

# **UNIVAC<sup>®</sup>** **Solid-State 90**

**TECHNICAL BULLETIN**

## ***X-6 ASSEMBLY SYSTEM***

### ***for TAPE Systems***

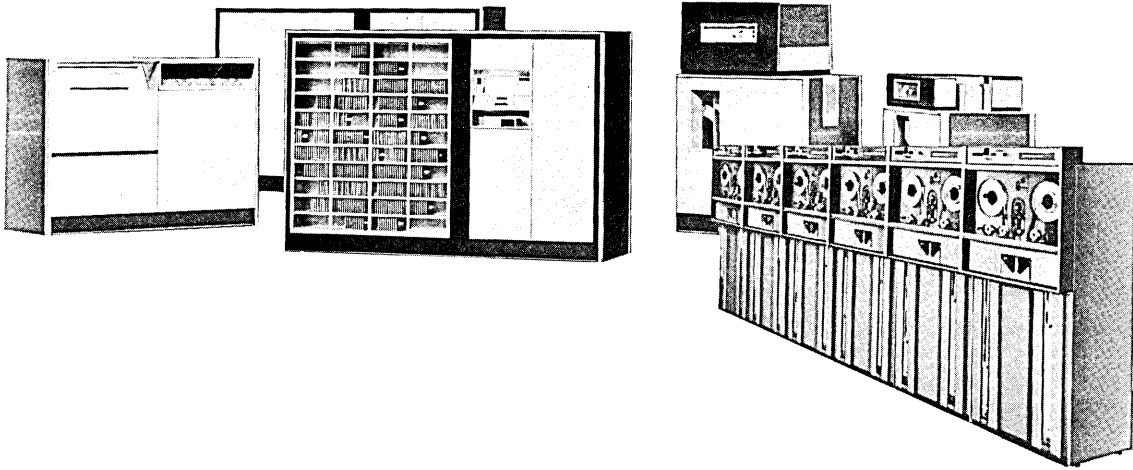
Modifications and refinements resulting from new research are continually making the X-6 Assembly System for the UNIVAC Solid-State 90 Magnetic Tape System more useful and economical. In order to keep X-6 System documentation as current as possible, supplementary technical information will appear in the UNIVAC Systems Routines Series published by Program Library Services.

February, 1961

# Contents

1. INTRODUCTION .....	1
2. THE X-6 PROGRAM .....	2
General Information .....	2
The X-6 Instruction .....	2
Spaces .....	3
Absolute Addresses .....	4
Register Addressing .....	5
3. SYMBOLIC AND RELATIVE ADDRESSING .....	6
General Information .....	6
Tags .....	6
Constants .....	9
Working Storage .....	11
Interlaces .....	12
Tables .....	15
Variable Addresses .....	16
4. PROGRAM ORGANIZATION .....	18
The Operation .....	18
Controls .....	20
5. X-6 INSTRUCTION CODES .....	23
General Information .....	23
Transfer Instructions .....	23
Arithmetic Instructions .....	25
Comparison Instructions .....	28
Editing Instructions .....	29
Control Instructions .....	31
Translate Instructions .....	31
Index Register Instructions .....	32
Input Instructions (High-Speed Reader) .....	33
Input-Output Instructions (Read-Punch Unit) .....	33
Output Instructions (High-Speed Printer) .....	34
Input-Output Instructions (Tape Synchronizer) .....	35
6. HOW X-6 WORKS .....	37
Input Processing .....	37
Optimization of Instructions .....	39
Clock Modification .....	41
7. PROGRAMMING PROCEDURE .....	47
Suggestions for Flow-Charting .....	47
Coding .....	47
Preparation for Assembly .....	47
Assembly .....	48
8. OPERATING INSTRUCTIONS .....	49
Loading and Assembling .....	49
Error Codes .....	49
Stop Codes .....	50
APPENDIX A. SUMMARY OF INSTRUCTION CODES .....	53
B. SUMMARY OF CARD TYPES .....	57
C. X-6 STORAGE LAYOUT .....	63
D. CARD FORMS .....	65
E. CODING FORMS .....	66
F. SAMPLE PROBLEM .....	68

## I. Introduction



This manual is intended to familiarize the programmer with the basic elements of the X-6 System for use with the UNIVAC Solid-State 90 Magnetic Tape Computer. The system is an efficient programming aid designed to facilitate the coding of data-processing applications by reducing both coding time and error frequency.

Coding time is reduced by allowing the optimum placement of instructions to be handled by X-6, thus freeing the programmer from exacting timing considerations. Further reductions in coding time result from the performance, by the assembler, of many jobs that would normally be undertaken by the programmer.

Low error frequency is achieved because sections of a problem may be coded individually and later assembled into a larger unified program. Moreover, a large application may be divided among several programmers to reduce over-all coding time.

Before attempting to employ the X-6 Assembly System, the programmer should be acquainted with the basic elements of the UNIVAC Solid-State 90 Magnetic Tape System. Such knowledge is *assumed* in this manual.

## 2. The X-6 Program

### GENERAL INFORMATION

Regardless of the form of its expression, a computer program is composed of interrelated sections of coding, each created to perform a definite function leading to the solution of a problem. These sections are commonly referred to as subroutines or *operations*.

The X-6 Assembly System is a master or executive routine that receives as input a series of these operations, created in the X-6 command language, and in one pass through the computer, produces a new deck in computer code. A side-by-side detailed correlation of the X-6 coding and the computer-coded object, or final, program is also produced as output on the High-Speed Printer (Figure 2-1).

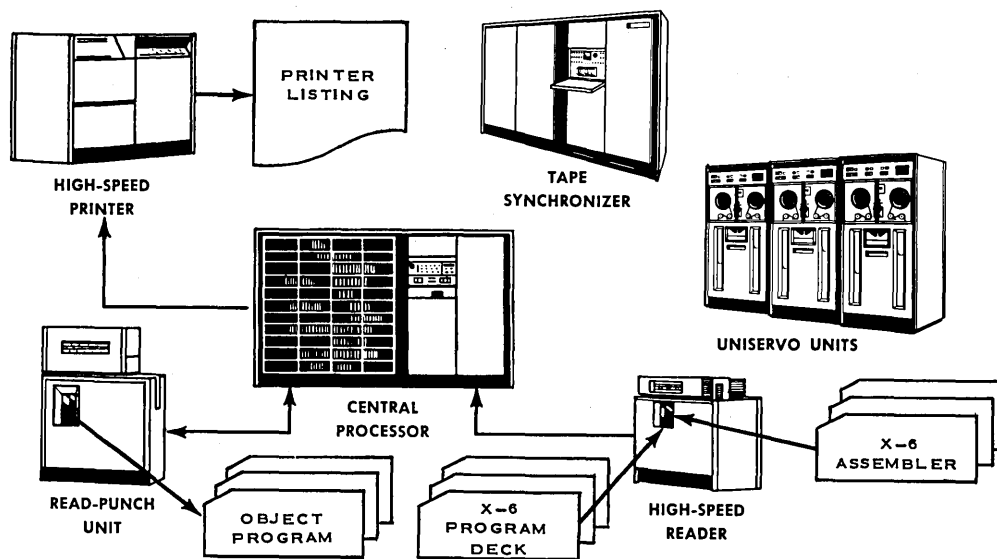


Figure 2-1. Information Flow from X-6 Program Deck to Computer-Coded Object Program.

After conversion to computer-acceptable language, the various elements of the X-6 program are assigned to storage as an integrated unit preparatory to the generation of final output on card or tape in either computer or XS-3 mode.

### THE X-6 INSTRUCTION

In the X-6 instruction, the *a*, *m*, and *c* addresses contain five digit specifications in the form:

aaaaa    III    mmmmm    ccccc

Here;        aaaaa    is the address of the instruction in storage. This address may be expressed in either computer or X-6 terminology, or it may be left blank allowing the assembler to assign a storage location.

**mmmm** is the address of the operand; the location of the next instruction to be executed; or can be ignored depending upon the instruction. This address may be expressed in either computer or X-6 terminology or it may be left blank, allowing the assembler to assign a storage location.

**cccc** is either the address of the next instruction to be executed, or can be ignored depending upon the instruction. This address may be expressed in either computer or X-6 terminology, or it may be left blank allowing the assembler to assign a storage location.

**III** is the three-character alphabetic instruction code.

## SPACES

If the generation of absolute addresses in the object program is to be left to the X-6 Assembly System, the *a*, *m*, or *c* portion involved in this assignment will be left blank. Spaces, indicated by deltas ( $\Delta$ ) in the *c* address of an instruction, will indicate to the assembler that the next, and only the next, consecutive line of coding is being referenced. The *a* address of the next instruction will be examined for spaces and assigned the same absolute address as the previous *c* address that referenced it. In this way, spaces provide a means of coding sequentially without the need for designating specific locations in storage for consecutive lines of coding. For example, there are two consecutive instructions in a program:

line	a	inst. code	m	c
1	xxxxx	LDA	yyyyy	$\Delta\Delta\Delta\Delta$
2	$\Delta\Delta\Delta\Delta$	STA	zzzzz	.....

The first instruction, located at xxxxx, specifies that register A is to be loaded with the contents of yyyyy. The second instruction specifies that the contents of register A are to be stored in zzzzz. By inserting spaces in the *c* address of the first instruction and in the *a* portion of the second instruction, the second will automatically succeed the first and the assigned storage locations will preserve the intended relationship between the two lines of coding.

Spaces may be employed in the *m* portion of an instruction when

- a. a transfer of control is to be effected from the *m* portion of the instruction being executed to the *a* portion of the next instruction in sequence.
- b. the next line of coding is the operand of the instruction being executed.

