minus zero (all ones), INS is identical to STA. The E register is set to the final contents 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.

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

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

$$\begin{array}{|c|c|}
\hline
\alpha_{COM} \quad T_j & \alpha\overline{[T_j]} \implies T_j \\
& T_j \text{ is permuted.} \\
\hline
\end{array}$$
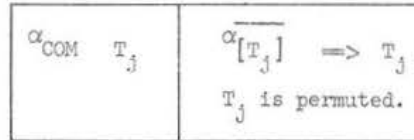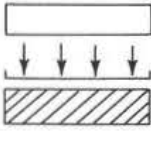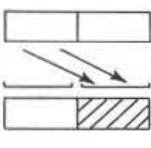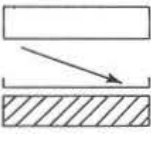
COM - Complement - performs two basic operations. The active subwords of T_j are complemented (one's complement - all ones become zeros and vice versa) (with sign extension) and all quarters are permuted whether active or not. Note that if all quarters are inactive, COM permutes all quarters of T_j without changing the data. PMT is another abbreviation - equivalent to COM.
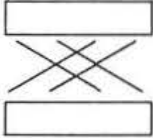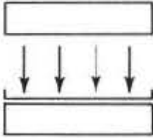
There are 4 basic steps:

1. $[T_j] \implies E$, permuted according to α.

2. Sign extension occurs in active subwords.

3. Active subwords are complemented. $\left(\alpha\overline{[E]} \implies {}^{\alpha}E\right)$

4. $[E] \implies T_j$ 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)	$\overline{[T_j]} \implies T_j$ $\overline{[T_j]} \implies E$	All of T_j is complemented
2	^{2}COM T_j	T_j (before) T_j (after)	$\overline{L[T_j]} \implies R(T_j)$ $R[T_j] \implies L(T_j)$	The halves are reversed and the right half is complemented.
3	^{16}COM T_j $[F_{16}] = 163$	T_j (before) T_j (after)	$\overline{q4[T_j]} \implies q1(T_j)$ $\overline{Sq4[T_j]} \implies q2,3,4(Tj)$	Quarters 2, 3, and 4 are set to the complemented sign extension.

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENTS
4	$^{\alpha}\text{COM } T_j$ $\alpha = 172$ (all inactive)	T_j T_j	$R[T_j] \Rightarrow L(T_j)$ $L[T_j] \Rightarrow R(T_j)$ (Simultaneously)	When all quarters are inactive, the data is not changed - it is merely permuted according to the given configuration.
5	$\text{COM } (T_k)_j^{*}$	$T_{k,j}$ $T_{k,j}$	$\overline{[T_{k,j}]} \Rightarrow T_{k,j}$ $\overline{[T_{k,j}]} \Rightarrow E$	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.
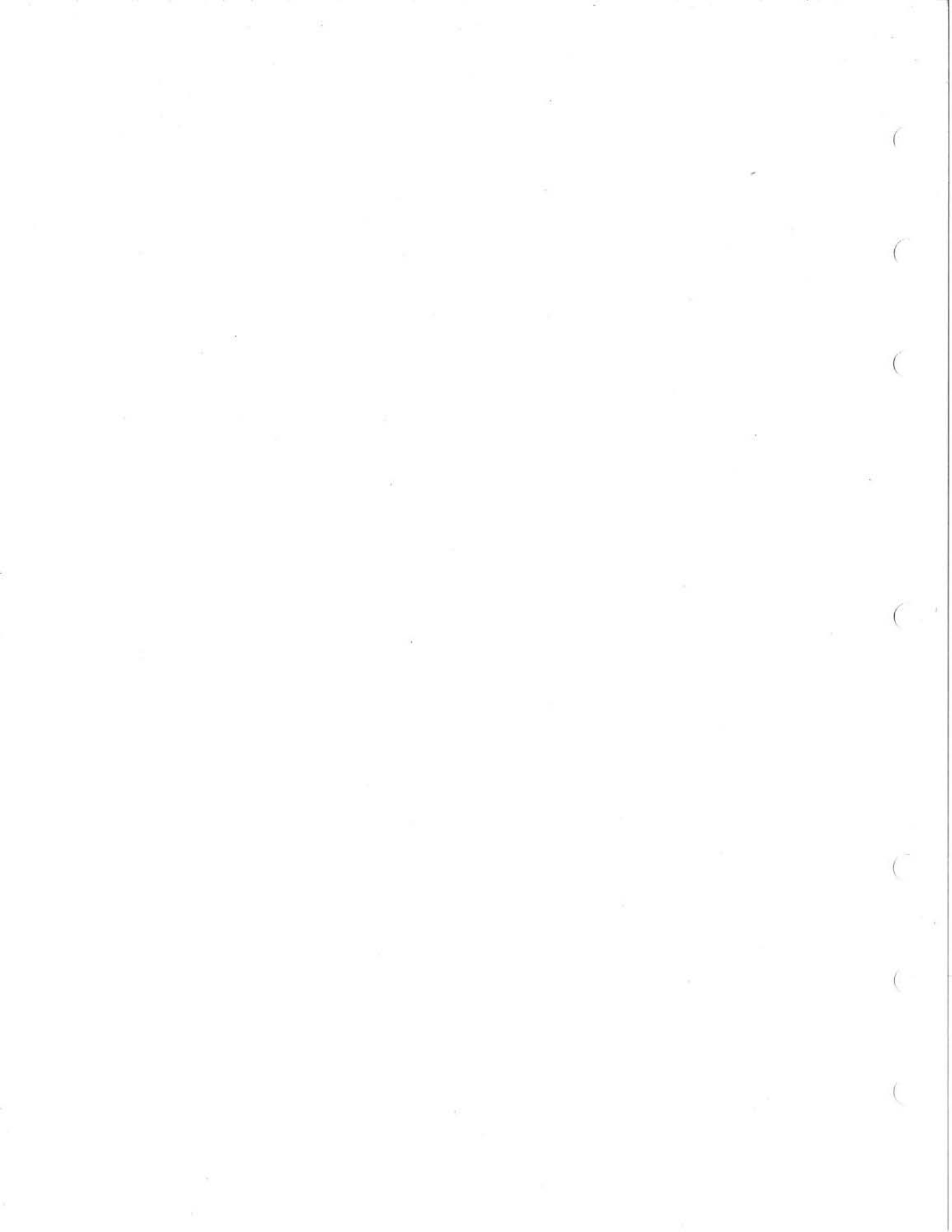
3-2.6 CONFIGURATION MEMORY CLASS

SPF
SPG
FLF
FLG

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

"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_0 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_0 is, of course, not changed. The F Memory address "c" is normally given in OCTAL

^{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.
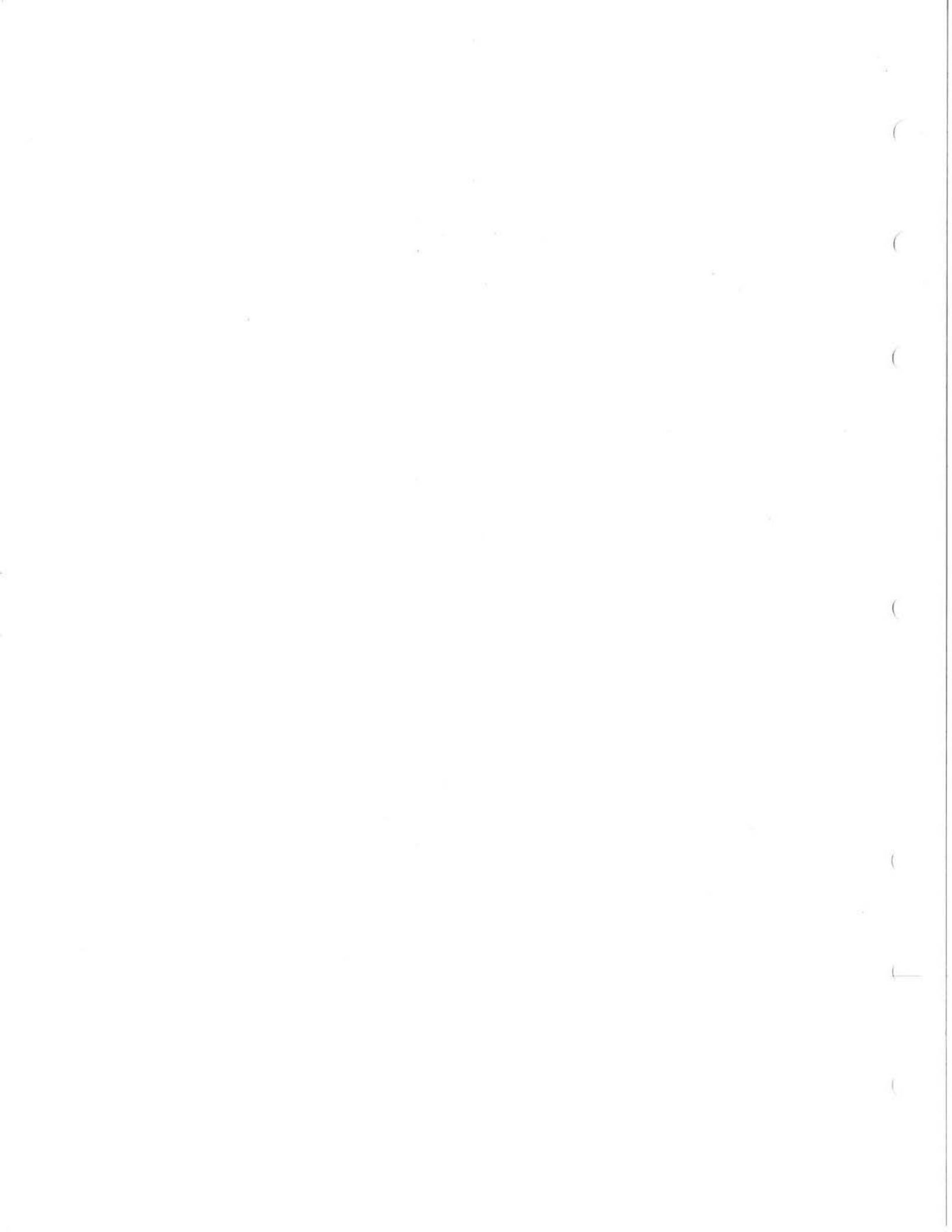
3-2.7 ARITHMETIC CLASS

ADD
SUB
MUL
DIV
TLY   (TALLY)
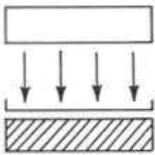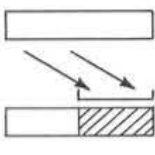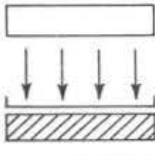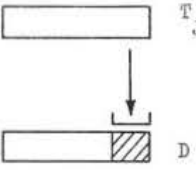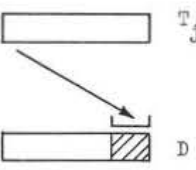
$^{\alpha}$ADD T_j	$^{\alpha}[A] + {^{\alpha}}[T_j] => {^{\alpha}}A$
$^{\alpha}$SUB T_j	$^{\alpha}[A] - {^{\alpha}}[T_j] => {^{\alpha}}A$

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 either 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	T_j D	$[A] + [T_j] => A$ $[A] \wedge [T_j] => C$ $[T_j] => D$ $[T_j] => E$	The expression $[A] \wedge [T_j] => C$ 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	T_j D	$R[A] + L[T_j] => R(A)$ $R[A] \wedge L[T_j] => R(C)$ $L[T_j] => R(D)$ $[T_j] => E$	The left half of the A, C, and D registers is unchanged. Z_2 is set if overflow occurs.
3	SUB T_j	T_j D	$[A] - [T_j] => A$ $[A] \wedge \overline{[T_j]} => C$ $[T_j] => D$ $[T_j] => E$	Z_4 is set if overflow occurs.

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENTS
4.	^{3}LDA (277) ^{3}ADD {307}	T_j / D	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}	T_j / D	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

$$\boxed{\alpha_{MUL} \quad T_j} \quad \boxed{\alpha_{[A]} \quad x \quad \alpha_{[T_j]} \Longrightarrow \alpha_{(AB)}}$$
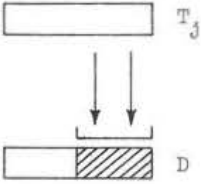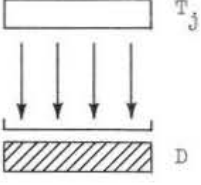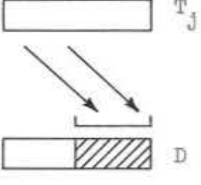
"MUL" forms the double-length, ones-complement product of $[A]$ and $[T_j]$ and stores it in A and B. The extra bit of B -- at the extreme right -- is set equal to the sign bit of the product, i.e., to ± 0.   (Bit 1.1 of B = Bit 4.9 of A after MUL.)
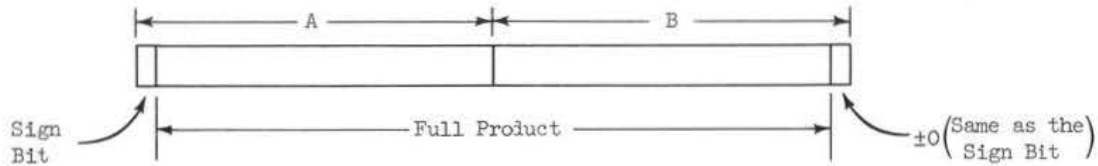
The use of configuration is similar to LDA and the relevant overflow indicator (corresponding to the active sign quarter) is cleared. No overflow can be generated. The active subwords of C are cleared to $+0$ and D is set as if an α_{LDD} T_j had been done. The E register is, as usual, set to the contents 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 $1 \cdot 1 (B)$ $+ 0 \Rightarrow Z_4$ $+ 0 \Rightarrow C$ $[T_j] \Rightarrow D$ $[T_j] \Rightarrow E$	"AB" is the double length register diagrammed above. It is also used with SAB, CAB, and DIV. Bit $1 \cdot 1$ of B is set to ± 0 -- depending on the sign of the product.
2	$^3LDA \ \{5\}$ $^3MUL \ \{4\}$		$000 \Rightarrow q\ 1\ (A)$ $050 \Rightarrow q\ 1\ (B)$ $000 \Rightarrow q\ 1\ (C)$ $004 \Rightarrow q\ 1\ (D)$ $0 \Rightarrow Z_1$	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)	T_J D	$+ 0 ==> R(A)$ $000030 ==> R(B)$ $+ 0 ==> R(C)$ $- 4 ==> R(D)$ $- 4 ==> E$ $0 ==> Z_2$	The left half words are not changed.
4.	LDA (3) MUL (-400)	T_J D	$- 0 ==> A$ $- 3000 ==> B$ $+ 0 ==> C$ $- 400 ==> D$ $- 400 ==> E$ $0 ==> Z_4$	
5.	^{2}LDA (3 ,, 0) ^{2}MUL (4 ,, 0)	T_J D	$+ 0 ==> R(A)$ $000030 ==> R(B)$ $+ 0 ==> R(C)$ $+ 4 ==> R(D)$ $(+4,,0) ==> E$ $0 ==> Z_2$	Only the right 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.

$$
\begin{array}{|c|c|}
\hline
 & {}^{\alpha}[\text{AB}] \ \div \ {}^{\alpha}[\text{T}_j] \quad ==> \text{A} \\
\cline{2-2}
{}^{\alpha}\text{DIV} \ \ \text{T}_j & \text{Remainder} \qquad ==> \text{B} \\
\hline
\end{array}
$$
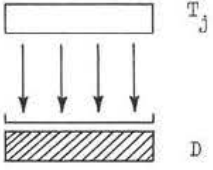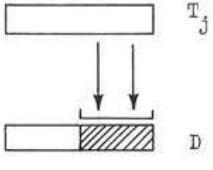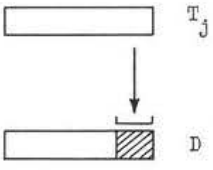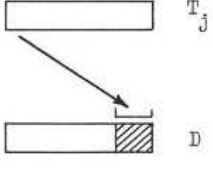
DIVIDE considers the contents of AB (except for the lowest order bit of B) as the numerator and the contents of T_j as the denominator. (Note that it is compatible with MUL.) Configuration is similar to ADD, LDA, etc. The Quotient is stored in A with the appropriate algebraic sign. The remainder is stored 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.)

$$
\frac{[\text{AB}]}{[\text{T}_j]} \equiv Q + \frac{R}{[\text{T}_j]} \qquad\qquad \begin{array}{l} Q ==> \text{A} \\ R ==> \text{B} \end{array}
$$

The relevant overflow indicator is cleared at the outset and an overflow will be generated if $|\ [\text{A}]\ |$ exceeds or equals $|\ [\text{T}_j]\ |$.

Note: 1.   If $|[\text{A}]| \ < \ 2 \cdot |[\text{T}_j]|$ 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 $[\text{AB}]$ and $[\text{T}_j]$ 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 $[\text{A}]$ and $[\text{B}]$ are useless.

5. $\dfrac{\text{N}}{+\ 0} = \overline{\text{N}}$, and Overflow is set. (This is true for any N.)

6. $\dfrac{\text{N}}{-\ 0} = \text{N}$, and Overflow is set. (Also true for any N.)

7.   Divide clears C (as if by ${}^{\alpha}\text{LDC (0) }$) and sets D (as if by ${}^{\alpha}\text{LDD } \text{T}_j$).

8.   The contents of the memory register go into E, as usual.

9.   See also note on page 3-61.

NO.	INSTRUCTION	CONFIGURATION DIAGRAM	ABBREVIATED DESCRIPTION	COMMENTS
1.	DIV T_j	T_j D	$[AB] \div [T_j] \Rightarrow A$ Remainder $\Rightarrow B$ $+0 \Rightarrow C$ $[T_j] \Rightarrow D$ $[T_j] \Rightarrow E$	Overflow, if any, sets Z_4.
2.	^{1}DIV T_j	T_j D	$R[AB] \div R[T_j] \Rightarrow R(A)$ Remainder $\Rightarrow R(B)$ $+0 \Rightarrow R(C)$ $R[T_j] \Rightarrow R(D)$ $[T_j] \Rightarrow E$	Overflow sets Z_2. The left half of the arithmetic unit is unchanged.
3.	^{3}LDA (000) ^{3}LDB (052) ^{3}DIV (5)	T_j D	$004 \Rightarrow q1(A)$ $001 \Rightarrow q1(B)$ $000 \Rightarrow q1(C)$ $005 \Rightarrow q1(D)$ $005 \Rightarrow E$ $0 \Rightarrow Z_1$	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,}	T_j D	$+4 \Rightarrow q1(A)$ $-1 \Rightarrow q1(B)$ $+0 \Rightarrow q1(C)$ $-5 \Rightarrow q1(D)$ $(-5,) \Rightarrow E$ $0 \Rightarrow Z_1$	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).

$$^{\alpha}\text{TLY} \quad T_j$$

$$^{\alpha}[T_j] \implies A$$

$$\text{count of ones} + [S_q D] \implies SqD$$

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		$[T_j] \implies A$ $n+q4[D] \implies q4D$ $[T_j] \implies E$	"n" is the number of ones in $[T_j]$. The addition is regular 9-bit ring addition with no overflow detection.
2.	TLY {+0}		$+\ 0 \implies A$ $+\ 0 \implies E$	The D register is not changed
3.	^{1}TLY {-0}		$-\ 0 \implies R(A)$ $18+q2[D] \implies q2D$ $-\ 0 \implies E$	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. $S \begin{array}{c} 1 \to 0 \\ 0 \to 1 \end{array} \bar{S}$ (complement of S).

Two representations of number zero $\left.\begin{array}{l} 0 = 00 \ldots 0 \\ \bar{0} = 11 \ldots 1 \end{array}\right\}$ 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
$\overline{S}$	Complement of S (sign bit complemented)	$\overline{S}$	101 100 010	000 1C0 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 1C1 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 1C1 101
n(S)	Normalizer of S (S signed fraction) $\frac{1}{2} \leq \| 2^{n(S)} \times S \| < 1$ Note: $n(0) = n(\overline{0}) = \lambda -1$. (Used as 9-bit number.)	n(S)	0	2
$\tau(S)$	Tally of S (number of ones in S) (used as 9-bit number.)	$\tau(S)$	5	6
		T	011 010 011	011 010 011
S $\wedge$ T	S and T for each bit b, b=1, 2,	S $\wedge$ T	010 010 001	011 010 010
S $\vee$ T	S or T ..., λ	S $\vee$ T	011 011 111	111 011 011
S $\oslash$ T	S or T but not both	S $\oslash$ T	001 001 110	100 001 001
S $\oplus$ T	λ-bit binary ring sum of S and T Note: $S \oplus \overline{S} \equiv \overline{0}$	$\oplus$	101 110 000	010 101 110
S $\ominus$ T	λ-bit binary ring difference = $(\overline{\overline{S} \oplus T})$	S $\ominus$ T	111 001 001	100 000 111

Enclosed expression applies to each active quarter of operand

Enclosed expression applies to each active subword of operand

A blank box indicates that no change is made.

TIME (over basic mem cycle)	i	ABBREV.	NAME	CONDITIONS	P ⑦	Q	X_i	W_n	$W_n^\#$	$W_{ni} = W_{ni} \oplus X_i$ W_{ni}	W_{ni}	Z(A)	A	B	C	D'	D	E
0	21	SPF	SPECIFY FORM ②															
	22	SPG	SPECIFY GROUP ②		$P+1$	$\lambda \oplus X_i$												④
1.2	31	FLF	FILE FORM							F_c ②								
2.8	32	FLG	FILE GROUP							G_w								
0	24	LDA	LOAD A										$W_{ni}^\#$					
	25	LDB	LOAD B											$W_{ni}^\#$				
	26	LDC	LOAD C												$W_{ni}^\#$			
	27	LDD	LOAD D														$W_{ni}^\#$	
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 ②							A			$W_{ni}^\#$					
2.4	55	INS	INSERT							(B∧A) $\vee(\overline{B}\wedge W_{ni}^\#)$								
0	41	ITA	INTERSECT A										$A \wedge W_{ni}^\#$					
	42	UNA	UNITE A										$A \vee W_{ni}^\#$					
	65	DSA	DISTINGUISH A										$A \oplus W_{ni}^\#$		$(A \wedge W_{ni}^\#)\vee C$			
	67	ADD	ADD									⑤	$A \oplus W_{ni}^\#$		$A \wedge W_{ni}^\#$		$W_{ni}^\#$	
	77	SUB	SUBTRACT									⑤	$A \ominus \overline{W_{ni}^\#}$		$A \wedge \overline{W_{ni}^\#}$			
15 11 8 5	76	MUL	MULTIPLY ⑦		$P+1$	$\lambda \oplus X_i$						0	$A \times W_{ni}^\#$		0			
73 55 37 16	75	DIV	DIVIDE ⑨ ④									⑥	$(AB/W_{ni}^\#)_{QUOT.}$	$(AB/W_{ni}^\#)_{REM.}$				
$(W_{ni}^\#)/_3$	60	CYA	CYCLE A										$\mathcal{P}^{(W_{ni}^\#)'} A$					
$(W_{ni}^\#)<9$ ⟹0	62	CAB	CYCLE AB											$\mathcal{P}^{(W_{ni}^\#)'} AB$				
	61	CYB	CYCLE B												$\mathcal{P}^{(W_{ni}^\#)'} B$			
$n/_3$	70	SCA	SCALE A	Z(A)=0								0	$2^{(W_{ni}^\#)'} A$			$\bar{0}$		
				Z(A)=1 (W_{ni}^\#)'>0									$2^{(W_{ni}^\#)'} \langle A\rangle$					
$n<9 \neq 0$				Z(A)=1 (W_{ni}^\#)'≤0									$2^{(W_{ni}^\#)'+1} \langle 2^{-1}A\rangle$					
	72	SAB	SCALE AB	Z(A)=0										$2^{(W_{ni}^\#)'} AB$			⑧	
				Z(A)=1 (W_{ni}^\#)'>0										$2^{(W_{ni}^\#)'} \langle AB\rangle$			$W_{ni}^\#$	
				Z(A)=1 (W_{ni}^\#)'≤0										$2^{(W_{ni}^\#)'+1} \langle 2^{-1}AB\rangle$				
	71	SCB	SCALE B												$2^{(W_{ni}^\#)'} B$			
$n/_3$	64	NOA	NORMALIZE A	Z(A)=0								0	$2^{m(A)} A$			$(W_{ni}^\#)'\ominus m(A)$		
				Z(A)=1 ⑦									$\langle 2^{-1} A\rangle$			$(W_{ni}^\#)' \ominus 1$		
	66	NAB	NORMALIZE AB	Z(A)=0									$2^{m(AB)} AB$			$(W_{ni}^\#)'\ominus m(AB)$		
				Z(A)=1 ⑦									$\langle 2^{-1} AB\rangle$			$(W_{ni}^\#)' \ominus 1$		
12 9 6 2	74	TLY	TALLY ⑦										$W_{ni}^\#$			$D' \oplus \sigma(W_{ni}^\#)$		
3.6	12 (REX)	SKK	SKIP ON INDEX	c<4	$P+1$		0 λ											④
					$P+2$		1 $-\lambda$											
							2 $X_i \oplus \lambda$											
							3 $X_i \ominus \lambda$											
							4-7 ⑦											
2.0	56	PMT	PERMUTE	α_6=17				W_n^α										④
		COM	COMPLEMENT	⑨				⑨ $\overline{W_n^\alpha}$										
0	20	LDE	LOAD E															$W_{ni}^\#$
1.2	30	STE	STORE E		$P+1$	$\lambda \oplus X_i$				E								
	40	ITE	INTERSECT E															$E \wedge W_{ni}^\#$
0	43	SED	SKIP IF E DIFFERS	ALL E ≠ $W_{ni}^\#$														
				ANY E ≠ $W_{ni}^\#$	$P+2$													
2.0	17	SKM	SKIP-MAKE	⑩	⑩	λ		⑩										④
-	-	·	·	·	·	·	·	·	·	·		·	·	·	·	·	·	·
0	11	RSX	RESET X				(W_n^α)											
1.2	16	DPX	DEPOSIT X					$S_0(X_j)X_j$										
1.2	14	EAX	EXCHANGE X		$P+1$	λ	(W_n^α)	$S_0(X_j)X_j$										④
.8	10	AUX	AUGMENT X				$X_j\oplus(W_n^\alpha)_b$ ⑫											
3.2	15	ADX	ADD X				$X_j\oplus(W_n^\alpha)_b$ ⑫	$X_j\oplus(W_n^\alpha)_b$										
-	-	·	·	·	·	·	·	·	·	·		·	·	·	·	·	·	·
3.2	04	JPX	JUMP ON POSITIVE X	$X_j<0$	$P+1$		⑬										(E_{91}): (E_{61}):	
	07	JNX	JUMP ON NEGATIVE X	$X_j>0$ $X_j<0$ $X_j\geq0$	λ $P+1$		$\sigma \oplus X_i$										$P+1$	
1.6	46	JPA	JUMP ON POSITIVE A	no A>0 any A>0	$P+1$												$P+1$	
	47	JNA	JUMP ON NEGATIVE A	any A<0 no A<0	$\lambda \oplus X_i$													
	44	JOV	JUMP ON OVERFLOW	no Z(A)=1 any Z(A)=1	P $\lambda \oplus X_i$												$P+1$	
1.2	05	JMP	JUMP	c even	λ												⑭	⑭
		BRC	BRANCH	c odd	$\lambda \oplus X_i$		⑭											
1.6	57	TSD	TRANSFER DATA	unit not ready	P												④	
				unit ready	$P+1$	$\lambda \oplus X_i$				⑮	⑮							

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 <u>EXA</u> has same effect as <u>STA</u>.

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

5.   <u>ADD</u>, <u>SUB</u> overflow conditions:

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

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

$$Z(A_{43}) \equiv Z(A_{42}) \equiv Z(A_{41}) \equiv Z(A_4) = Z_4$$

$$Z(A_3) = Z_3$$

$$Z(A_{21}) \equiv Z(A_2) = Z_2$$

$$Z(A_1) = Z_1$$

6.   <u>DIV</u> Conditions:

CONDITIONS		$Z(A)$	A	B
$|W_{rj}{}^*| \neq 0$	$|W_{rj}{}^*| > |AB|$	0	QUOT	REM
	$|W_{rj}{}^*| \leq |AB|$	1	JUNK	JUNK
$|W_{rj}{}^*| = 0$	$|AB| = 0$	1	$\overline{A}$	
	$|AB| \neq 0$		$\overline{A} \oslash W_{rj}{}^*$	$R^{-1} B$

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

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

7.   Exceptions in <u>MUL</u>, <u>DIV</u>, <u>NOA</u>, <u>NAB</u>, <u>TLY</u>:

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.   <u>CYCLE</u>, <u>SCALE</u>, and <u>NORMALIZE</u> instructions begin, in effect, with <u>LDD</u>.

9.   <u>PMT</u>, <u>COM</u> consist of 3 consecutive steps:

$W_{rj}{}^* \implies E$
$\overline{W_{rj}{}^*} \implies E$
$E \implies W_{rj}$
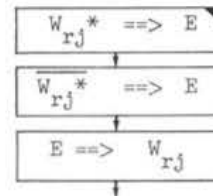
10. <u>SKM</u> variations:

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

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

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. <u>ADX</u> , <u>AUX</u> consist of sequence of steps:
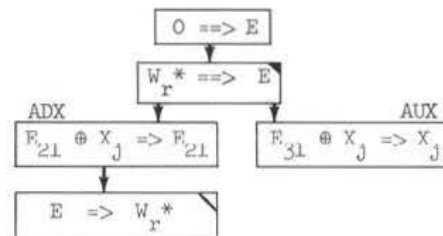
13. $\underline{c}$ is 18-bit signed integer expansion of c. $(0 \leq c \leq 37 ; -17 \leq \underline{c} \leq +17)$

14. <u>JMP</u>, <u>BRC</u> variations:

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

15. <u>TSD</u> (Unit Ready)

TABLE OF CONTENTS

CHAPTER 4

TX-2 IN-OUT SYSTEM

4-1 INTRODUCTION:

TX-2 was designed for 33 "IN-OUT" devices (see chart 7-1). Each channel is identified by its "Sequence Number" - Zero for "STARTOVER" and $40-77_{(8)}$ for "normal" channels. (Sequence Numbers are usually given in Octal.)

The basic In-Out set includes:

For Input:	Photoelectric Paper Tape Reader
	Keyboard and Reader of Lincoln Writer
	Datrac Analog Sampler
For Output:	Xerox Printer
	High Speed Paper Tape Punch
	Printer and Punch of Lincoln Writer
	Display Scopes
For Bulk Storage:	Variable Speed Addressable Magnetic Tape
	(4 units, manually selected at first, about 2 million words per unit.)

The subprograms associated with INOUT units can be written so that the waiting time for one unit is automatically used as computation time for others. Only one subprogram is in operation at any specific time, although the interleaved operation of several subprograms makes it possible for several INOUT units to be in operation simultaneously.

In a typical program, a subprogram will continue to run until it must wait for its associated unit to complete a data transfer or until it is interrupted to allow a subprogram of higher priority to run. Each subprogram has a "placekeeper" to remember where it should resume operation and an indicator ("FLAG") to tell when it is ready to run again. Since it is likely that more than one subprogram will be ready (i.e., more than one Flag will be up,) at any given time, a priority system is provided and is adjustable (by rewiring the "Priority Plugboard").

Each INOUT channel has, therefore, a "Sequence Number" (40-77 octal) for identification, a placekeeper (the correspondingly numbered index register), and a one bit register - its "FLAG" for signaling. Channel number zero is a special case in that its "unit" is the STARTOVER and CODABO pushbuttons, its "placekeeper" is the Toggle Start Point Register (TSP), and its Priority is the highest and cannot be changed. (The pushbuttons - STARTOVER and CODABO - raise Flag #0. "CODABO" also clears alarms, presets all control flip-flops, lowers all other Flags, and starts the computer. "STARTOVER" does NO MORE than to raise Flag #0.)

Sequence Numbers 76 and 77 have been reserved for non-INOUT purposes. Flags 76 and 77 must be raised and/or lowered by programmed instructions. With the standard priority plugboard, they have the lowest Priority position. (Sequence number 40 has the highest. The K register, a 6 bit FF register, holds the sequence number of the currently operating subprogram.) A hTSD using a non-INOUT Sequence No. will cycle memory one place to the left.

4-2 TX-2 INOUT JARGON

4-2.1 SEQUENCE - SUBPROGRAM - PROGRAM

TX-2 is indeed a "Multiple Sequence" or "Multiple Subprogram" machine. This is to say that it can interleave subprograms - i.e., it can keep track of several interleaved program sequences. This does not say that it can run several interleaved independent programs. So much colusion and cooperation would be required to interleave unrelated programs that they should probably be done by the same person. One could then argue that the result would be better described as a multi-purpose program.

The word "Sequence" is often used as a synonym for "Inout Channel". Sometimes it refers to "Sequence Number". (We often say "Sequence" 77 rather than "Sequence Number" 77). And it is used in the "normal" sense - i.e., "subprogram".