For example, load register A with a constant of 00000 00128 and register L with the contents of storage location yyyyy. Compare the two quantities for equality. If equal, store the contents of register A in storage location zzzzz; otherwise, go to location wwwww for further processing. Here xxxxx, ppppp, yyyyy, zzzzz, and wwwww are addresses defined in the X-6 program.

line	a	inst. code	m	c
1	x x x x x	LDA	ΔΔΔΔΔ	p p p p p
2	ΔΔΔΔΔ	...	0 0 0 0 0	0 0 1 2 8
3	p p p p p	LDL	y y y y y	ΔΔΔΔΔ
4	ΔΔΔΔΔ	TEQ	ΔΔΔΔΔ	w w w w w
5	ΔΔΔΔΔ	STA	z z z z z	.....

In final conversion to the machine-coded object program, these five lines of coding might appear in the following manner.

line	a	inst. code	m	c
1	0200	25	0202	0204
2	0202	00	0000	0128
3	0204	30	4206	0208
4	0208	82	0210	0410
5	0210	60	4412	....

Spaces may not be entered in both the *m* and *c* portions of an instruction unless either of these portions is normally ignored; for example, translate instructions which ignore the *m* address, or instructions which load registers with zeros and which ignore the *c* addresses. For all instructions in which both *m* and *c* are important, only one portion may be spaces.

### ABSOLUTE ADDRESSES

Although the normal mode of procedure in an X-6 program is to allow the assembler to assign absolute addresses, it may be necessary for the programmer to reserve an address in storage. An absolute location used in the *a*, *m*, or *c* portion of an instruction, is placed in the least significant digit positions of the portion to which it applies. The unused positions of the most significant digits of the address are filled with either spaces (Δ) or zeros. For example, an instruction at xxxxx to load register A with the contents of storage location 956 would appear as:

a	inst. code	m	c
xxxxx	LDA	ΔΔ956	.....
or			
xxxxx	LDA	00956	.....

Spaces or zeros are empty-column indicators employed mainly for the convenience of the key punch operator. As greater familiarity is gained with the system, however, filler symbols may be omitted, since they never appear in final output.

When examining addresses, the assembler will examine both the fifth and the first digit positions of the address in that order. If neither is an alphabetic character, the address will be considered absolute and will be carried over to the object program without modification. Normal optimization of instructions will occur after a correction factor is employed to allow for the specification of the absolute location.<sup>1</sup>

**NOTE:** If an absolute address is to be assigned by the programmer, the particular location *must be restricted from use* by the assembly system in its normal address assignment. This will prevent the system from assigning an already used location. The method of restriction will be discussed in the section *Program Organization*, page 18.

## REGISTER ADDRESSING

As a result of their addressability in the Solid-State System, registers A, X, and L may be specified in the *m* portion of many, and the *c* portion of all instructions. The only restriction is that they cannot be employed in the *m* portion of instructions that specify a transfer from *register to storage*. The three registers are designated and addressed as  $\Delta\Delta\Delta RA$ ,  $\Delta\Delta\Delta RX$ ,  $\Delta\Delta\Delta RL$ .

As an example of register addressability, an instruction at xxxxx to load the contents of register A into register X would appear in the following manner.

	<b>a</b>	<b>inst. code</b>	<b>m</b>	<b>c</b>
	xxxxx	LDX	$\Delta\Delta\Delta RA$	.....

It is recommended that when an instruction is being executed *in* a register, the next line of coding be entered on the coding paper with the particular register as the *a* address. Although not punched as output, this line will be shown on the printed listing and will allow the assembler to optimize more efficiently.

For example:

<b>line</b>	<b>a</b>	<b>inst. code</b>	<b>m</b>	<b>c</b>
1	xxxxx	LDA	04211	$\Delta\Delta\Delta RL$
2	$\Delta\Delta\Delta RL$	ADD	00204	00406

<sup>1</sup>Optimum placement of instructions will be discussed in the section, *How X-6 Works*, page 37.

### 3. Symbolic and Relative Addressing

#### GENERAL INFORMATION

The use of relative and symbolic addresses eliminates the necessity of referring to absolute storage locations around the drum and, in large measure, frees the programmer from timing considerations. Unlike spaces which can relate only two successive lines of coding, a relative or symbolic address can relate one line of coding with another that either has been specified, or will be specified at some point in the program. Simply stated, a relative address indicates a relationship between a line being referenced and another line whose location, in storage, has already been determined. A symbolic address is any arbitrary combination of characters defined within a system to represent a storage location.

#### TAGS

Tags are symbolic designations for storage locations that will be assigned absolute addresses either by the assembly system or the programmer. These symbolic designations are entered as the *a* portion of lines of coding that are to be referenced by the *m* or *c* portions of other lines of coding. Like spaces, they free the programmer from having to decide, as he codes his routine, exactly where in storage a line of coding is to go. The programmer has the added advantage of specifying whether a tag is to receive an address in standard or high-speed access storage. The X-6 Assembly System makes provision for two types of tags, *temporary* and *permanent*.

#### TEMPORARY TAGS

As mentioned previously, an X-6 program is divided into smaller sections of coding called *operations*. Temporary tags provide connecting links within, and *only within*, operations. That is to say, a temporary tag assigned to a particular line of coding is meaningful only within the operation in which that line of coding is located and cannot be referred to by coding in another operation. The format for the temporary tag is;

$\Delta\Delta xxi$

where  $\Delta\Delta$  is ignored by the system.

$xx$  is the tag identifier. These two digits may be numeric, alphabetic, or alpha-numeric. A maximum of fifty temporary tags may be specified in any operation.

$i$  is N if the tag is to be given an address in *standard* access storage.  
is F if the tag is to be given an address in *high-speed* access storage.



For example, add a constant of 00000 00xxx to the contents of 4211 and store the result in 4236.

line	a	inst. code	m	c
1	.....	LDA	Δ4211	ΔΔΔΔΔ
2	ΔΔΔΔΔ	ADD	ΔΔΔΔΔ	ΔΔG2N
3	ΔΔΔΔΔ	...	00000	00xxx
4	ΔΔG2N	STA	Δ4236	.....

Tag ΔΔG2N, in the c address of line 2, permits communication with line 4 after the arithmetic computation occurs. Line 4 may be communicated with from any point within this operation by specifying a transfer of control to ΔΔG2N. The N indicator will instruct the assembler to assign this line of coding to standard access storage.

Although, in the five digit format of the temporary tag, only the three low order digits are examined by the assembler, the fourth low order digit may be utilized by the programmer if he so desires. For example, a temporary tag may be specified as

ΔG39N

If this is done, however, the three low order characters must be unique in the operation for, as was indicated, only these three digits will be examined by the assembler and ΔG39N will be treated as ΔΔ39N. Regardless of what the fourth low-order character specifies (i.e., ΔS39N, Δ539N, and so forth) this tag will be recognized as ΔΔ39N.

#### PERMANENT TAGS

Permanent tags not only serve as communication links within operations, but also enable the programmer to communicate with lines of coding in other operations. Unlike temporary tags, permanent tags may be assigned absolute addresses by the programmer if he so desires. The format for the permanent tag is;

nnnxi

where **nnn** is the tag identifier and may be alphabetic, numeric, or alpha-numeric. A maximum of 300 permanent tags may be assigned in a program.

**i** is N if the tag is to be given an address in *standard* access storage.

is F if the tag is to be given an address in *high-speed* access storage.

Frequently, the nnn specification is entered as the number of the operation in which the tag is initially specified. For example, the permanent tag

2054F

indicates that tag 4 in operation 205 is to be assigned an address in *high-speed* access storage. And similarly, the permanent tag

SIN5N

indicates that this is tag 5 in operation SIN and is to be assigned an address in *standard* access storage.

## Q AND P TAGS

Two additional tag specifications are provided by the X-6 System to handle overflow resulting from either an arithmetic computation or from an abnormal condition in an input or output unit. These two specifications, designated Q and P tags, permit transfer of control either normally to *c* or, when necessary, to *c* + 1 if an overflow condition is present.

Q and P specifications may either be permanent or temporary and the basic rules applicable to temporary and permanent tag assignment will also apply here. It will be recalled that the two basic tag formats are

ΔΔxxi

for temporary tags, and

nnnxi

for permanent. The same basic format exists for the overflow tags with the exception that the low order digit position (*i*) on each type of tag will now contain an Q to indicate the line to be executed if overflow *does not* occur, or a P to indicate the line to be executed if overflow *does* occur (*c* + 1).

For example, two quantities are located in storage locations 4211 and 4216. Add them together, store the result in 4236, and go to GREGN for further processing.

line	a	inst. code	m	c
1	ΔΔ S1N	LDA	Δ4211	ΔΔΔΔΔ
2	ΔΔΔΔΔ	ADD	Δ4216	ΔΔΔΔΔ
3	ΔΔΔΔΔ	STA	Δ4236	GREGN

If the arithmetic computation results in an overflow condition, a constant of 00000 00001 is to be stored in 4246 and control is to be transferred to AAR4N for further processing.

line	a	inst. code	m	c
1	ΔΔ S1N	LDA	Δ 4 2 1 1	ΔΔΔΔΔ
2	ΔΔΔΔΔ	ADD	Δ 4 2 1 6	ΔΔG1 <u>Q</u>
3	ΔΔG1 <u>Q</u>	STA	Δ4236	GREGN
4	ΔΔG1P	LDA	ΔΔΔΔΔ	ΔΔ S2N
5	ΔΔΔΔΔ	...	00000	00001
6	ΔΔS2N	STA	Δ 4 2 4 6	AAR4N

Tag ΔΔG1Q in line 2 indicates that control is to be transferred to line 3 if overflow does not occur. If overflow does occur, however, line 4, tagged ΔΔG1P, will be executed as the *c* + 1 line.

It should be noted that neither the Q or P line has to physically follow the instruction which may cause an overflow condition as long as the tag is unique within the operation, if it is a temporary tag, or unique within the program, if it is permanent. Furthermore, these tags, whether actually executed or not, must be subtracted from the total allowable number of tags in an X-6 program.