4-2.2 PLACEKEEPERS, PROGRAM COUNTERS, AND THE P REGISTER

The placekeepers - all 33 of them counting #0, (the Toggle Start Point,) - are memory devices whose purpose is to remember where each subprogram is to resume operation when it gets a chance. Placekeepers $40\text{-}77_{(8)}$ are index registers. Placekeeper "ZERO" is the Toggle Start Point register (TSP) (a row of toggle switches on the computer console).

The P register is an 18 bit flip-flop control register that always holds either the location number of the current instruction or that of the next instruction. It corresponds to the "program counter" or "instruction counter" of other machines.

Index Registers 40-77, the placekeepers, are often called the "program counters". Occasionally the P register is called "The program counter".

4-2.3 SELECT, CONNECT, TURN ON

To "connect", or "Turn on" an INOUT unit means to set the control flip-flop of the channel so that data can be transferred, and so that the INOUT unit has access to its Flag. The unit is said to be "connected to the computer". Each regular INOUT unit has a "C" flip-flop - and a corresponding console indicator - to show whether it is "connected" or not.

The word "select" is often used as a synonym for "connect" but it is also more or less reserved for the day when two or more units must share the same channel. This will be true, for example, in the magnetic tape bulk storage system.

3 TX-2 INOUT CONTROL LANGUAGE

TSD - "Transfer Data" and IOS - "INOUT SELECT" are the only INOUT operations. The channel used for data transfer depends on the "sequence number" in use rather than the unit connected, for many units may be connected, but only one subprogram is in operation at the time a given data transfer is initiated.

Control of the <u>interleaving</u> - not strictly an INOUT function is done through:
    The hold bit(#4.9), a syllable of every instruction,
    Resetting placekeepers via X Memory operations, and
    Drop out - permanent or temporary. (See 4-3.4)

4-3.1 CHANGE OF SEQUENCE NUMBER

A change of sequence number occurs <u>whenever</u>:

a)   A high Priority INOUT channel takes over by "BREAKING" or interrupting a lower priority subprogram.

b)   A subprogram drops out (either permanently, or to wait for its unit to get ready for another data transfer) <u>and</u> a lower priority subprogram takes over. If no other subprogram is ready, no change of sequence number occurs. The computer goes into "LIMBO", a condition where it repeatedly scans all the Flags until one is up. If the same old Flag (as indicated by the K register) comes up, no change of sequence occurs.

When a change of sequence number occurs, several internal registers are affected:

	OLD #	NEW #	SET FROM THE P REGISTER

The E Register $\Longrightarrow$

Old Placekeeper $\Longrightarrow$ Reset from the P register same as the right half of E. This will be "p+1" (one more than the location number of the last instruction) unless the last instruction changed P directly. (E.g., by SKX, SKM, JMP, JNA, JPA, JOV, JMP, SED, JNX, JPX, or TSD) TSD will leave "p" rather than "p+1" if the data transfer can not take place.

P Register $\Longrightarrow$ Set from the new placekeeper.

K Register $\Longrightarrow$ Set to the new sequence number.

Note that the current placekeeper is changed only when the sequence number is changed. It can therefore be used as an ordinary index register while its subprogram is in operation.

4-3.2 THE HOLD BIT

A typical INOUT subprogram is usually written so that it can be interrupted at any time by another subprogram of higher priority. To do this completely, one would have to refrain from using the Arithmetic Unit and the E register. Since this is too severe a restriction, the "hold syllable" or "hold bit" is provided. A hold bit insures that no "break" or interruption will occur following the completion of the held instruction.

A break can occur before a "held TSD", but only when the INOUT unit is unable to handle the data transfer. (This is called "DISMISS and WAIT".)

Since instructions using the E register must nearly always be held, the assembly program automatically inserts the hold syllable. (LDE, ITE, and JPX, JNX.) (JPX and JNX are included because their automatic dismiss is usually not wanted. The hold syllable cancels "dismiss" whether built in (as in TSD, JNX, JPX) or programmed (as in ^{20}SKX, ^{20}IOS, ^{20}JMP)).

4-3.3 START POINTS

To start a subprogram we need only set its placekeeper to the starting place and raise its Flag. If the computer is running, the subprogram will start as soon as it has highest priority among those that are ready.

"Starting" is particularly easy for sequence number zero. Its placekeeper, "TSP", is set by hand. If the computer is running or in "LIMBO" the STARTOVER push-button will suffice. Flag zero will go up and a change of sequence number to #0 will occur as soon as an instruction is performed that has no hold bit (or when a hTSD that can not be initiated is encountered). CODABO is used when the computer is not running, or when the user wants to stop all other subprograms and start subprogram #0 only.

A subprogram using sequence number zero has highest priority and therefore can not be interrupted. Sequence number zero is used primarily to start other subprograms. This amounts to setting placekeepers for the others and raising the Flags of those that should start. The following operations are used:

<div style="margin-left:3em">

For setting placekeepers: RSX, SKX - i.e., the instructions normally used to change the X Memory.

For raising Flag "F": ^{10}SKX$_F$ or IOS$_F$ 50 000

For permanent Drop Out: The dismiss bit (4.8) - a syllable of SKX, JMP, and IOS only. The built in Dismiss feature of TSD, JNX, and JPX can also be used for permanent drop out.

</div>

Note that the single instruction "^{30}SKX$_\alpha$ 101" (in sequence zero) would start the subprogram that is at 101 operating under sequence number "α". (Providing, of course, that α is not zero.) In fact, the ^{30}SKX$_\alpha$ 101 will work from any sequence number other than α. (It can not be made to look like "JMP 101".)

4-3.4   DROP OUT - PERMANENT AND TEMPORARY

When a subprogram is finished, it can drop out permanently through the DISMISS syllable (bit 4.8) of IOS, SKX, or JMP. When TSD has initiated an output data transfer or when it has completed an input data transfer the built in dismiss will cause drop out if "hold" was not used. This drop out will be temporary - the INOUT unit will raise the Flag. For input units the Flag is raised when the next datum is ready (e.g., when the next key is pressed or the next line of tape comes up). For output units the Flag is raised when the data transfer is complete and the unit is ready for another (e.g., when the character has been printed, or the paper tape has been punched).

Drop out always lowers the current Flag. It is considered "temporary" if the unit is about to raise the Flag and "permanent" if the Flag will be raised by another subprogram (or if the subprogram is finished for good). Temporary drop out can also occur when a TSD operation is not possible - i.e., when an output unit is still busy

or when there is no datum available from an input unit (e.g., when the next line has not yet arrived). This form of temporary drop out is called "DISMISS and WAIT" and can not be prevented by using the "hold bit". In this case, the TSD that caused the drop out has not been done, the P register is not advanced, and the TSD is done when the subprogram resumes operation.

4-3.5 THE IOS OPERATION - "INOUT SELECT"

The primary functions of IOS are "Connection" and "Disconnection" of INOUT units, and the specification of operating modes. Some units have several modes - for example, the user has the option of punching tape with or without a 7th hole on each line. IOS is also used for raising and lowering Flags and will eventually be used for selecting mag tape drives.

The basic IOS operations are:

IOS_J 20 000   -   Disconnect Unit "J" from the computer

IOS_J 3XXXX   -   Connect Unit J (if not already connected), and set to Mode XXXX

IOS_J 40 000   -   Lower Flag J

IOS_J 50 000   -   Raise Flag J

IOS_J 60 XXX   -   Select Unit XXX (Not used yet)

IOS_J 20 000, Disconnect, has no effect on interleaving except that a TSD that tries to initiate a data transfer will not be performed. (A "Dismiss and Wait" will occur - waiting for the unit to be ready to transfer the data. In most cases this amounts to a permanent drop out.)

IOS_J 3XXXX, Connect, has one peculiarity. It will raise Flag J whenever:
                        Unit J is an OUTPUT unit,
                and Unit J was not already connected.
When a mode change takes much time, the unit involved will generate a raise flag signal to indicate that the change has been made, and no data transfer will be accepted during the intervening interval.   (i.e., the "buffer is busy".)

IOS_J 40 000, LOWER FLAG J, is not equivalent to drop out if J is the current sequence number. The subprogram currently in operation will continue to run until it drops out or until it is interrupted by a unit of higher priority. If such a BREAK occurs, the interrupted subprogram will not resume operation, for Flag J is indeed lowered. If J is not the current sequence number, IOS_J 40 000 prevents

subprogram J from resuming operation until Flag J is raised somehow - (perhaps by
unit J or by another subprogram).

IOS_J 50 000, (and $^{10}SKX_J$) will <u>raise FLAG J</u>, but as before, "J = current sequence"
is a special case. Note that:

$$\begin{array}{ll} SKX_J & Y \\ ^{20}IOS_J & 50\ 000 \end{array} \quad \text{or} \quad ^{30}SKX_J \quad Y$$

Will change the P register and therefore be similar to a JMP if J is <u>not</u> the current
sequence number. But if J <u>is</u> the current sequence number, no change of sequence
number is ordered and the RAISE FLAG cancels the DISMISS. There is effectively no
change. (Except that $^{30}SKX_J$ Y will set X_J to "Y" but this will be wiped out by the
next change of sequence number.)

The Flag of the <u>current sequence</u> is never used. Following each instruction
that is <u>not held</u>, control scans the Flags having higher priority but goes no further.
It does not consider the current Flag. If the instruction <u>was held</u> no scan is made
at all. When a subprogram drops out, <u>all</u> the Flags are scanned until a raised Flag
is found. When no Flags are up, and this scanning is taking place, the computer is
in "LIMBO". As soon as a Flag is found, a change of Sequence Number (see 4-3.1)
takes place and normal operation is resumed.

4-3.6 THE REPORT BIT

A simple IOS has no effect on the E register. If bit 4.4 is set to 1, (a
$^{1}IOS_J$ 0 for example) the control flip-flops of the chosen unit are copied into E
<u>before</u> the rest of the instruction is performed. Thus, if $^{1}IOS_J$ 3XXXX is used, E
will contain information on the state of affairs <u>before</u> the mode change. Unused
portions of E are cleared.

The standard report is as follows:

Bit 3.1 to 3.6	-	Sequence Number of Reporting Unit
" 2.9	-	Flag
" 2.8	-	Buffer Status - 1 = not busy
		0 = busy
" 2.7	-	Maintenance
" 2.6	-	Connect
" 2.5	-	EIA - Equipment Inability Alarm
" 2.4	-	MISIND - Missed Data Indicator
" 2.3 - 1.1	-	Mode flip-flops - same as in the IOS 3XXXX for **most** units.
" 3.7 - 4.9	-	Special indicators - cleared if not used.

4-3.7   TSD - TRANSFER DATA

With a few exceptions (41, 42, 55, 75) each INOUT unit has an INOUT Buffer Register (IOB) and a Status FF. STATUS = 1 means it is the computer's turn to use the buffer, STATUS = 0 means that the "BUFFER is BUSY" - i.e., the unit is working on an uncompleted data transfer. The buffers range in size from 6 to 24 bits. TSD - Transfer Data - means either "copy from IOB_k to memory" or "copy memory to IOB_k" where k is the current sequence number (i.e., contents of the K register). Thus for input units, TSD completes the data transfer and for output units, TSD initiates the data transfer. (For input, the transfer is "unit-to-buffer," then "buffer-to-memory"(via TSD) and for output it is "memory-to-buffer"(via TSD), then "buffer-to-unit".)

Except where TSD is used in ASSEMBLY mode, permutation and activity can be used in the normal manner. There is no sign extension - subword form is ignored. "Inactive" portions of an output buffer are filled with +0. The buffer is considered to be at the far right unless otherwise stated in the unit descriptions.

TSD has two built-in DISMISS features. If the buffer is busy, the TSD can not be performed and drop out occurs whether a hold is used or not. This is called "dismiss and wait" and comes before the P register index point in the control cycle. (P is not advanced.) Once the TSD operation is done, the other built-in DISMISS occurs but this time "hold" is effective. Such a hold is used on input devices to insure use of the new datum as soon as possible and on output devices to utilize the processing time without changing the sequence number. It is possible in either case to use so much time that lower priority subprograms never have time to operate.

If an INOUT unit is not connected, a TSD will find the buffer "busy" and "Dismiss and wait" will occur. If the unit is subsequently connected by another subprogram, the flag of the first will be raised and the TSD will be performed as soon as normal interleaving will allow.

If a TSD is done using sequence number 0, 76, or 77, the specified memory word will be cycled left once. The configuration syllable is not used - the cycle is a full 36 bit operation. Unless a "hold" was used, the automatic dismiss syllable of TSD will take effect. (This is also true for sequence numbers for which there is no INOUT unit as yet.)

Note that for sequence number 75 (Miscellaneous Outputs), TSD does not cycle. (It will still dismiss if not "held", however.)

IOS

^{0}IOS$_{J}$ 0 — Has no effect except to take time. (But note that ^{1}IOS is "Report".)

^{0}IOS$_{J}$ 20000 — Disconnect Unit J

^{0}IOS$_{J}$ 3XXXX — Connect Unit J, Set Mode, Raise Flag J if Unit J was a disconnected Output Unit.

^{0}IOS$_{J}$ 40000 — Lower Flag J - Not DISMISS, (i.e., will not cause drop out.)

^{0}IOS$_{J}$ 50000 — Raise Flag J

SKX

^{10}SKX$_{J}$ N — Raise Flag J, Set X$_{J}$ to "N".

^{20}SKX$_{J}$ N — DISMISS, Set X$_{J}$ to "N".

^{30}SKX$_{J}$ N — Both of the above for J ≠ k. If J = k, there is no drop out. (k = current sequence number.)

DISMISS

Bit 4.8 for IOS, JMP, SKX (e.g., ^{20}SKX, ^{20}IOS)

"Built in" as part of TSD, JNX, JPX

(Otherwise not available.)

Change of Sequence Number Affects:

Register E → | OLD # | NEW # | CONTENTS OF P |

Old Placekeeper → Contents of Register P

Register K → New Sequence Number

Register P → Contents of New Placekeeper X

Report - ^{1}IOS - (bit 4.4 of IOS)

*Not Used, Except by Magnetic Tape.

Sequence Number of Reporting Unit.

2.4 - MISIND
2.5 - EIA
2.6 - Connect
2.7 - Maintenance
2.8 - Buffer Status (0 = busy)
2.9 - Flag

*2.3 - 1.1 Mode Bits

* Register E is cleared before the report. Therefore, all un-used bits are zero.

If a program uses but one unit there is no need to interleave any subprograms and the entire program can be performed using one sequence number. Even if two or more units are to be used, it is sometimes better to use them one after the other rather than simultaneously. If the above conditions are true, the only pitfall that may be overlooked is premature drop out. Careful use of the dismiss bit and built in dismiss features will prevent this error.

Interleaved operation of subprograms requires sharing the following:

> Main Memory
> X Memory
> F Memory
> Arithmetic Element (A,B,C,D, and Overflow FF).
> TIME

(Listed in order of increasing difficulty)

Main Memory and X Memory must usually be partitioned, except, of course where they are used for common data. The F Memory can usually be set at the start to some "Standard Configurations" and left unchanged. Two approaches to sharing TIME and AE are given below.

4-4.1   BRUTE FORCE HOLDING

Whenever the INOUT units involved are slow enough, or are not free-running (i.e., do not dictate timing) a brute force method may be used. (The Lincoln Writer Printer and the High Speed Punch are two such units.) The lower priority subprograms can use a hold bit on all instructions where a break is intolerable, (assuming a BREAK will change the Arithmetic Element). The only limit is that they can't hold on _all_ instructions. The highest priority subprogram has no problem other than the fact that it must drop out now and then to give the others a chance. If it must wait for a lower unit it can do so by dropping out and relying on the other subprogram for restarting. Synchronization can be automatic only if the high priority loop contains a temporary drop out. The easiest way to obtain a temporary drop out is through regular and, if need be, dummy TSD operations (e.g., non-printing keys on the Typewriter, blank tape on the punch, etc.). Another method would be to use the Interval Timer (Unit #54).

4-4.2   HIGH—LOW—MEDIUM PRIORITY SUBPROGRAMS

The Brute Force Holding method will not work if the timing of one unit requires that it receive attention soon after its Flag comes up. In many cases it is necessary to restrict holding to no more than three consecutive operations. Fortunately, the index memory operations can be used in place of the Arithmetic Element operations for many applications. This means that all but the lowest priority subprograms do not need the Arithmetic Element. The rules for this method are as follows:

For Lowest Priority Subprogram - The only one to use the Arithmetic Element.

a) No more than "n" consecutive holds.[*] (It should be possible to limit this lowest priority subprogram to holding only on JPX or JNX.) ("Consecutive", here refers to TIME, not storage.)

For Medium Priority Subprograms -

a) No more than "n" consecutive holds.[*]

b) No use of the Arithmetic Element

For Highest Priority -

a) No use of the Arithmetic Element unless it is saved and restored.

b) "Hold" should be needed only on JNX, JPX, and TSD. In other places, it has no effect. When used on TSD, care should be taken to insure that some time remains for other units.

---

[*]"n" the number of permissable consecutive "holds" is determined by the timing requirements of the highest priority subprogram. "n" = 3 is enough to allow considerable flexibility in the other subprograms.

IN-OUT ALARMS

DESCRIPTION:

This "device" enables operation of an Alarm Subprogram whenever an IN-OUT alarm occurs. FLAG 41 is raised upon EIA (Equipment Inability) or MISAL (Missed Data) for any of the devices listed below. TSD is used to determine the type and source of the alarm.

MODE SELECTION:

IOS_{41} 30000	Connects alarm circuitry to central computer. FLAG 41 will now be raised upon alarms. TSD will now report alarming conditions. "MISAL" (Alarm) is suppressed. See Note 1 below.

TSD INSTRUCTION:

TSD T_j, ‖α OR ^{α}TSD T_j		TSD does not clear alarm. The offending unit must be disconnected (IOS 20000). See Note 3. TSD copies IOB_{41} into T_j.

BUFFER BIT ALLOCATIONS:

Bit	Corresponding Octal Integer	In-Out Alarm	
1.1	001	MISAL - Datrac	50
1.2	002	MISAL - PETR	52
1.3	004	MISAL - Mag. Tape	46
1.4	010	EIA - Mag. Tape	46
1.5	020	EIA - Camera	60
1.6	040	EIA - Punch	63
1.7	100	EIA - Xerox	51
1.8	200	EIA - Lincoln Wtr.	65,66
1.9	400	EIA - Lincoln Wtr.	71,72

NOTE: 1. An unsupressed "MISAL" will stop the computer. The two forms of supression, program and manual, are independent - both must be off to remove the supression. Programmed supression does not light the yellow console light, but the red light and gong still work.

2. EIA - "Equipment Inability Alarm" - does not stop the computer but it may ring a buzzer or stop the unit involved.

3. If an additional alarm is generated before the first has been cleared, IOB_{41} will be set but FLAG 41 will not be raised. TSD can be used again to see if this has occurred. Note that an IOS 30000 following IOS 20000 will raise FLAGS 60, 63 and 51, but not FLAGS 50 or 52. FLAG 46 is a special case. TSD should be used before disconnecting the offending unit for it can not report conditions of units that are not connected.
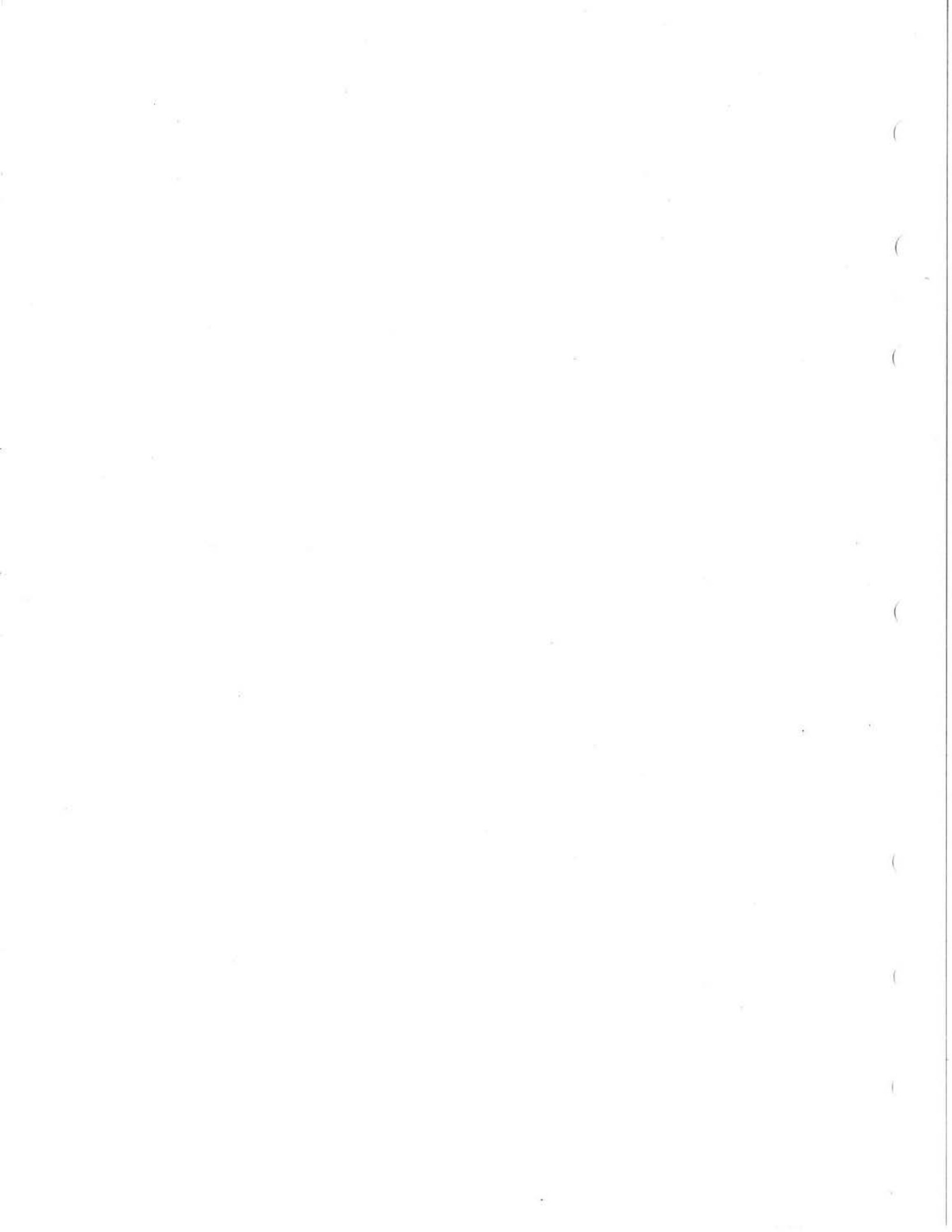
TRAP

The TRAP circuits can be set to raise FLAG 42 (thereby starting a special subprogram) whenever a metabit is encountered in the operation of other subprograms. FLAG 42 can also be raised on change of sequence number or upon a signal from the TX-2 Sync System. The circuits can also set metabits. Since metabits can be encountered in 3 basic ways, there are several TRAP modes. See below. TSD is not used (but retains its cycle left and dismiss features). Combined modes are allowed. For example, IOS₄₂ 30007 will set to trap on all metabits encountered, whether by instruction, defer cycle, or operand.

MODES

(Programmed)   (All Pushbuttons OFF.)

IOS$_{42}$ 20000 or IOS$_{42}$ 30000	Clear	Clears Mode Selection
IOS$_{42}$ 30001	Trap on Marked Instruction	FLAG 42 is raised before the end of the marked instruction.
IOS$_{42}$ 30002	Trap on Deferred Address	FLAG 42 is raised before the end of the instruction using a marked deferred address.
IOS$_{42}$ 30004	Trap on Operand	FLAG 42 is raised after a delay of one to several instructions, depending on overlap conditions.
IOS$_{42}$ 30010	Trap on Change Sequence	FLAG 42 is raised during change sequence cycle if new Place Keeper (index register) is marked (2.9 = 1). The "new" (marked) sequence number goes into quarter 3 of the E Register, and the "old" into quarter 4. There is no trap when leaving number 42.

The three "set metabits" modes below are partly manual in that the "set metabits" pushbutton switch on the console must be "ON". These modes do not raise FLAG 42.

IOS$_{42}$ 30100	Set Metabits of Instructions	Sets metabit of all instructions performed.
IOS$_{42}$ 30200	Set Metabits of Deferred Addresses	Sets metabit of all deferred addresses used.
IOS$_{42}$ 30400	Set Metabits of Operands	Sets metabit of all operands used.

MANUAL MODE: (Trap on Sync System Signal)

FLAG 42 can be raised by a signal from the Sync System. The requirements are:

1. "Sync 1" and "Sync 2" pushbutton switches should be OFF.

2. The "Sync to Trap" pushbutton switch should be "ON". While it is "ON" all other trapping modes are not effective. Setting modes still work. When "Sync to Trap" is turned "OFF", the original mode is reinstated.

3. "Gate 1 to Sync Jacks" pushbutton switch should be ON. (This is located on the Sync System Panel.)
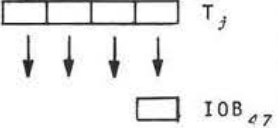
MISCELLANEOUS INPUTS

Nine one-bit independent input channels - each with a BCN jack and an ON-OFF toggle switch are provided. A standard TX-2 transition from -3V to ground will set the chosen buffer digit and raise FLAG 47. The buffer is cleared upon copying into memory via TSD. Two Schmidt Triggers with filters are provided on the panel itself, and a 3 channel push-button pulse generator is also available as a separate, movable unit.

MODE SELECTION

IOS_{47} 30000	CONNECT	This allows inputs to raise FLAG 47. It does nothing else.

TSD

TSD T_j ‖α OR $^{\alpha}$TSD T_j	▭▭▭▭ T_j ↓ ↓ ↓ ↓ ▭ IOB_{47}	TSD reads IOB into T_J and clears IOB. Permutation is operative - there is no sign extension - quarter 1 must be active - activity of q2, 3, 4 is not relevant.

MANUAL CONTROLS:

Filter switches

Schmidt Triggers

Three channel push-button pulse generator

Toggles ("on-off" for each.)

BCN Connectors - (Standard TX-2 Inout Transition)
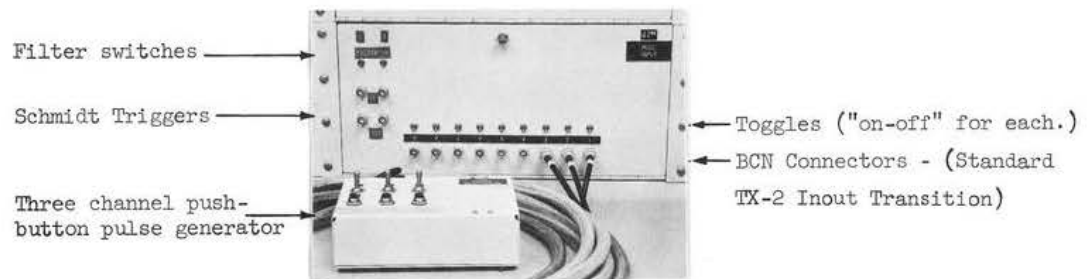
Notes: 1. The input signal must be a "Standard TX-2 Transition" (i.e. from -3 volts to ground with a rise time less than 0.2 microseconds).

2. The Schmidt triggers are completely independent of the rest of the circuitry and must be cabled to the channel desired. The input to the Schmidt trigger must be a smooth transition from ground to -3 volts. Since the normal open circuit voltage is about -3 volts, a sine wave of about 5 volts RMS, or an opening switch contact can be used. The filter should be used with the switch contact input or with any other noisy source. The Schmidt trigger inverts the input signal producing a standard TX-2 Inout Transition as its output. (The circuit switches at -0.9 volts going down and at about -2.2 volts going up. The rise and fall time is about 0.15 microseconds.)

July 1961
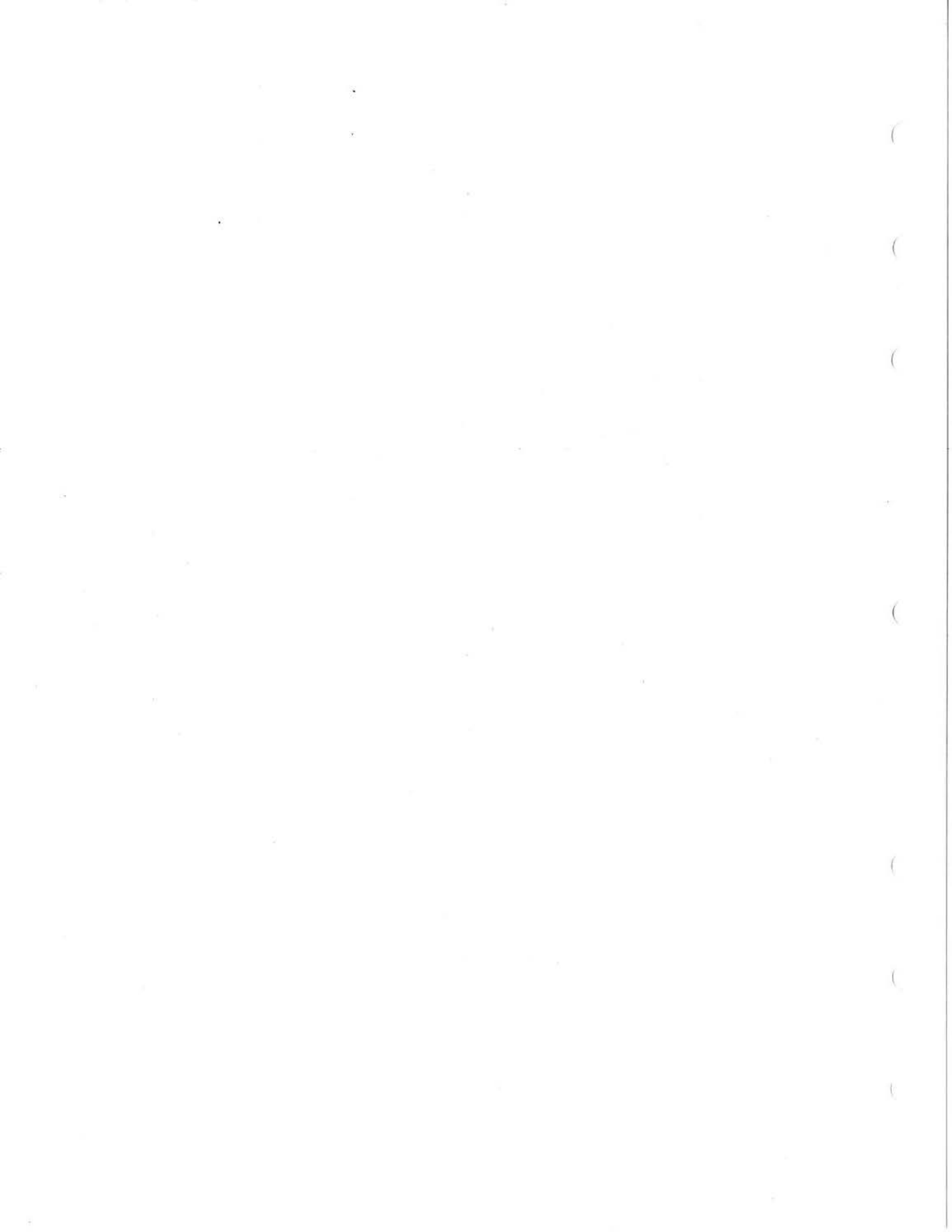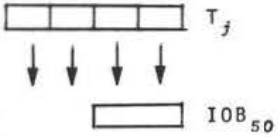
DATRAC - Analog to Digital Numerical Input

The DATRAC is an analog to digital converter made by Epsco, Inc. It provides numerical samples of a continuous electric signal. A measurement or sample is started upon receipt of a "trigger pulse" from the computer or from an external source. (Such as the Interval Timer - See No. 54.) Pertinent parameters are as follows: (and see notes below.)

Maximum Sampling Rate:	27 Kilocycles (37 usec per sample)
Measuring Time (Trigger to Raise Flag):	22 usec.
Nominal Input Signal:	-1 volt to +1 volt (can be set to ± 10 volt or ± 100 volt behind the panel.)
Output	Signed 18 bit Ones Complement Fraction

OPERATIONS

IOS_{50} 30000	CONNECT	IOS_{50} 30000 permits FLAG 50 to be raised and sends a trigger pulse to the datrac control panel, (where it may be switched to the DATRAC, or not as desired by the user.)
TSD $T_j \parallel \alpha$ OR $^{\alpha}$TSD T_j		TSD copies an 18 bit signed ones complement fraction into T_j along permuted pathways if so specified. The reliable precision is, however, only 8 to 11 bits - (at the left end). There is no sign extension in T_j. TSD also sends a trigger to the DATRAC control panel (where it may be switched to the DATRAC or not as desired.)

NOTE: 1. Do not trigger the Datrac more often than at 37 usec intervals. It is possible to damage the circuits.

2. TSD copies the measurement taken at the time of the last trigger.

3. A MISAL is created when a trigger arrives at the Datrac before the previous sample has been transferred to the computer via TSD. (See IN-OUT #41.)