### The Tag-Equals Card (Card Type 3)

The tag-equals card makes it possible for the programmer to assign absolute storage addresses to *permanent* tags if he wishes. Up to seven entries may be made per card with no restrictions on the number of cards used. The last valid entry will be followed by a word of nines (9999999999).

CARD TYPE											ENTRY 1										ENTRY 2										ENTRY 3										
	3											t t t t t Δ n n n n	t t t t t Δ n n n n	t t t t t Δ n n n n																											
										1 2 3 4 5 6 7 8 9 10	11 12 13 14 15 16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31 32 33 34 35 36 37 38 39 40	41 42 43 44 45																											
Printed in U.S.A. - REMINGTON RAND	ENTRY 4										ENTRY 5										ENTRY 6										ENTRY 7										
	t t t t t Δ n n n n										t t t t t Δ n n n n	t t t t t Δ n n n n	t t t t t Δ n n n n																												
										46 47 48 49 50 51 52 53 54 55	56 57 58 59 60 61 62 63 64 65	66 67 68 69 70 71 72 73 74 75	76 77 78 79 80 81 82 83 84 85	86 87 88 89 90																											

Here        **t t t t t**    is the permanent tag.  
              **Δ n n n n**    is the absolute address in storage.

### CONSTANTS

There are two basic methods of specifying constants in an X-6 program. The first is to include the particular constant as a line within the coding. For example, load register A with the contents of storage location 4211 and add a constant of 00000 00001.

line	a	inst. code	m	c
1	1N	LDA	4211	
2		ADD		2N
3			00000	00001
4	2N	[NEXT INSTRUCTION]		

The second is to pool all constants used in a program in an area in storage called the K area and reference them with a five-digit symbolic address. The format of this five-digit specification is:

**KΔxxx**

where **K** indicates a reference to the constant area.

**xxx** is the number assigned to the particular constant in the K area, from  $\Delta\Delta 0$  to 299 thus allowing a maximum of 300 pooled constants in any one program.

For example, 4216 contains a quantity to which a constant of 00000 00002 is to be added. With the constant entered into the K area as the first constant ( $K\Delta\Delta\Delta 0$ ) the coding may be written in the following manner.

line	a	inst. code	m	c
1	1N	LDA	4216	
2		ADD	K 0	

The constant pool may be entered into the program as a separate operation and designated as operation KKK. Each constant is then entered with its five-digit K specification as the *a* address. It should be noted, however, that no particular sequence must be preserved in referencing constants. Also, a constant designated, for example,  $K\Delta\Delta\Delta 8$  or  $K\Delta 299$  may be specified as the only constant in the program with no preceding constant entries.

All K area constants are automatically assigned to high-speed access storage unless the storage locations in this area have been used, or unless the programmer specifies otherwise. That is, the programmer may specify, if he chooses, an absolute location for any K area constant by entering the five-digit specification as a permanent tag on the Tag-Equals<sup>2</sup> card and assigning an absolute location to it. Normally, the assembler would assign an absolute address to the K specification where the constant is first referenced in the X-6 program.

Constants may either be data or instructions. A data constant, whether stored in the K area or included with the coding, is entered with the *a* portion containing any legitimate X-6 address, the instruction code positions containing spaces, and the *m* and *c* portions containing the absolute value of the constant which will be carried over to final output without modification.

For example, a K area constant of 00000 00030 might be entered as;

	a	inst. code	m	c
	K 61	...	00000	00030

Instruction constants utilize the entire thirteen digit positions of the instruction code and the *m* and *c* portions. These constants are entered in symbolic form and, therefore, must be converted before they are carried over to final output. For example, a constant designated as  $K\Delta\Delta 21$  that is an instruction to load register A with GIN4F and then go to SIN4F would be entered as:

<sup>2</sup> See Tag-Equals Card, page 9.

	a	inst. code	m	c
	K 21	LDA	GIN4F	SIN4F

The X-6 assembler distinguishes between data and instruction constants by the presence or absence of spaces in the instruction code. It should be noted that if a constant is tagged with a symbolic specification other than a K address, it may not be entered in the constant pool.

## WORKING STORAGE

Working storage locations (locations utilized for holding data in anticipation of some future computation in the program) are referred to as W areas and can be referenced by a five-digit symbolic address. The format for this specification is;

WΔxxx

where W indicates a reference to a working storage location.

xxx is the number assigned to a particular working storage location in the W area, from ΔΔ0 to 299 thus allowing 300 working storage locations in any one program.

For example, a working storage location, designated WΔΔ10, contains a quantity to which the contents of KΔΔ31 are to be added. Store the result in SIN1N.

line	a	inst. code	m	c
1	1N	LDA	W 10	
2		ADD	K 31	
3		STA	SIN1N	

The working storage areas utilized in a program may be entered, like K area constants, as a separate operation and designated as operation WWW. Each working storage location used is then specified with its five-digit W designation as the a address, and the initial condition of the location specified in the digit positions of the instruction code and the m and c portions. As is the case with K area constants, no particular order must be preserved in designating working storage locations and WΔΔΔ0 or WΔ299 could equally be the first location with which communication is made.

All W locations are automatically assigned an address in high-speed access storage unless the storage locations in this area have been used, or unless the programmer specifies otherwise. The programmer may assign an absolute address to a W location on a Tag-Equals card. Normally, however, a W specification would be assigned an absolute address, by the assembler, where it is first referenced in the program.

## INTERLACES

Information entering the computer as input or leaving as output, is stored in a fixed pattern of storage locations called an interlace pattern. Each unit has a designated pattern of locations into which data is read or from which data is written, punched, or printed. A five-digit symbolic address may be employed in the *m* portion of an instruction to communicate with a particular word in an interlace. The format for this symbolic designation is

**unwxx**

- Here;
- u** is the particular interlace being referenced.
    - H** for the read interlace of the High-Speed Reader.
    - R** for the read interlace of the Read-Punch Unit.
    - O** for the punch interlace of the Read-Punch Unit.
    - P** for the print interlace of the High-Speed Printer.
    - T** or **Z** for the tape interlace. Either or both may be employed.
  
  - n** is the number of the interlace pattern. This may be 0 through 9 thus allowing ten separate interlace patterns for each unit. A total of twenty interlace patterns are allowed for tapes, ten for T and ten for Z.
  
  - w** is the word part.
    - U** for the unprimed part of the card word.
    - P** for the primed part of the card word.
    - N** for the numeric part of the translated card word or the numeric portion of a tape specification.
    - Z** for the zone part of the translated card word or the zone portion of a tape specification.
  
  - xx** is the word being referenced.
    - 10-19 for card words 0 through 9, respectively, sensed at read station 1 of the High-Speed Reader and stored in a read interlace.
    - 20-29 for card words 0 through 9, respectively, sensed at read station 2 of the High-Speed Reader and stored in a read interlace.

10-19 for card words 0 through 9, respectively, sensed at read station 1 of the Read-Punch Unit and stored in the punch interlace.

20-29 for card words 0 through 9, respectively, sensed at read station 2 of the Read-Punch Unit and stored in the punch interlace.

10-19 for card words 0 through 9, respectively, stored in the punch interlace, to be punched as output.

01-13 for words to be printed as output and stored in the print interlace.

00-71 for words 1 through 72, respectively, of a block of data in UNIVAC XS-3 mode stored in a tape interlace.

00-99 for words 1 through 100, respectively, of a block of data in USS mode stored in a tape interlace.

**NOTE:** When a complete tape interlace is addressed wxx will always be 000.

The five-digit symbolic specification in the *m* portion of an instruction that will address the *primed image* in the *eighth word* location of a card read at the *first read station* of the High-Speed Reader and stored in an input band assigned to contain *interlace number 2*, would be

**H2P17**

To address the *unprimed portion* of the same data after it has been read at *read station 2*, the following five-digit specification would be entered in the *m* portion of the instruction;

**H2U17**

#### **Addressing A Print Interlace**

When a word in a print interlace is addressed, only the desired word location is entered in the xx digits of the symbolic address since there is no need to refer to a read or punch station. Because a print interlace contains thirteen word locations, corresponding to the thirteen possible word positions on a printed line, the last two digits of the address may be specified as any word from 01 to 13. For example, for the *numeric image* in the *fourth* word location of a print interlace assigned as interlace number 2, the address in the *m* portion would be;

**P2N03**

Similarly, to address the zone portion of the same word, one would write the specification as;

**P2Z03**

#### **Addressing A Complete Print Interlace**

When addressing a *complete* print interlace, the five digit specification is in a somewhat different format from the usual interlace specification. The format for this address is;

Pn0aa

where P indicates the print interlace.  
n indicates the number of the interlace (0-9).  
aa indicates the number of lines that the paper in the printer is to be advanced (00-79).

For example, using PRN as the mnemonic instruction code to print out, the instruction to advance the paper twelve lines and print the contents of print interlace number 2 would be;

	a	inst. code	m	c
.....		PRN	P2012	.....

**NOTE:** When using any High-Speed Reader or Read-Punch Unit service routine, the program will not deal with the normal interlace positions but rather with working storage areas designated as interlace positions within the particular routine. Card images are entered and retrieved from these areas by the individual service routine, and transferred to the normal interlace positions. Therefore, when employing any service routine, the programmer *cannot* use a symbolic interlace designation but must refer to the actual working storage address utilized in the particular routine.

**Interlace Card (Card Type 4)**

All symbolic interlace specifications in a problem are entered on this card. Up to seven entries may be made per card with no restriction on the number of cards. The last valid entry is followed by a word of nines (999999999).

CARD TYPE	ENTRY 1										ENTRY 2										ENTRY 3										12 34 56 78 9																																																										
	4	t n Δ Δ Δ x b b 0 0										t n Δ Δ Δ x b b 0 0										t n Δ Δ Δ x b b 0 0																																																																			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
ENTRY 4	ENTRY 5										ENTRY 6										ENTRY 7										12 34 56 78 9																																																										
	t n Δ Δ Δ x b b 0 0										t n Δ Δ Δ x b b 0 0										t n Δ Δ Δ x b b 0 0											t n Δ Δ Δ x b b 0 0																																																									



Here,            **t**    is the type of interlace (R, P, O, H, T or Z)  
                   **n**    is the number of the interlace (0-9).  
                   **x**    is 0 for a two part untranslated interlace (unprimed and primed).<sup>3</sup>  
                               is 1 for a two part translated interlace (zone and numeric).  
                               is 2 if both kinds are specified.  
                   **bb**    is the absolute address of the band (bb must be an even number).