MANUAL CONTROLS:

DATRAC Control Panel

1.   Trigger Inputs (Standard TX-2 Inout
2.   Transition. i.e. -3
3.   to ground in less than
4.   0.2 microseconds.)

Osc.   Oscillator Inputs (3 volts RMS required)

Osc.

TSD   "Up" connects programmed triggers to

IOS   DATRAC. These are internally connected.

General Radio Oscillator (Not an integral part
of the DATRAC system.)

DATRAC Signal Input. (Cannon Connector XL311
required. Pin 1 - Shield, pin 2 - Ground,
pin 3 - Signal.)

"External Trigger Input" This trigger input is
not compatible with TX-2. Use the inputs cn the
"Control Panel" above. They are internally
connected to this input.

DATRAC Power Supply The Datrac power is not
left ON. A warm-up time of from 5 to 30 minutes
is required. (30 minutes is enough to insure that
it is as stable as it will be.) Users should be
sure to turn the power OFF when they are finished.

XEROX PRINTER

The XEROX is an electrostatic, high speed (960 lines per minute) printer. It is basically a charactron display with automatic continuous xerographic recording. Through electronic compensation, the display area or frame is held "stationary" for about 45 milliseconds, and then moved down about 0.1 inch to catch up with the paper. The programmer must specify the x,y position as well as the code number for each character to be printed.

OPERATIONS

IOS_{51} 30000	CONNECT	"Connect" turns on the xerographic recording apparatus and raises FLAG 51 when the equipment is ready. (Warm up time is about 5 seconds.)

IOS_{51} 30010	FRAME SYNC	If the unit is not connected, "Frame Sync" will "connect" it. FLAG 51 is raised at the start of the next Frame period. (Frame period is 45 milliseconds.)

TSD T_j ‖α OR $^\alpha$TSD T_j		TSD causes one character to be printed. Since TSD takes about 750 usec, only 60 characters can be printed within one FRAME interval. See diagram below for Xerox buffer layout.

BUFFER LAYOUT

X Position   Y Position   Character Code

Notes:

1. The "Frame Area" is a rectangle approximately 5 by 1 1/4 inches.

2. The "Origin" is at the left end, centered vertically.

3. The X position is given by a 9 bit positive integer. (000 to 777_8)

4. The Y position is given by a 6 bit ones-complement signed numeral. (-37 to $+37_8$)

5. The First TSD starts the paper motion. The paper will continue to move until 15 seconds after the last TSD or until 15 seconds after the Xerox is disconnected via IOS_{51} 20000.

6. A TSD that must start up the paper drive is <u>unpredictable</u> due to noise generated by the paper mechanism. The safe procedure is to print one or two "blank" characters (e.g. code 100) and an extra Frame Sync (IOS_{51} 30010) to ensure that the paper is moving smoothly when the data is displayed.

July 1961

Numerical Parameters:

Frame Interval	-	45 milliseconds (approximately)
Frame Reset Interval	-	2 milliseconds
Character Print Time	-	750 microseconds
Frame Size	-	1 1/4 inches high by 5 inches wide
Character Size	-	Nominally 3/32 high by 1/16 wide
Paper Speed	-	2 inches per second

Calculated Parameters:

Characters per line		60
Vertical deflection		.02 inches per unit
Horizontal deflection		.01 inches per unit
Frame spacing	=	.094 inches
Lines per inch	=	10.65
Suggested X increment	-	8 units

MANUAL CONTROLS

ON OFF
SWITCH

Low paper buzzer alarm
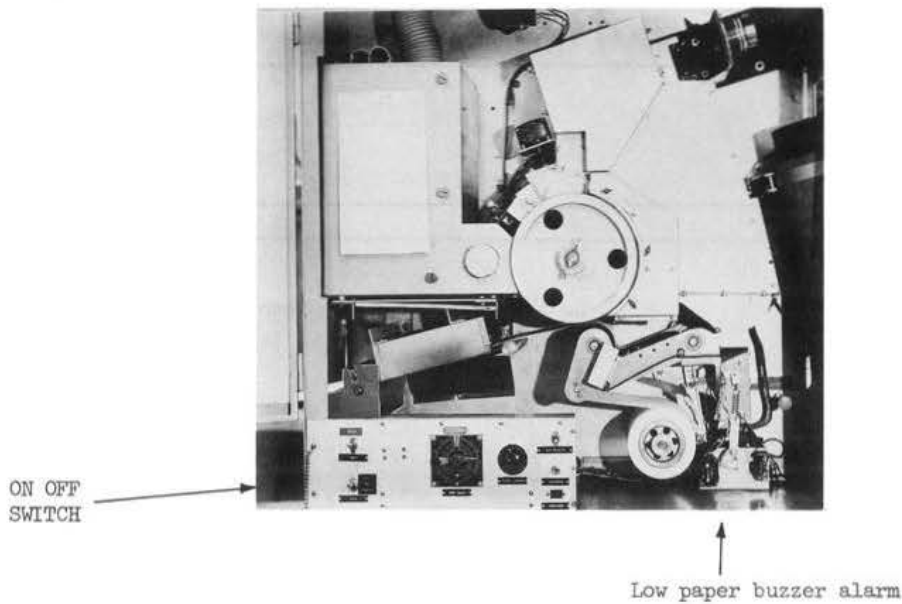
Note: "Low Paper" alarm causes an EIA indication and can raise FLAG 41 if IO alarm circuits are "connected". See IN-OUT Unit 41.

XEROX PRINTER CHARACTER CODES

CHARACTER		OCTAL CODE		CHARACTER		OCTAL CODE	
A		154		j		122 (107)	
B		142		h		324 (034)	
C		361 (056)(071)(346)		n		323 (033)	
D		352		p		024	
E		313 (043)		q		111	
F		344 (054)		t		112	
G		302 (012)		w		173	
H		354		x		174	
I		172 (157)		y		163	
J		144		z		164	
K		143		α		310 (040)	
L		332 (047)(062)(317)		β		311 (041)	
M		360 (055)(070)(345)		γ		333 (063)	
N		370 (355)		Δ		203	
O		353		ε		334 (064)	
P		312 (042)		λ		023	
Q		160 (145)		1		001	
R		371 (356)		2		002	
S		322 (017)(032)(307)		3		003	
T		153		4		004	
U		362 (057)(072)(347)		5		020 (005)	
V		152		6		021 (006)	
W		343 (053)		7		022 (007)	
X		161 (146)		8		300 (010)	
Y		342 (052)		9		301 (011)	
Z		162 (147)		0	(ZERO)	000	
h		132 (117)		?		202	
↓		133		,	(COMMA)	204	
<		220 (205)		∩		120 (105)	
>		221 (206)		⊃		121 (106)	
•	(PERIOD)	222 (207)		×		113	
+		351		○	(CIRCLE)	714 (444)	
−		372 (357)		~		373	
∨		340 (050)		∧		341 (051)	
'		363 (073)		#		364 (074)	
|		730 (445)(460)(715)		‖		731 (446)(461)(716)	
Σ		703 (413)		□		704 (414)	
}		720 (415)(430)(705)		{		721 (416)(431)(706)	
☞		150		→		151	
_		570 (555)		—		571 (556)	
(		140		)		141	
=		114		≡		130 (115)	
/		131 (116)		∗		102	
∪		103		⊂		104	

Note: Bit 1.9 of the Xerox Character Code is a "size control bit". "1" means large, and "0" means small. The codes are given above with the "proper" size.

MANUAL CONTROLS
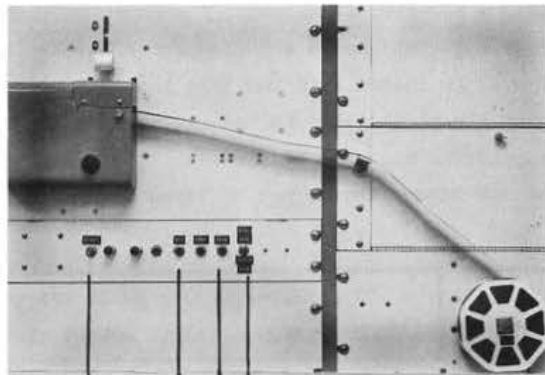
Note: The manual control pushbuttons discon-
nect the PETR. (This is equivalent to the $IOS_{52}20000$ instruction.)

Reel/Strip Toggle Switch - Selects the drive for tape motion in the "Reel" direction for both manual and computer operation.

  "Reel" - Up to 2000 lines per sec. Drive provided by the reeler.

  "Strip" - About 400 lines per sec. Drive provided by the internal capstan.

Wind Pushbutton - Starts tape movement in the "Reel" direction. Countermands any computer originated operation. Tape is not read by computer.

Bin Pushbutton - Starts tape movement into the bin, at about 400 lines/sec via internal capstan drive. When the "End Mark" (Code 73, no 7th.) is encountered tape motion is stopped with the mark on the bin side of the reading point. Any computer originated operation is countermanded. Tape is not read.

Maintenance Switch

TAPE DIAGRAMS

(Read toward PETR, inside to outside.)

6 7 1 0 0 1 3 0 2 2 0 6

Inside Edge

Outside Edge

7th Hole

End Mark

0 1 1 1 0 0

To the PETR

"Normal" (One line)

(Read toward the 7th. hole - i.e. From inside to outside.)

"Assembly" (Six lines)

To the Reeler
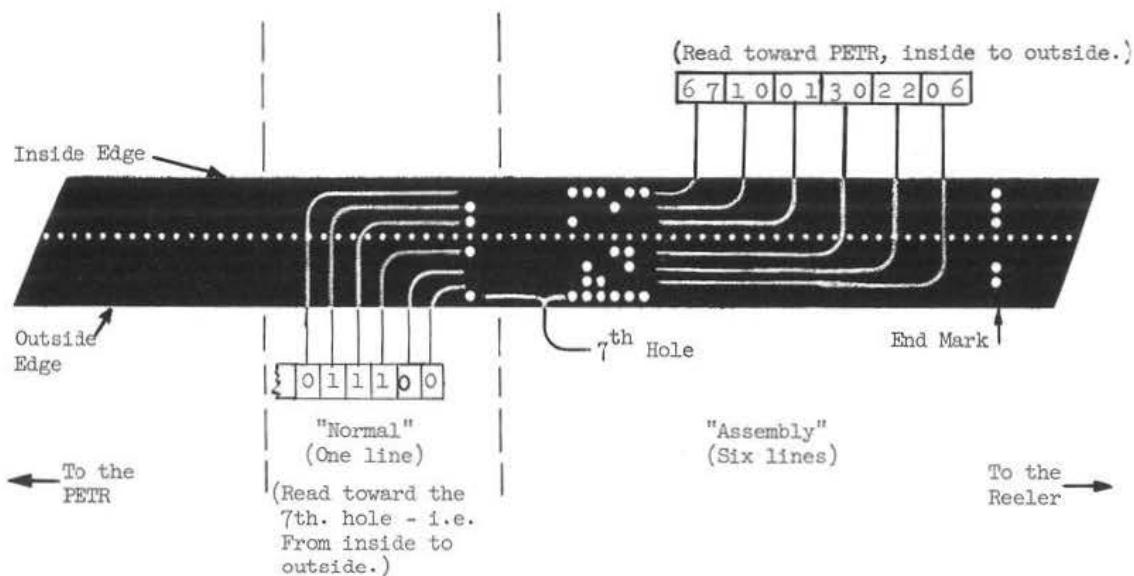
INTERVAL TIMER

The INTERVAL TIMER is essentially a counter that passes every n^{th} pulse of a pulse oscillator (the "End Carry Pulse"). The basic counting rate, timed interval, start time, and stop time are controllable. The output, a string of accurately spaced pulses, can be used to raise FLAG 54, and (or) to trigger an external device (such as the DATRAC for example). Control is partly manual, partly by program.

OPERATIONS

IOS_{54} 30000	STOP COUNTER (and Connect)*	STOPS the counter and hence the output string. The count for the interval will now be reset repeatedly from the buffer (at the counting rate). (Counting rate is selected manually.)
IOS_{54} 30100	START COUNTER (and Connect)*	STARTS the counter. The pulse string will start after one counted interval and will continue until it is stopped. In this mode, the string is available only at the "EC OUTPUT" jack on the console.
IOS_{54} 30200	SET TO RAISE FLAG 54 (and Connect)*	CONNECTS output string to raise FLAG 54 at the end of each timed interval. This mode is used when the interval timer is to be started by hand or by an external trigger.
IOS_{54} 30300	START and RAISE FLAG (and Connect)*	This is a combination of the two operations just above. The first output pulse and raising of FLAG 54 come after one interval as specified by the Buffer. (The buffer can be set manually from toggles, or by a TSD in the program.)

*All IOS_{54} 30000 instructions "connect" the unit if it is not already connected.

Notes: 1. The buffer is "busy" during "end carry time" - i.e. when it is in use. For 10 kc., 100 kc., and "Ext. Osc." this is equivalent to the basic counting interval. For 1 mc., the buffer is "busy" during the last 16 counts. If the reset value is less than 16, the counter must be stopped before the buffer can be changed.

2. Any change in the buffer becomes effective only at the end of the current interval unless the change is made with the counter stopped.

3. Manual control overrides program control.

SCOPE DISPLAY

The "scope" is a cartesian coordinate, high speed (20 to 80 usec) display with 10 bit precision in (x, y), controllable intensity (4 levels), and a phosphor persistancy of about 2 seconds. Each point must be specified separately and the display must be repeated endlessly if it is to be viewed rather than photographed. A camera mount, several cameras, and a film index instruction are provided. The usable display area is 7 by 7 inches.

OPERATIONS

IOS_{60} 30000	SELECT SCOPE (CONNECT)	If the scope is <u>unselected</u>, FLAG 60 is raised. This instruction gives lowest intensity and a centered origin. See other IOS_{60} 30000 type operations below.
IOS_{60} 30000 IOS_{60} 30010 IOS_{60} 30020 IOS_{60} 30030	SELECT SCOPE and set INTENSITY	The scope "intensity" is controlled by the <u>duration</u> of the spot rather than beam intensity. 30000 - Low - 10 μsec. 30010 - Med. Low - 20 μsec. 30020 - Med. High - 40 μsec. 30030 - High - 80 μsec.
IOS_{60} 30000 IOS_{60} 30100 IOS_{60} 30200 IOS_{60} 30300	SELECT SCOPE and set ORIGIN LOCATION	The origin can be at the center, at the left or bottom edge, or at the lower left corner. 30000 - Center - ⊡ 30100 - Bottom Center - ⊡ 30200 - Left Center - ⊡ 30300 - Lower Left Corner - ⊡
IOS_{60} 30004	INDEX FILM	The IOB is busy until the return signal comes back from the camera. The return signal also raises FLAG 60.

NOTE: The IOS_{60} 30004 (Index Film) instruction causes an "EIA" (Equipment Inability Alarm) when the film supply in the camera magazine is low. This raises flag 41 <u>if unit 41 is connected</u>, lights the "End of Film" light, and rings a buzzer. (See next page.) It does not stop the computer. The scope and camera can still be used until the film runs out completely. When there is no film at all, the return signal that frees the buffer is not generated, TSD operations find the buffer "busy", and "Dismiss and Wait" occurs.

SCOPE DISPLAY

TSD T$_j$ ‖α OP ^{α}TSD T$_j$	 x coord. y coord. (4.9 - 3.9) (2.9 - 1.9)	TSD copies from T$_j$ to the scope buffer. The 10 bit coordinates are interpreted as signed ones complement numerals. Therefore, there are two zeros - plus zero (all zeros) and minus zero (all ones). Moved origin modes are realized by automatic complementing of the appropriate sign bit. Thus +0 = -0 in centered origin mode, and $0777_{(8)}$ = $1000_{(8)}$ in moved modes. Permutation and activity may be used.

NOTES

1. In most cases it is easier to use 9 bit arithmetic. In 18 bit arithmetic, one can use "Fractions" and sense end carry by the SKM instruction, or one can use "integers" and sense overflow the same way. In the latter case, one must cycle or scale to the left so that the 10 bits will be in the buffer position.

MANUAL CONTROLS
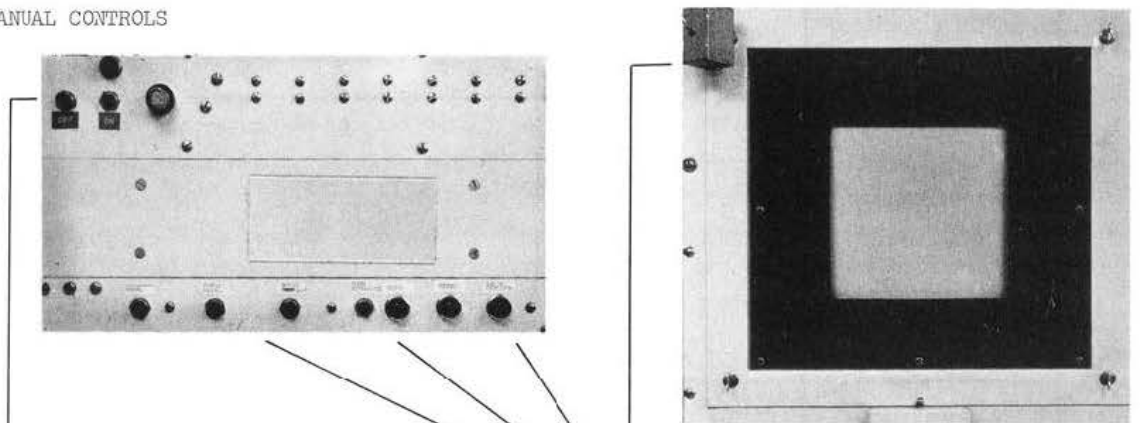
On-Off Pushbuttons - Display power comes on after a 65 second delay. It is best for it to be brought on while the computer is stopped, for it often causes a spurious raise flag signal.

For best resolution, it is necessary to wait about 20 minutes for the circuits to reach thermal equilibrium.

Camera Inversion Switch - "Toward the wire" is "Normal". "Away", gives a vertically inverted display to compensate for the mirror inversion in the camera mount.

End of Film Light
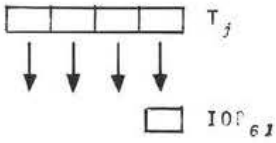
End of Film Acknowledgement Pushbutton This button will stop the alarm buzzer, but does not clear the EIA flip-flop. (See In-Out #41.)

Manual Film Index - Moves the film one frame.

RANDOM NUMBER GENERATOR

The Random Number Generator assembles a 9 bit number "at random" from a radioactive Cesium Source. The average time required is 57.6 usec - minimum time 28.8 usec.

OPERATIONS

IOS_{61} 30000	SELECT AND TRIGGER	The select operation also triggers the generation of a random number. FLAG 61 is raised as soon as the number is ready.

TSD $T_j \parallel \alpha$ OR $^{\alpha}$TSD T_j	T_j IOP_{61}	TSD copies the generated number into T_j. (Permutation is allowed, and quarter one must be active.) TSD also triggers the generation of the next number. FLAG 61 will be raised when it is ready.

MANUAL CONTROLS

On-Off Pushbuttons - There is a 60 second warm-up delay. Note: The Random Number Generator should be left OFF when not in use.

Note: The meters should read about half scale. They are used for maintenance purposes only. The maintenance switch is inside the box above the control panel.
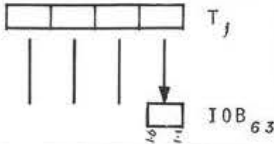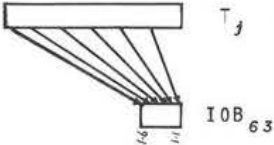
PAPER TAPE PUNCH

The PUNCH is the counterpart unit to the PETR. It is a line-by-line device and can be programmed to punch at speeds up to 180 lines per second. A TSD must be given for each line. Four modes are defined below.

OPERATIONS

IOS_{63} 30000	NORMAL NO 7th	Sets to punch 6 channels - no 7th hole (used for blank tape, end marks, and visual pattern punching.)
IOS_{63} 30004	NORMAL WITH 7th	Sets for 6 channels with automatic 7th hole punch on each line. (Used for tapes to be listed on off line Lincoln Writers.)
IOS_{63} 30002	ASSEMBLY NO 7th	Sets for splayed punching - Six TSD instructions punchout a 36 bit computer word.
IOS_{63} 30006	ASSEMBLY WITH 7th	Sets for splayed punching with automatic 7th hole. Used primarily for Binary Output.

NOTE: All the above IOS_{63} 30000 instructions "CONNECT" (SELECT) the punch and raise FLAG 63 if (and only if) the PUNCH was __unconnected__ prior to the instruction.

TSD T_j ‖α OR $^{\alpha}$TSD T_j		IN NORMAL mode, permutation and/or activity may be used. Only 6 bits of T_j are copied, T_j is not affected.
TSD T_j ‖α OR $^{\alpha}$TSD T_j		In ASSEMBLY mode the configuration syllable is ignored. The datum is copied as shown (from bits 4.9, 4.3, 3.6, 2.9, 2.3, 1.6) and after the copy the full 36 bit word is cycled left once (in T_j). Six "TSD T_j" operations will copy a 36 bit word from T_j to tape and will __leave__ T_j __cycled 6 places to the left.__

MANUAL CONTROLS:

Buzzer - Sounds when tape is low or jammed.

Buzzer Suppression Switch - Normally to the left
(i.e., Unsuppressed.)

Maintenance Switch - "On Maintenance" is to the
right.

Tape Feed Pushbutton - There is a 5 sec. delay
before tape is fed. (Button must be held down.)

Alarm Indicator Light (Low Tape, or Tape Jam)

Maintenance Indicator Light

Punch Reeler

Reeler Control Arm - Normally operated by the
tape as it is being reeled but can be moved
up by hand to energize the reeler brake when
needed.

Reeler Switch

"Chad" Basket

Punch Motor Switch - Normally up -
(For use on Maintenance mode only.)

TAPE DIAGRAM:

To the
Punch

Outside   Inside

Last
TSD

First
TSD

Normal Mode -
(Read toward the
7th. hole - i.e.,
from inside to
outside edge.)

| 1 | 0 | 1 | 0 | 0 | 0 |

| 6 | 7 | 1 | 0 | 0 | 1 | 3 | 0 | 2 | 2 | 0 | 6 |

7th
Hole

To the
Punch Reeler

Assembly mode - (Read from bottom
to top, inside to outside.)

LINCOLN WRITER

NOTE: Two Lincoln Writers can be ON LINE at once - "65, 66" or "71, 72"

REFERENCE: Group Report 51-8 (6 October 1959)

DESCRIPTION:

The Lincoln Writer Input consists of a double keyboard with automatic case change and a Soroban mechanical tape reader. They are interlocked so that only one can be used at a time - the keyboard is inactive while the reader is running. The Output is an IBM electric typewriter and a Friden paper tape punch.

Manual controls on the Lincoln Writer permit on-line or off-line use and seemingly both at once. The simplest connection for coding is "pure on-line" i.e., keyboard and reader connected to the computer alone - not to punch or writer.

In this "pure on-line"mode, there are no timing considerations that can cause trouble. TSD operations can be written without regard to the elapsed time between them. The only complication that must be remembered is that "carriage return" resets the keyboard to "lower case" and "normal script" without transmitting any case or script code.

Note also that a "carriage return" sent to the writer via TSD using Sequence # 66 or 72 will also affect the KEYBOARD'S automatic case memory in same manner. The Lincoln Writer input is not completely independent of its output! The situation becomes more complex when the keyboard is connected to writer and/or punch as well. These complex cases are to be discussed later in a supplement.

Automatic case codes are generated whenever the user changes from one keyboard to another. The key will remain locked down until two TSD operations have been performed - the first to accept the case code and the second for the character itself.
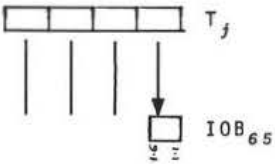
TX-2 LINCOLN WRITER CODES

00	0	☞	40	Q	α
01	1	∑	41	R	Δ
02	2	|	42	S	ρ
03	3	▮	43	T	ε
04	4	/	44	U	ħ
05	5	×	45	V	⊃
06	6	#	46	W	β
07	7	→	47	X	∧
10	8	<	50	Y	λ
11	9	>	51	Z	~
12	_	‾	52	(	{
13	○	□	53	)	}
14	READ IN		54	+	⊞
15	BEGIN		55	–	≡
16	NO		56	,	'
17	YES		57	•	*
20	A	≡	60	CAR RETURN	
21	B	⊂	61	TAB	
22	C	∨	62	BACK SPACE	
23	D	∂	63	COLOR BLACK	
24	E	γ	64	SUPER	
25	F	t	65	NORMAL	
26	G	ω	66	SUB	
27	H	×	67	COLOR RED	
30	I	∫	70	SPACE	
31	J	y	71	WORD EXAM	
32	K	≠	72	LINE FEED DOWN	
33	L	?	73	LINE FEED UP	
34	M	∪	74	LOWER CASE	
35	N	∩	75	UPPER CASE	
36	O	∮	76	STOP	
37	P	ħ	77	NULLIFY	

KEYBOARD

OPERATIONS:

IOS_{65} 30000 (IOS_{71} 30000 for other unit)	CONNECT KEYBOARD	This instruction selects the keyboard. <u>Pressing a key</u> will now raise FLAG 65 (or 71) if the keyboard is connected to the computer through the Lincoln Writer manual controls.

$TSD\ T_j\ \|\alpha$ OR $^{\alpha}TSD\ T_j$	T_j IOB_{65}	TSD copies the code number of the depressed key into T_j. Permutation may be used, quarter one must be active. The key is released after the copy. If an automatic code for case change (75 to UPPER, 74 to LOWER) was generated, the key will be released by the <u>second</u> TSD.

MANUAL CONTROLS
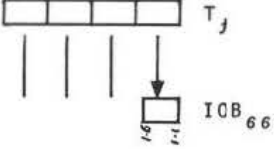
1. The <u>reader</u> must be started by hand via the "start reader" pushbutton. It will then read the line at the read station, advance one line, and wait for the datum to be accepted (presumably via TSD). The <u>keyboard</u> is inactivated while the reader is on. "STOP Reader" will re-activate the keyboard Maximum speed is 19 lines/sec and if the keyboard is connected to the computer alone, it can be programmed to run as slowly as desired.

2. Other manual controls are more or less self-explanatory and are well covered in Group Report 51-8.

3. Only the six bits (1.1 - 1.6) corresponding to the buffer are changed by TSD.

TYPEWRITER OUTPUT

OPERATIONS

IOS_{66} 30000 (Unit 1) IOS_{72} 30000 (Unit 2)	SELECT (CONNECT)	This operation selects the output of the Lincoln Writer - Typewriter and/or Punch. FLAG 66 (or 72) is raised if (and only if) the unit was unselected prior to the instruction.
TSD T_j ‖α OR ^{α}TSD T_j	T_j IOB_{66}	TSD copies 6 bits from T_j to the Lincoln Writer, where it is printed and/or punched depending on manual controls. The Buffer remains busy until the printing or punching is over and at that time FLAG 66 (or 72) is raised Permutation may be used and quarter one should be active.

MANUAL CONTROL

1. See Group Report 51-8 for details.

2. "Computer Output" should be switched to "Punch" and/or "Writer".

NOTES: 1. Carriage Return (Code #60) not only returns the carriage and advances the paper, but it also resets the Lincoln Writer to Lowercase and Normal Script. The Keyboard case relay is changed too. (In this respect, the keyboard and writer are not independent devices.)

2. Certain character codes (14,15,16,17,71,76,77) do not print. (They are labeled on the keyboard as "WORD EXAM", "READ IN", etc.) When such a code is sent to the WRITER, it is accepted, and takes about the same time as a regular character, but nothing is printed.

July 1961

MISCELLANEOUS OUTPUTS

Nine one-bit computer controlled relay contacts with G.R. Terminals are provided. Channel No. 9 has an high speed output as well. The nine bit word can be shifted left (ring or open) under manual control at a 500 KC rate if only high speed output is required or at 500 cycles per second if the relay contacts are to be sensed. TSD is not used. (It will not cycle the memory word, but it will DISMISS if no "hold" is used, and it will change E as if it were a ^{o}LDE operation.)

OPERATIONS

IOS$_{75}$ 30 000	Clear	The nine output channels are set to correspond to quarter 1 (the righthand nine bits) of the instruction. IOS$_{75}$ 30000 therefore clears all nine - IOS$_{75}$ 30777 sets all nine.	
" 001	SET 1.1		
" 002	" 1.2		
" 004	" 1.3		
" 010	" 1.4		
" 020	" 1.5	There is no raise flag indication. It may be assumed that the relay has changed after 2 milliseconds. The high speed output "MOM-9" will change before the instruction is over.*	
" 040	" 1.6		
" 100	" 1.7		
" 200	" 1.8		
" 400	" 1.9		

*Note: Changing bit 1.9 from "1" to "0" produces a standard TX-2 IN-OUT transition (-3 V to ground) at "MOM-9". Going from "0" to "1" produces a similar transition from ground to -3. The rise-fall time for these transitions is less than 0.2 microseconds.

MANUAL CONTROLS

Relay Contact Terminals

Shift Control

Maintenance Switch

High Speed "MOM-9" Output

Test Pushbutton - Sets relays to correspond to the Toggle Reset Register.

Shift-rate Inputs and Input Selector Switch

Toggle Reset Register
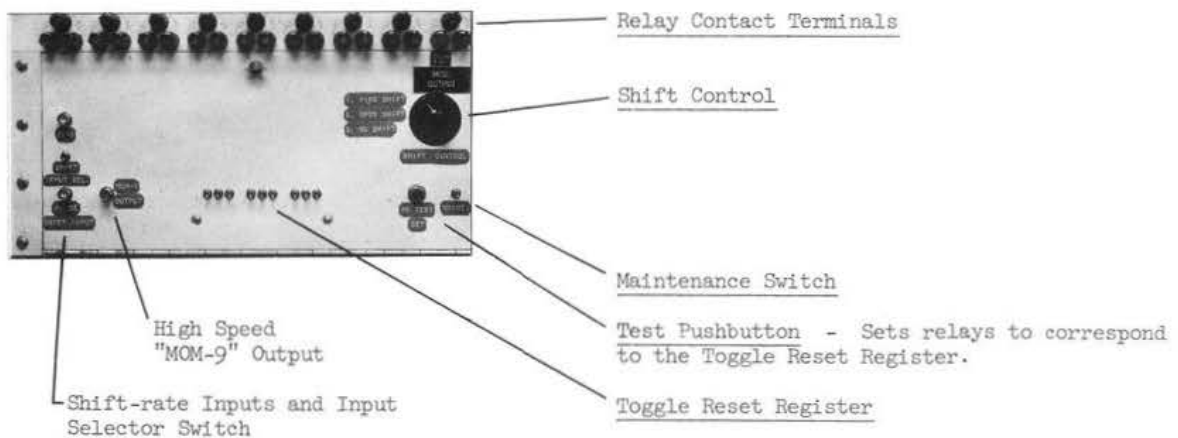
RELAY CONTACTS

C. P. Clare High Speed Relay - HGS - 1009 - Make before break.

Up to 1/4 amp non-inductive load.

Up to 1 amp reactive load with suppressor only. (Plug-in suppressors are available.)

Cycle rate approximately 500 cycles - 2 millisec. period.

SHIFT CONTROL

1. Ring Shift Left
2. Open End Shift Left (all nine, 1.9 is not a sign bit.)
3. No shift

SHIFT INPUT SELECTOR

The shift rate is determined by the external shift input which may be a sinewave or a pulse train. The shift input selector is a toggle switch which should be thrown toward the source used - up for "SINE", down for "PULSE".

SINE WAVE INPUT

A 15V RMS sine wave is required (e.g. GR Oscillator 1304). Maximum rate 500 KC for High Speed ("MOM-9") output, 500 cycles for relay output. Each cycle produces a one bit shift if "SHIFT CONTROL" is in position 1 or 2.

PULSE SHIFT INPUT

A standard TX-2 Inout Transition (-3 to Gnd) is required. It gives a one bit shift if "SHIFT CONTROL" is in position 1 or 2. Maximum rate: 500 KC for High Speed Output (MOM-9), 500 cycles for relay output. "MOM-9" should NOT be used to trigger the shift, for it may not change bit 1.9 or 1.1. (It would shift the others reliably.)

NOTE:

1. The shift and pushbutton inputs are not interlocked and can interfere with programmed use. Miscellaneous inputs can be used to synchronize the program and the shift input in use.