## TABLES

A table may be defined as an area in storage in which the various entries (they may be either data or instructions) are separated by a given increment or, more precisely, a specified number of storage locations. X-6 provides three table areas designated as S, U, or V. Each table area may contain a maximum of ten tables with up to 1000 entries per table.

To communicate with a particular entry in a table, a five-digit symbolic address is employed. The format for this specification is;

**tnxxx**

where            **t**    is the name of the table area (S, U, or V).  
                   **n**    is the number of the particular table (0-9).  
                   **xxx**    is the number of the particular *entry* in the table (000-999).

For example, the instruction to load the *fifteenth* entry of the *second* table in table area S into register A would be

a	inst. code	m	c
.....	LDA	S2014	.....

### Tables Card (Card Type 5)

All table specifications employed in a program must be entered on this card. Up to three entries may be made per card with no restriction on the number of cards except the implied physical restriction on the number of table specifications. The last valid entry is followed by a word of nines (999999999).

<sup>3</sup> The notations here for x are not applicable to the print or tape interlace and x will always equal 0.

CARD TYPE	ENTRY 1										ENTRY 2																																		
	5	t n Δ Δ Δ Δ s s s s s i i i Δ Δ Δ x x x x										t n Δ Δ Δ Δ s s s s s										12																							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
PRINTED IN U.S.A. REMINGTON RAND	ENTRY 2 (cont.)					ENTRY 3																																							
		i i i Δ Δ Δ x x x x					t n Δ Δ Δ Δ s s s s s i i i Δ Δ Δ x x x x										12																												
	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90

Here,            t    is the name of the table area (S, U, or V).  
                  n    is the number of the particular table (0-9).  
                  ssss is the absolute address of the first entry.  
                  iii   is the increment between entries.  
                  xxxx is the total number of entries in the table.

**VARIABLE ADDRESSES**

Variable addresses are employed in operations that are to be coded as library routines. Basically, a library routine is a routine, of some commonly-used function, coded so that it may be employed in many varying applications thus avoiding the necessity of repeatedly coding the function each time it is needed. Any operation may be specified as a library routine; however, all references to tables, interlaces, constants, working storages, and so forth, are generalized by the substitution of variable addresses for specific locations. These variable addresses can be particularized, when the library routine is employed in a particular application, by cross-referencing them to specific X-6 addresses. The variable address is a five-digit symbolic specification entered in the a, m, or c portion of an instruction. The format for this specification is;

$$X \Delta \Delta nn$$

where           X    is an indication that this is a variable address.  
                  nn   is the number of the variable address within the operation. This may range from Δ1 to 20 for any single operation.





### Detail Card (Card Type 8)

The Detail Cards contain lines of coding or constants. The card numbers are in ascending sequence starting one higher than the Header card number. An operation may contain a maximum of 999 Detail cards.

CARD TYPE	OPER. NO.	CARD NO.	<i>a</i>					CONTROL	<i>m</i>				<i>c</i>								
8	h h h	y y y	a	a	a	a	a	1	1	1	1	1	m	m	m	m	c	c	c	c	12
																					34
																					56
																					78
																					90
<b>COMMENTS - ANY DESCRIPTIVE ENGLISH</b>																				12	
																				34	
																				56	
																				78	
																				90	

Printed in U.S.A. REMINGTON RAND

- Here,
- h h h** is the operation number.
  - y y y** is the card number.
  - a a a a a** is the X-6 *a* address
  - 1 1 1** is the mnemonic instruction code.
  - m m m m m** is the X-6 *m* address.
  - c c c c c** is the X-6 *c* address.

Column 16 on the Detail card is designated as a control column and may contain the following entries;

- 1, 2, or 3** for the appropriate index register if index register modification is indicated.
- U or P** for the unprimed or primed word of an alphabetic constant.
- N or Z** for the numeric or zone portion of an alphabetic constant.
- 2** for a negative numeric constant.
- Δ** for a positive numeric constant.

**NOTE:**

Whether pooled or included with the coding, data constants are recognized by the absence of any entry in the columns reserved for a mnemonic instruction code. The ten-digit constant is listed in the *m* and *c* address columns on the Detail card. Positive numeric constants are indicated by a space ( $\Delta$ ) in the control column and negative ones by a two (2).

Sometimes, alphabetic constants are needed in either a two-part translated or untranslated form for printed or punched output. As a convenience, X-6 allows the programmer to write the alphabetic constant twice with N and Z, or with U and P in the control column. This makes it unnecessary for the coder to break up the alphabetic characters into the bit configurations which will recreate the desired alphabets when the data is to be printed or punched.

**End-Operation Sentinel Card (Card Type 9)**

One End-Operation Sentinel card must follow the last Detail card in an operation.

CARD TYPE	OPER. NO.	CARD NO.																																													
9	h h h	y y y																																													
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
			COMMENTS - ANY DESCRIPTIVE ENGLISH																																												
			46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90

Printed in U.S.A. REMINGTON RAND

Here, **h h h** is the operation number.  
**y y y** is the card number. This will be one higher than the number on the last Detail card of the operation.

**CONTROLS**

Certain cards are used as controls for the program organization and assembly. The *Label* card and the *End-Input* card, like the Header, Detail, and End-Operation Sentinel cards, *must be specified* in a program. *Restrict* cards, like the summary cards specified in the previous chapter (Tag-Equals, Tables, and so forth), are optional and dependent upon the particular application.



CARD TYPE	ENTRY 1										ENTRY 2										ENTRY 3																							
	i i n n n n s s s s										i i n n n n s s s s										i i n n n n s s s s										12													
2																															34													
																															56													
																															78													
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
Printed in U.S.A. - REMINGTON RAND	ENTRY 4										ENTRY 5										ENTRY 6										ENTRY 7													
	i i n n n n s s s s										i i n n n n s s s s										i i n n n n s s s s										i i n n n n s s s s										12			
																																									34			
																																									56			
																																									78			
																																									9			
46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90

Here,  $ii$  is the increment between restricted storage locations.  
 $nnnn$  is the total number of restricted locations.  
 $sssss$  is the absolute address of the first location in a restricted area.

**End-Input Card (Card Type 10)**

This card alerts the computer that all X-6 input has been received. It contains the first instruction of the assembled program which will be executed after the program is fully loaded. There is only one such card per program.

CARD TYPE	CONTROL																INST. CODE			$m$						$C$																				
																	I I I			m m m m m c c c c c						c c c c c c						12														
10																																34														
																																56														
																																78														
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45		
Printed in U.S.A. - REMINGTON RAND	COMMENTS - ANY DESCRIPTIVE ENGLISH																																													
																																														12
																																														34
																																														56
																																														78
																																														9
46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90		



## 5. X-6 Instruction Codes

### GENERAL INFORMATION

In addition to the use of relative and symbolic addressing, the X-6 System further facilitates the coding of data-processing runs by allowing the use of mnemonic instruction codes which are easily recognizable by the functions they cause to be performed in the computer. For example, applying the term *load* to transfers from storage to registers and using LD as the mnemonic contraction for load, the instruction code that would cause the transfer of data from some storage location to register A would be LDA. Similarly, applying the term *store* to transfers from registers to storage and using ST as the mnemonic contraction for store, the instruction code that would cause the transfer of data from register X to some storage location would be STX.

The following pages contain a repertory of X-6 mnemonic instruction codes, descriptions of each, and the timing, in word times, for the execution of each instruction. A *hyphen* in the *m* or *c* portion of an instruction indicates that the computer ignores that portion when the instruction is executed.

### TRANSFER INSTRUCTIONS

#### LOAD REGISTER A

LDA <i>m c</i>	Transfer the contents of storage location <i>m</i> to register A.	4
----------------	---	---

#### LOAD REGISTER X

LDX <i>m c</i>	Transfer the contents of storage location <i>m</i> to register X.	4
----------------	---	---

#### LOAD REGISTER L

LDL <i>m c</i>	Transfer the contents of storage location <i>m</i> to register L.	4
----------------	---	---

#### STORE REGISTER A

STA <i>m c</i>	Transfer the contents of register A to storage location <i>m</i> .	4
----------------	--	---

<b>STORE REGISTER X</b>			
STX	<i>m c</i>	Transfer the contents of register X to storage location <i>m</i> .	4
<b>STORE REGISTER L</b>			
STL	<i>m c</i>	Transfer the contents of register L to storage location <i>m</i> .	4
<b>TRANSFER REGISTER A TO L</b>			
ATL	- <i>c</i>	Transfer the contents of register A to register L.	3
<b>TRANSFER REGISTER C TO A</b>			
CTA	<i>m -</i>	Transfer the contents of register C to register A. The location of the next instruction is at <i>m</i> .	3
<b>CLEAR REGISTER A</b>			
CLA	<i>m -</i>	Clear register A to zeros and set sign to plus. The location of the next instruction is at <i>m</i> .	3
<b>CLEAR REGISTER X</b>			
CLX	<i>m -</i>	Clear register X to zeros and set sign to plus. The location of the next instruction is at <i>m</i> .	3
<b>CLEAR REGISTER L</b>			
CLL	<i>m -</i>	Clear register L to zeros and set sign to plus. The location of the next instruction is at <i>m</i> .	3
<b>CLEAR REGISTERS A AND X</b>			
CAX	<i>m -</i>	Clear register A and register X to zero (register A and register X assume sign of register L). The location of the next instruction is at <i>m</i> .	14
<b>CLEAR REGISTER A</b>			
CAA	<i>m -</i>	Clear register A to zero and retain original sign. The location of the next instruction is at <i>m</i> .	3

### Sample Problems

- Place the contents of storage location 4211 into working storage.

line	a	inst. code	m	c
1	1N	LDA	4211	
2		STA	W 1	.....

2. Transfer the contents of table element S3043 to register A and the contents of W $\Delta\Delta\Delta$ 1 to register L and clear W $\Delta\Delta\Delta$ 1 to zeros. The first instruction should be a temporary tag line in standard access storage.

line	a	inst. code	m	c
1	1N	LDA	S3043	
2		LDL	W 1	
3		CLX		.....
4		STX	W 1	.....

3. Transfer the contents of storage location 4331 to register L and a constant 000000128 to register A. Also, store the constant in the K area and zero fill location 4331. Tag the first instruction with a permanent tag to be assigned to high-speed access storage. It will be the first permanent tag in operation AAR.

line	a	inst. code	m	c
1	AAR1F	LDL	4331	
2		LDA		1N
3		...	00000	00128
4	1N	STA	K 1	
5		CLX		.....
6		STX	4331	.....

## ARITHMETIC INSTRUCTIONS

### ADD

**ADD  $m$   $c$**                       Add algebraically the contents of storage location  $m$  to the contents of register A and place the sum in register A.                      5

### SUBTRACT

**SUB  $m$   $c$**                       Subtract algebraically the contents of storage location  $m$  from the contents of register A and store the difference in register A.                      5

### MULTIPLY

**MUL  $m$   $c$**                       Multiply the contents of register L by the contents of location  $m$  and store the ten most significant digits of the product in register A and the ten least significant digits in register X. Both register A and                      105

register X will have the sign of the product. Multiplication can be controlled by placing a sentinel in multipliers having less than ten digits. The sentinel, placed just to the left of the most significant digit of the multiplier, stops the multiplication after the last significant multiplier digit is used. The sentinels 0101 or 1101 may be indicated by a nonnumeric A or F. If a program sentinel is used, the number of significant digits of the product that will appear in register X is equal to the number of significant digits in the multiplier. For example, if a program sentinel is used, in a multiplication that results in a five-digit product (xxxxx) and the multiplier is a three-digit number, the entire product would appear in registers A and X in the following manner.

Register A

0 0 0 0 0 0 0 0 X X

Register X

X X X 0 0 0 0 0 0 0

**DIVIDE**

**DIV *m c***

Divide the contents of storage location *m* by the contents of register L. The quotient with its sign is placed in register A unrounded and the remainder is placed in register X. If the divisor is zero, or it is less than or equal to the dividend, overflow occurs. Division may be controlled by placing a sentinel of 0101 (a nonnumeric A) in register X. The sentinel will control the number of digits developed in the quotient. This number must always be even. Therefore, to develop a two-digit quotient, the sentinel would be placed in digit position three of register X; a four-digit quotient, digit position five; a six-digit quotient, digit position seven, and so forth. To develop ten digits of the quotient, no sentinel is needed and the contents of register X need not be changed. It must be ascertained,

115

however, that no bit configuration of 0101 exists in register X at the time of the arithmetic computation. If there is a possibility that such a configuration does exist in register X, the register should be filled with zeros.

**NOTE:** If an overflow occurs as the result of an add, subtract, or divide instruction, the location of the next instruction is at  $c+1$ . If the  $c$  address is at word level 199, overflow will cause control to revert to the instruction at word level 000 of the same band. If an arithmetic register is used as the  $c$  address of an instruction in which overflow occurs, the next instruction is still taken from that register after a delay of one word time.

### Sample Problems

1. Add the contents of storage location 4211 and  $K\Delta 146$ . Place the sum in 4103.

line	a	inst. code	m	c
1	1N	LDA	4211	
2		ADD	K 146	
3		STA	4103	.....

2. Reduce the contents of  $W\Delta 102$  and  $W\Delta 205$  by 7. Place the result in two consecutive entries in table U4.

line	a	inst. code	m	c
1	1N	LDL		2N
2		...	00000	00007
3	2N	LDA	W 102	
4		SUB	RL	
5		STA	U4001	
6		LDA	W 205	
7		SUB	RL	
8		STA	U4002	

3. Multiply the contents of storage location 4331 by the contents of K $\Delta$ 49. Store the product in two consecutive working storage areas and clear registers A and X to zeros.

line	a	inst. code	m	c
1	1N	LDL	4331	
2		MUL	K 49	
3		STA	W 1	
4		STX	W 2	
5		CAX	.....	.....

### COMPARISON INSTRUCTIONS

#### TEST EQUALITY

TEQ m c	3
Compare the contents of register A and the contents of register L. If the contents of both registers are equal, the location of the next instruction is specified in m; if they are unequal, the location of the next instruction is specified in c.	

#### TEST GREATER

TGR m c	3
Compare the contents of register A and the contents of register L. If the contents of register A are greater than the contents of register L, the location of the next instruction is specified in m; if not, the location of the next instruction is specified at c.	

#### Sample Problems

1. Quantity Y is in storage location 4251 and quantity Z in 4371. If Y equals Z, add the two quantities and store the sum in W $\Delta$ 29. If Y is greater than Z, subtract Z from Y and store the difference in W $\Delta$ 30. If Y is less than Z, store Y in W $\Delta$ 31 and Z in W $\Delta$ 32 and clear 4351 and 4371 to zeros. Jump to ASINF after all housekeeping functions are performed.

line	a	inst code	m	c
1	1N	LDA	4251	
2		LDL	4271	
3		TEQ		2N
4		ADD	RL	
5		STA	W 29	ASINF

line	a	inst. code	m	c	(continued)
6	2N	TGR		3N	
7		SUB	RL		
8		STA	W 30	ASINF	
9	3N	STA	W 31		
10		STL	W 32		
11		CLX			
12		STX	4351		
13		STX	4371	ASINF	

## EDITING INSTRUCTIONS


### BUFF

**BUF *m c*** Superimpose or buff the 1 bits of the word whose location is specified in *m*, onto the contents of register A and leave the result in register A. The sign of register A remains unchanged. 4


### ERASE

**ERS *m c*** Change the bits in each digit position of register A to binary zero wherever the word, in storage location *m* has a zero in the corresponding bit position. The sign of register A remains unchanged. 4

### SHIFT RIGHT

**SHR *m c***  

 Shift the contents of register A *nn* digit positions to the right into register X which is also shifted *nn* digit positions to the right into register A. Here, *nn* = 00 through 10. The signs of both registers remain unchanged. 3 + *nn*

### SHIFT LEFT

**SHL *m c***  

 Shift the contents of register A *nn* digit positions to the left losing the most significant digits and bringing in zeros to the least significant digit positions. The sign of register A remains unchanged. Here, *nn* = 00 through 10. 3 + *nn*

**ZERO SUPPRESS**

**ZUP - c**

Suppress zeros and commas preceding the first significant digit of a field by inserting spaces. Before execution, register A will contain the numeric portion and Register X the zone. Results will be in register A and register X.

4

**Sample Problems**

1. Store a field in W $\Delta\Delta\Delta$ 1 in the form;

0000123456

The field is initially stored as

xx123456xx

in storage location 4211. Here, x is an unknown quantity to be edited out.

line	a	inst. code	m	c
1	1N	LDA	4211	
2		SHR	00002	
3		ERS		2N
4		...	0000H	HHHHH
5	2N	STA	W 1	.....

2. Store a field in W $\Delta\Delta\Delta$ 1 in the form;

0000123456

The field is initially stored in two locations;

x123xxxxxx

in location 4013 and;

xxxxxxx456

in location 4033. Again, x is an unknown quantity to be edited out.



line	a	inst. code	m	c
1	1N	LDA	4013	
2		SHR	00003	
3		ERS		2N
4		...	0000H	HH000
5	2N	ATL	.....	
6		LDA	4033	
7		ERS		3N
8		...	00000	00HHH
9	3N	BUF	RL	
10		STA	W 1	

## CONTROL INSTRUCTIONS

### STOP

STP *m c*

Stop the computer. The computer stops with the stop instruction in register C. This occurs before the next instruction is started. Normally, when the computer is restarted, the first step will be to search for the next instruction specified by the *c* address. In this case, the *m* digits are ignored, and may be used as a code to indicate the reason for stopping. However, if desired, the *m* address may be used as an alternate restart location by depressing the *m* button on the control panel.

Ind.

### JUMP

JMP *m -*

Jump to the instruction whose address is specified in *m*.

2

## TRANSLATE INSTRUCTIONS

### CARD-TO-MACHINE CODE

CTM - *c*

Translate from card code to machine (computer) code. Before the command is given, register A must contain the unprimed word and register X the primed word of the field to be translated. After the command is executed, register A will contain the numeric and register X the zone in computer code. The signs remain unchanged.

3

## MACHINE-TO-CARD CODE

**MTC - c** Translate from machine (computer) code to card code. Before the command is given, register A must contain the numeric and register X the zone in computer code. After the command has been executed, the unprimed word is in register A, and the primed word in register X. The signs of registers A and X are positive. 3

## TRANSLATE XS-3 TO MACHINE CODE

**TXM - c** Translate UNIVAC XS-3 to machine (computer) code. Before this command is given, register A must contain the numeric portion of the word to be translated. The zone is the same for both XS-3 and computer code. The sign remains unchanged. 3

## TRANSLATE MACHINE TO XS-3 CODE

**TMX - c** Translate machine (computer) code to XS-3 code. Before this command is given, register A must contain the numeric portion of the word to be translated. The zone is the same for both computer and XS-3 code. The sign remains unchanged. 3

## INDEX REGISTER INSTRUCTIONS

### LOAD INDEX REGISTER

**LIR *m c*** Load *m* portion into the appropriate index register. 3

### INCREMENT INDEX REGISTER

**IIR *m c*** Add *m* to the contents of the specified index register. The sum is entered in the specified index register and in register A in digit positions 3 through 6. The remainder of register A is cleared to zeros, and the sign of register A set to plus. 4

**NOTE:** The *m* addresses of the LIR and the IIR instructions do not refer to actual storage locations. If this address happens to be the same as that of an actual location, the contents of that location will be unaffected.