November 1961

TX-2 HANDBOOK

CHAPTER 5

LIGHTS AND BUTTONS

TABLE OF CONTENTS

Fig. 5-1, TX-2 Floor Plan - March 1963

5-1.1 Frame Contents - Floor Plan

F1 - Console - See Figure 5-3, 5-4, 5-5, 5-6, 5-7, 5-8, 5-9

F2 - Main Frame

B - Arithmetic Element, and E Register

C - Control, X Memory, P, Q, M, and N Registers

D - Sequence Control, Thin Film Memory

E - U and T Memories

F - IN-OUT Switch

F3 - S Memory Register Selection Circuits

F4 - S Memory Stack

F5 - S Memory Digit Plane Drivers and Sense Amplifiers

F6 - TX-2 Mag Tape Drivers and Timing Track Writing Equipment

F7 - IBM Tape Control, TX-2 Tape Control, Plotter Control, TX-2 Power

F8 - Lincoln Writer Controls, αβ Clock, Display Box, Misc. Inputs Box,
     Speech Filters

F9 - Plugboards, Datrac, Interval Timer, Misc. Input, Misc. Output, Ampex
     Mag Tape

5-1.2   Power ON-OFF Procedures
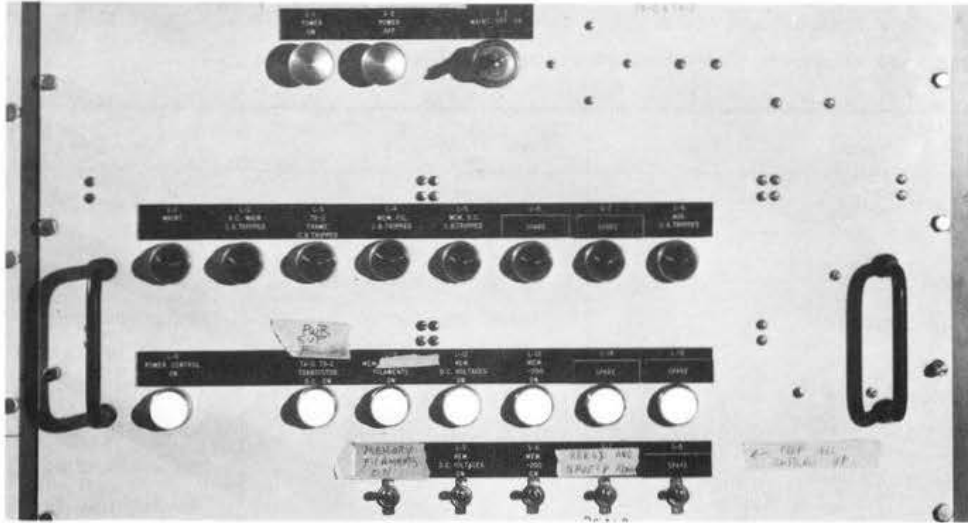
Fig. 5-2 - Power Panel (in the Power Room)

Power On:

    Hold the button in until the warning horn stops. The computer will
    be "ready" in about 1 1/2 minutes. Run "clear memory" once or twice
    with parity alarms suppressed. Un-suppress the alarms. the computer
    should be ready for use. Log the time. (Fig. 5-3) Turn on the Lincoln
    Writer(s), and the IBM Tape Units. (They have their own power switches.)

Power Off:

    Be sure TX-2 tape is at "MAT 0000", before pushing "OFF" button. Log
    the time. (Fig. 5-3) Turn off the Lincoln Writer(s), and IBM Tape Units.

5-1.3   Power Alarms - Breakers
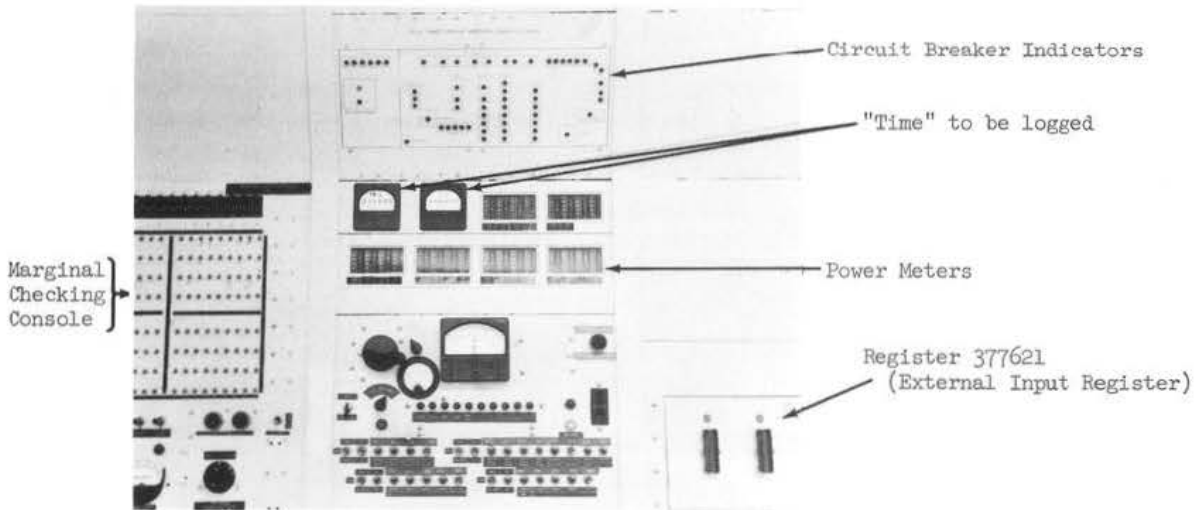
Fig. 5-3 - Maintenance Console

The circuit breakers are located at the top of each frame. When a breaker "lets go", a horn sounds and a light comes on at the breaker panel and at the power panel on the console. The accepted procedure is as follows:

Frame:	What to do:	If that fails:
S, T or U Memory : F2, 3, 4, or 5	a) Dump Power b) Reset the breaker c) Bring Power On	Call for help.
Computer Frames:	Reset Breaker	Call for help.
IN-OUT Equipment	Reset Breaker	Call for help - Set Maintenance Switch on Breaker Panel and do not use failing unit.

In any event, log the incident stating the time it occurred, which breaker it was, and what was done.

5-1.4 Air Conditioning

There is a room temperature thermometer on the column at frame 9. It usually reads about 70°F. There are two thermometers for the S Memory Stack. One is behind the stack, the other is in the power room. The power room meters should read as follows: (They are to the left as you enter from the computer room.)

	NORMAL	CALL for HELP	DUMP POWER
Memory Stack	60 - 70	72 or more	75 or more
"MIXED AIR"	48 - 58	75 or more	80 or more
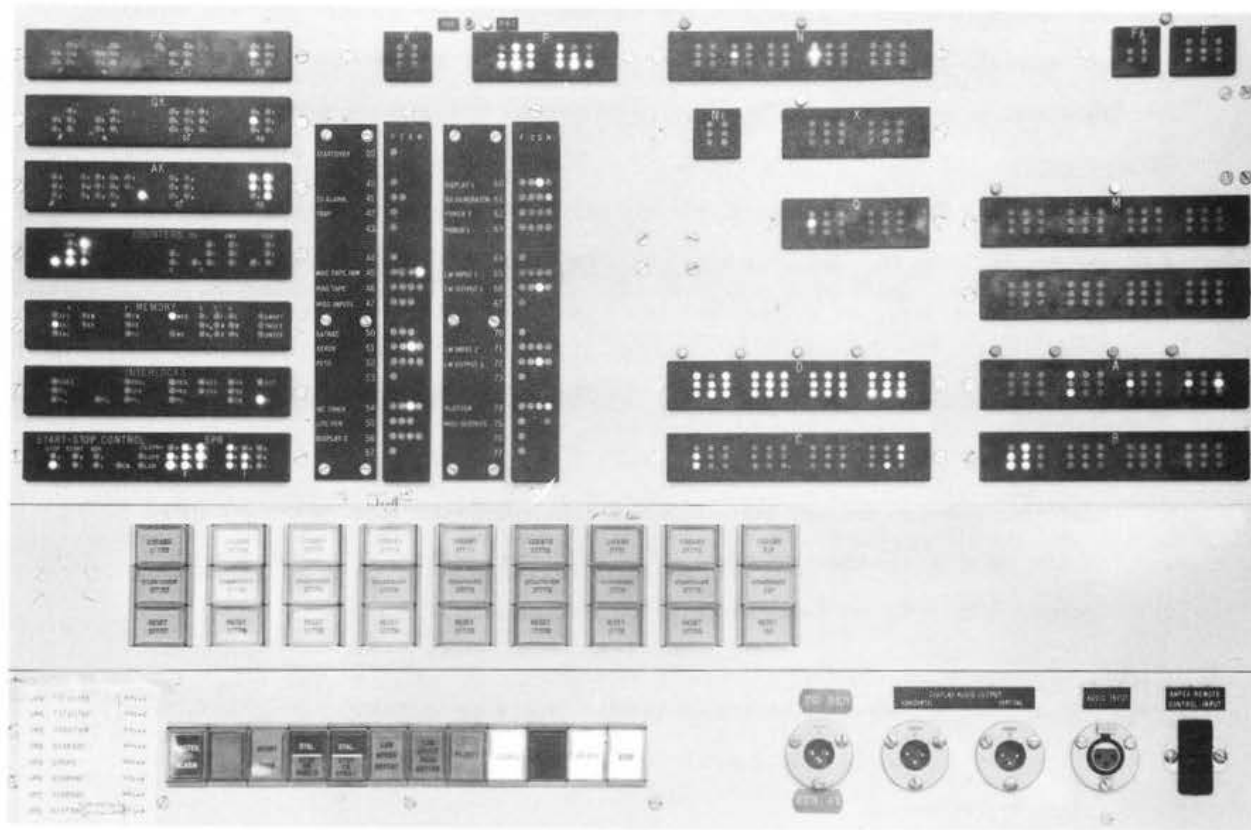
Fig. 5-4, - Console Indicator Panel

5-2.1 Primary Indicators

The indicator lights for the central machine registers are arranged to be read easily in OCTAL by grouping the binary indicators in columns of 3 lights each. - The least significant bit at the bottom. For example:

0 1 2   3 4 5   6 7 0   1 1 1
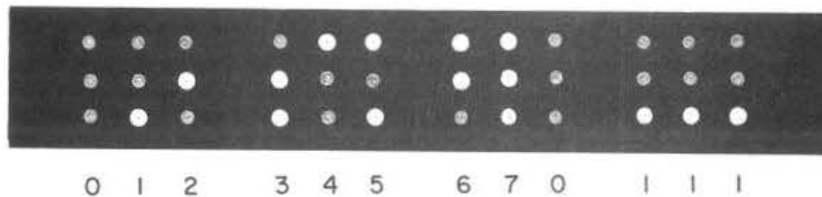
Filament bulbs are used and therefore vary in light output while the computer is running. (Note also that the bulbs will occasionally burn out and give an erroneous indication.)

The table below summarizes the information given by the lights.

Registers A, B, C, D, E, K

These indicators always show the contents of the associated flip-flop registers. (K shows the sequence number last used. - See Chapter 4.) They are not affected by alarms or pushbuttons. The overflow indicators are just above A.

PK Indicators

PK_α and PK_β are of interest for the most part to the Technicians. They show the PK_1 PK_2, and PK_3 timing levels. PK_h, PK_{cf}, and PK_{op} show the hold bit, configuration bits, and Operation Code of the instruction whose read out cycle is in progress.

QK Indicators

QK_α and QK_β show the QK timing levels. QK_{cf} and QK_{op} show the configuration read from F Memory and the last operation that required a data reference. QK_{cf} is also used to remember the original Index Register Number on deferred address cycles. (The original Index Register is used last.)

AK Indicators

AK_α and AK_β show the AK timing levels. (There is an indicator for each level. AK_{cf} and AK_{op} show the configuration (minus permutation) Operation Code of the last Arithmetic Operation.

ASK - The Arithmetic Step Counter

ASK is used to count steps for Arithmetic Operations such as Multiply, Tally, and Divide which are different for different word lengths.

XWK - The X Memory Counter

XWK sets the timing levels for the writing of the Index Registers.

CSK - The Change Sequence Counter

CSK sets the timing levels for a Change of Sequence. (CSK $\gtrless$ $10_{(octal)}$ are the steps that indicates LIMBO.)

Memory Indicators - Interlocks

Most of these are not of interest to programmers. The PI_1 PI_2, PI_3, PI_5, and DFA are of use in interpreting N (see page 5-10). PI_2 indicates a defer cycle is in progress. DFA indicates completion of a defer cycle.

START-STOP Control - SPR, Start Point Register

The left half of this indicator shows the start-stop interlocks and is of
interest mainly to Technicians. The right half is the Start Point Register. It
is set by the RESET, STARTOVER, and CODABO pushbuttons and is used to set the P
register when a change to sequence zero is performed. (See PK_2 in the table below.)

Registers P and N

P and N are the selector and buffer for readout of instructions from STUV Memory.
P is also called the "Central Program Counter" and N the "Instruction Register".
At the start of any instruction, they are, of course, compatible - P gives the
address of the contents of N. As the instruction is performed, both are changed.
The extent of such change depends on when, in the cycle, the computer was stopped.

There are two indicators to help interpret P. Their use is given below:

Indicators		Contents of P
"POD"	"P+1"	
0	0	P gives the address of the last instruction read out of STUV Memory. (It may have been changed in N - see PK_2 in table on page 5-9.)
0	1	P has been indexed, but not yet used for read out. It gives the address of the next instruction.
1	0	"PmODified" - P has been changed radically and probably bears no relation to N. (As by a jump, skip, or sequence change.)
1	1	This situation should not occur. Take a picture of it.

The interpretation of the N register depends upon the type of operation being
performed, and how far the computer has gone before stopping. Instructions require from
one to five basic cycles. The first cycles for one instruction can be overlapped with the
final cycles of the previous instruction and it is therefore possible for two cycles to
be running at the same time. The stop system (stop button, sync system, and slow speed
control) is synchronized so that once a cycle has started, it must proceed to completion.
There are indicators that tell what cycle is next, but one must exercise ingenuity to
determine which one has just finished. The basic cycles are abbreviated as follows:

PK_1	-	Instruction Readout Cycle
(PKAK)	-	(Used by AOP instruction - Instruction Readout followed by Arithmetic Cycle)
PK_2	-	Intermediate Address Cycle (Deferred Addressing)
PK_3	-	Final Address Cycle (Deferred Addressing)
Q_K	-	Data Reference Cycle
QKAK	-	Data Reference followed by Arithmetic Operation Cycle. They are inseparable, but another QK could start before the AK part is over.
CSK	-	Change Sequence Cycle.

In practice, their order of occurrence depends on the operations being performed and on overlap conditions. The table below shows the cycles for the three basic types of computer operations.

Type	Op Code (See Chart 7-3)		Cycles Required
1	0 - 7	(Jumps and IOS)	PK_1
	44, 46, 4,	With deferred address	PK_1 , PK_2 , PK_3
2	10 - 57 (but not 44, 46, 47)	(Non AE, Non Jump)	PK_1 , QK
		With deferred address	PK_1 , PK_2 , PK_3 , QK
3	60 - 77	(AE operations)	PK_1 , QKAK
		With deferred address	PK_1 , PK_2 , PK_3, QKAK

The CSK cycle can occur only at the end of an instruction - i.e., only after PK_1, PK_3, QK, or QKAK - never between PK_1 and PK_2, nor between PK_2 and PK_3.

The effect of these cycles on N is as follows:

Cycle	Effect on N
PK$_1$	This is the initial instruction readout. N will contain the instruction located at the address indicated by P. EXCEPT when the operation is JNX or JPX (codes 6 and 7), for on these two operations the right half of N is used for the sign extended index increment (18 bits). (BUT if the JNX or JPX is deferred, the increment is not added until PK$_3$ so N is $\underline{not}$ changed during PK$_1$.
PK$_2$	The intermediate deferred address cycle (PK$_2$) is always followed by PK$_3$. The intermediate address is read out into N using Q as the selector. All 36 bits of N are changed, but the $\underline{initial}$ index register number was saved (in QKIR$_{CF}$) to be used last (in the PK$_3$ cycle). After a PK$_2$ cycle, N contains the contents of the memory register given by Q. Bit 2.9 of Q will be 1 due to the defer bit.
PK$_3$	The final deferred address cycle does not do a memory readout. It is known as the "Ultimate Cycle". DFA will be set to 1. N is further changed by adding in the index contents to get the final address. Then the original N$_J$ bits are restored and the instruction continues. The rest of N is cleared.
QK	The QK cycle of all index memory operations and SKM (all op codes 10-17) clears the right half of N. For RSX, EXX, AUX, and ADX it is subsequently set from the right half of E which was in turn set from memory and may have sign extension. ADX puts the augend from memory there. The next PK cycle can not be overlapped with the QK cycle of these operations.
QKAK	QKAK does not change N
CSK	The change sequence cycle always changes N$_J$ to the old sequence number. If the new number is zero, the right half of N is set to the contents of SPR (Start Point Register). The rest of N is not changed by CSK. CSK must be followed by PK$_1$.

"Control" does not really care about what has been done. It is interested only in what it is allowed to do. Once it has started a cycle, it goes merrily on to completion, but before starting one, it must get past a number of interlocks, one of which is the start-stop system. We can therefore tell what cycle is about to start and with that information, together with a program manuscript, the P register, and the P + 1 and POD indicators, we should be able to deduce where it stopped, and therefore what is in N. The conditions for starting are foretold by indicators PI_1, PI_2, PI_3, PI_5, and DFA as follows:

| "Interlock Indicators" | | | | | Next Cycle | Stopped |
PI_1	PI_2	PI_3	PI_5	DFA	is	After
0	0	0	0	X	PK_1	?
0	1	0	1	0	PK_2	PK_1
0	1	0	0	0	PK_3	PK_2
0	0	0	0	1	PK_1	PK_3 or QK, CSK
0	0	1	0	X	CSK	?
1	0	0	0	X	QK or QKAK	?

Note: "X" means "It can be 0 or 1, it does not matter."
"?" means "Any cycle <u>but</u> PK_2".

Registers Q and M

Except for defer cycles, Q and M are the selector and buffer for data references to STUV memory. The memory references for deferred intermediate addresses use Q as the selector and <u>N</u> as the buffer (Cycles PK_2 and PK_3).

Indicators N_j and X

The N_j lights are copies of the index tag bits of N (bits 3.6 - 3.1). The X register is the index memory buffer. N_j and X will always be compatible, for the X memory is read out even if it is not used.

Registers FA and F

FA and F are the selector (F Address) and buffer for the Configuration Memory. They are always compatible.
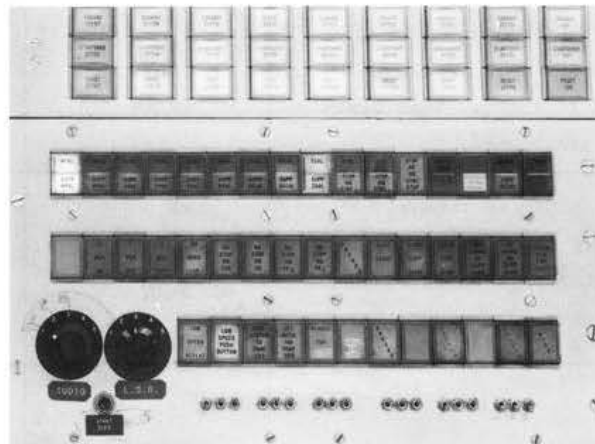
Fig. 5-5, - Alarms, Conditions and Action Pushbuttons

The top row of pushbutton indicators in Fig. 5-5 above is used for the ten TX-2 Alarms, two special indicators, and the Sync System (Section 5-4). All of the alarms except TSAL, USAL, and the "Mousetrap" can be suppressed by pushing the indicator. These pushbuttons have two lights each. The upper light indicates the alarm, the lower light shows that it is suppressed. Suppression of an alarm merely keeps it from stopping the computer. In the case of parity alarms, the suppression allows the computer to use the incorrect information and a new, presumably correct, parity is computed and stored. The light will always come on. The chime must be suppressed separately if not wanted. (See 5-3.1 for chime details.) "MISAL" - the Missed Information Alarm - can also raise flag 41 (In Out Alarms). See Chapter 4, Section 5.

Detailed information on each alarm is given in the table below:

ALARM NAME	CAUSE	HAPPENS DURING COMPUTER CYCLE **	WHAT TO DO
MPAL	Parity Alarm on readout of data from STUV Memory into the M Register. Q gives the address of STUV Memory register. This alarm can not be programmed.	QK or QKAK	Take a picture. Report failing bit and whether it was "dropped" or "picked up". (if you know what should have been read out.) Try again.
NPAL	Parity Alarm on readout of instruction into N (location given by P), or on readout of deferred intermediate address into N (location given by Q).	PK_1 or PK_2	Take a picture. Report failing bit as "drop out" or "pick-up". Try again, alarm not programmable.
**	Because of overlap, another cycle may be running concurrently and the computer will continue until both are completed.		

ALARM NAME	CAUSE	HAPPENS DURING COMPUTER CYCLE [**]	WHAT TO DO
XPAL	Parity alarm on readout of index register into X. N_j tells which index register. This alarm can not be programmed.	Any Cycle	Take a picture. Report failing bit as "dropout" or "pickup" if you can. Try again.
FPAL	Parity alarm on readout from F Memory (configuration) into the F register. FA tells which F memory register.	PK_1 QK or QKAK	Take a picture. Report failing bit as "dropout" or "pickup". Try again.
PSAL	P register is set to an illegal address.	PK_1 CSK	Check your program, this alarm can be programmed. If machine malfunction is suspected, take a picture and try again.
QSAL	Q register is set to an illegal address - either a data reference or a deferred address. Check chart on page 5-10.	PK_2, QK, or QKAK	Check your program. This alarm can be programmed and is not likely to be a machine malfunction.
OCSAL	Operation Selection Alarm: An illegal instruction was readout into N.	PK_1	Check the program. Take a picture.
IOSAL	In Out Alarm: This happens on an IOS instruction. The selected device is either broken, on "maintenance", or non-existent. The IOS has had no affect, even if the alarm was suppressed. The N_j indicators should tell what unit was selected.	PK_1	Check the device you are selecting or the indicator panel. The in out device is probably on "maintenance". Unless there is a "hands off" sign, throw the maintenance switch down (i.e., not maintenance) and try again.

[**] Because of overlap, another cycle may be running concurrently and the computer will continue until both are completed.

November 1963

ALARM NAME	CAUSE	HAPPENS DURING** COMPUTER CYCLE	WHAT TO DO
MISAL	Missed Information Alarm: This occurs when the program is too slow for the in-out device, and a new datum or output opportunity has come along before the last was used. MISAL is automatically suppressed if sequence 41 (in Out Alarms) is connected.	Any Time	Probably program trouble. Can happen with PETR, TX-2 Mag Tape, A/D Converter, or IBM Mag Tape. Take a picture.
TSAL	The T memory selection currents have not died out before a new register selection was demanded. (T memory is 200,000 to 207,777)	Any Cycle Except CSK	Take a picture, report that it happened, and hope it will go away. It can not be programmed or suppressed.
USAL	Same as TSAL, but for the U Memory. (210,000 - 217,777)	Any cycle except CSK	Same as for TSAL.
Mouse-trap	This is an extra alarm designed to trap any mouse that may be causing computer trouble. It will be set differently from time to time. As of now, it is set to catch a missed control pulse.	Any cycle	Same as for TSAL - USAL.
The following indicators are not true alarms.			
Priority Patch Indicator	A non-standard priority plug-board is in use. (The standard priority is consecutive numerical order--lowest number having highest priority.)	It doesn't happen - it exists.	Replace the standard plugboard at Frame 2, Bay D.
Limbo	The computer is running, but all selected sequences are waiting for a flag.	It doesn't happen - it exists.	For most inter-leaved programs, the LIMBO light will be on, for some waiting time is almost unavoidable. If the program seems to have stopped completely, check the interleaving.

**   See Footnote on page 5-12.

5-2.3   IN-OUT Indicators

    The IN-OUT indicators common to most units -
i.e., "Flag", "Connect", "Status", and "Maintenance" -
are on the main indicator panel - Fig. 5-4 and Fig. 5-6
(to the right). All sequence numbers have a FLAG, but
some have no associated IN-OUT unit and hence no "Connect",
"Status", or "Maintenance" indicators ("F", "C", "S", and
"M"). The indicators are interpreted as follows: (see
also Chapter 4)
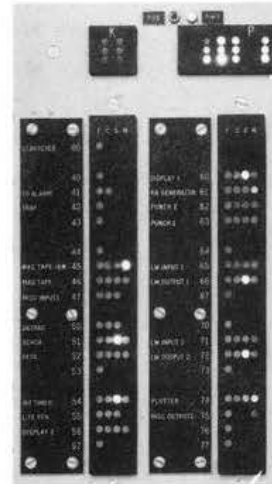
Fig. 5-6   IN-OUT Indicators

INDICATOR	MEANING
F - FLAG	The Flag is up - The associated program will be operated as soon as priority conditions allow.
C - CONNECT	The Associated IN-OUT unit is "connected"; i.e., selected for use.
S - STATUS	If STATUS = 1 ("ON"), a TSD can be performed. If STATUS = 0 ("OFF"), a TSD will have to wait, for the IN-OUT buffer is still busy processing the last datum.
M - MAINTENANCE	If M = 1, the Maintenance Switch (at the unit) is up. A select instruction (IOS) will cause an IOSAL (IN-OUT Select Alarm). The unit can not be connected.

    The IN-OUT Buffers and special indicators are on a separate panel shown on next page:
(See Chapter 4 also).
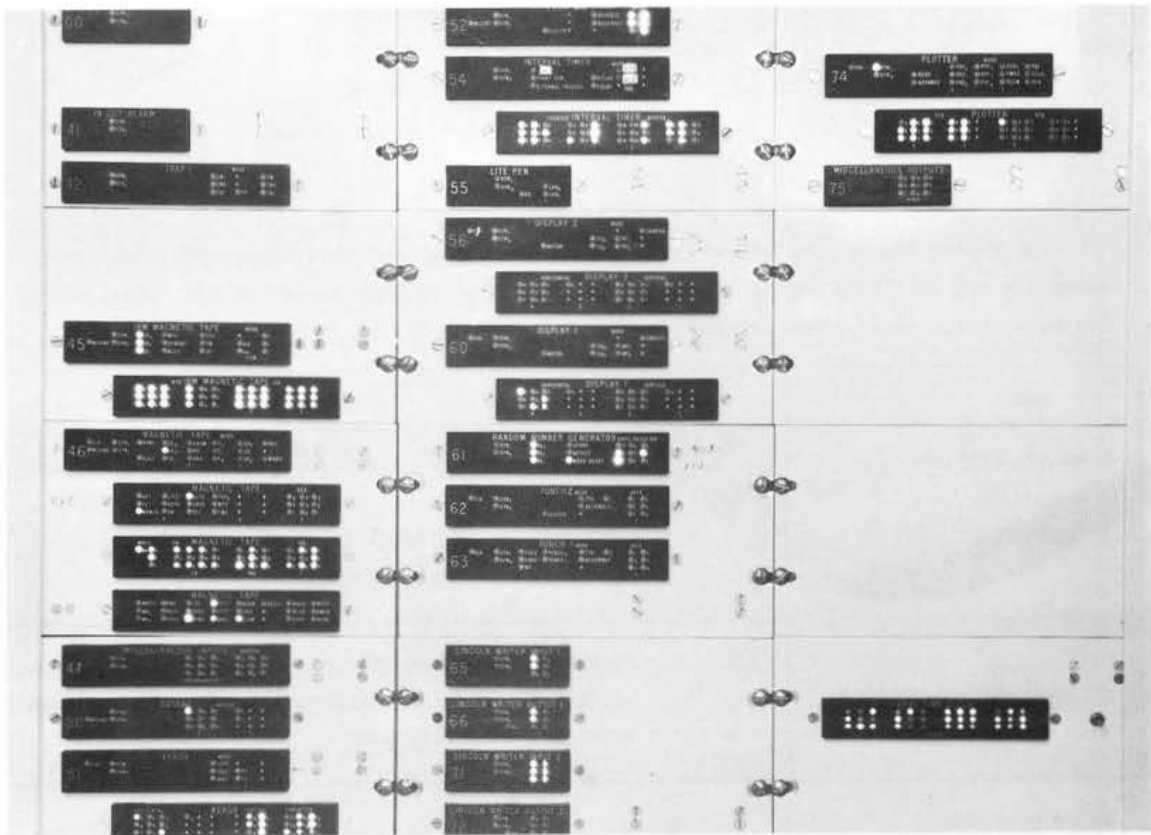
Fig. 5-7 - IN-OUT Buffers and Special Indicators
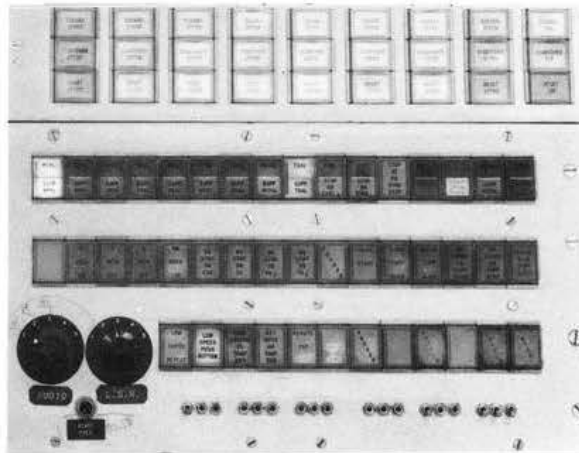
Fig. 5-5, - Alarms, Conditions and Action Pushbuttons

5-3.1   Condition Pushbuttons (Fig. 5-5)

The center row of pushbuttons and some of the bottom row will change the mode of operation and will light up as an indication that the computer is not in its normal mode. The table below shows what they do:

NAME	FUNCTION
U Memory Off T Memory Off S Memory Off	These prevent the program from using the U, S, or T memories. When the indicators are ON, a PSAL or QSAL will be generated if an attempt is made to use the suppressed memory.
No Overlap	Overlapped programs will run slower with "NO OVERLAP" on, but the indicator lights - especially N - should be easier to interpret.
No Stop on CSK No Stop on QK No Stop on PK_2 No Stop on PK_1	The computer will not stop <u>before</u> the selected cycle(s).
PASOFA AUTO START	"Preset and Start over from Alarm" and "Auto Start" are usually used together. Auto Start alone is equivalent to pushing CALACO about a second after the alarm. PASOFA is equivalent to an automatic CODABO after alarm. (Except that the alarm is not cleared.) They are used primarily for maintenance and computer repair.

November 1963

NAME	FUNCTION
Low Speed Repeat	This circuit inserts a variable delay between the computer cycles. It operates in conjunction with the NO STOP buttons (i.e., it does not insert a delay before the selected cycle(s)). There can be no OVERLAP when this mode of operation is used. The inserted delay (and therefore the effective computer speed) is controlled by the right-hand switch-knob at the bottom left corner of the control panel (Fig. 5-5). (It is labeled L.S.R.)
Low Speed Pushbutton	This circuit inserts a "STOP" before each computer cycle unless the "NO STOP" buttons are on. There can be no overlap.
Hold on LSPB	"Hold on Low Speed Pushbutton" - In this mode, all instructions are treated as if their hold bit were set. This allows step-by-step operation of a low priority program without any interruption due to a change of sequence.
Remote TSP	There is a portable control panel that contains some of the condition and action buttons and another 18-switch toggle START register. It can be plugged in at Frames 9, 3, and 2, and also behind the console. (It contains condition buttons: Low Speed Repeat, Low Speed Pushbutton, Remote TSP, the Sync Stops; and action buttons: CODABO, PRESET, CAIACO, and STOP.
No Chime on SUPP ALMS No Chime on $\overline{\text{SUPP}}$ ALMS	$\overline{\text{SUPP}}$ means "not suppressed". The circuits were designed for two different chimes but only one tone is commercially available at present. These condition buttons have no other effect.
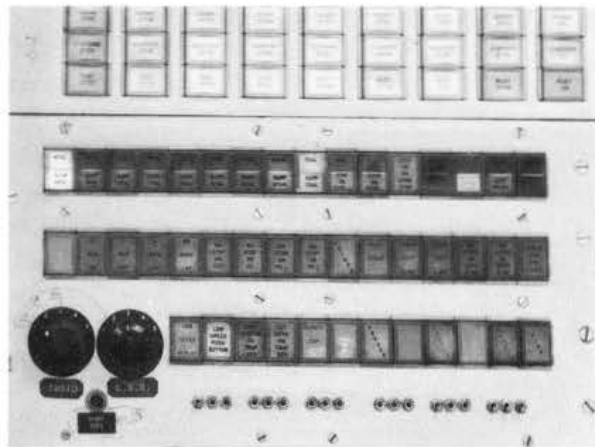
Fig. 5-5 - Alarms, Conditions and Action Pushbuttons

There are six buttons that actually do something. Their use is outlined in the table below. No information registers (Memory or AE) are affected. There is no clear memory button.

BUTTON	FUNCTION
CODABO	"Count Down and Blast Off" - CODABO is the most commonly used start button. It is roughly equivalent to STOP, CLEAR ALARMS, PRESET, STARTOVER, and CALACO in that order. Its effect is to clear all flags, preset all interlocks, and start the computer at the memory location given by the Toggleswitch Start Register (TSP) or the remote TSP, if selected. There are 9 CODABO buttons - 8 for the fixed addresses - 377710 to 377717 and the ninth for the two toggle START registers (Console and Remote). CODABO leaves the SPR (Start Point Register) set to the chosen starting place.
STOP	"STOP" is synchronized so that the computer will complete the cycles it has started. Except for start-stop interlocks, no registers or indicators are directly affected.