## INPUT INSTRUCTIONS (High-Speed Reader)


### HIGH-SPEED READER CARD CYCLE

<b>HCC</b> <i>m c</i>	Initiate card movement in the High-Speed Reader. The card fed to the Reader is sensed and its image stored in the buffer band. The next instruction is normally at <i>c</i> except when a HCC is given before the preceding HCC instruction has had a chance to feed a card. In this case, the second HCC instruction is not executed; the contents of register C go to register A, and the next instruction is specified at <i>m</i> .	3 (4 if <i>m</i> )
-----------------------	---	-----------------------


### HIGH-SPEED READER BUFFER TEST

<b>HBT</b> <i>m c</i>	Test the buffer of the High-Speed Reader. If it is loaded, the contents of register C are transferred to register A and the location of the next instruction is specified by <i>m</i> . If the buffer is not loaded, the location of the next instruction is specified by <i>c</i> and the contents of register A are not altered.	3 (4 if <i>m</i> )
-----------------------	--	-----------------------

### HIGH-SPEED READER BUFFER UNLOAD


<b>HBU</b> <i>m c</i>  <b>Hn00d</b>	Transfer the contents of the High-Speed Reader buffer to storage according to the predetermined interlace pattern. Here: <i>n</i> = interlace number (0-9) <i>d</i> = 0 for normal translation <i>d</i> = 1 for automatic translation	203 <sup>4</sup>
--	--	------------------

### HIGH-SPEED READER STACKER SELECTION

<b>HSS</b> <i>m c</i>  <b>ΔΔn00</b>	Select the output stacker of the High-Speed Reader. Here: <i>n</i> = Stacker 0, 1, or 2.	3
--	---	---

## INPUT-OUTPUT INSTRUCTIONS (Read-Punch Unit)

### READ-PUNCH UNIT CARD CYCLE

<b>RCC</b> <i>m c</i>  <b>0n00d</b>	Initiate card movement in the Read-Punch Unit. Here: <i>n</i> = interlace number (0-9). <i>d</i> = 0 for normal translation. <i>d</i> = 1 for automatic translation.	103 <sup>4</sup>
--	---	------------------

<sup>4</sup>Word-time applicable only when *d* = 0; when *d* = 1 word time is 208 for RPU and 207 for HSR.

## READ-PUNCH UNIT BUFFER TEST

RBT *m c*

Test the input buffer of the Read-Punch Unit. If it is loaded, transfer the contents of register C to register A. The location of the next instruction is specified by *m*. If the buffer is not loaded, the location of the next instruction is specified by *c* and the contents of register A are unaltered.

3  
(4 if *m*)

## READ-PUNCH UNIT BUFFER UNLOAD

RBU *m c*

  
Rn00*d*

Transfer the contents of the Read-Punch Unit buffer to storage according to the predetermined interlace pattern. Here: *n* = interlace number (0-9).  
*d* = 0 for normal translation.  
*d* = 1 for automatic translation.

203 <sup>5</sup>

## READ-PUNCH UNIT STACKER SELECTION

RSS - *c*

Select output stacker 1 of the Read-Punch Unit. Stacker 0 is automatically selected if no specification is made.

3

## OUTPUT INSTRUCTIONS (High-Speed Printer)

### PRINT-BUFFER TEST

PBT *m c*

Test to see if the High-Speed Printer is free for use. If it is free, the contents of register C go to register A and the location of the next instruction is specified by *m*. If the printer is not free for use, control is transferred to *c* and the contents of register A remain unaltered.

3  
(4 if *m*)

### PAPER FEED

PF D *m c*

  
ΔΔΔ*yy*

Advance the paper in the printer *yy* lines. Here *yy* may vary from 00 to 79.

4

### PRINT

PRN *m c*

  
Pn0*yy*

Advance the paper *yy* lines and print one line. Registers A and X are used for the transfer and, therefore, their contents are destroyed.

Here: *yy* = number of lines to advance (00-79).  
*n* = interlace number (0-9).

592

<sup>5</sup>Word-time applicable only when *d* = 0; when *d* = 1 word-time is 208.

## INPUT-OUTPUT INSTRUCTIONS (Tape Synchronizer)

### TAPE READ

TRD *m c*

$\Delta\Delta xyz$

Read one block of information from tape onto the tape buffer.

17

Here: *x* = Servo number (0-9).  
*y* = 0 if USS mode.  
*y* = 5 if UNIVAC XS-3 mode.  
*z* = Direction and gain—  
0—forward normal.  
1—forward low.  
2—forward high.  
5—backward normal.  
6—backward low.  
7—backward high.

### TAPE WRITE

TWR *m c*

$\Delta\Delta xy0$

Write one block from the tape buffer band onto tape.

17

Here: *x* = Servo number (0-9).  
*y* = Mode and density—  
0—USS 250 cpi.<sup>6</sup>  
5—UNIVAC 250 cpi.  
6—UNIVAC 125 cpi.

### TAPE BUFFER TEST

TBT *m c*

Test the Tape Buffer to determine whether it is available, or in use. If the buffer is available the location of the next instruction is specified by *m*. If the buffer is currently being used, the location of the next instruction is specified by *c*.

3  
(5 if *m*)

### TAPE SERVO TEST

TST *m c*

Test for servo availability. If the test indicates that a tape-handling instruction is in progress, control is transferred to *c* for the next instruction. If the servo is ready for a new instruction, transfer the contents of register C to register A and go to *m* for the location of the next instruction.

3  
(4 if *m*)

### TAPE BUFFER UNLOAD

TBU *m c*

$Tn000$   
 $Zn000$

Transfer the contents of the tape-buffer band to storage.

205

Here: *n* = interlace number (0-9).

<sup>6</sup> Characters-per-inch

**TAPE BUFFER LOAD**

TBL *m c*  
     $\overbrace{Tn000}$   
     $Zn000$

Transfer the contents of the specified  
tape interlace to the tape-buffer.      205  
Here:    *n* = interlace number (0-9).

**TAPE REWIND**

TRW *m c*  
     $\overbrace{\Delta\Delta xy0}$

Rewind the tape to the first block condi-      600 *ms*  
tion.  
Here:    *x* = servo number  
          *y* = 0 to rewind without interlock.  
          *y* = 2 to rewind with interlock.

## **6. How X-6 Works**

### **INPUT PROCESSING**

Programmers will be able to make more effective use of X-6 if they understand how it performs its function.

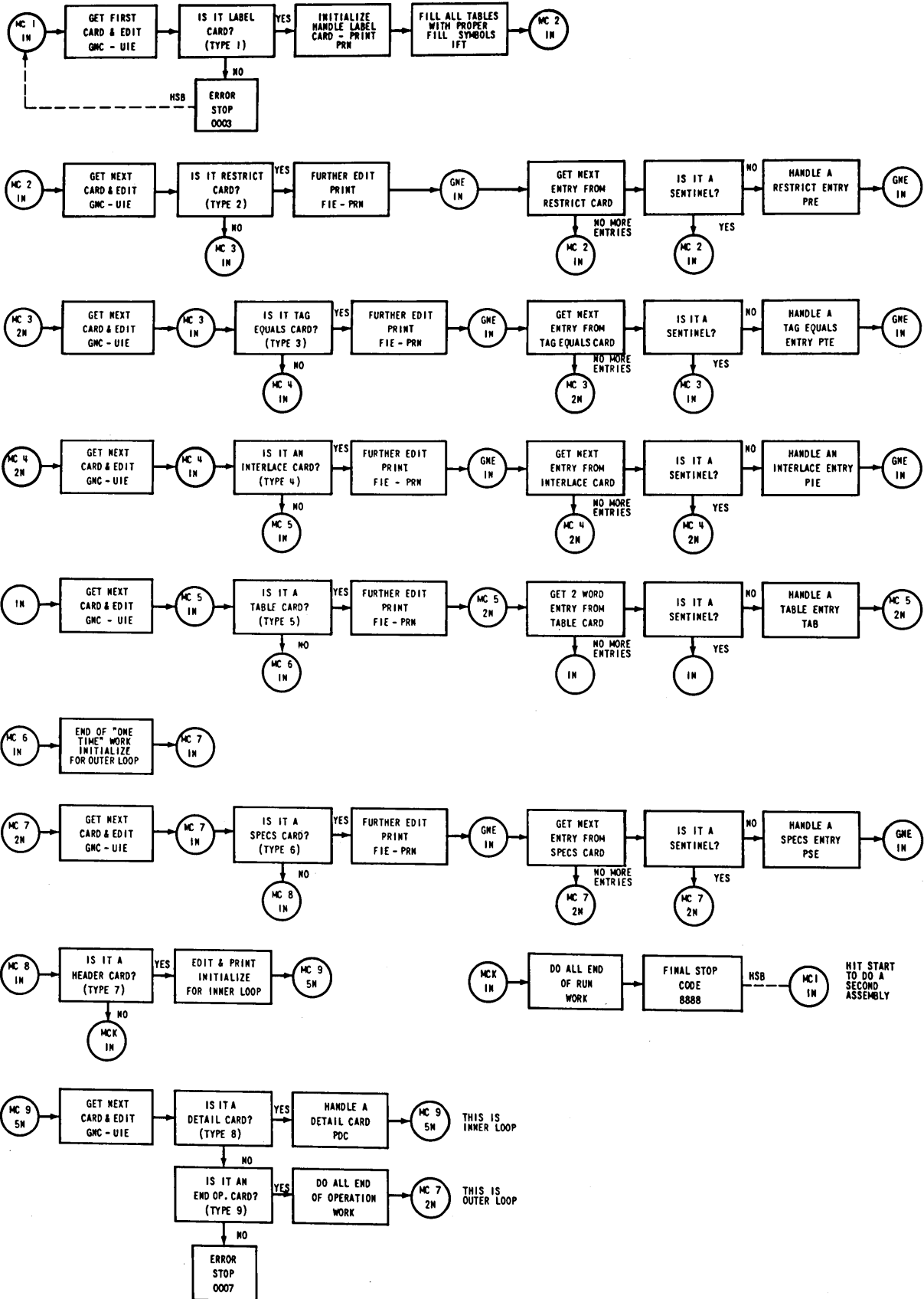
Each input card type follows a different path. A brief statement of the steps performed on each card type follows:

1. The fields in the Label card (type 1) are carried over to the output without modification.
2. The entries in Restrict cards (type 2) are used to mark off locations in a storage availability table. X-6 will not allocate any restricted addresses.
3. The entries in Tag-Equals cards (type 3) are filed in internal tables equated to the given absolute addresses. The absolute addresses are marked off in the storage availability table.
4. The entries in Interlace cards (type 4) are used to mark off interlace positions in the storage availability table, and the origins are filed for future use.
5. The entries in Table cards (type 5) are handled in a similar fashion. The only change is that increments as well as origins are saved for future use.