BUTTON	FUNCTION
CALACO	"Clear ALarms and COntinue" - CALACO merely resumes operation where it left off. If no flags are up, the computer will go into LIMBO. The combination of STOP and CALACO has no effect on a single sequence non-InOut program, but will probably upset IN-OUT and interleaved programs because of the timing.
RESET	There are nine RESET buttons - eight of them load the SRP with the fixed addresses 377710 to 377717. The ninth loads SRP from the selected Toggle Start Register (Console or Remote). RESET has NO OTHER EFFECT. The SRP is, in effect, a partial placekeeper for sequence zero. If the program raises flag zero, sequence zero starts at the place indicated by SRP. SRP is not changed when sequence zero drops out as the other place-keepers are. It can be changed only by pushbuttons.
STARTOVER	Nine STARTOVER buttons are available. They are equivalent to RESET plus a "Raise Flag Zero". STOP followed by STARTOVER will not do much, for STARTOVER does not start the computer. If the computer is running or in LIMBO, STARTOVER will be effective for Flag Zero has priority over all others no matter which priority plugboard is in use. STARTOVER followed by CALACO is similar to CODABO, but does not clear the Flags and interlocks.
PRESET	There is but one PRESET button. Like RESET, it is seldom used by programmers. It clears all flags and In-Out "Connect" flip-flops, and sets all interlocks and indicators to their proper "PRESET" value. This button is interlocked so that it is ineffective unless the computer is stopped.
Clear SUPP ALMS Clear $\overline{\text{SUPP}}$ ALMS	Suppressed Alarms are handled by separate circuits and a pushbutton is supplied for each type. $\overline{\text{SUPP}}$ means "not suppressed".
Clear Real Time Clock (Reg. 377630)	The Real Time Clock is indexed automatically every 10 microseconds. It will clear itself every 7.6 days or so if it is left alone. (The toggle switch to the right of the indicator turns the indicator lights off but has no effect on the Clock Register.)

5-3.4   Miscellaneous Console Items:
Audio Controls

Selector Switch
Mike Input
Mike Level Adj.
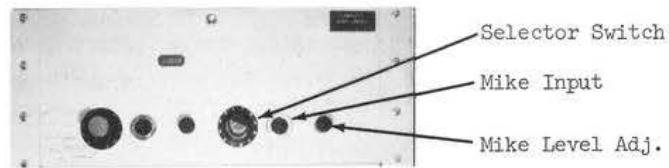
Fig. 5-8 - Audio Controls

For the convenience in trouble-shooting, to reassure users that the machine is running, and to further the progress of research, TX-2 has been made audible via two separate, independent, and identical Audio Systems. The Selector switches have ten positions, five of which are currently in use and wired as follows:

1.   Quarter 1 of the X Register (Analog signal decoded from indicator Circuit.)

2.   Quarter 2 of the X Register (Analog signal decoded from indicator Circuit.)

3.   Vertical Display Decoder (Sequence 60)

4.   Horizontal Display Decoder (Sequence 60)

5.   The Patch Panel at Frame 9.

The inside knob of the selector switch is the main volume control. The microphone input is mixed in at all selector settings and has its own level control.

Knob Register - (377620)

Register 377620 - The Knob Register - also called the "Shaft Encoded" Register is located just below display #1. It is similar to a toggle register except that it is set by four knobs - one for each quarter. The metabit is a lighted pushbutton switch. (Eight revolutions cover the range 000-777.)

External Input Register - 377621)

Register 377621 - The External Input Register - is a set of four plugs just to the right of the marginal check panel (Fig. 5-3). There exists a box with 37 pushbuttons intended for use with (or as) the external register. These pushbuttons are directly analogous to toggles except they must be held down if they are to stay a "1".   (Unlike the keyboard, any number may be down together.)

Note: Contact bounce is about the same as the toggle contact bounce - a delay of 10 ms allows a small safety factor.

Clock Register - (377630)

Register 377630 is a 36-bit counter indexed every 10 microseconds by an external oscillator. It can be cleared by pushbutton (Fig. 5-5), but not by a programmed instruction (such as STA or DPX).
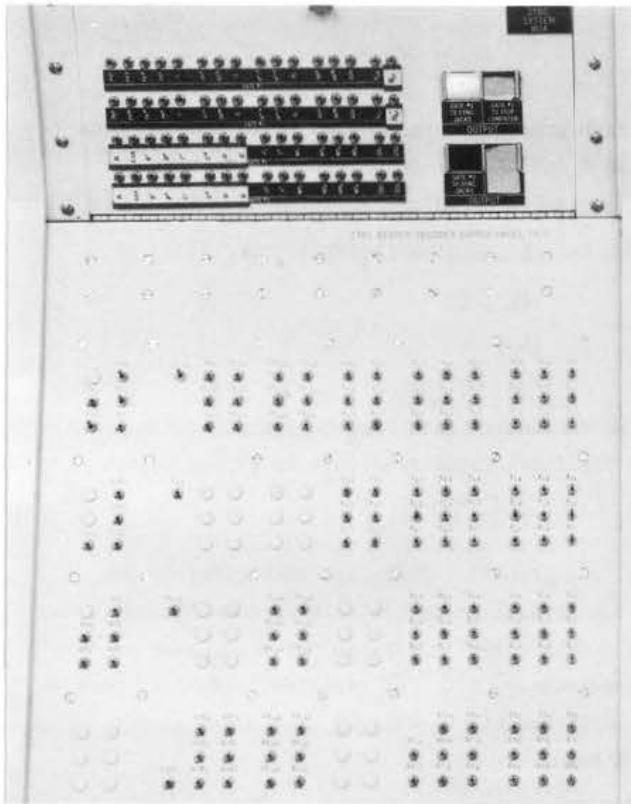
Fig. 5-9   -   TX-2 Sync System

The Sync System produces an output signal when certain manually preselected conditions are met. For example:

a)   When (or if) the program gets to a prespecified memory location. (i.e., $[P]$ = preset value)

b)   When a certain memory register is used for data or deferred address. (i.e., $[Q]$ = preset value)

c)   When a certain operation is used. (i.e., $[PK_{op}]$ = preset value)

d)   When a certain sequence number is used.

Certain combinations are possible. The output can be switched to any or all of the following:

OUTPUT	SWITCH LOCATION	COMMENT
Stop on SYAL #1	Top Row Pushbuttons (Fig. 5-5)	There are two alarms and two condition selectors, but only one set of condition parameter switches.
Stop on SYAL #2	Right next to SYAL 1	
Signal to Sync Jacks	On Sync Panel (Fig 5-9 above)	Uses for scope sync during computer repair and maintenance.
Raise Flag 42 (trap sequence)	Bottom Row Pushbuttons (Fig. 5-5)	See Chapter 4 - Trap Sequence. This button over-rides other trap modes.
Sync Stop to Arithmetic Element	Top Row Pushbuttons (Fig. 5-5)	Used mainly for naintenance to Stop AE operations within a cycle.

Figure 5-9 shows the two SYNC SYSTEM panels. The lower panel contains the Sync Parameter Switches; the upper panel the Sync Condition Selecting Switches and the output-to-Sync jacks pushbutton switches.

Parameter Switches

There is but one set of switches for each parameter even though there are two sets of selectors. The parameter switches are laid out in four major rows:

        INSTRUCTION   (PK , N)

        CHANGE SEQUENCE COUNTER, and INSTRUCTION LOCATION (CSK, P)

        DATA CYCLE   (QK , Q)

        ARITHMETIC CYCLE   (AK)

Each parameter set is grouped, like the indicators (see Fig. 5-4, and Section 5-2.1) in columns of 3 for Octal interpretation. The least significant bit is at the bottom.

Condition Selector Switches

There are two SYnc stop ALarms (SYAL 1 and SYAL 2). These are controlled by two "GATES" (Gate 1 and Gate 2). The "GATES" are controlled by two sets of condition switches - 32 switches each. Either gate or both can supply output pulses to the sync jacks or stop circuits. (Only the alarm indicators are separate.) ALL the selected conditions must be met for the output to be generated. (The GATES are AND circuits.)

The conditions available are described below:

CONDITION	COMMENT
PK_α	See upper left corner - Fig. 5-9. PK_α refers to the 32 possible time steps (levels) of the PK cycle counter and therefore determines <u>when</u> the sync signal will be generated. A setting of $16_{(oct)}$ is recommended, for it provides a definite time to stop, and is used by all instructions. PK_α is recommended when any of the "P"-type conditions are used.
PK_{op}	This compares the PK_{op} Parameter switch setting with the operation most recently read out of memory. (Bits 4.3 to 3.7)
PK_{cf}	This compares the configuration switch setting with the configuration bits most recently read out. (Bits 4.8 to 4.4)
PK_h	"h" refers to the hold bit of the instruction most recently read out. (Bit 4.9)

CONDITION	COMMENT
P	P refers, of course, to the P register.[*] The Parameter switches are in the second major row.
QK_α	This condition allows selection of the time step when the sync pulse is generated. A setting of 2 is recommended. QK_α and PK_α should not be used concurrently unless a particular type of overlap condition is sought.
QK_{op}	This set looks for a particular operation, just as does PK_{op}, but only those instructions that require a data reference will ever get into QK_{op}. (The QK_{op} indicator lights are at the left - Fig. 5-4. They are sometimes helpful in debugging, for they tell what operation made the last data reference.)
Q	Q is used for intermediate deferred addresses as well as data references, but these two can be separated somewhat via QK_α. With Q <u>and</u> QK_α selected, the sync output will occur only for data references since the <u>PK counter</u> is used for the defer cycles.
$N^{4.10}$	The circuitry is able to detect <u>set</u> metabits on instructions but <u>not</u> zero metabits. If the parameter switch is down and the selector switch is up, no sync pulse can be generated for the gate in use.
$M^{4.10}$	The data reference metabit ($M^{4.10}$) can be detected only when <u>set</u> (just as $N^{4.10}$ above). Note that it can be changed without a memory reference for it serves as the metabit of the A, B, C, D, and E registers. (i.e., $MKC_{4.10}$ A or $MKC_{4.10}$ B will change bit 4.10 of M.
N_j	"N_j" refers to the "j" bits of the N register and hence to the index register in use, or to the bit selection of an SKM operation.

[*] If the instruction that has been interrupted used a deferred address, CALACO will not continue the program until the third time it is used. (Since P is not changed until the last moment, a Sync Stop occurs during the intermediate cycle, and again during the "ultimate" cycle.)

CONDITION	COMMENT
AK_α	"AK_α" is the Arithmetic Instruction Time Level counter. There is one switch for each level (it therefore makes little sense to have more than one up). The recommended setting is OFF - it is mainly for maintenance use.
AK_{op}	"AK_{cp}" is a 6-bit register that holds the most recent <u>arithmetic</u> operation (code values are all <u>above</u> 57). It is not changed until another arithmetic operation is performed.
ASK	ASK is a 7-bit counter used for arithmetic operations that require repetitive steps - for example, multiply and divide. It clicks along during shifts and cycles, but is not used.
$X_{2.9}$	$X_{2.9}$ is the sign bit of the X memory buffer. It can be used, for example, to detect completion of a JNX or JPX loop.
$N_{2,1}$	$N_{2,1}$ is the right half of the N register. It is especially useful for detecting a jump to a specified location.
K	K holds the current sequence number. It is often useful in conjunction with CSK below.
CSK	CSK - The Change Sequence Counter - will remain zero until a change of sequence occurs. A setting of 10 ($CSK_4 = 1$) detects a change into LIMBO, a setting of 1 is recommended if K is used for the Old Sequence; a setting of 6 if K is to be set to the New Sequence. In either case, the CSK cycle will be completed before the computer stops.
B, C, D, F, MT, IOI	These letters refer to open cables at Bays B, C, F, of Frame 2, the Mag Tape Frames (F6, 7, 8), and Frame 9 (IOI). They are used by the maintenance technicians for special conditions cooked up as the need arises.

5-5.1   Paper Tape Read-in Programs -

Plugboard Memory[*] (377740 - 377777) contains three standard programs. They are as
follows:

CODABO POINT	NAME OF PROGRAM	COMMENT
377770(8)	"Clear Memory" or "Smear Memory"	All of S,T, and U Memory is set to +0 in the Left Half word and each register's own location in the right. Metabits are not changed. This program proceeds automatically to 377750 - Set Standard Config - and then to 377760 - Read in Reader Leader. (See below.)
377750(8)	Set Configuration	All of F Memory is set to the standard configurations and the program proceeds automatically to 377760 - Read in Reader Leader
377760(8)	Read In	This program reads the first 21 words from paper tape into registers 3 through 24 of S Memory, and then goes to register 3. All binary tapes start with the "Reader Leader", a block of 21 words that is the TX-2 Read in Program. The TX-2 Read-in program will read any standard binary block and check the sum-check at the end. (If the check fails, the program tries again.) The meta-bit of each word being stored is cleared. See listings below. See Section 6-3.4, page 6-23 for Binary Format.

* There are two plugboards: Plugboard "B" - registers 377740 to 377757, and Plugboard "A" -
registers 377760 to 377777.

"READER LEADER"

TX-2 Paper Tape Input Program [For Binary Format]

LOCATION	INSTRUCTION*	NUMERICAL FORM (OCTAL)	
0	0	000000	000000
1	1	000000	000001
2	2	000000	000002
3	$^{1}RSX_{54}5$	011154	000005
4	$^{36}JMP_{54}20$	360554	000020
5	$h^{2}RSX_{53}0$	421153	000000
6	$^{1}STE\ 11$	013000	000011
7	$^{36}JMP_{54}17$	360554	000017
10	$hLDE\ 0$	402000	000000
11	$STE_{53}34$	003053	000034
12	$h^{1}JNX_{53}7$	410753	000007
13	$^{36}JMP_{54}20$	360554	000020
14	$hJPX_{56}377760$	400656	377760
15	$hJNX_{56}377760$	400756	377760
16	$^{14}JPQ\ 27$	140500	000027
17	$^{2}MKZ_{4.12}400011$	021712	400011
20	$^{1}RSX_{57}3$	011157	000003
21	$hTSD\ 0$	405700	000000
22	$^{36}JPX_{57}21$	360657	000021
23	$^{1}AUX_{56}0$	011056	000000
24	$h^{2}AUX_{56}0$	421056	000000
25	$^{1}STE\ 16$	013000	000016
26	$^{15}BPQ_{54}0$	150554	000000
27	$^{1}IOS_{52}20000$	010452	020000

* Registers 0, 1, and 2 are not part of the Reader Leader itself, but are used as temporary storage.

PLUGBOARD PROGRAMS

LOCATION	INSTRUCTION	NUMERICAL FORM (OCTAL)	
377740	760,342,,340,000	760342	340000
377741	410,763,.762,761	410763	762761
377742	160,142,,140,411	160142	140411
377743	202,163,,162,1.61	202163	162161
377744	732,232,,230,200	732232	230200
377745	605,731,,730,733	605731	730733
377746	320,670,,750,600	320670	750600
377747	604,331,,330,333	604331	330333
377750	SPG 377740	002200	377740
377751	^{4}SPG 377741	042200	377741
377752	^{10}SPG 377742	102200	377742
377753	^{14}SPG 377743	142200	377743
377754	^{20}SPG 377744	202200	377744
377755	^{24}SPG 377745	242200	377745
377756	^{30}SPG 377746	302200	377746
377757	^{34}SPG 377747	342200	377747

Plugboard "B"

LOCATION	INSTRUCTION	NUMERICAL FORM (OCTAL)	
377760	^{1}SKX$_{54}$23	011254	000023
377761	REX$_{52}$377763	001252	377763
377762	^{21}IOS$_{52}$30106	210452	030106
377763	REX$_{53}$5	001253	000005
377764	hTSD$_{54}$26	405754	000026
377765	h^{36}JPX$_{53}$377764	760653	377764
377766	h^{1}JNX$_{54}$377763	410754	377763
377767	^{14}JPQ 3	140500	000003
377770	REX$_{77}$207777	001277	207777
377771	DPX$_{77}$777776	001677	777776
377772	^{14}JPQ 377773	140500	377773
377773	REX 777610	001200	777610
377774	h^{36}JPX$_{77}$377771	760677	377771
377775	^{30}SKN$_{4.12}$377744	301712	377744
377776	77,,0	000077	000000
377777	^{14}JPQ 377750	140500	377750

Plugboard "A"

[*] The X Memory is not changed, but each register is "exercised" to remove possible XPAL alarms.

5-5.3   Tape Preparation
         (See Group Report 51-8, dated 6 October 1959)

         An abbreviated manuscript and associated tape is shown below:

This end is inside the reel.
A yard or so of blank tape
is sufficient to reach the
reeler.

"End Mark" - i.e. code 73
(without seventh hole.)

About 2 inches for convenience
Initial Carriage Return

Initial Carriage Return

**TEST
LDA GG$_i$
ADD FT$_n$

Manuscript

Final Carriage Return

About 2 inches for convenience

STOP CODE - 76 With a 7th hole

About a foot of blank tape for
protection. This is the outside
end of the reel.

TABLE OF CONTENTS

CHAPTER 6

TX-2 UTILITY SYSTEM

6-1   INTRODUCTION - TYPICAL USE OF M4 ASSEMBLY PROGRAM

The TX-2 Assembly Program "M4" is a conventional symbolic assembler, but has considerable flexibility and two types of special features - Meta-Language for control of the program, and Macro Instructions, a feature that gives M4 the essential characteristics of a compiler. The symbolic tags for address sections can be nearly any combination of letters, symbols, and numerals (with a few restrictions). Tags used for the configuration and index syllables are nearly as flexible. M4 will assign all tags that have not been assigned by the user. The program is designed for on-line keyboard input and control as well as paper tape input. After checkout has started, a program can be kept in symbolic form in magnetic tape bulk storage.

Typical use of M4 begins with off-line tape preparation using a Lincoln Writer (See Group Report 51-8.). During debugging, the program can be preserved in symbolic and/or binary form on paper tape or in mag tape bulk storage as the user wishes. The symbolic form saved by the program is called a "DIRECTIVE" and is essentially the same as the original manuscript. Additions, insertions, relocation, rearrangement, and deletion are all handled by the M4 system - it is not necessary to retype the manuscript.

In addition to DIRECTIVE output (via Xerox, Lincoln Writer, Paper Tape, or Magnetic Tape), one can also get a LISTING (via Xerox, Lincoln Writer, or Punch). A LISTING is a copy of the program in absolute as well as symbolic format (side-by-side). It includes an alphabetically ordered tag table and a FORMAT ERROR notice if any errors were found. A LISTING can be obtained on punched paper tape for off-line Lincoln Writer printout, but this tape is not acceptable as input.

The binary form of the users program can be stored directly in the computer memory, punched in binary format on paper tape, or stored in magnetic tape bulk storage. When stored directly, either on mag tape or in memory, the M4 program area is protected and the storage may be incomplete. If a DIRECTIVE exists in core memory it too is protected.

6-1.1   MANUSCRIPT - DIRECTIVE - LISTING

A "Manuscript" is any program prepared off-line. It may exist in printed, hand written, or punched tape form.

A DIRECTIVE is the symbolic form created by M4. It may exist within M4 temporary storage, in magnetic tape bulk storage, or in printed or punched form. A DIRECTIVE closely resembles the manuscript. The following changes are worth noting:

1.) Any corrections and/or insertions have been made.

2.) All definitions and equalities are at the beginning. (Equalities may be anywhere on a manuscript.)

3.) Redundancies such as extra spaces are removed.

4.) Fractions are converted to the equivalent integer. The Numeral System is preserved.

5.) A check sum is added at the end.

A LISTING is a program output in absolute as well as symbolic format. The format is as follows:

> Tag Table (Alphabetical)
> Equalities
> Macro Definitions
> Format Errors
> Program (in symbolic and absolute)
> RC Words (unless the RC block location was specified within the program.)

(The Tag Table, Errors, absolute program, and RC block are not part of a Directive.)

6-1.2 META-LANGUAGE

The control of the M4 program is accomplished through the M4 Meta-Language instructions. All meta-language commands are to be preceeded by ⌐⌐ (two hands). When used on a manuscript, meta-commands are obeyed on read-in and do not appear on the directive (Except for those like ⌐⌐ RC, which is used to specify the location of the "RC Block" - (Register Containing).).

The basic types of Meta-Language Commands are:

> Input
> Correction-making
> Output
> Mag Tape
> Format
> Macro Definition
> Direct Storage
> Single Pushbutton

6-1.3 MACRO INSTRUCTIONS

A macro instruction is essentially a convenient flexible abbreviation for a similarly convenient and flexible subprogram. The user writes the subprogram once - with dummy parameters - as a "MACRO DEFINITION". Tags, and equalities used in the definition are kept separately and are not part of the program proper. When a macro instruction is used, only those parameters that are needed should be specified. The portions of the defining subprogram that refer to unspecified parameters are left out when the macro is converted. For this reason, and since the parameter of one macro can be the abbreviation for another, a different set of instructions will usually be compiled for each use.

Some standard macro instructions will be built into the M4 system. When they are used on a manuscript, the definition will appear on the M4 Directive and Listing. Since Macro Instructions can be redefined, it will not be necessary to avoid using the standard names except to avoid confusion when reading a program later on.

6-2 M4 PROGRAMMING LANGUAGE

6-2.1 INSTRUCTION WORDS

A TX-2 instruction word has 4 basic syllables and three special indicators as shown in the diagram below:
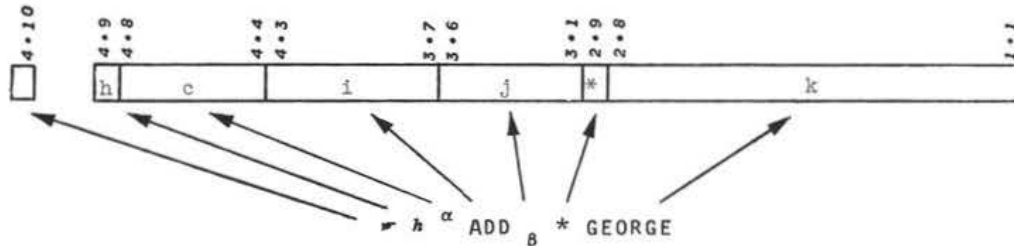
The three indicators are preassigned symbols as follows:

BIT 4·10   � = meta bit (not part of binary tape format)

BIT 4·9 h = hold bit

            $\overline{h}$ = no hold bit (Needed because h is automatically included
                                with LDE, ITE, JPX and JNX.)

BIT 2·9   * = defer bit

They must be in normal script, and may appear anywhere in the word.

The four basic syllables are as follows:

"c" - "Configuration" syllable - a 5 bit word (bits 4.4 - 4.8) used as the F memory address or as an extension of the instruction syllable (JMP, IOS, SKM, SKX). This syllable must be in superscript or preceeded by ‖ . It can be numerical or symbolic, but no spaces are allowed. For SKM, JMP, and SKX it is specified automatically by the supernumerary mnemonics. (See Table 7-3.)

"i" - "Instruction" syllable - a 6 bit word. This syllable is a normal script, 3 letter standard mnemonic abbreviation for the instruction. It is terminated by a space as well as the standard symex terminators. The mnemonic abbreviations include the configuration syllable as well for JMP, SKM, and SKX. (See Chart 7-3.) The instruction syllable may be specified numerically or with a normal symex but in these cases it is not terminated by a space. A regular symex terminator must be used. (See 6-2.2, rule #8.)

"j" - "Index" syllable - a 6 bit word used as the X memory address, i.e., the index register tag. (Except for SKM where it is used for bit designation.) The "j" syllable is normally in subscript. It may be a numeral or a symex, but no spaces are allowed.

"k" - "Base Address" - a 17 bit word. The base address may be symbolic or numerical and spaces may be used as part of a symex. It is given in normal script. Redundant spaces are removed upon conversion.

It is not necessary to use the order shown above. Any ordering is allowed if the script and symex conventions are carefully used. For example:

$$^{\alpha}ADD \; T_j \quad \text{or} \quad ADD^{\alpha}T_j \quad \text{or} \quad ADD \; T_j \| \alpha \quad \text{or} \quad ^{\alpha}T_j \; ADD$$

6-2.2 SYMEX DEFINITION - TAGS - EQUALITIES - AUTOMATIC ASSIGNMENT

A "Symex" is a symbolic expression. It is converted to a numeral by M4 when a program is stored, punched in binary format, or listed. A "TAG" is a symex used as a name for a place in a program. A tag is always terminated by an arrow (→), and is set to the numerical location of the word that it tags.

A symex can be set equal to a numeral directly - e.g., "apple = 6", or to any 36 bit word. For example a symex may be set equal to an instruction. When such a symex is used as the instruction syllable in a normal word, it must be terminated by a symex terminator - not by "space" (Only Standard mnemonics are terminated by space.)

A symex that is not used as a Tag nor defined by equal sign will be assigned by M4 according to its use within the program. See chart below:

Unassigned Symexes:

Used to Specify:	Automatically Assigned as or to:
Configuration Only	Zero
Index Only	The lowest numerical index register value not already used. (Except Zero and no higher than 77.)
Configuration and Index	Zero
Address only	The numerical memory location of the next place in the RC words block. The contents of this RC word are set to zero. This provision is useful in assigning temporary storage.
Configuration and Address	Zero
Config., Index, and Address	Zero
Index and Address	Same as Index Only.
Origin (i.e., Memory Location of Block)	"N", where N is the numer of words in the program including the RC Block.

A symex assigned by M4, or by equals sign may be redefined at any point in a program manuscript, and the latest definition will be used throughout. If the symex was initially assigned as a Tag - i.e., with an arrow, a re-definition will be recorded as a double definition error, and will be accepted, but not corrected. The only way to remove it from the directive is to use meta-language (REPLACE) and refer to it with a relative address based on a different tag.

6-2.3 RULES FOR SYMEX FORMATION

1. A symex must contain at least one non-numerical character.
2. It may contain as many legal characters as desired.
3. The single letters A, B, C, D and E are preassigned to the numerals 377604 - 377610. (i.e., the AE addresses.)
4. The three letter mnemonic instruction abbreviations can not be used as symexes.
5. The preassigned abbreviations and single letters can be used as part of a symex if they are not separately terminated. Note that space bar terminates op codes and single letter AE addresses but does not terminate other symexes.

e.g. "ATYPE" or "TYPE A" are allowed.

"A TYPE" is equivalent to "377604 + TYPE".

"ADDY" is allowed - "ADD Y" is not. (ADDY is a legitimate Symex -
ADD Y is a two syllable instruction.)

6. The legitimate symbols are:

    0 1 2 3 4 5 6 7 8 9

    A through Z

    α β γ Δ ϵ λ

    i j k n p q t w x y z   (NOT *h*)

    · (PERIOD) ' (APOSTROPHE)

    _ ‾ □ O   and Space Bar.

7. Compound Characters are allowed when the following restrictions are applied:

    Only one backspace.

    Two or three characters only.

    Space bar is allowed.

    Any sequence of characters is legal. (Except ⊘ , ⊖ , ⊕)

8. The following symbols terminate symexes:

    ⇒ / × ∨ ∧

    ⊂ ≡ ~ < > ∪ ∩

    , (COMMA) = → |

    # *h* + – ?

    SCRIPT CHANGES, TAB, CARRIAGE RETURN, COLOR CHANGES,
LINE FEED UP, DOWN

    { }) (‖ * Σ

6-2.4 NUMERICAL FORMAT - USE OF COMMAS

M4 will accept integers or fractions in Decimal or Octal. It will not accept
mixed numbers, except as a SKM bit designation. The details are handled by the
position of the period. See chart below:

Periods	Numeral Type	Example	Equivalent Octal Integer			
None	Octal Integer	431	000	000	000	431
Preceeding	Octal Fraction	.431	214	400	000	000
Following	Decimal Integer	431.	000	000	000	657
Both	Decimal Fraction	.431.	156	254	020	303
Centered	SKM Bit Designation	2.10	000	000	000	052
				(i.e., 10 1010_2)		

Note: The SKM Designation is usually given in subscript and is, therefore, moved
to bits 3.1 - 3.6.

Note: a)   The meta-command ☛DECIMAL reverses the meaning of a "following" period. ☛OCTAL restores it to the above.

   b)   The two parts of an SKM Bit Designation are Decimal integers.

   c)   Numbers may be preceeded by Plus (+) or Minus (-). For example: (Octal Integer 400 000 000 000 = - 377 777 777 777.)

   d)   M4 converts fractions to Integer form for Directives. The Numeral System is preserved.

Commas are used to specify separate subwords as follows:

   a)   The word is set to +0. (Any unspecified portions will therefore stay at +0.)

   b)   The word is assembled from left to right.

   c)   The numeral (or any other word) is converted to a 36 bit binary word.

   d)   This 36 bit word is inserted into Memory according to the comma chart.

COMMA CHART

COMMAS BEFORE	COMMAS AFTER	CRAM DIAGRAM	EXAMPLE			
0	0		444	333	222	111
0	1		111	-	-	-
0	2		222	111	-	-
0	3		333	222	111	-
1	0		-	-	-	111
1	1		-	222	111	-
1	2		-	111	-	-
1	3		444	333	222	-
2	0		-	-	222	111
2	1		-	-	111	-
2	2		222	111	444	333
2	3		-	-	444	333
3	0		-	-	-	111
3	1		-	-	-	444
3	2		-	444	-	-
3	3		-	-	444	333

For example: To specify 1/2 in each quarter, write

           200   200   200   200
    or   200,   200,, 200,   200
    or   ,$^{.4}$,,,$^{.4}$,,$^{.4}$,,,$^{.4}$,

To specify an instruction in the right half word as well as its normal position
in the left, write

$$.\,^{\alpha}\text{LDA}_{\beta}\,,,,\,^{\alpha}\text{LDA}_{\beta}\,,,,$$

6-2.5 MEMORY LOCATION OF PROGRAM - ORIGINS

The location of the first word of a block is to be specified in numerical or
symex form and is terminated by a <u>vertical bar</u>. It may be on a line by itself, as
it will be on DIRECTIVES, or it may preceed a normal word. The symex may be
recursive - i.e., set equal to another symex. If the recursive symex is circular
(i.e., eventually equal to itself,) or if it is undefined, the block will be located
incorrectly, and no alarm is generated. (It will appear to be located a "n", where
"n" is the number of words in the program, but the RC words are assigned as if it
were located at zero.) If there is no origin (i.e. no vertical bar), the whole
program is located (correctly) at 200 000$_{(8)}$. See the examples below.

HAG6|	HAG6=77
	LDA BOSY
	STA HUPG
22JAC→	JPQ MOUSE

HAG6+100|	
	+413,,5
	−563,51
516|	LDE TOMM
	STE JERRY
H→	JPQ HEHE

If the origin is specified by a symex (as "HAG6" above), the block may be moved by
redefining the symex via equal sign. If the origin was assigned numerically (e.g.
516\| above) it can be changed by REPLACE, counting back from an honest tag. For
example, to move the block at 516 to location 5516, the proper metacommand would be
➤➤REPLACE   H−3   5516\|
Note that the Origin itself counts as a full line.

Since it is often more convenient to specify an operand directly rather than by
its address, M4 interprets any word within brackets, e.g. {3456} as an operand by
providing a register containing the bracketed word, and using its address wherever
the same word is used within brackets.   (Bracketed words are called "RC Words"
from "Register Containing".)

The "RC Block" is made up of RC words, i.e. bracketed expressions and temporary
storage assigned to unassigned symexes used in address sections.   It is located at
the end of the last program block unless otherwise specified by meta language.

Examples:

SUB {w}   "w" can be any 36 bit word.

                                  Words within the RC Block may be Tagged.   If
                                  such a word is changed via meta language, the
                                  change should be made where it is used (i.e. at
                                  "DICK") rather than inside the RC Block
DICK→ LDE {GEORGE→$^{\alpha}$LDA T$_j$}   (i.e. at "GEORGE").

                                  A change made within the RC Block will not
                                  appear on a subsequent DIRECTIVE.   To be
                                  lasting, such a change should be made outside
                                  the block.

━━REP   DICK   LDA {GEORGE→ 3}   **PERMANENT
━━REP   GEORGE   GEORGE→3   **TEMPORARY

"RC Words" may contain other RC Words - i.e. the brackets may be "nested".
For example:

LDA {LDE {-13}}

The brackets must balance - there must be as many right hand brackets as there are
left hand ones.

6-2.7 WORD ASSEMBLY

The address syllable is formed first as a 36 bit integer using normal integer arithmetic. It may contain parentheses, the arithmetic symbols: $+ - \times /$, and the logic symbols $\wedge$ ("and"), $\vee$ ("or"), $\otimes$ ("exclusive or"). The symbols are interpreted <u>from left to right</u>. (i.e. not quite "normal" algebra)

For example:

$$4G + 5R/6 \qquad \text{is interpreted as } \frac{4G + 5R}{6}$$

$$4G + (5R/6) \qquad \text{is interpreted as } 4G + \frac{5R}{6}$$

Parenthetical expressions may be nested, but the parentheses must balance - i.e. there must be as many left parentheses as right. For example:

$$(77 - (4G + (5R/6)))$$

The use of another symex is equivalent to using parentheses. e.g.,

$$4G + \alpha \text{ is interpreted as } 4G + \frac{5R}{6} \text{ if } \alpha = 5R/6$$

The 36 bit address syllable is <u>united</u> (inclusive OR) with the others (configuration, operation, index). Extra syllables of the latter group are also <u>united</u> into the word. The one bit syllables are set last, "not hold" ($\bar{h}$) being the final one.

6-2.8 SPECIAL SYMBOLS

"The current Location" - The symbol "#" is a special symex which always is equal to the current location. Thus "JMP #" is a jump to itself, "JMP # + 1" is a "jump-to-the-next-register". If # is used within brackets, it refers to the RC BLOCK rather than the current address.

**   "Start of Comments" - A double asterisk - ** - is used before comments or annotations. All symbols are legal in the comment section except ⟜⟜XXX and }, and comments are saved and included in Listing and Directive Printouts. A carriage return terminates the comment section. A comment may be used within an RC word and another on the same line outside the bracket.

$p|q$ The notation $_{p|q}T$ is equivalent to $_p\{T_q\}*$. The $p|q$ must be in subscript.

For example: $REX_{\alpha|\beta}TAGG$ is equivalent to $REX_\alpha\{TAGG_\beta\}*$.

-   A carriage return immediately preceded by minus sign (-) will not terminate the line. This feature is needed because complex nested Macro-instructions often require more than one line of print. It can not be used for comments.

The M4 conversion - assembly process is designed for input and control from the keyboard or from paper tape.   Since keyboard use is more flexible and tape is faster, the normal procedure is to use both.   "Metacommands" are instructions directed to the M4 program itself covering the following areas:

> Paper Tape Input
> Alterations
> M4 Output
> Direct Storage
> Format Variation
> Magnetic Tape Bulk Storage
> Macro Definition (Section 6-4)

6-3.1 METACOMMAND FORMAT

Metacommands may require one line or several, and no address section, or as many as two.   The address section refers to the "Directive" of a program and may specify one line, or a block of consecutive lines.   Note that a line of a directive may correspond to several program words (c.f. MACRO, Section 6-4) or no program words at all (e.g., origins and comments).

There are three formats for address sections:

1.
   AA   - The line at "AA".

   AA+n   - The nth line after AA.

   AA−n   - The nth line before AA.

   "AA" should be an honest tag (defined by an arrow), rather than a numeral or symex defined by equal sign.

2.
   AA|n   n lines beginning with the line at AA

   AA±q|n   (or AA ± q).

3.
   AA → BB   The block of lines from AA to BB including AA but <u>not</u> <u>BB</u>.   (Or from AA ± q to BB ± p.)

A typical metacommand is as follows:

   ⚏REP   GEORGE+7   hSTE TT

It will replace the line at "GEORGE + 7" with the word "hSTE TT".

Similarly, the meta-command

⌐MOVE   AA→BB   GEORGE

will move the block "AA to BB" to just before George.

To reduce typing, the following conventions are allowed:

1. **#**   is equivalent to the end of the program.
2. **AA→**   means "From before AA to the end".   (Do not use this
                with DELETE.)
3.   **→AA**   means "From the beginning up to AA".   (Not including AA)
4. The name of a metacommand can be abbreviated to just the first three
    letters.
5. Tab is used to terminate syllables.

6-3.2 M4 OPERATION - NAME, CLEAN, LW READ, RECONVERT, STORE, GOTO

There are two versions of the M4 program - "M4 from mag. tape" on the <u>Golden</u>
<u>Reel</u> and "M4 from paper" on the <u>White Reel</u>.   They are essentially identical except
that the paper tape version will ignore all metacommands that use magnetic tape
bulk storage.

M4 is located at registers 160 000 - 174 010 (octal) and uses the rest of
memory as temporary storage.   157 777 down towards zero is used for storage of
your Directive.   174 000 - 200 000 is used for various tables.

There are two CODABO start points:

160 000 - Fresh start, new program, M4 is reset completely, a New Name
          is required.
160 001 - Continue same old program.   M4 is reset to OCTAL, and T = TAB.
          (see section 6-3.4)

Upon read-in of either reel, M4 will type NAME and then will wait for the user
to respond by typing his "M4 Name" on the keyboard.   The situation is the same as
that produced by the metacommand "NAME" described below:

⌐NAME   DAD

NAME is the metacommand used to identify the user.   The users "name" is
required by M4 to identify output and to determine which part of the M4
Magnetic Tape is to be used.   All paper tape, and printed output is identified

by the user's M4 initials and a 4 character word derived from the check sum of his directive.

When "M4 from tape" is used, the users name determines which portion of Bulk Storage is available.   Most individual users are assigned tape storage equivalent to one S memory.   Groups of users may combine by using the same name, thereby extending their available bulk storage to several memories. Bulk storage is used in blocks of $200_{(8)}$ words.   See section 6-3.7 for further details.

Note:
1. M4 names are three characters long.
2. The special name "FRE" is reserved for users who have no name.
3. "NAME" does not clear M4.

⇝CLEAN

CLEAN (CL)   restores M4 to its pristine state.   It is equivalent to CODABO 160 000 except that CODABO 160 000 types NAME and waits for one, while CLEAN does not.

⇝LW READ

LW READ (LW) (Special Key "Readin") switches M4 input from Keyboard to PETR.   The PETR will be in its "Bin and Read" mode ($IOS_{52}30104$).   When stop code (76) is read by the PETR, M4 input is switched back to the keyboard. All Metacommands may be used on tape except ⇝ READ and ⇝ RECONVERT.

⇝RECONVERT

RECONVERT (REC)   expands the stored directive into free storage in a form similar to its printed or punched version.   It then forms a new directive. The result is the same as punching a paper tape directive, cleaning M4, and reading in the new directive.   All redundancies are removed, the RC Block is corrected, - the directive is shorter than before.   When there is insufficient free storage, this command will produce QSAL and do nothing.

⇝BINARY STORE   AA

BINARY STORE (BIN), (Special Key "BEGIN") completes the conversion process and stores the program directly in memory.   If an address "AA" is given, a "h JPQ AA" is performed with sequence number 70 selected.   The user can therefore save the return point, run his program, and return to M4 automatically.   "AA" must be within the program area.

"BINARY STORE" does not destroy M4 or the directive.   If a program is to be located within these areas of storage, a binary tape should be made - see section 6-3.4 under "PUNCH BINARY".

⇝GO TO   AA

"GO TO" executes an "h JPQ AA" with sequence 70 selected.   It makes it possible to go to the user's program and return automatically to M4.   The address "AA" must be within the program area.   NOTE: M4 leaves STANDARD CON-FIGURATIONS in F memory only when its use is terminated by GOTO or BIN.   When the user starts his own program with CODABO F register #37 is not standard.

```
►►INSERT        AA      ONE LINE
    OR
►►INSERT        AA

    ≡≡≡

►►END
```

INSERT (INS) puts the new program lines <u>just</u> <u>before</u> "AA".   When only one line is to be inserted or when the next line will be a metacommand from this group, the terminating command "END" is not needed.

```
►►DELETE       AA

►►DELETE       AA→BB

►►DELETE       AA|n
```

DELETE (DEL) removes a section of the directive.   Symexes that were assigned within the removed area are now undefined.   The RC words used only within the deleted area are <u>not</u> removed from the RC Block.   If an Origin is removed, subsequent words are located relative to the Origin preceding the deleted area.

```
►►REPLACE      AA
►►REPLACE      AA→BB
►►REPLACE      AA|n
```

REPLACE (REP) is a combination of DELETE and INSERT.   It can remove an arbitrary number of lines and put a different arbitrary set in their place. Note that it deals with <u>lines</u> - not words.   For example, to replace an in-struction at a tagged location, one must be sure to replace the tag also, for the whole line is removed.   If only one line is to be inserted, the new line can be typed on the same line as the REP instruction.   If several are required they must go on succeeding lines and END (or another meta-command from this group) must be used as a terminator.   See examples below.

```
►►REP          G6          G6→ᵅLDA T_j      (To correct one line.)

►►REP          AP→AP+6

----                                       (To replace one set of lines
----                                        with a different set.)
----

►►END
```

```
►►MOVE         AA          CC
►►MOVE         AA→BB       CC
►►MOVE         AA|n        CC
```

MOVE (AA to CC in this case) is a combination of delete and insert where nothing is lost.

In the example below, a few "second thoughts" have been indicated in hand written form.

← ** SAMPLE PROGRAM For User's Handbook

```
START|                  START=400

TEST→           SKZ SKN₃.₁ PR
COM  Comm→      JPQ BL              ³⁰SKX₆₆ #+1
                                    IOS₆₆ 30000
OUT→            TSD TABL_α
                                    SKZ₃.₃ IND
MKN₃.₁ PR                           JPQ OOPS        BL = 1000
                                                    OOPS = 2000
RE→             LDA {0} {-1}
                STA RESET
```

The correction tape for the above is shown below.   A complete listing of the corrected program is given in the next section.

```
BL=1000
OOPS=2000
►►REP           TEST                TEST→SKZ₃.₁PR
►►REP           OUT-1→OUT+1
COMM→           JPQ BL
                ³⁰SKX₆₆#+1
                IOS₆₆30000
OUT→            TSD TABL_α
                SKZ₃.₃IND
                JPQ OOPS
►►END
►►REP           RE                  RE→LDA {-1}
►►INS           TEST-1      **SAMPLE PROGRAM FOR USERS HANDBOOK
```

Note: The symex "START" could not be used in the last metacommand because it is not an honest, arrow-defined tag.   To insert before an origin, one must count back from an honest tag.

☞LIST	AA	
☞LIST	AA→BB	LIST (LI) (Special Key - WORD EXAM) - List pro-
☞LIST	AA|n	duces a "M4 Listing" via the Xerox Printer.
☞PLIST	AA	
☞PLIST	AA→BB	Plist produces a listing via Punched paper tape.
☞PLIST	AA|n	
☞TYPE	AA	
☞TYPE	AA→BB	Type produces a listing via the Lincoln Writer.
☞TYPE	AA|n	

A LISTING tells what M4 did with the Manuscript or Directive received. Macro instructions are given in expanded form. The program is given in OCTAL as well as symbolic and is usually preceded by various tables. The overall format is as follows:

<div align="center">LISTING FORMAT</div>

SYMEX TABLE	Given only when the first word of the
EQUALITIES TABLE	Program is included in the area requested
MACRO DEFINITIONS	by the Metacommand.
ERRORS	

PROGRAM:

    SYMBOLIC   ABSOLUTE

RC BLOCK

SUMCK   (Paper tape versions only.)

The SYMEX TABLE is printed in octal, in three columns and is in alphabetical order (4 letters) as if there were but one page. Symexes with an asterisk were assigned by M4 because they were left unassigned on the manuscript, or because they were used within a Macro Definition. Symexes with a hand (☞) are MACRO names. Symexes used only within macros, and those defined by equals sign, are listed, but no equivalent numeral is given.

The EQUALITIES TABLE is in one column in alphabetic order. (First Letter) Symexes with an asterisk are those assigned by M4. An equality definition can not be deleted but it will be replaced if repeated. The last definition is the one that is used, and such repetition does not constitute an error.

MACRO DEFINITIONS are listed as defined originally. They can not be deleted or changed but they may be re-defined.

The ERROR PRINTOUT is of two types - FORMAT ERROR and DOUBLE ASSIGNMENT. An error printout is given on the first listing only - it is not repeated on subsequent listings even though the error may still exist. A line containing a FORMAT ERROR is reprinted in the error printout and the error deleted from the Directive. (It therefore can not reoccur.) A DOUBLE ASSIGNMENT occurs when a symex is assigned both by equal sign and by arrow, or twice by arrow. The printout gives the location of the first assignment. The second assignment is used and can be found in the symex table. If both definitions were by arrow, the offending tag will appear in two places on the directive and on the listing, but only the last assignment will be used throughout the program and in the Symex table. To delete the first one, it is necessary to count from some other tag. If the second one was wrong, it can be deleted directly, and the first one will take over in subsequent Listings and Directives.

The error printout format is shown by the examples below:

DOUBLE DEFINITION ‖   **BADTAG**   ****FIRST LOCATION**

FORMAT ERROR ‖   **LOCATION**   ****BAD LINE**

Format Errors include such things as an attempt to define an op code (e.g. LDA = DAD 6), extra meta-commands (especially END), and improper symbols in MACRO DEFINITIONS. In meta-commands, the line is removed; in other cases, only the offending character is deleted.

The PROGRAM is printed in symbolic form at the left, on two lines if necessary. The octal numeral form is given at the right with its octal memory location.

The SUMCHECK occurs only on paper tape versions of Listings and Directives (Plist and Dir). It is given as a meta-command, e.g. ⇀SUM   **436521** The sum is checked on M4 Read-in (⇀LW READ) and if an error is found, the word "SUMCK" is printed on the Lincoln Writer. The user can proceed at his own risk, or he can try again.

⇀ERRORS

ERRORS will type the error block if there are less than 8, or if there are more, it will print them on the Xerox. Once this is done, the error block will not appear on subsequent listings.

Example 1

Here is a listing of the program of the previous section - <u>before</u> the correction tape was used.

BL=000407	RE=000404	TABL=000412
OUT=000402	RESET=000411	TEST=000400
PR=000410		
START= 400		

$\alpha= 1$

FORMAT ERROR‖ TEST+1**COM→		JPQ BL

START‖			
TEST→	SKN $_{3 \cdot 1}$ PR	|301761 000410|000400
	COM JPQ BL	|145700 000407| 401
OUT→	TSD TABL$_\alpha$	|005701 000412| 402
	MKN $_{3 \cdot 1}$ PR	|031761 000410| 403
RE→	LDA { 0}	|002400 000406| 404
	STA RESET	|003400 000411| 405
	0	|000000 000000| 406
BL→	0	|000000 000000| 407
PR→	0	|000000 000000|000410
RESET→	0	|000000 000000| 411
TABL→	0	|000000 000000| 412

↠SUM | 037602

Example 2

Here is a listing of the same example, made _after_ the correction tape was used.   Note that the error notice is always printed once, even if the error is corrected later on.   If a listing is made _between_ tapes, the error notice is not included on the second listing.

```
                          OUT=000404
         COMM=000401      PR=000415              TABL=000417
         IND=000414       RE=000410              TEST=000400
                          RESET=000416

         BL= 1000


         OOPS= 2000


         START= 400


         α= 1


         FORMAT ERROR‖ TEST+1**COM→             JPQ BL


                          **SAMPLE PROGRAM FOR USERS HANDBOOK
         START|
         TEST→            SKZ 3.1 PR            |201761 000415|000400
         COMM→            JPQ  BL               |140500 001000|  401
                          30 SKX 66 #+1         |301266 000403|  402
                          IOS 66 30000          |000466 030000|  403
         OUT→             TSD  TABL α           |005701 000417|  404
                          SKZ 3.3 IND           |201763 000414|  405
                          JPQ  OOPS             |140500 002000|  406
                          MKN 3.1 PR            |031761 000415|  407
         RE→              LDA  {-1}             |002400 000413|000410
                          STA  RESET            |003400 000416|  411


                          0                     |000000 000000|  412
                          -1                    |777777 777776|  413
         IND→             0                     |000000 000000|  414
         PR→              0                     |000000 000000|  415
         RESET→           0                     |000000 000000|  416
         TABL→            0                     |000000 000000|  417



         ☛SUM             056632
```

☞ DIR	AA	
☞ DIR	AA→BB	Directive via Punched paper tape.
☞ DIR	AA | n	
☞ TDIR	AA	
☞ TDIR	AA→BB	Directive via Typewriter.
☞ TDIR	AA | n	
☞ LDIR	AA	
☞ LDIR	AA→BB	Directive via Xerox.
☞ LDIR	AA | n	

The format of a DIRECTIVE is as follows:

EQUALITIES TABLE	(No Asterisks or Hands)
MACRO DEFINITIONS	
SYMBOLIC PROGRAM	
☞☞ SUMCHECK	432170 (Looks like a Meta-command)

There is no symex table, error table, octal program, or RC Word block. The sumcheck is a 6 digit octal number. If the tape has been damaged or spliced so that the sumcheck is wrong, M4 will type "SUMCK" on the Lincoln Writer, and control of M4 will return to the Keyboard.

Partial Listings and Directives

If no address section is given in the Listing and Directive type meta-commands, the entire program is included in the output. In a partial Listing or Directive, the tables and macro definitions are included only if the first word of the program is included in the section indicated by the address given.

(Directive <u>Before</u> Corrections)

```
START= 400

  START|
TEST→              SKN  3.1  PR
                   COM  JPQ  BL
OUT→               TSD  TABL_α
                   MKN  3.1  PR
RE→                LDA  { 0}
                   STA  RESET



►►SUM              011516
```

(Directive <u>After</u> Corrections)

```
BL= 1000

OOPS= 2000

START= 400


                   **SAMPLE PROGRAM FOR USERS HANDBOOK
  START|
TEST→              SKZ  3.1  PR
COMM→              JPQ  BL
                   30 SKX  66 #+1
                   IOS  66  30000
OUT→               TSD  TABL_α
                   SKZ  3.3  IND
                   JPQ  OOPS
                   MKN  3.1  PR
RE→                LDA  {-1}
                   STA  RESET



►►SUM              021762
```

☛☛PUNCH
☛☛PUNCH   AA

PUNCH produces a punched paper tape in Binary Format.   If an address is given, a JPQ AA goes into register $27_{(oct.)}$.   If there is no given address, JPD 27 is used.   Since the readin program ends with an $IOS_{52}20000$ (DISCONNECT PETR) in register $26_{(oct.)}$ the "AA" is essentially a starting address for the tape.   AA must be within the program.

Binary Tape Format is as shown below:

To the Bin   To the Reeler
←——   ——→

Last		First	Start	Reader	
·Program· · · · · · · · · · · · · · · · · ·		·Program· · · ·	Block for	· · · · · Leader · · · · · · · · ·	
Block*		Block*	Register $27_{(oct)}$	(23 words)	

7th Hole

End Mark
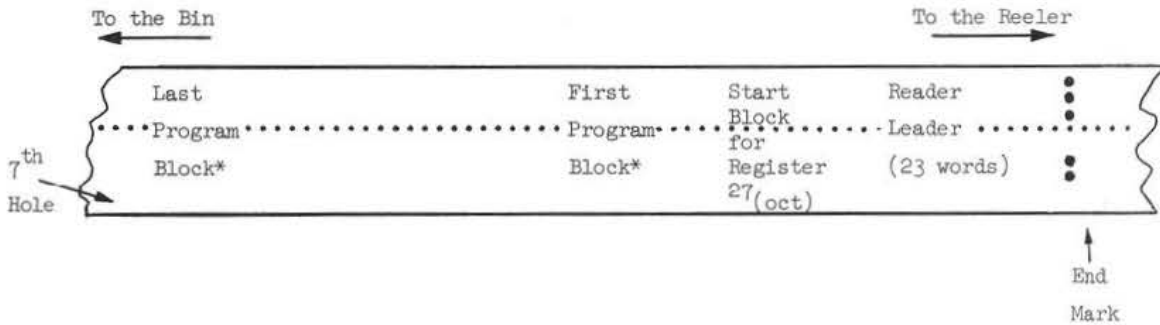
*There is, in general, one block per M4 Origin.

The "Reader Leader" is the binary version of the Readin Program itself.   The Readin Plugboard Program reads the reader leader into registers $0-26_{(octal)}$ and then jumps to register 3, the start of the readin program.   The first word of a block tells where the block is to be located, and how many words are therein.   The last word is an 18 bit check sum and an address telling the readin program where to go next.   For all but the last block, this address will be "3", the start of the readin program.   For the last block it will be $26_{(oct.)}$.   Register 26 contains "$IOS_{52}20000$" and 27 will be either JPQ AA (if AA is given) or ^{14}JPD 27 if no address was given.   The readin program therefore will jump to the new program with sequence number 52 chosen, and with the PETR DISCONNECTED.   If there is no given starting address, the readin program leaves the computer in LIMBO.

Note: Metabits are not set by the readin program.   They are cleared wherever the readin program stores new words.

Note also: The special block for register 27 comes before the program on the tape.   Program material that is to go in register 27 therefore super-cedes this special block created by M4.

6-3.5 M4 FORMAT VARIATIONS - DEC, OCT, T=CR, T=TAB, RC STORE, XXX

➤➤DECIMAL   (➤➤DEC)
➤➤OCTAL   (➤➤OCT)

All numerals in an M4 manuscript are considered to be of the same numeral system unless they are followed by a period (in which case, the other numeral system is used).   DEC and OCT remain effective until changed and are not saved for inclusion in the Directive.   M4 preserves Decimal Integers by reproducing them with a "following period".   Octal is the "normal" mode - CLEAN or CODABO 160000 (Pushbutton) will reset the numeral interpretation to Octal.   Note also that the right hand numeral in a SKM bit designation is always Decimal.

➤➤T=CR
➤➤T=TAB

When a table of constants is part of a manuscript, it may be easier to type it in several columns by "tabulation".   "T=CR" allows this by making TAB a word termi-nator similar to Carriage Return (CR).   In this mode, tabs can not be used with-in words.   T=TAB returns M4 to normal.

Note: These meta-commands are not included on a Directive.   A table that was typed in several columns will be reproduced as one column on the Directive.

➤➤RC STORE   (➤➤RC)

RC STORE means "Put the RC BLOCK here".   The RC Block will be at the end of the program if this meta-command is not used.   RC STORE can be inserted via ➤➤INS and need not be deleted for M4 automatically deletes an existing RC STORE when a new one is inserted.

⟵⟵XXX (or Special Key "NO")

XXX has complete control. It wipes out all input information back to the previous carriage return and forward to the next CR (or tab if T=CR). It can be used on any line, anywhere even within a comment, but not after backspace.

6-3.6 USE OF SPECIAL KEYS

The top row of keys on the Lincoln Writer has five special keys which have no machine function or associated character. They are extras, and can be used for any purpose. The M4 system uses them as abbreviations for certain meta-commands. These special keys terminate the line as far as M4 is concerned. If an address is to be used, it is to be typed first. The keys are assigned as follows:

WORD EXAM (71)	LIST
YES (17)	MOST RECENT OF TYPE, TDIR, DIR, LIST, PLIST, OR LDIR.
NO (16)	XXX (This works backwards only - It does not delete forward.)
BEGIN (15)	BIN (BINARY STORE)
READ IN (14)	LW READ
STOP (76)	The stop key means "stop". It is always active. If M4 is printing too much, if it is "hung-up", or if it is performing a function that is no longer wanted, the stop key will return control to the keyboard and stop whatever is going on.

The M4 Magnetic Tape Bulk Storage reel contains a copy of the M4 program itself (in Binary and Directive form, and with 3 Binary back-up copies), and working space about 30 times the size of S Memory. Working space is assigned in 200 word blocks, the average allotment being 1000 blocks - i.e. one S Memory. M4 does all the detailed tape coding. A standard tape format is used with an 18 bit check sum for error detecting. Tape malfunction is automatically reported on the typewriter - the report includes missed data detected by the alarm circuits (Sequence #41) as well as check sum errors. The users "M4 Name" is used to determine the proper working space, and M4 automatically protects the rest of the tape. The four Meta-commands listed below permit the user to store and retrieve his program or other material in Directive Format (Save-Read) or in straight Binary (Tape-Core).

M4 Answers to Mag Tape Meta-Commands

When M4 can not perform the given command because of programming limitations, it types NO in red on the Lincoln Writer, and it ignores the command. No data is transferred.

When M4 can and does complete the given command, it tells the user which tape area was used, and the associated four character identification derived from the check-sum. The tape area is specified by the block number of the first block and the block number of the next free block. For example:

$$0100 - 0107 \qquad TY7J$$

means that blocks 100 through 106 were used. "TY7J" is the identifying word.

If there is a tape equipment malfunction while M4 is in operation, the words "TAPE ERROR" are printed in red on the Lincoln Writer followed by pertinent data. This print-out should be saved for the Tape Engineers, and the incident should be duly reported.

SAVE   *100-200*   ****SAVE CURRENT DIRECTIVE**

If all is well, M4 stores the current directive on Mag tape beginning at the specified block number (100 in this case), and reports back via the typewriter the tape area used, and the four character identification. This will be the same identification as that used on Listings. Saving and retrieving a Directive on Magnetic tape does not "clean it up" the way it does when paper tape is used. The

Directive comes back exactly as it was. To "clean it up", the meta-command "RECONVERT" should be used before SAVE. The second block number is protected. If the directive will not fit in the specified area, M4 types "NO" in red. If the second number is omitted, the remainder of the user's allotted working space is assumed to be available. If both numbers are omitted, address "zero" is assumed. If M4 types "NO", the command has been ignored.

⇝READ   *100*   ****READ DIRECTIVE FROM MAG TAPE**

If the address given is the start of a Directive, "READ" will clean M4 and read in the Directive from tape. It will then type the tape location (i.e. first block and the one after the last block), and the four character identification.

If the address given is not the start of a Directive, "READ" types "NO" in red and does not clean M4 nor read from tape.

TAPE - CORE (Binary Storage and Retrieval)

TAPE and CORE deal with Binary (i.e. Absolutely Numerical) information and therefore require two address sections - one for the working space on tape and the other for core memory. (The word "to" is understood - i.e. it is TO TAPE and TO CORE.) Both address sections must be numerical and the tape address comes first.

⇝TAPE   *200-300*   *0-17777*
                    (TAPE AREA)   (MEMORY AREA, INCLUSIVE)

TAPE copies from core to Tape. Working space on tape is used a block at a time - i.e. in $200_{(8)}$ word sections. If the second tape block number is omitted, the rest of the user's allotted working space is assumed to be available. If the data from core will not fit, none of it is copied and M4 types "NO" in red. If it does fit, M4 types the first block number, the one after the last, and the four character identification derived from the check sum that is used on tape. The rest of a partially filled block is set to zero. The memory address is inclusive.

⇝CORE   *100*   *0-17777*
                    (TAPE AREA)   (MEMORY AREA, INCLUSIVE)

CORE copies from tape to core until the specified core area is full. If all is well, M4 will type the usual message - i.e. tape area used, and four character identification word. If the meta-command asks for words beyond the user's allotted

space, or if the M4 program itself is threatened, M4 types "NO" in red.   (M4 is located at 160000 - 174010.)   The identification word will be different if only part of a section is retrieved or if a program is stored in several pieces and retrieved by one command.

6-3.8   META-COMMAND SUMMARY

Clean		
LW Read		Input
Reconvert		
Name		

Insert		
Delete		Changes
Replace		
Move		

List		
Type		
Plist		
Directive punch		
Tdir		Output
Ldir		
Binary Store		
Punch Binary		
Goto		

Decimal		
Octal		
T=CR		
T=Tab		Format
End		
RC		
Sumck		

Save		
Read		Mag Tape
Tape		
Core		

Define		Macro
EMD		

Demo

BIN	- Binary Store
CLEAN	- Clear M4 Directive Storage
CORE	- Tape to Core
DEC	- Decimal
DEF	- Define
DEL	- Delete
DEMO	- Demonstrate
DIR	- Directive Punch
END	- End of Multiple Word Meta-Command
EMD	- End of Macro Definition
GOTO	- Go To User's Program
INS	- Insert
LDIR	- List Directive on Xerox
LIST	- Print Listing on Xerox
LW	- Lincoln Writer Read-in
MOVE	- Move Program Block
NAME	- Set User Identification
OCT	- Octal
PLIST	- Punch Listing
PUN	- Punch Binary
RC	- RC Store
READ	- Read Directive from Tape
REC	- Reconvert
REP	- Replace
SAVE	- Save Directive on Tape
SUMCK	- Sum Check
TAPE	- Core to Tape
TDIR	- Type Directive
TYPE	- Type Listing
T=CR	- Tab equals Carriage Return
T=TAB	- Tab equals Tab.

A macro-instruction is an abbreviation for a <u>flexible subprogram</u> which is written by the <u>user</u> (as a Macro Definition) and is inserted into the program by M4 wherever the Macro Instruction is used.   The subprogram is written in terms of dummy parameters and when it is copied by M4, only those portions that correspond to <u>specified</u> parameters are used. For example:

If the <u>definition</u> of "DO A,B,C,D" is

```
►►DEF                    DO|A,B,C,D
                         A
                         B
                         B
                         C
                         D
►►EMD
```

And if the <u>program</u> is:

```
100|
LINE 1→                  DO|LDA T_J,ADD TT, ADD BB, STA CC
LINE 2→                  DO|LDB T_α
LINE 3→                  DO|,6
```

Then M4 will produce:

```
100|
LINE 1→         DO|LDA  T_J,ADD   TT,ADD   BB,STA   CC
                LDA  T_J           |002401 000112|000100
                ADD    TT          |006700 000113|    101
                ADD    TT          |006700 000113|    102
                ADD    BB          |006700 000110|    103
                STA    CC          |003400 000111|    104
LINE 2→         DO|LDB  T_α
                LDB  T_α           |002502 000112|    105
LINE 3→         DO,6
                6                  |000000 000006|    106
                6                  |000000 000006|    107
```

6-4.1   Macro Definitions — Meta-commands "DEFINE" and "EMD"

As shown by the example above, two meta-commands are used with Macro Definitions — ➤➤ DEFINE (DEF is enough) and ➤➤ EMD (End of Macro Definition.)   A macro definition has two parts — the abbreviation itself, and the defining subprogram.   The Macro Definition must precede the use of a Macro instruction.

6-4.2   The Mnemonic Abbreviation Line of A Macro Definition

A Macro Definition starts with the "Macro Name" and dummy parameters as follows:

> ➤➤DEF   DO|A,B,C,D

The "Macro Name" here is "DO", the "dummy parameters" are A,B,C and D, and commas were used as "Macro Terminators".   A Macro Definition must be terminated by the Meta-command " ➤➤EMD" (End of Macro Definition).

6-4.3   Macro Names

There are two kinds:

a.   Any Symex may be used as a Macro Name.   It may be used alone, or followed by a terminator and parameters, (each of which is separated from the other by termi-nators).

b.   A compound character may be used.   It may consist of two or three superposed non-alpha-numeric characters — e.g., ⊞ or ≠ or ➤.   It may not be ⊘, ⊕, or ⊖. (These are reserved for M4.)   The characters may be typed in any order.   A compound macro name is itself a terminator — i.e. parameters may come before as well as after.   For example:

> ➤➤DEF   A , α ⊞ β →DOUG
> -------
> -------
> -------
> ➤➤EMD

The Macro Name is ⊞ .

6-4.4 Dummy Parameters

Dummy Parameters may be any symex (even three letter mnemonic codes and the single letters A, B, C, D and E). A Dummy Parameter may be included as a mnemonic aid and need not be used in the defining subprogram. Dummy Parameters must be separated by macro terminators.

6-4.5 Macro Terminators

The following symbols may be used.

$$\blacktriangleright \; , \; = \; \rightarrow \; | \; \supset \; \equiv \; \sim \; < \; > \; \cap \; \cup \; / \; \times \; \vee \; \wedge$$

Other symbols may not be used.

6-4.6 The Defining Subprogram

The defining subprogram is written using the Dummy Parameters and must be terminated by ►► EMD (End of Macro Definition). Note the following rules and conventions.

1. Symexes defined by equal sign (=) or by arrow (→) within the macro definition are not part of the program proper and refer only to the macro subprogram.

2. A symex that is not defined within the MACRO will refer to the main program and if it's not assigned there, it will be assigned automatically in the RC Block. (But only if the Macro is used in the program proper.)

3. The single letter symexes A, B, C, D, E will refer to the AE unless they are used as Dummy Parameters.

4. An instruction in a definition may use a parameter harmlessly so that it will be left out when the parameter is not used. One way to do this is as follows:

$$\text{LDA }^{\alpha} \text{T }_j \; + \; (DP) \; - \; (DP)$$

("DP" is the dummy parameter.)

5. A Dummy Parameter may not be used as a tag within the defining subprogram. (You can, of course, write JPQ DP, but not:

$$DP \rightarrow \qquad\qquad \text{LDA }^{\alpha}T_j$$

6. A line that uses two Dummy Parameters will be left out if either is left out when the macro is used.

7. A line that uses a dummy parameter may be kept in with that parameter equal to zero when the parameter is not used. This is done by using another symex that is set equal to the dummy in question. For example:

```
        LDA DUM1 +DUM2
```

can be written as:

```
        LDA DUM1 + G6
        G6 = DUM2
```

6-4.7 Use of Macro Instructions

A macro may be used as a Pseudo Instruction by itself, or "nested" as a parameter of another macro. It may even be used as a parameter of itself. It may be an RC word. When used as an RC Word, it will use several registers of the RC Block and the location of the <u>first</u> of these will be the associated address. Consider the examples below.