Card types 1-5 must have been received in order, and after the first type 6 or 7 card, no additional 1-5 cards will be accepted. At this point the initial phase of X-6 is complete, and from this point on the routine expects card types 6-9 or 7-9 on a per-operation basis.

6. The entries in Specifications cards (type 6) are filed in tables for direct substitution later.
7. The Header card (type 7) is used to initialize for the detail cards which follow.
8. Detail cards (type 8) encompass all of the lines of coding and constants which make up a routine. Only detail cards cause output punching. Processing these cards is the primary function of X-6.
9. The End-Operation Sentinel card (type 9) signifies that the last card in a group of detail cards has been received.
10. The End-Input card (type 10) indicates the end of a run. It contains the instruction which will be used by the loading routine to start the execution of the assembled program.

# X-6 ASSEMBLY SYSTEM BLOCK CHART





## OPTIMIZATION OF INSTRUCTIONS

Optimization, if the X-6 assembler is free to optimize, is achieved with the use of a working storage location designated as the Clock. The format of this location is:

00 0000 0xxx

where xxx designates the current band-relative address and may vary from 000 to 199. The Clock location is initially set to

00 0000 0000

and its value increases as each instruction is assigned to storage. By adding the current Clock reading and the word time lapse for execution to a particular band origin the *tentative best address* (TBA) is established for each new address encountered.<sup>7</sup> If the address is available, it is marked off in a storage availability table and, after assignment is made, the band-relative position of the TBA becomes the new Clock reading. If the address is not available, the assembler keeps searching until an available address is located.

The following is a general outline of the manner in which the Clock is employed when optimizing.

- A. The *a* address is moved to working storage W $\Delta\Delta\Delta$ 0 and analyzed in the following manner.
  1. If the *a* address contains spaces, it will be assigned the same absolute address as the previous *m* or *c* address which referenced it.
  2. If the *a* address contains a legitimate symbolic address (tag, K or W specification) the appropriate table is searched to determine whether or not the symbolic specification has been previously assigned an absolute address either by the assembler or the programmer. If it has, the band-relative position of the previously assigned absolute address becomes the new Clock reading. If the symbolic address has not been previously assigned an absolute address, X-6 will assign one based on the current reading of the Clock.
- B. The mnemonic instruction code is translated to its machine code equivalent.
- C. The word-time lapse that will occur between the execution of the *a* and the execution of the *m* address is added to the Clock to form the TBA.
- D. The *m* address is moved to working storage W $\Delta\Delta\Delta$ 0 and analyzed in the following manner.
  1. If the *m* address contains spaces, an absolute address is assigned based on the current band-relative position indicated by the Clock. A switch is set so that the succeeding *a* address will be examined and assigned the same absolute location in storage.
  2. If the *m* address contains a legitimate symbolic address, the analysis is the same as for A-2.
- E. The word-time lapse that will occur between the execution of the *m* and the execution of the *c* address is added to the Clock of form the TBA.

---

<sup>7</sup> Table 1. Instruction Code Information Words contains the execution time for X-6 instruction codes.

TABLE 1. INSTRUCTION CODE INFORMATION WORDS

X-6 INST. CODE	ACTION CODE	EXECUTION TIME BEFORE <i>m</i>	EXECUTION TIME BEFORE <i>c</i>
ADD	0	002	003
BUF	0	002	002
DIV	0	002	113
ERS	0	002	002
LDA	0	002	002
LDL	0	002	002
LDX	0	002	002
MUL	0	002	103
STA	0	002	002
STL	0	002	002
STX	0	002	002
SUB	0	002	003
LIR	0	000	002
IIR	0	000	003
TRD	1	000	017
TWR	1	000	017
TRW	1	000	150
TMX	1	000	003
TXM	1	000	003
ATL	1	000	003
CTM	1	000	003
MTC	1	000	003
ZUP	1	000	004
HSS	1	000	003
RSS	1	000	003
CLA	2	003	000
CLL	2	003	000
CLX	2	003	000
JMP	2	002	000
CAA	2	003	000
CAX	2	014	000
CTA	2	002	000
PFD	3	222*	003
SHL	3	111†	003
SHR	3	111†	003
HBU	4	198	203
PRN	4	197	592
RBU	4	098	203
RCC	4	098	203
TBU	4	048	103
TBL	4	198	205
HBT	5	004	003
HCC	5	004	003
PBT	5	004	003
RBT	5	004	003
STP	5	003	003
TEQ	5	003	003
TGR	5	003	003
TBT	5	005	003
TST	5	004	003

If control column indicates index register modification, add one more word time before *m*.

\* is a code not affecting timing.

† use amount of shift.

F. The symbolic *c* address is moved to working storage W $\Delta\Delta\Delta$ 0 and analyzed in the following manner.

1. If the *c* address contains spaces, an absolute address is assigned based on the current band-relative position indicated by the Clock. A switch is set so that the next *a* address is examined and assigned the same absolute location in storage.
2. If the *c* address contains a legitimate symbolic address, the analysis is the same as for A - 2.

G. Exit.

### CLOCK MODIFICATION

The Clock-modification option is provided to allow the programmer to interrupt the normal sequence of address assignment by altering the reading of the Clock. The modification option affords the following capabilities:

1. The programmer can direct the routine to add or subtract a specific number of word times from the band-relative address that would normally be assigned to the *a*, *m*, and *c* portion of an instruction. The programmer may also specify a new reference point from which the adjustment is to be made. This point may have been previously identified or it may still remain to be assembled by the routine.
2. The programmer can direct the X-6 System to reset the Clock to one of the following conditions after assignment has been made.
  - a. The reading before the adjustment was made.
  - b. A reading based on the adjusted address.
  - c. A reading based on the adjusted address and further modified by the addition of a specified number of word times.

### CLOCK MODIFICATION INSTRUCTIONS

All clock-modifying instructions will contain the word CLOCK in the *a* address. This will indicate to the assembler that a modification to an address is to be made. The modification instruction will immediately precede the instruction containing the address to be adjusted. The following are internal instructions to the X-6 system. These instructions are to be key-punched in the same way as any X-6 instruction. The card containing the modification instruction is filed immediately before the card containing the instruction to be modified. No card number will be entered on the modification card thereby allowing their insertion into an operation without disturbing the Detail card sequence.

$\Delta\Delta\Delta$	$\underbrace{\quad m \quad}$	$\underbrace{\quad c \quad}$	Modify the succeeding <i>a</i> address.
	$\underbrace{\quad sssss \quad}$	$\underbrace{\quad 00\ iii \quad}$	Here: $sssss$ is any legitimate X-6 symbolic address.
			$iii$ is a numeric increment that is to be added to the absolute equivalent of this symbolic designation.

The *a* address of the instruction to be modified will be assigned an absolute address that is *iii* word times from the absolute location of *sssss*.

ADA  $\overset{m}{\underbrace{\quad\quad\quad}}_{xxx0n}$   $\overset{c}{\underbrace{\quad\quad\quad}}_{00iii}$

Modify the succeeding  $a$  and  $m$  addresses.

Here:  $iii$  is a numeric increment added to the Clock reading which will be assigned to the next  $a$  address.

$xxx$  is a numeric increment that is to be added to the new Clock reading to derive the  $m$  address of the next instruction.

$n$  is 0 if the adjusted Clock reading is to be used to derive the  $m$  address of the next instruction.

$n$  is 1 if the Clock reading prior to adjustment is to be used to derive the  $m$  address of the next instruction.

If  $xxx$  is zeros (000), the next  $m$  address will be derived normally, either from the adjusted or the preadjusted Clock reading. This will depend upon whether  $n$  equals 0 or 1.

ADM  $\overset{m}{\underbrace{\quad\quad\quad}}_{xxx0n}$   $\overset{c}{\underbrace{\quad\quad\quad}}_{00iii}$

Assign the next  $a$  normally and modify the succeeding  $m$  and  $c$  addresses.

Here:  $iii$  is a numeric increment added to the Clock reading which will be assigned to the next  $m$  address.

$xxx$  is a numeric increment that is to be added to the new Clock reading to derive the  $c$  address of the next instruction.

$n$  is 0 if the adjusted Clock reading is to be used to derive the  $c$  address of the next instruction.

$n$  is 1 if the Clock reading prior to adjustment is to be used to derive the  $c$  address of the next instruction.

If  $xxx$  is zeros (000), the  $c$  address of the next instruction will be derived normally, either from the adjusted or the preadjusted Clock reading. This will depend upon whether  $n$  equals 0 or 1.

ADC  $\overset{m}{\underbrace{\quad\quad\quad}}_{xxx0n}$   $\overset{c}{\underbrace{\quad\quad\quad}}_{00iii}$

Assign the next  $a$  and  $m$  normally and modify the succeeding  $c$  address.