Example 1. A Macro used within brackets — i.e. as an "RC Word"

```
➡DEF                   TBS|α
                        α
                        α
                        α
                        α
                        α
➡EMD
100|
USE➡                   LDA {TS➡TBS| 0}     **5 BLANK RC WORDS
                       LDA TOMM
                       STA TS+3
```

The program is expanded as follows:

```
100|
USE➡                   LDA {TS➡TBS|0}     |002400 000103|000100
                       LDA  TOMM          |002400 000110|   101
                       STA  TS+3          |003400 000106|   102
TS➡                    TBS|0
                        0                 |000000 000000|   103
                        0                 |000000 000000|   104
                        0                 |000000 000000|   105
                        0                 |000000 000000|   106
                        0                 |000000 000000|   107
TOMM➡                   0                 |000000 000000|000110
```

Example 2. A Macro used to generate a table of squares.

      If the manuscript is as follows:

            ▸▸DEF   SQ|A

                                        A

                                      A

            ▸▸EMD

            100| NSQ= (#−TABL) × (#−TABL)

            TABL→   SQ| (SQ| (SQ| (NSQ)))

M4 will produce the program shown in the "Plisting" below.

                                   TABL=000100

NSQ= (#−TABL) × (#−TABL)

   ▸▸DEF   SQ|A

                       A

                       A

   ▸▸EMD

    100|

   TABL→   SQ| (SQ| (SQ| (NSQ)))

                 (NSQ) |000000 000000|000100

                 (NSQ) |000000 000001| *101*

                 (NSQ) |000000 000004| *102*

                 (NSQ) |000000 000011| *103*

                 (NSQ) |000000 000020| *104*

                 (NSQ) |000000 000031| *105*

                 (NSQ) |000000 000044| *106*

                 (NSQ) |000000 000061| *107*

   ▸▸SUM   *025015*

Example 3. An open subroutine for index memory "integer" multiplication.

The macro below finds the righthand 18 bits of the full product of two X Memory words (α & β), provided that said product is no larger than 17 bits and sign. The product goes into X Register "α", "TXX" is cleared, X Register "β" is ruined, and the symexes "TXX" and "FX1" are "used up". (Since symex "S" is defined within the macro, it is <u>not</u> "used up".)

		TXX=000110		USE=000100

FX1= 1

S=

```
DEF        MULX,α×β
           DPX TXX
          ¹EXXₐ TXX
          ⁴SZZ₁.₁ TXX
S→         INXₐ|β
           INXᵦ|β
           RSX_FX1 TXX
           JPX_FX1 S
EMD
```

100|

```
USE→       MULX,1×2
           DPX   TXX        |001600 000110|000100
          ¹EXX ₁ TXX        |011401 000110|  101
          ⁴SZZ ₁.₁ TXX      |261721 000110|  102
           INX ₁|2          |021201 400107|  103
           INX ₂|2          |021202 400107|  104
           RSX_FX1 TXX      |001101 000110|  105
           JPX_FX1 S        |400601 000102|  106
```

```
                            |000002 000000|  107
              ²
TXX→          0             |000000 000000|000110
```

SUM   037643

Example 4. An open subroutine for "exclusive or" using a <u>compound macro name</u>.

In the macro below, the result goes into X Register "α", TXX is set to (α), $c(\beta)$, and X Register "β" is not changed. An underline was used in the macro name because the symbols $\circledcirc$, $\ominus$, and $\oplus$ are not available as macro names.

	TXX=000106	USE=000100

BILL= *1*

TOMM= *2*

➤➤DEF $\alpha\underline{\circledcirc}\beta$
$^1DPX_\beta TXX$
$ ^2 DPX_\alpha TXX$
$ ^{17} COM\ E$
$ ITE\ TXX$
$ RSX_\alpha E$
$ ^2 AUX_\alpha E$

➤➤EMD

100|

USE→   TOMM$\underline{\circledcirc}$BILL

$^1DPX_{BILL}\ TXX$	|011601 000106|000100
$ ^2DPX_{TOMM}\ TXX$	|421602 000106| 101
$ ^{17}COM\ E$	|575600 377610| 102
$ ITE\ TXX$	|404000 000106| 103
$ RSX_{TOMM}\ E$	|401102 377610| 104
$ ^2AUX_{TOMM}\ E$	|021002 377610| 105

TXX→   *0* |000000 000000| 106

➤➤SUM   *036551*

TABLE OF CONTENTS

IN-OUT SEQUENCE ASSIGNMENTS FOR TX-2

00	STARTOVER		
40		60	DISPLAY NO· 1
41	ALARM, IN-OUT	61	RANDOM NUMBER GENERATOR
42	TRAP	62	PUNCH NO· 2
43		63	PUNCH NO· 1
44		64	
45	IBM MAG TAPE	65	LINCOLN WRITER INPUT NO· 1
46	MAG TAPE BULK STORAGE	66	LINCOLN WRITER OUTPUT NO· 1
47	MISCELLANEOUS INPUTS	67	
50	DATRAC	70	
51	XEROX	71	LINCOLN WRITER INPUT NO· 2
52	PETR	72	LINCOLN WRITER OUTPUT NO· 2
53		73	
54	INTERVAL TIMER	74	PLOTTER
55	LITE PEN	75	MISCELLANEOUS OUTPUTS
56	DISPLAY NO· 2	76	
57		77	

Table 7-1   October 1961

TABLE 7-2
STANDARD CONFIGURATION SET

α	F|α|	DESCRIPTOR	α	F|α|	DESCRIPTOR
0	000	4 3 2 1	20	200	4 3·2 1
1	340	4 3·2 1	21	230	4 3·2 1
2	342	2 1·4 3	22	232	2 1·4 3
3	760	4·3·2·1	23	732	2·1·4·3
4	761	1·4·3·2	24	733	3·2·1·4
5	762	2·1·4·3	25	730	4·3·2·1
6	763	3·2·1·4	26	731	1·4·3·2
7	410	4 3 2·1	27	605	1·2·3·4
10	411	1 4 3·2	30	600	4·3·2·1
11	140	4 3 2 1	31	750	4·3·2·1
12	142	2 1 4 3	32	670	4·3·2·1
13	160	4 3 2 1	33	320	4 3·2 1
14	161	1 4 3 2	34	333	3 2·1 4
15	162	2 1 4 3	35	330	4 3·2 1
16	163	3 2 1 4	36	331	1 4·3 2
17	202	2 1·4 3	37	604	3·4·1·2

STANDARD PERMUTATIONS							
0	1	2	3	4	5	6	7

TABLE 7-2A[*]

STANDARD CONFIGURATION SET

α	$F\|\alpha\|$	DESCRIPTOR		α	$F\|\alpha\|$	DESCRIPTOR
0	000	4 3 2 1		20	200	4 3·2 1
1	340	4 3·2 1		21	230	4 3·2 1
2	342	2 1·4 3		22	232	2 1·4 3
3	760	4·3·2·1		23	671	1·4·3·2
4	761	1·4·3·2		24	672	2·1·4·3
5	762	2·1·4·3		25	673	3·2·1·4
6	763	3·2·1·4		26	604	3·4·1·2
7	410	4 3 2·1		27	605	1·2·3·4
10	411	1 4 3·2		30	600	4·3·2·1
11	140	4 3 2 1		31	750	4·3·2·1
12	142	2 1 4 3		32	730	4·3·2·1
13	160	4 3 2 1		33	670	4·3·2·1
14	161	1 4 3 2		34		To be assigned by the user
15	162	2 1 4 3		35		" " " " " "
16	163	3 2 1 4		36		" " " " " "
17	202	2 1·4 3		37		To be assigned by the user, but also used by Executive System. (So use hold bit.)

STANDARD PERMUTATIONS

0	1	2	3	4	5	6	7

Numerical Order	Alphabetical Order	Supernumerary Mnemonics

Numerical Order

```
0
1
2
3
4  - IOS
5  - JMP (1)
6  - JPX
7  - JNX
10 - AUX
11 - RSX
12 - SKX (2)
13
14 - EXX
15 - ADX
16 - DPX
17 - SKM (3)
20 - LDE
21 - SPF
22 - SPG
23
24 - LDA
25 - LDB
26 - LDC
27 - LDD
30 - STE
31 - FLF
32 - FLG
33
34 - STA
35 - STB
36 - STC
37 - STD
40 - ITE
41 - ITA
42 - UNA
43 - SED
44 - JOV
45
46 - JPA
47 - JNA
50
51
52
53
54 - EXA
55 - INS
56 - COM
57 - TSD
60 - CYA
61 - CYB
62 - CAB
63
64 - NOA
65 - DSA
66 - NAB
67 - ADD
70 - SCA
71 - SCB
72 - SAB
73
74 - TLY
75 - DIV
76 - MUL
77 - SUB
```

Alphabetical Order

```
ADD - 67
ADX - 15
AUX - 10
COM - 56
CAB - 62
CYA - 60
CYB - 61
DIV - 75
DPX - 16
DSA - 65
EXA - 54
EXX - 14
FLF - 31
FLG - 32
INS - 55
IOS -  4
ITA - 41
ITE - 40
JMP -  5
JNA - 47
JNX -  7
JOV - 44
JPA - 46
JPX -  6
LDA - 24
LDB - 25
LDC - 26
LDD - 27
LDE - 20
MUL - 76
NAB - 66
NOA - 64
RSX - 11
SAB - 72
SCA - 70
SCB - 71
SED - 43
SKM - 17
SKX - 12
SPF - 21
SPG - 22
STA - 34
STB - 35
STC - 36
STD - 37
STE - 30
SUB - 77
TSD - 57
TLY - 74
UNA - 42
```

Supernumerary Mnemonics

(1)
JMP - ^{0}JMP
BRC - ^{1}JMP
JPS - ^{2}JMP
BRS - ^{3}JMP
JPQ - ^{14}JMP
BPQ - ^{15}JMP
JES - ^{16}JMP
JPD - ^{20}JMP
BRD - ^{21}JMP
JDS - ^{22}JMP
BDS - ^{23}JMP

(2)
REX - SEX - ^{0}SKX
INX - ^{2}SKX
DEX - ^{3}SKX
SXD - ^{4}SKX
SXL - ^{6}SKX
SXG - ^{7}SKX
RXF - ^{10}SKX
RXD - ^{20}SKX
RFD - ^{30}SKX

(3)
SKM - ^{0}SKM

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

Skip
SKU - ^{10}SKM
SUC - ^{11}SKM
SUZ - ^{12}SKM
SUN - ^{13}SKM

Skip on Zero
SKZ - ^{20}SKM
SZC - ^{21}SKM
SZZ - ^{22}SKM
SZN - ^{23}SKM

Skip on One
SKN - ^{30}SKM
SNC - ^{31}SKM
SNZ - ^{32}SKM
SNN - ^{33}SKM

Rotate
CYR - ^{4}SKM
MCR - ^{5}SKM
MZR - ^{6}SKM
MNR - ^{7}SKM
SNR - ^{34}SKM
SZR - ^{24}SKM
SUR - ^{14}SKM

OPERATION CODE MNEMONICS

Table 7-3

7-4 META-COMMAND MNEMONICS

Clean			BIN — Binary Store
LW Read	} Input		CLEAN — Clear M4 Directive Storage
Reconvert			CORE — Tape to Core
Name			DEC — Decimal
			DEF — Define
Insert			DEL — Delete
Delete	} Changes		DEMO — Demonstrate
Replace			DIR — Directive Punch
Move			END — End of Multiple Word Meta-Command
			EMD — End of Macro Definition
List			GOTO — Go To User's Program
Type			INS — Insert
Plist			LDIR — List Directive on Xerox
Directive punch			LIST — Print Listing on Xerox
Tdir	} Output		LW — Lincoln Writer Read-in
Ldir			MOVE — Move Program Block
Binary Store			NAME — Set User Identification
Punch Binary			OCT — Octal
Goto			PLIST — Punch Listing
			PUN — Punch Binary
Decimal			RC — RC Store
Octal			READ — Read Directive from Tape
T=CR			REC — Reconvert
T=Tab	} Format		REP — Replace
End			SAVE — Save Directive on Tape
RC			SUMCK — Sum Check
Sumck			TAPE — Core to Tape
			TDIR — Type Directive
Save			TYPE — Type Listing
Read	} Mag Tape		T=CR — Tab equals Carriage Return
Tape			T=TAB — Tab equals Tab.
Core			
Define	} Macro		
EMD			
Demo			

XEROX PRINTER CHARACTER CODES

CHARACTER	OCTAL CODE		CHARACTER	OCTAL CODE
A	154		ʃ	122 (107)
B	142		k	324 (034)
C	361 (056)(071)(346)		n	323 (033)
D	352		p	024
E	313 (043)		q	111
F	344 (054)		t	112
G	302 (012)		w	173
H	354		x	174
I	172 (157)		y	163
J	144		z	164
K	143		α	310 (040)
L	332 (047)(062)(317)		β	311 (041)
M	360 (055)(070)(345)		γ	333 (063)
N	370 (355)		Δ	203
O	353		ε	334 (064)
P	312 (042)		λ	023
Q	160 (145)		1	001
R	371 (356)		2	002
S	322 (017)(032)(307)		3	003
T	153		4	004
U	362 (057)(072)(347)		5	020 (005)
V	152		6	021 (006)
W	343 (053)		7	022 (007)
X	161 (146)		8	300 (010)
Y	342 (052)		9	301 (011)
Z	162 (147)		0 (ZERO)	000
ʰ	132 (117)		?	202
↓	133		, (COMMA)	204
<	220 (205)		∩	120 (105)
>	221 (206)		⊃	121 (106)
· (PERIOD)	222 (207)		×	113
+	351		○ (CIRCLE)	714 (444)
−	372 (357)		~	373
∨	340 (050)		∧	341 (051)
'	363 (073)		#	364 (074)
		730 (445)(460)(715)		‖	731 (446)(461)(716)
Σ	703 (413)		□	704 (414)
}	720 (415)(430)(705)		{	721 (416)(431)(706)
☛	150		→	151
—	570 (555)		—	571 (556)
(	140		)	141
=	114		≡	130 (115)
/	131 (116)		*	102
∪	103		⊂	104

Note: Bit 1.9 of the Xerox Character Code is a "size control bit". "1" means large, and "0" means small. The codes are given above with the "proper" size.

Table 7-5   October 1961

TX-2 LINCOLN WRITER CODES

00	0	☛	40	Q	α
01	1	Σ	41	R	Δ
02	2	|	42	S	ϸ
03	3	‖	43	T	ϵ
04	4	/	44	U	ℏ
05	5	×	45	V	⊃
06	6	#	46	W	β
07	7	→	47	X	∧
10	8	<	50	Y	λ
11	9	>	51	Z	~
12	_	⁻	52	(	{
13	○	□	53	)	}
14	READ IN		54	+	≡
15	BEGIN		55	–	=
16	NO		56	,	'
17	YES		57	.	*
20	A	n	60	CAR RETURN	
21	B	c	61	TAB	
22	C	v	62	BACK SPACE	
23	D	q	63	COLOR BLACK	
24	E	γ	64	SUPER	
25	F	t	65	NORMAL	
26	G	w	66	SUB	
27	H	x	67	COLOR RED	
30	I	i	70	SPACE	
31	J	y	71	WORD EXAM	
32	K	z	72	LINE FEED DOWN	
33	L	?	73	LINE FEED UP	
34	M	∪	74	LOWER CASE	
35	N	∩	75	UPPER CASE	
36	O	j	76	STOP	
37	P	k	77	NULLIFY	

COMMAS BEFORE	COMMAS AFTER	CRAM DIAGRAM	EXAMPLE
0	0		444 333 222 111
0	1		111 - - -
0	2		222 111 - -
0	3		333 222 111 -
1	0		- - - 111
1	1		- 222 111 -
1	2		- 111 - -
1	3		444 333 222 -
2	0		- - 222 111
2	1		- - 111 -
2	2		222 111 444 333
2	3		- - 444 333
3	0		- - - 111
3	1		- - - 444
3	2		- 444 - -
3	3		- - 444 333

Table 7-7   October 1961

This duration chart was made by TX-2 by timing the duration of 8000 repetitions of each operation with various combinations of memories.   The columns are labled as follows:

P MEM	-	The memory used for the instructions.
OP Code	-	The instruction being timed.
Δ	-	The memory used for intermediate deferred address (if any).
Q MEM	-	The memory used for final operand (if any).
MMS	-	Average duration in microseconds.

The abbreviations used within the columns are as follows:

S	-	S memory
T	-	T memory
VFF	-	Flip-Flop part of V memory (A,B,C,etc.)
VT	-	Toggle Memory

The instructions are listed in numerical order (by op codes).

P MEM	OP CODE	Δ	Q MEM	MMS
S	AOP		167000	8·0
T	AOP		167000	6·0
S	IOS		0	9·2
T	IOS		0	7·2
S	JMP			7·6
T	JMP			5·6
S	JPA			8·0
T	JPA			6·0
S	JNA			8·0
T	JNA			6·0
S	JOV			8·0
T	JOV			6·0
S	JNX			9·6
T	JNX			7·6
S	JPX			9·6
T	JPX			7·6
S	SKX			10·0
T	SKX			8·0
S	SKX	S		20·4
T	SKX	S		18·4
S	SKX	T		18·4
T	SKX	T		16·4
S	AUX		S	13·6
T	AUX		S	11·6
S	AUX		VFF	12·4
T	AUX		VFF	10·4
S	AUX		VT	12·0
T	AUX		VT	10·0
S	AUX		T	11·6
T	AUX		T	9·6
S	RSX		S	12·8
T	RSX		S	10·8
S	RSX		T	10·8
T	RSX		T	8·8
S	RSX		VFF	11·6
T	RSX		VFF	9·6
S	RSX		VT	11·2
T	RSX		VT	9·2
S	ADX		S	16·0
T	ADX		S	10·0
S	ADX		T	10·0
T	ADX		T	12·0
S	ADX		VFF	10·8
T	ADX		VFF	8·8

Table 7-8, pg. 1   October 1961

P MEM	OP CODE	Δ	Q MEM	MMS
S	ADX		VT	10•4
T	ADX		VT	8•4
S	DPX		S	14•0
T	DPX		S	7•6
S	DPX		T	7•6
T	DPX		T	10•0
S	DPX		VFF	8•4
T	DPX		VFF	6•4
S	DPX		VT	8•0
T	DPX		VT	6•0
S	EXX		S	14•0
T	EXX		S	11•2
S	EXX		T	11•2
T	EXX		T	10•0
S	EXX		VFF	11•6
T	EXX		VFF	9•6
S	EXX		VT	11•2
T	EXX		VT	9•2
S	SKM		S	14•8
T	SKM		S	9•6
S	SKM		T	9•6
T	SKM		T	10•8
S	SKM		VFF	10•4
T	SKM		VFF	8•4
S	SKM		VT	10•0
T	SKM		VT	8•0
S	SKM	S	S	25•2
T	SKM	S	S	20•0
S	SKM	S	T	20•0
T	SKM	S	T	21•2
S	SKM	S	VFF	20•8
T	SKM	S	VFF	18•8
S	SKM	S	VT	20•4
T	SKM	S	VT	18•4
S	SKM	T	S	23•2
T	SKM	T	S	18•0
S	SKM	T	T	18•0
T	SKM	T	T	19•2
S	SKM	T	VFF	18•8
T	SKM	T	VFF	16•8
S	SKM		0	14•8
T	SKM		0	9•6
S	LDA		S	12•8
T	LDA		S	6•4

P MEM	OP CODE	Δ	Q MEM	MMS
S	LDA		T	6•8
T	LDA		T	8•8
S	LDA		VFF	6•8
T	LDA		VFF	5•2
S	LDA		VT	6•8
T	LDA		VT	4•8
S	LDB		S	12•8
T	LDB		S	6•4
S	LDB		T	6•8
T	LDB		T	8•8
S	LDB		VFF	6•8
T	LDB		VFF	5•2
S	LDB		VT	6•8
T	LDB		VT	4•8
S	LDC		S	12•8
T	LDC		S	6•4
S	LDC		T	6•8
T	LDC		T	8•8
S	LDC		VFF	6•8
T	LDC		VFF	5•2
S	LDC		VT	6•8
T	LDC		VT	4•8
S	LDD		S	12•8
T	LDD		S	6•4
S	LDD		T	6•8
T	LDD		T	8•8
S	LDD		VFF	6•8
T	LDD		VFF	5•2
S	LDD		VT	6•8
T	LDD		VT	4•8
S	LDE		S	12•8
T	LDE		S	6•4
S	LDE		T	6•8
T	LDE		T	8•8
S	LDE		VFF	6•8
T	LDE		VFF	5•2
S	LDE		VT	6•8
T	LDE		VT	4•8
S	SPF		S	12•8
T	SPF		S	9•6
S	SPF		T	9•6
T	SPF		T	8•8
S	SPF		VFF	10•4
T	SPF		VFF	8•4

Table 7-8, pg. 3   October 1961

P MEM	OP CODE	Δ	Q MEM	MMS
S	SPF		VT	10•0
T	SPF		VT	8•0
S	SPG		S	12•8
T	SPG		S	9•6
S	SPG		T	9•6
T	SPG		T	8•8
S	SPG		VFF	10•4
T	SPG		VFF	8•4
S	SPG		VT	10•0
T	SPG		VT	8•0
S	STA		S	14•0
T	STA		S	7•6
S	STA		T	6•8
T	STA		T	10•0
S	STA		VFF	6•8
T	STA		VFF	5•2
S	STA		VT	6•8
T	STA		VT	4•8
S	STB		S	14•0
T	STB		S	7•6
S	STB		T	6•8
T	STB		T	10•0
S	STB		VFF	6•8
T	STB		VFF	5•2
S	STB		VT	6•8
T	STB		VT	4•8
S	STC		S	14•0
T	STC		S	7•6
S	STC		T	6•8
T	STC		T	10•0
S	STC		VFF	6•8
T	STC		VFF	5•2
S	STC		VT	6•8
T	STC		VT	4•8
S	STD		S	14•0
T	STD		S	7•6
S	STD		T	6•8
T	STD		T	10•0
S	STD		VFF	6•8
T	STD		VFF	5•2
S	STD		VT	6•8
T	STD		VT	4•8
S	STE		S	14•0
T	STE		S	7•6

P MEM	OP CODE	Δ	Q MEM	MMS
S	STE		T	6·8
T	STE		T	10·0
S	STE		VFF	6·8
T	STE		VFF	5·2
S	STE		VT	6·8
T	STE		VT	4·8
S	EXA		S	14·0
T	EXA		S	7·6
S	EXA		T	6·8
T	EXA		T	10·0
S	EXA		VFF	6·8
T	EXA		VFF	5·2
S	EXA		VT	6·8
T	EXA		VT	4·8
S	ITA		S	12·8
T	ITA		S	6·4
S	ITA		T	6·8
T	ITA		T	8·8
S	ITA		VFF	6·8
T	ITA		VFF	5·2
S	ITA		VT	6·8
T	ITA		VT	4·8
S	UNA		S	12·8
T	UNA		S	6·4
S	UNA		T	6·8
T	UNA		T	8·8
S	UNA		VFF	6·8
T	UNA		VFF	5·2
S	UNA		VT	6·8
T	UNA		VT	4·8
S	DSA		S	12·8
T	DSA		S	6·4
S	DSA		T	6·8
T	DSA		T	8·8
S	DSA		VFF	6·8
T	DSA		VFF	5·2
S	DSA		VT	6·8
T	DSA		VT	4·8
S	ITE		S	12·8
T	ITE		S	6·4
S	ITE		T	6·8
T	ITE		T	8·8
S	ITE		VFF	6·8
T	ITE		VFF	5·2

Table 7-8, pg. 5   October 1961

P MEM	OP CODE	Δ	Q MEM	MMS
S	ITE		VT	6・8
T	ITE		VT	4・8
S	SED		S	12・8
T	SED		S	9・6
S	SED		T	9・6
T	SED		T	8・8
S	SED		VFF	10・4
T	SED		VFF	8・4
S	SED		VT	10・0
T	SED		VT	8・0
S	FLF		S	14・0
T	FLF		S	7・6
S	FLF		T	6・8
T	FLF		T	10・0
S	FLF		VFF	6・8
T	FLF		VFF	6・0
S	FLF		VT	6・8
T	FLF		VT	4・8
S	FLG		S	15・6
T	FLG		S	8・8
S	FLG		T	8・4
T	FLG		T	11・6
S	FLG		VFF	8・4
T	FLG		VFF	8・0
S	FLG		VT	8・4
T	FLG		VT	6・8
S	TSD		S	14・4
T	TSD		S	8・8
S	TSD		T	7・6
T	TSD		T	10・4
S	TSD		VFF	7・6
T	TSD		VFF	8・8
S	TSD		VT	7・6
T	TSD		VT	8・8
S	INS		S	15・2
T	INS		S	8・8
S	INS		T	6・8
T	INS		T	11・2
S	INS		VFF	6・8
T	INS		VFF	6・4
S	INS		VT	6・8
T	INS		VT	6・0
S	COM		S	14・8
T	COM		S	8・4

P MEM	OP CODE	Δ	Q MEM	MMS
S	COM		T	6·8
T	COM		T	10·8
S	COM		VFF	6·8
T	COM		VFF	6·4
S	COM		VT	6·8
T	COM		VT	6·0
S	ADD		S	12·8
T	ADD		S	6·4
S	ADD		T	6·8
T	ADD		T	8·8
S	ADD		VFF	6·8
T	ADD		VFF	5·6
S	ADD		VT	6·8
T	ADD		VT	4·8
S	SUB		S	12·8
T	SUB		S	6·4
S	SUB		T	6·8
T	SUB		T	8·8
S	SUB		VFF	6·8
T	SUB		VFF	5·6
S	SUB		VT	6·8
T	SUB		VT	4·8
S	MUL		S	20·8
T	MUL		S	20·8
S	MUL		T	19·6
T	MUL		T	20·0
S	MUL		VFF	19·6
T	MUL		VFF	20·8
S	MUL		VT	19·6
T	MUL		VT	19·2
S	[7]MUL		S	17·6
T	[7]MUL		S	17·6
S	[7]MUL		T	16·4
T	[7]MUL		T	15·2
S	[7]MUL		VFF	16·4
T	[7]MUL		VFF	16·0
S	[7]MUL		VT	16·4
T	[7]MUL		VT	16·0
S	[1]MUL		S	16·0
T	[1]MUL		S	14·4
S	[1]MUL		T	11·6
T	[1]MUL		T	12·0
S	[1]MUL		VFF	13·2
T	[1]MUL		VFF	12·8

Table 7-8, pg. 7   October 1961

P MEM	OP CODE	Δ	Q MEM	MMS
S	[1]MUL		VT	13·2
T	[1]MUL		VT	12·8
S	[3]MUL		S	12·8
T	[3]MUL		S	11·2
S	[3]MUL		T	10·0
T	[3]MUL		T	8·8
S	[3]MUL		VFF	10·0
T	[3]MUL		VFF	9·6
S	[3]MUL		VT	10·0
T	[3]MUL		VT	9·6
S	DIV		S	80·0
T	DIV		S	80·0
S	DIV		T	77·2
T	DIV		T	77·6
S	DIV		VFF	78·8
T	DIV		VFF	78·4
S	DIV		VT	77·2
T	DIV		VT	78·4
S	[7]DIV		S	60·8
T	[7]DIV		S	60·8
S	[7]DIV		T	59·6
T	[7]DIV		T	60·0
S	[7]DIV		VFF	59·6
T	[7]DIV		VFF	60·8
S	[7]DIV		VT	59·6
T	[7]DIV		VT	59·2
S	[1]DIV		S	43·2
T	[1]DIV		S	43·2
S	[1]DIV		T	42·0
T	[1]DIV		T	40·8
S	[1]DIV		VFF	42·0
T	[1]DIV		VFF	41·6
S	[1]DIV		VT	42·0
T	[1]DIV		VT	41·6
S	[3]DIV		S	22·4
T	[3]DIV		S	22·4
S	[3]DIV		T	19·6
T	[3]DIV		T	20·0
S	[3]DIV		VFF	21·2
T	[3]DIV		VFF	20·8
S	[3]DIV		VT	19·6
T	[3]DIV		VT	20·8
S	TLY		S	19·2
T	TLY		S	19·2

P MEM	OP CODE	Δ	Q MEM	MMS	
S	TLY		T	16·4	
T	TLY		T	16·8	
S	TLY		VFF	18·0	
T	TLY		VFF	17·6	
S	TLY		VT	18·0	
T	TLY		VT	17·6	
S	^{7}TLY		S	16·0	
T	^{7}TLY		S	16·0	
S	^{7}TLY		T	13·2	
T	^{7}TLY		T	13·6	
S	^{7}TLY		VFF	14·8	
T	^{7}TLY		VFF	14·4	
S	^{7}TLY		VT	13·2	
T	^{7}TLY		VT	14·4	
S	^{1}TLY		S	12·8	
T	^{1}TLY		S	11·2	
S	^{1}TLY		T	10·0	
T	^{1}TLY		T	12·0	
S	^{1}TLY		VFF	10·0	
T	^{1}TLY		VFF	11·2	
S	^{1}TLY		VT	10·0	
T	^{1}TLY		VT	9·6	
S	^{3}TLY		S	12·8	
T	^{3}TLY		S	8·0	
S	^{3}TLY		T	6·8	
T	^{3}TLY		T	8·8	
S	^{3}TLY		VFF	6·8	
T	^{3}TLY		VFF	8·0	
S	^{3}TLY		VT	6·8	
T	^{3}TLY		VT	8·0	
S	SCA		S	12·8	107·2
T	SCA		S	8·0	107·2
S	SCA		T	6·8	104·4
T	SCA		T	8·8	104·8
S	SCA		VFF	6·8	
T	SCA		VFF	8·0	
S	SCA		VT	6·8	106·0
T	SCA		VT	8·0	105·6
S	^{7}SCA		S	12·8	107·2
T	^{7}SCA		S	8·0	107·2
S	^{7}SCA		T	6·8	104·4
T	^{7}SCA		T	8·8	104·8
S	^{7}SCA		VFF	6·8	
T	^{7}SCA		VFF	8·0	

Table 7-8, pg. 9   October 1961

P MEM	OP CODE	Δ	Q MEM	MMS	
S	^{7}SCA		VT	6•8	106•0
T	^{7}SCA		VT	8•0	105•6
S	^{1}SCA		S	12•8	107•2
T	^{1}SCA		S	8•0	107•2
S	^{1}SCA		T	6•8	104•4
T	^{1}SCA		T	8•8	104•8
S	^{1}SCA		VFF	6•8	
T	^{1}SCA		VFF	8•0	
S	^{1}SCA		VT	6•8	106•0
T	^{1}SCA		VT	8•0	105•6
S	^{3}SCA		S	12•8	107•2
T	^{3}SCA		S	8•0	107•2
S	^{3}SCA		T	6•8	104•4
T	^{3}SCA		T	8•8	104•8
S	^{3}SCA		VFF	6•8	
T	^{3}SCA		VFF	8•0	
S	^{3}SCA		VT	6•8	106•0
T	^{3}SCA		VT	8•0	105•6
S	SCB		S	12•8	107•2
T	SCB		S	8•0	107•2
S	SCB		T	6•8	104•4
T	SCB		T	8•8	104•8
S	SCB		VFF	6•8	106•0
T	SCB		VFF	8•0	105•6
S	SCB		VT	6•8	106•0
T	SCB		VT	8•0	105•6
S	SAB		S	12•8	107•2
T	SAB		S	8•0	107•2
S	SAB		T	6•8	104•4
T	SAB		T	8•8	104•8
S	SAB		VFF	6•8	
T	SAB		VFF	8•0	
S	SAB		VT	6•8	106•0
T	SAB		VT	8•0	105•6
S	^{7}SAB		S	12•8	107•2
T	^{7}SAB		S	8•0	107•2
S	^{7}SAB		T	6•8	104•4
T	^{7}SAB		T	8•8	104•8
S	^{7}SAB		VFF	6•8	
T	^{7}SAB		VFF	8•0	
S	^{7}SAB		VT	6•8	106•0
T	^{7}SAB		VT	8•0	105•6
S	^{1}SAB		S	12•8	107•2
T	^{1}SAB		S	8•0	107•2

P MEM	OP CODE Δ	Q MEM	MMS	
S	[1]SAB	T	6·8	104·4
T	[1]SAB	T	8·8	104·8
S	[1]SAB	VFF	6·8	
T	[1]SAB	VFF	8·0	
S	[1]SAB	VT	6·8	106·0
T	[1]SAB	VT	8·0	105·6
S	[3]SAB	S	12·8	107·2
T	[3]SAB	S	8·0	107·2
S	[3]SAB	T	6·8	104·4
T	[3]SAB	T	8·8	104·8
S	[3]SAB	VFF	6·8	
T	[3]SAB	VFF	8·0	
S	[3]SAB	VT	6·8	105·0
T	[3]SAB	VT	8·0	105·6
S	CYB	S	12·8	107·2
T	CYB	S	8·0	107·2
S	CYB	T	6·8	104·4
T	CYB	T	8·8	104·8
S	CYB	VFF	6·8	106·0
T	CYB	VFF	8·0	105·0
S	CYB	VT	6·8	106·0
T	CYB	VT	8·0	105·6
S	[7]CYB	S	12·8	107·2
T	[7]CYB	S	8·0	107·2
S	[7]CYB	T	6·8	104·4
T	[7]CYB	T	8·8	104·8
S	[7]CYB	VFF	6·8	106·0
T	[7]CYB	VFF	8·0	105·6
S	[7]CYB	VT	6·8	106·0
T	[7]CYB	VT	8·0	105·6
S	[1]CYB	S	12·8	107·2
T	[1]CYB	S	8·0	107·2
S	[1]CYB	T	6·8	104·4
T	[1]CYB	T	8·8	104·8
S	[1]CYB	VFF	6·8	106·0
T	[1]CYB	VFF	8·0	105·6
S	[1]CYB	VT	6·8	106·0
T	[1]CYB	VT	8·0	105·6
S	[3]CYB	S	12·8	107·2
T	[3]CYB	S	8·0	107·2
S	[3]CYB	T	6·8	104·4
T	[3]CYB	T	8·8	104·8
S	[3]CYB	VFF	6·8	106·0
T	[3]CYB	VFF	8·0	105·6

Table 7-8, pg. 11   October 1961

P MEM	OP CODE	Δ	Q MEM	MMS	
S	[3]CYB		VT	6·8	106·0
T	[3]CYB		VT	8·0	105·6
S	CYA	·	S	12·8	107·2
T	CYA		S	8·0	107·2
S	CYA		T	6·8	104·4
T	CYA		T	8·8	104·8
S	CYA		VFF	6·8	38·2
T	CYA		VFF	8·0	38·8
S	CYA		VT	6·8	106·0
T	CYA		VT	8·0	105·6
S	CAB		S	12·8	107·2
T	CAB		S	8·0	107·2
S	CAB		T	6·8	104·4
T	CAB		T	8·8	104·8
S	CAB		VFF	6·8	
T	CAB		VFF	8·0	38·8
S	CAB		VT	6·8	106·0
T	CAB		VT	8·0	105·6
S	NOA		S	19·2	12·8
T	NOA		S	19·2	8·0
S	NOA		T	18·0	6·8
T	NOA		T	18·4	8·8
S	NOA		VFF	18·0	6·8
T	NOA		VFF	19·2	8·0
S	NOA		VT	18·0	6·8
T	NOA		VT	17·6	8·0
S	NAB		S	33·6	12·8
T	NAB		S	33·6	8·0
S	NAB		T	32·4	6·8
T	NAB		T	32·8	8·8
S	NAB		VFF	32·4	6·8
T	NAB		VFF	33·6	8·0
S	NAB		VT	32·4	6·8
T	NAB		VT	32·0	8·0

TX-2 USERS HANDBOOK

CHAPTER 7 - VARIOUS TABLES

TABLE OF CONTENTS

IN-OUT SEQUENCE ASSIGNMENTS FOR TX-2

00 STARTOVER	
40	60 DISPLAY NO· 1
41 ALARM, IN-OUT	61 RANDOM NUMBER GENERATOR
42 TRAP	62 PUNCH NO· 2
43	63 PUNCH NO· 1
44	64
45 IBM MAG TAPE	65 LINCOLN WRITER INPUT NO· 1
46 MAG TAPE BULK STORAGE	66 LINCOLN WRITER OUTPUT NO· 1
47 MISCELLANEOUS INPUTS	67
50 DATRAC	70
51 XEROX	71 LINCOLN WRITER INPUT NO· 2
52 PETR	72 LINCOLN WRITER OUTPUT NO· 2
53	73
54 INTERVAL TIMER	74 PLOTTER
55 LITE PEN	75 MISCELLANEOUS OUTPUTS
56 DISPLAY NO· 2	76
57	77

Table 7-1   October 1961

TABLE 7-2

STANDARD CONFIGURATION SET

α	F|α|	DESCRIPTOR	α	F|α|	DESCRIPTOR
0	000	4 3 2 1	20	200	4 3·2 1
1	340	4 3·2 1	21	230	4 3·2 1
2	342	2 1·4 3	22	232	2 1·4 3
3	760	4·3·2·1	23	732	2·1·4·3
4	761	1·4·3·2	24	733	3·2·1·4
5	762	2·1·4·3	25	730	4·3·2·1
6	763	3·2·1·4	26	731	1·4·3·2
7	410	4 3 2·1	27	605	1·2·3·4
10	411	1 4 3·2	30	600	4·3·2·1
11	140	4 3 2 1	31	750	4·3·2·1
12	142	2 1 4 3	32	670	4·3·2·1
13	160	4 3 2 1	33	320	4 3·2 1
14	161	1 4 3 2	34	333	3 2·1 4
15	162	2 1 4 3	35	330	4 3·2 1
16	163	3 2 1 4	36	331	1 4·3 2
17	202	2 1·4 3	37	604	3·4·1·2

STANDARD PERMUTATIONS

0	1	2	3	4	5	6	7

Numerical Order	Alphabetical Order	Supernumerary Mnemonics

Numerical Order

```
0
1
2
3
4  - IOS
5  - JMP  (1)
6  - JPX
7  - JNX
10 - AUX
11 - RSX
12 - SKX  (2)
13
14 - EXX
15 - ADX
16 - DPX
17 - SKM  (3)
20 - LDE
21 - SPF
22 - SPG
23
24 - LDA
25 - LDB
26 - LDC
27 - LDD
30 - STE
31 - FLF
32 - FLG
33
34 - STA
35 - STB
36 - STC
37 - STD
40 - ITE
41 - ITA
42 - UNA
43 - SED
44 - JOV
45
46 - JPA
47 - JNA
50
51
52
53
54 - EXA
55 - INS
56 - COM
57 - TSD
60 - CYA
61 - CYB
62 - CAB
63
64 - NOA
65 - DSA
66 - NAB
67 - ADD
70 - SCA
71 - SCB
72 - SAB
73
74 - TLY
75 - DIV
76 - MUL
77 - SUB
```

Alphabetical Order

```
ADD - 67
ADX - 15
AUX - 10
COM - 56
CAB - 62
CYA - 60
CYB - 61
DIV - 75
DPX - 16
DSA - 65
EXA - 54
EXX - 14
FLF - 31
FLG - 32
INS - 55
IOS -  4
ITA - 41
ITE - 40
JMP -  5
JNA - 47
JNX -  7
JOV - 44
JPA - 46
JPX -  6
LDA - 24
LDB - 25
LDC - 26
LDD - 27
LDE - 20
MUL - 76
NAB - 66
NOA - 64
RSX - 11
SAB - 72
SCA - 70
SCB - 71
SED - 43
SKM - 17
SKX - 12
SPF - 21
SPG - 22
STA - 34
STB - 35
STC - 36
STD - 37
STE - 30
SUB - 77
TSD - 57
TLY - 74
UNA - 42
```

Supernumerary Mnemonics

(1)
- JMP - ^{0}JMP
- BRC - ^{1}JMP
- JPS - ^{2}JMP
- BRS - ^{3}JMP
- JPQ - ^{14}JMP
- BPQ - ^{15}JMP
- JES - ^{16}JMP
- JPD - ^{20}JMP
- BRD - ^{21}JMP
- JDS - ^{22}JMP
- BDS - ^{23}JMP

(2)
- REX - SEX - ^{0}SKX
- INX - ^{2}SKX
- DEX - ^{3}SKX
- SXD - ^{4}SKX
- SXL - ^{6}SKX
- SXG - ^{7}SKX
- RXF - ^{10}SKX
- RXD - ^{20}SKX
- RFD - ^{30}SKX

(3)
- SKM - ^{0}SKM

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

Skip
- SKU - ^{10}SKM
- SUC - ^{11}SKM
- SUZ - ^{12}SKM
- SUN - ^{13}SKM

Skip on Zero
- SKZ - ^{20}SKM
- SZC - ^{21}SKM
- SZZ - ^{22}SKM
- SZN - ^{23}SKM

Skip on One
- SKN - ^{30}SKM
- SNC - ^{31}SKM
- SNZ - ^{32}SKM
- SNN - ^{33}SKM

Rotate
- CYR - ^{4}SKM
- MCR - ^{5}SKM
- MZR - ^{6}SKM
- MNR - ^{7}SKM
- SNR - ^{34}SKM
- SZR - ^{24}SKM
- SUR - ^{14}SKM

OPERATION CODE MNEMONICS

Table 7-3   October 1961

Clean		
LW Read	}	Input
Reconvert		
Name		

Insert		
Delete	}	Changes
Replace		
Move		

List		
Type		
Plist		
Directive punch		
Tdir	}	Output
Ldir		
Binary Store		
Punch Binary		
Goto		

Decimal		
Octal		
T=CR		
T=Tab	}	Format
End		
RC		
Sumck		

Save		
Read	}	Mag Tape
Tape		
Core		

Define	}	Macro
EMD		

Demo

BIN	-	Binary Store
CLEAN	-	Clear M4 Directive Storage
CORE	-	Tape to Core
DEC	-	Decimal
DEF	-	Define
DEL	-	Delete
DEMO	-	Demonstrate
DIR	-	Directive Punch
END	-	End of Multiple Word Meta-Command
EMD	-	End of Macro Definition
GOTO	-	Go To User's Program
INS	-	Insert
LDIR	-	List Directive on Xerox
LIST	-	Print Listing on Xerox
LW	-	Lincoln Writer Read-in
MOVE	-	Move Program Block
NAME	-	Set User Identification
OCT	-	Octal
PLIST	-	Punch Listing
PUN	-	Punch Binary
RC	-	RC Store
READ	-	Read Directive from Tape
REC	-	Reconvert
REP	-	Replace
SAVE	-	Save Directive on Tape
SUMCK	-	Sum Check
TAPE	-	Core to Tape
TDIR	-	Type Directive
TYPE	-	Type Listing
T=CR	-	Tab equals Carriage Return
T=TAB	-	Tab equals Tab.

XEROX PRINTER CHARACTER CODES

CHARACTER	OCTAL CODE		CHARACTER	OCTAL CODE	
A	154		j	122 (107)	
B	142		k	324 (034)	
C	361 (056)(071)(346)		n	323 (033)	
D	352		p	024	
E	313 (043)		q	111	
F	344 (054)		t	112	
G	302 (012)		w	173	
H	354		x	174	
I	172 (157)		y	163	
J	144		z	164	
K	143		α	310 (040)	
L	332 (047)(062)(317)		β	311 (041)	
M	360 (055)(070)(345)		γ	333 (063)	
N	370 (355)		Δ	203	
O	353		ε	334 (064)	
P	312 (042)		λ	023	
Q	160 (145)		1	001	
R	371 (356)		2	002	
S	322 (017)(032)(307)		3	003	
T	153		4	004	
U	362 (057)(072)(347)		5	020 (005)	
V	152		6	021 (006)	
W	343 (053)		7	022 (007)	
X	161 (146)		8	300 (010)	
Y	342 (052)		9	301 (011)	
Z	162 (147)		0 (ZERO)	000	
h	132 (117)		?	202	
ł	133		, (COMMA)	204	
<	220 (205)		∩	120 (105)	
>	221 (206)		⊃	121 (106)	
• (PERIOD)	222 (207)		×	113	
+	351		○ (CIRCLE)	714 (444)	
–	372 (357)		~	373	
v	340 (050)		∧	341 (051)	
'	363 (073)		#	364 (074)	
|	730 (445)(460)(715)		‖	731 (446)(461)(716)	
Σ	703 (413)		□	704 (414)	
}	720 (415)(430)(705)		{	721 (416)(431)(706)	
▬	150		→	151	
—	570 (555)		—	571 (556)	
(	140		)	141	
=	114		≡	130 (115)	
/	131 (116)		*	102	
∪	103		⊂	104	

Note: Bit 1.9 of the Xerox Character Code is a "size control bit". "1" means large, and
"0" means small. The codes are given above with the "proper" size.

Table 7-5   October 1961

TX-2 LINCOLN WRITER CODES

00	0	☛	40	Q	α
01	1	∑	41	R	∆
02	2	|	42	S	ρ
03	3	‖	43	T	ε
04	4	/	44	U	ℏ
05	5	×	45	V	⊃
06	6	#	46	W	β
07	7	→	47	X	∧
10	8	<	50	Y	λ
11	9	>	51	Z	~
12	_	⎯	52	(	{
13	○	☐	53	)	}
14	READ IN		54	+	≡
15	BEGIN		55	−	=
16	NO		56	,	'
17	YES		57	.	*
20	A	n	60	CAR RETURN	
21	B	c	61	TAB	
22	C	∨	62	BACK SPACE	
23	D	q	63	COLOR BLACK	
24	E	γ	64	SUPER	
25	F	t	65	NORMAL	
26	G	w	66	SUB	
27	H	x	67	COLOR RED	
30	I	i	70	SPACE	
31	J	y	71	WORD EXAM	
32	K	z	72	LINE FEED DOWN	
33	L	?	73	LINE FEED UP	
34	M	∪	74	LOWER CASE	
35	N	∩	75	UPPER CASE	
36	O	ʝ	76	STOP	
37	P	k	77	NULLIFY	

COMMAS BEFORE	COMMAS AFTER	CRAM DIAGRAM	EXAMPLE
0	0		444 333 222 111
0	1		111 - - -
0	2		222 111 - -
0	3		333 222 111 -
1	0		- - - 111
1	1		- 222 111 -
1	2		- 111 - -
1	3		444 333 222 -
2	0		- - 222 111
2	1		- - 111 -
2	2		222 111 444 333
2	3		- - 444 333
3	0		- - - 111
3	1		- - - 444
3	2		- 444 - -
3	3		- - 444 333

Table 7-7   October 1961

7-8 AVERAGE DURATION OF INSTRUCTIONS

This duration chart was made by TX-2 by timing the duration of 8000 repetitions of each operation with various combinations of memories.   The columns are labled as follows:

    P MEM   - The memory used for the instructions.

  OP Code - The instruction being timed.

      Δ   - The memory used for intermediate deferred address (if any).

  Q MEM   - The memory used for final operand (if any).

    MMS   - Average duration in microseconds.

The abbreviations used within the columns are as follows:

    S   - S memory

    T   - T memory

  VFF - Flip-Flop part of V memory (A,B,C,etc.)

  VT   - Toggle Memory

The instructions are listed in numerical order (by op codes).

P MEM	OP CODE	Δ	Q MEM	MMS
S	AOP		167000	8·0
T	AOP		167000	6·0
S	IOS		0	9·2
T	IOS		0	7·2
S	JMP			7·6
T	JMP			5·6
S	JPA			8·0
T	JPA			6·0
S	JNA			8·0
T	JNA			6·0
S	JOV			8·0
T	JOV			6·0
S	JNX			9·6
T	JNX			7·6
S	JPX			9·6
T	JPX			7·6
S	SKX			10·0
T	SKX			8·0
S	SKX	S		20·4
T	SKX	S		18·4
S	SKX	T		18·4
T	SKX	T		16·4
S	AUX		S	13·6
T	AUX		S	11·6
S	AUX		VFF	12·4
T	AUX		VFF	10·4
S	AUX		VT	12·0
T	AUX		VT	10·0
S	AUX		T	11·6
T	AUX		T	9·6
S	RSX		S	12·8
T	RSX		S	10·8
S	RSX		T	10·8
T	RSX		T	8·8
S	RSX		VFF	11·6
T	RSX		VFF	9·6
S	RSX		VT	11·2
T	RSX		VT	9·2
S	ADX		S	16·0
T	ADX		S	10·0
S	ADX		T	10·0
T	ADX		T	12·0
S	ADX		VFF	10·8
T	ADX		VFF	8·8

Table 7-8, pg. 1   October 1961

P MEM	OP CODE	Δ	Q MEM	MMS
S	ADX		VT	10·4
T	ADX		VT	8·4
S	DPX		S	14·0
T	DPX		S	7·6
S	DPX		T	7·6
T	DPX		T	10·0
S	DPX		VFF	8·4
T	DPX		VFF	6·4
S	DPX		VT	8·0
T	DPX		VT	6·0
S	EXX		S	14·0
T	EXX		S	11·2
S	EXX		T	11·2
T	EXX		T	10·0
S	EXX		VFF	11·6
T	EXX		VFF	9·6
S	EXX		VT	11·2
T	EXX		VT	9·2
S	SKM		S	14·8
T	SKM		S	9·6
S	SKM		T	9·6
T	SKM		T	10·8
S	SKM		VFF	10·4
T	SKM		VFF	8·4
S	SKM		VT	10·0
T	SKM		VT	8·0
S	SKM	S	S	25·2
T	SKM	S	S	20·0
S	SKM	S	T	20·0
T	SKM	S	T	21·2
S	SKM	S	VFF	20·8
T	SKM	S	VFF	18·8
S	SKM	S	VT	20·4
T	SKM	S	VT	18·4
S	SKM	T	S	23·2
T	SKM	T	S	18·0
S	SKM	T	T	18·0
T	SKM	T	T	19·2
S	SKM	T	VFF	18·8
T	SKM	T	VFF	16·8
S	SKM		0	14·8
T	SKM		0	9·6
S	LDA		S	12·8
T	LDA		S	6·4

P MEM	OP CODE	Δ	Q MEM	MMS
S	LDA		T	6·8
T	LDA		T	8·8
S	LDA		VFF	6·8
T	LDA		VFF	5·2
S	LDA		VT	6·8
T	LDA		VT	4·8
S	LDB		S	12·8
T	LDB		S	6·4
S	LDB		T	6·8
T	LDB		T	8·8
S	LDB		VFF	6·8
T	LDB		VFF	5·2
S	LDB		VT	6·8
T	LDB		VT	4·8
S	LDC		S	12·8
T	LDC		S	6·4
S	LDC		T	6·8
T	LDC		T	8·8
S	LDC		VFF	6·8
T	LDC		VFF	5·2
S	LDC		VT	6·8
T	LDC		VT	4·8
S	LDD		S	12·8
T	LDD		S	6·4
S	LDD		T	6·8
T	LDD		T	8·8
S	LDD		VFF	6·8
T	LDD		VFF	5·2
S	LDD		VT	6·8
T	LDD		VT	4·8
S	LDE		S	12·8
T	LDE		S	6·4
S	LDE		T	6·8
T	LDE		T	8·8
S	LDE		VFF	6·8
T	LDE		VFF	5·2
S	LDE		VT	6·8
T	LDE		VT	4·8
S	SPF		S	12·8
T	SPF		S	9·6
S	SPF		T	9·6
T	SPF		T	8·8
S	SPF		VFF	10·4
T	SPF		VFF	8·4

Table 7-8, pg. 3   October 1961

P MEM	OP CODE	Δ	Q MEM	MMS
S	SPF		VT	10·0
T	SPF		VT	8·0
S	SPG		S	12·8
T	SPG		S	9·6
S	SPG		T	9·6
T	SPG		T	8·8
S	SPG		VFF	10·4
T	SPG		VFF	8·4
S	SPG		VT	10·0
T	SPG		VT	8·0
S	STA		S	14·0
T	STA		S	7·6
S	STA		T	6·8
T	STA		T	10·0
S	STA		VFF	6·8
T	STA		VFF	5·2
S	STA		VT	6·8
T	STA		VT	4·8
S	STB		S	14·0
T	STB		S	7·6
S	STB		T	6·8
T	STB		T	10·0
S	STB		VFF	6·8
T	STB		VFF	5·2
S	STB		VT	6·8
T	STB		VT	4·8
S	STC		S	14·0
T	STC		S	7·6
S	STC		T	6·8
T	STC		T	10·0
S	STC		VFF	6·8
T	STC		VFF	5·2
S	STC		VT	6·8
T	STC		VT	4·8
S	STD		S	14·0
T	STD		S	7·6
S	STD		T	6·8
T	STD		T	10·0
S	STD		VFF	6·8
T	STD		VFF	5·2
S	STD		VT	6·8
T	STD		VT	4·8
S	STE		S	14·0
T	STE		S	7·6

P MEM	OP CODE	Δ	Q MEM	MMS
S	STE		T	6·8
T	STE		T	10·0
S	STE		VFF	6·8
T	STE		VFF	5·2
S	STE		VT	6·8
T	STE		VT	4·8
S	EXA		S	14·0
T	EXA		S	7·6
S	EXA		T	6·8
T	EXA		T	10·0
S	EXA		VFF	6·8
T	EXA		VFF	5·2
S	EXA		VT	6·8
T	EXA		VT	4·8
S	ITA		S	12·8
T	ITA		S	6·4
S	ITA		T	6·8
T	ITA		T	8·8
S	ITA		VFF	6·8
T	ITA		VFF	5·2
S	ITA		VT	6·8
T	ITA		VT	4·8
S	UNA		S	12·8
T	UNA		S	6·4
S	UNA		T	6·8
T	UNA		T	8·8
S	UNA		VFF	6·8
T	UNA		VFF	5·2
S	UNA		VT	6·8
T	UNA		VT	4·8
S	DSA		S	12·8
T	DSA		S	6·4
S	DSA		T	6·8
T	DSA		T	8·8
S	DSA		VFF	6·8
T	DSA		VFF	5·2
S	DSA		VT	6·8
T	DSA		VT	4·8
S	ITE		S	12·8
T	ITE		S	6·4
S	ITE		T	6·8
T	ITE		T	8·8
S	ITE		VFF	6·8
T	ITE		VFF	5·2

Table 7-8, pg. 5   October 1961

P MEM	OP CODE	Δ	Q MEM	MMS
S	ITE		VT	6.8
T	ITE		VT	4.8
S	SED		S	12.8
T	SED		S	9.6
S	SED		T	9.6
T	SED		T	8.8
S	SED		VFF	10.4
T	SED		VFF	8.4
S	SED		VT	10.0
T	SED		VT	8.0
S	FLF		S	14.0
T	FLF		S	7.6
S	FLF		T	6.8
T	FLF		T	10.0
S	FLF		VFF	6.8
T	FLF		VFF	6.0
S	FLF		VT	6.8
T	FLF		VT	4.8
S	FLG		S	15.6
T	FLG		S	8.8
S	FLG		T	8.4
T	FLG		T	11.6
S	FLG		VFF	8.4
T	FLG		VFF	8.0
S	FLG		VT	8.4
T	FLG		VT	6.8
S	TSD		S	14.4
T	TSD		S	8.8
S	TSD		T	7.6
T	TSD		T	10.4
S	TSD		VFF	7.6
T	TSD		VFF	8.8
S	TSD		VT	7.6
T	TSD		VT	8.8
S	INS		S	15.2
T	INS		S	8.8
S	INS		T	6.8
T	INS		T	11.2
S	INS		VFF	6.8
T	INS		VFF	6.4
S	INS		VT	6.8
T	INS		VT	6.0
S	COM		S	14.8
T	COM		S	8.4

P MEM	OP CODE	Δ	Q MEM	MMS
S	COM		T	6.8
T	COM		T	10.8
S	COM		VFF	6.8
T	COM		VFF	6.4
S	COM		VT	6.8
T	COM		VT	6.0
S	ADD		S	12.8
T	ADD		S	6.4
S	ADD		T	6.8
T	ADD		T	8.8
S	ADD		VFF	6.8
T	ADD		VFF	5.6
S	ADD		VT	6.8
T	ADD		VT	4.8
S	SUB		S	12.8
T	SUB		S	6.4
S	SUB		T	6.8
T	SUB		T	8.8
S	SUB		VFF	6.8
T	SUB		VFF	5.6
S	SUB		VT	6.8
T	SUB		VT	4.8
S	MUL		S	20.8
T	MUL		S	20.8
S	MUL		T	19.6
T	MUL		T	20.0
S	MUL		VFF	19.6
T	MUL		VFF	20.8
S	MUL		VT	19.6
T	MUL		VT	19.2
S	[7]MUL		S	17.6
T	[7]MUL		S	17.6
S	[7]MUL		T	16.4
T	[7]MUL		T	15.2
S	[7]MUL		VFF	16.4
T	[7]MUL		VFF	16.0
S	[7]MUL		VT	16.4
T	[7]MUL		VT	16.0
S	[1]MUL		S	16.0
T	[1]MUL		S	14.4
S	[1]MUL		T	11.6
T	[1]MUL		T	12.0
S	[1]MUL		VFF	13.2
T	[1]MUL		VFF	12.8

Table 7-8, pg. 7   October 1961

P MEM	OP CODE	Δ	Q MEM	MMS
S	[1]MUL		VT	13·2
T	[1]MUL		VT	12·8
S	[3]MUL		S	12·8
T	[3]MUL		S	11·2
S	[3]MUL		T	10·0
T	[3]MUL		T	8·8
S	[3]MUL		VFF	10·0
T	[3]MUL		VFF	9·6
S	[3]MUL		VT	10·0
T	[3]MUL		VT	9·6
S	DIV		S	80·0
T	DIV		S	80·0
S	DIV		T	77·2
T	DIV		T	77·6
S	DIV		VFF	78·8
T	DIV		VFF	78·4
S	DIV		VT	77·2
T	DIV		VT	78·4
S	[7]DIV		S	60·8
T	[7]DIV		S	60·8
S	[7]DIV		T	59·6
T	[7]DIV		T	60·0
S	[7]DIV		VFF	59·6
T	[7]DIV		VFF	60·8
S	[7]DIV		VT	59·6
T	[7]DIV		VT	59·2
S	[1]DIV		S	43·2
T	[1]DIV		S	43·2
S	[1]DIV		T	42·0
T	[1]DIV		T	40·8
S	[1]DIV		VFF	42·0
T	[1]DIV		VFF	41·6
S	[1]DIV		VT	42·0
T	[1]DIV		VT	41·6
S	[3]DIV		S	22·4
T	[3]DIV		S	22·4
S	[3]DIV		T	19·6
T	[3]DIV		T	20·0
S	[3]DIV		VFF	21·2
T	[3]DIV		VFF	20·8
S	[3]DIV		VT	19·6
T	[3]DIV		VT	20·8
S	TLY		S	19·2
T	TLY		S	19·2

P MEM	OP CODE	Δ	Q MEM	MMS	
S	TLY		T	16·4	
T	TLY		T	16·8	
S	TLY		VFF	18·0	
T	TLY		VFF	17·6	
S	TLY		VT	18·0	
T	TLY		VT	17·6	
S	[7]TLY		S	16·0	
T	[7]TLY		S	16·0	
S	[7]TLY		T	13·2	
T	[7]TLY		T	13·6	
S	[7]TLY		VFF	14·8	
T	[7]TLY		VFF	14·4	
S	[7]TLY		VT	13·2	
T	[7]TLY		VT	14·4	
S	[1]TLY		S	12·8	
T	[1]TLY		S	11·2	
S	[1]TLY		T	10·0	
T	[1]TLY		T	12·0	
S	[1]TLY		VFF	10·0	
T	[1]TLY		VFF	11·2	
S	[1]TLY		VT	10·0	
T	[1]TLY		VT	9·6	
S	[3]TLY		S	12·8	
T	[3]TLY		S	8·0	
S	[3]TLY		T	6·8	
T	[3]TLY		T	8·8	
S	[3]TLY		VFF	6·8	
T	[3]TLY		VFF	8·0	
S	[3]TLY		VT	6·8	
T	[3]TLY		VT	8·0	
S	SCA		S	12·8	107·2
T	SCA		S	8·0	107·2
S	SCA		T	6·8	104·4
T	SCA		T	8·8	104·8
S	SCA		VFF	6·8	
T	SCA		VFF	8·0	
S	SCA		VT	6·8	106·0
T	SCA		VT	8·0	105·6
S	[7]SCA		S	12·8	107·2
T	[7]SCA		S	8·0	107·2
S	[7]SCA		T	6·8	104·4
T	[7]SCA		T	8·8	104·8
S	[7]SCA		VFF	6·8	
T	[7]SCA		VFF	8·0	

Table 7-8, pg. 9   Octoter 1961

P MEM	OP CODE	Δ	Q MEM	MMS	
S	^{7}SCA		VT	6·8	106·0
T	^{7}SCA		VT	8·0	105·6
S	^{1}SCA		S	12·8	107·2
T	^{1}SCA		S	8·0	107·2
S	^{1}SCA		T	6·8	104·4
T	^{1}SCA		T	8·8	104·8
S	^{1}SCA		VFF	6·8	
T	^{1}SCA		VFF	8·0	
S	^{1}SCA		VT	6·0	106·0
T	^{1}SCA		VT	8·0	105·6
S	^{3}SCA		S	12·8	107·2
T	^{3}SCA		S	8·0	107·2
S	^{3}SCA		T	6·8	104·4
T	^{3}SCA		T	8·8	104·8
S	^{3}SCA		VFF	6·8	
T	^{3}SCA		VFF	8·0	
S	^{3}SCA		VT	6·8	106·0
T	^{3}SCA		VT	8·0	105·6
S	SCB		S	12·8	107·2
T	SCB		S	8·0	107·2
S	SCB		T	6·8	104·4
T	SCB		T	8·8	104·8
S	SCB		VFF	6·8	106·0
T	SCB		VFF	8·0	105·6
S	SCB		VT	6·8	106·0
T	SCB		VT	8·0	105·6
S	SAB		S	12·8	107·2
T	SAB		S	8·0	107·2
S	SAB		T	6·8	104·4
T	SAB		T	8·8	104·8
S	SAB		VFF	6·8	
T	SAB		VFF	8·0	
S	SAB		VT	6·8	105·0
T	SAB		VT	8·0	105·6
S	^{7}SAB		S	12·8	107·2
T	^{7}SAB		S	8·0	107·2
S	^{7}SAB		T	6·8	104·4
T	^{7}SAB		T	8·8	104·8
S	^{7}SAB		VFF	6·8	
T	^{7}SAB		VFF	8·0	
S	^{7}SAB		VT	6·8	106·0
T	^{7}SAB		VT	8·0	105·6
S	^{1}SAB		S	12·8	107·2
T	^{1}SAB		S	8·0	107·2

P MEM	OP CODE	Δ	Q MEM	MMS	
S	[1]SAB		T	6.8	104.4
T	[1]SAB		T	8.8	104.8
S	[1]SAB		VFF	6.8	
T	[1]SAB		VFF	8.0	
S	[1]SAB		VT	6.8	105.0
T	[1]SAB		VT	8.0	105.6
S	[3]SAB		S	12.8	107.2
T	[3]SAB		S	8.0	107.2
S	[3]SAB		T	6.8	104.4
T	[3]SAB		T	8.8	104.8
S	[3]SAB		VFF	6.8	
T	[3]SAB		VFF	8.0	
S	[3]SAB		VT	6.8	106.0
T	[3]SAB		VT	8.0	105.6
S	CYB		S	12.8	107.2
T	CYB		S	8.0	107.2
S	CYB		T	6.8	104.4
T	CYB		T	8.8	104.8
S	CYB		VFF	6.8	106.0
T	CYB		VFF	8.0	105.0
S	CYB		VT	6.8	106.0
T	CYB		VT	8.0	105.6
S	[7]CYB		S	12.8	107.2
T	[7]CYB		S	8.0	107.2
S	[7]CYB		T	6.8	104.4
T	[7]CYB		T	8.8	104.8
S	[7]CYB		VFF	6.8	106.0
T	[7]CYB		VFF	8.0	105.6
S	[7]CYB		VT	6.8	106.0
T	[7]CYB		VT	8.0	105.6
S	[1]CYB		S	12.8	107.2
T	[1]CYB		S	8.0	107.2
S	[1]CYB		T	6.8	104.4
T	[1]CYB		T	8.8	104.8
S	[1]CYB		VFF	6.8	106.0
T	[1]CYB		VFF	8.0	105.6
S	[1]CYB		VT	6.8	106.0
T	[1]CYB		VT	8.0	105.6
S	[3]CYB		S	12.8	107.2
T	[3]CYB		S	8.0	107.2
S	[3]CYB		T	6.8	104.4
T	[3]CYB		T	8.8	104.8
S	[3]CYB		VFF	6.8	106.0
T	[3]CYB		VFF	8.0	105.6

Table 7-8, pg. 11   October 1961

P MEM	OP CODE	Q MEM	Δ	MMS
S	[3]CYB	VT	6·8	106·0
T	[3]CYB	VT	8·0	105·6
S	CYA	S	12·8	107·2
T	CYA	S	8·0	107·2
S	CYA	T	6·8	104·4
T	CYA	T	8·8	104·8
S	CYA	VFF	6·8	38·2
T	CYA	VFF	8·0	38·8
S	CYA	VT	6·8	106·0
T	CYA	VT	8·0	105·6
S	CAB	S	12·8	107·2
T	CAB	S	8·0	107·2
S	CAB	T	6·8	104·4
T	CAB	T	8·8	104·8
S	CAB	VFF	6·8	
T	CAB	VFF	8·0	38·8
S	CAB	VT	6·8	106·0
T	CAB	VT	8·0	105·6
S	NOA	S	19·2	12·8
T	NOA	S	19·2	8·0
S	NOA	T	18·0	6·8
T	NOA	T	18·4	8·8
S	NOA	VFF	18·0	6·8
T	NOA	VFF	19·2	8·0
S	NOA	VT	18·0	6·8
T	NOA	VT	17·6	8·0
S	NAB	S	33·6	12·8
T	NAB	S	33·6	8·0
S	NAB	T	32·4	6·8
T	NAB	T	32·8	8·8
S	NAB	VFF	32·4	6·8
T	NAB	VFF	33·6	8·0
S	NAB	VT	32·4	6·8
T	NAB	VT	32·0	8·0