Here:  $iii$  is a numeric increment added to the Clock reading which will be assigned to the next  $c$  address.

$xxx$  is a numeric increment that is to be added to the new Clock reading to derive the  $a$ ,  $m$ , or  $c$  portion of the instruction directly succeeding the next  $c$  address.

$n$  is 0 if the adjusted Clock reading is to be used to derive the address to which the next  $c$  will transfer control.

*n* is 1 if the Clock reading prior to adjustment is to be used to derive the address to which the next *c* will transfer control.

If *xxx* equals zeros (000), the address to which the next *c* address will transfer control will be derived either from the adjusted or the preadjusted Clock reading. This will depend upon whether *n* equals 0 or 1.

It should be noted that if the next *c* is spaces, the *a* address following it will be assigned the same absolute location in storage.

SEA  $\overset{m}{\underbrace{\quad\quad\quad}}_{xxx0n}$   $\overset{c}{\underbrace{\quad\quad\quad}}_{sssss}$

Modify the succeeding *a* address.

Here: *sssss* is any legitimate X-6 symbolic address.

*xxx* is a numeric increment that is to be added to the band-relative position of the symbolic address to derive the next *a* address.

*n* is 0 if the next *m* address will be derived normally from the adjusted Clock reading.

*n* is 1 if the next *m* address will be derived normally from the Clock reading prior to adjustment.

If *xxx* equals zeros (000), the band-relative position of symbolic designation *sssss* will be assigned to the next *c* address.

SEM  $\overset{m}{\underbrace{\quad\quad\quad}}_{xxx0n}$   $\overset{c}{\underbrace{\quad\quad\quad}}_{sssss}$

Assign the next *a* address normally and modify the succeeding *m* address.

Here: *sssss* is any legitimate X-6 symbolic address.

*xxx* is a numeric increment that is to be added to the band-relative position of the symbolic address to derive the next *m* address.

*n* is 0 if the next *c* address will be derived normally from the adjusted Clock reading.

*n* is 1 if the next *c* address will be derived normally from the Clock reading prior to adjustment.

If *xxx* equals zeros (000), the band-relative position of symbolic designation *sssss* will be assigned to the next *m* address.

SEC  $\underbrace{m}_{xxx0n}$   $\underbrace{c}_{sssss}$  Assign the next *a* and *m* normally and modify the succeeding *c* address.

Here: *sssss* is any legitimate X-6 symbolic address.  
*xxx* is a numeric increment that is to be added to the band-relative position of the symbolic address to derive the next *c* address.  
*n* is 0 if the adjusted Clock reading is to be used to derive the address to which the next *c* will transfer control.  
*n* is 1 if the Clock reading prior to adjustment is to be used to derive the address to which the next *c* will transfer control.

If *xxx* equals zeros (000), the band-relative position of symbolic designation *sssss* will be assigned as the next *c* address.

**NOTE:** The number of word times used to modify an address is *always added* to the Clock reading. Therefore, to decrement the Clock reading, the number of word times is subtracted from 200, and the result is used as the modifying information.

#### Examples Employing Clock Modification

The following examples illustrate some of the possible uses of the Clock modification.

- Two quantities, in  $K\Delta\Delta 29$  and  $W\Delta\Delta 10$ , are to be compared. Control will be transferred on the basis of this comparison. If  $K\Delta\Delta 29$  is either greater than or equal to  $W\Delta\Delta 10$  control will be transferred to 4N for further processing. If neither of these conditions is met, control will be transferred to 7N.

line	a	inst. code	m	c
1	1N	LDA	K 29	
2		LDL	W 10	
3		TEQ	4N	
4		TGR	4N	7N

The X-6 assembler might produce the following computer coding.

line	a	inst. code	m	c	W/T
1	0200	25	4002	0204	4
2	0204	30	4406	0208	4
3	0208	82	0411	0211	3/3
4	0211	87	0411	0214	200/3

It can be seen that it could take a possible 211 word times to execute these four lines of coding if both *m* addresses, in each comparison, have a common transfer point. However, by inserting a Clock modification instruction in the X-6 coding, the execution time for these four lines can be reduced.

line	a	inst. code	m	c
1	1N	LDA	K 29	
2		LDL	W 10	
	CLOCK	ADM	00001	00003
3		TEQ	4N	
4		TGR	4N	7N

Three word times are added to the reading that the Clock would normally assign to tag 4N and then the Clock is restored to its initial value to derive the *c* address. The maximum number of word times it would now take to execute these four instructions would be 17 or a saving of 194 word times.

line	a	inst. code	m	c	W/T
1	0200	25	4002	0204	4
2	0204	30	4406	0208	4
3	0208	82	0414	0211	6/3
4	0211	87	0414	0214	3/3

2. Two quantities, located in WΔ99 and WΔ183, are to be multiplied together. Optimum execution time, based on the sentinel placement in the multiplier, should be 40 word times between *a* and *c* of the multiplication line. Modify the multiplication line so that this optimization will be achieved in the final absolute address assignment.

line	a	inst. code	m	c
1	1N	LDL	W 99	
	CLOCK	ADC	00001	00135
2		MUL	W 183	

Since normal assignment of *c* would be based on a reading that is 105 word times greater than *a*, and since it is desired that execution time between *a* and *c* be only 40 word times, the Clock would have to be decremented by 65 word times before *c* is assigned.<sup>8</sup> This is accomplished by subtracting the desired decrement from 200, adding the result to the Clock, and deriving *c* from that reading. The coding that would be produced by the assembler might appear in the following manner.

line	a	inst. code	m	c	W/T
1	0200	30	4002	0204	4
2	0204	85	4006	0444	40

---

<sup>8</sup>*X-6 always assigns 105 word times in a multiplication.*



## **7. Programming Procedure**

### **SUGGESTIONS FOR FLOW-CHARTING**

1. Define your operations as you flow-chart; keep them short.
2. Use large circles for communication links between operations; assign permanent tags to these circles.
3. Use smaller circles for communication links within operations; assign temporary tags to these circles.
4. Use X-6 symbology in the flow chart. Assign table and interlace symbols, and working storage addresses at this time.

### **CODING**

1. Start each operation with a header, card type 7, on a new piece of coding paper.
2. Code the main chain first and then the lesser used branch paths. Since each address is allocated the first time it is encountered, this technique will produce better minimization.
3. Use the comments columns liberally. The X-6 edited listing will be more valuable if full comments are appended. Limit your comments to numerics and alphabets.
4. Use the card number as a cross reference to the box on the flow chart.
5. End each operation with an end-operation sentinel, card type 9.
6. Be sure all working storages are filled properly initially. Main storage is often filled with stop orders rather than zeros.
7. Buffer tests must be inserted by the programmer when required. Accurate estimates can be made by consulting the table *Instruction Codes Information Words*.

### **PREPARATION FOR ASSEMBLY**

1. Have all operations keypunched and verified.
2. Obtain any needed library routines and prepare specification cards.
3. Prepare card types 1, 2, 3, 4, 5, and 10 if this has not already been done. Be sure to restrict the area used by the standard loading routine.
4. Arrange the input deck in the desired order. If the program is very large, place the most important operations first; they will get better minimization.
5. Sight check the separate operations to make certain that card types 7, 8, and 9 within each operation are identically punched in columns 3-5 (operation number).
6. Either manually or by machine, check that card numbers are ascending within operations with no omissions.

## **ASSEMBLY**

1. Follow the X-6 operating instructions.
2. Check the edited listing carefully. All detected input data errors are coded and tabulated in print word 01 on the listing. These errors must be corrected before debugging can commence.
3. The output program deck is complete in stacker zero of the Read-Punch Unit. Any cards in stacker 1 should be destroyed.

## 8. Operating Instructions

### LOADING AND ASSEMBLING

1. Load X-6 Program. If the deck is in the three instruction per card format use PLD01. If it is in the one instruction per card format use PTA01.
2. After X-6 is loaded, or earlier:
  - a. Feed blank cards through to all stations of the RPU.
  - b. Advance paper in HSP so six free holes show above the paper holding clamps.
  - c. Put X-6 input program deck in the HSR.
3. To assemble a program:
  - a. Go on continuous, general clear, and run.
  - b. Successful stop is 67888cccc.
  - c. Error stops are listed on the following pages along with error codes which do not stop the computer.
4. Get a memory dump to preserve the information accumulated during the assembly which will be useful for debugging.
5. The following routines might also be used, after an X-6 assembly, if desired.
  - a. X-6LNU – Produces a list of all storage locations not used by the assembled program. This routine should follow the memory dump.
  - b. MAC02 – Produces a listing of all storage locations with page and line number of the program contents.
6. To reassemble, load the last 110 cards of the X-6 deck.

### ERROR CODES

These error codes appear on the printer listing as shown on page 51.

CODE	ORIGINATES IN OP.	MEANS
A	PTS	More than 300 perm. tags. Address 9999 has been assigned.
B	TTS	More than 50 temp. tags. Address 9999 has been assigned.
C	KWS	Address higher than K 299 or W 299 has been requested. 9999 has been assigned.
D	MAR	No more storage – have assigned 9999.

CODE	ORIGINATES IN OP.	MEANS
F	STS	Nothing in specs table matches this "X" symbolic address. Absolute 9999 has been assigned.
G	AAR	An incorrect a address. Previous instruction had blanks in m or c part. This a should have been blank. This a has been processed properly – the previous line must be fixed.
H	PDC (AC 2)	Spaces in m and c. Spaces in m will be assumed to be in error.
I	ICA	Invalid instruction code.
J	IAP	Reference has been made to a word part in an interlace which was not properly restricted in summary card type 4. Address of 9999 has been assigned.

### STOP CODES

These stop codes appear in the *m* portion of a STP order on the printer listing.

CODE	ORIGINATES IN OP.	MEANS
0001	GN2	The card being diverted to HSR Stacker #2 has failed to pass read check. Reposition cards and hit start to try again.
0002	GNC	Malfunction in HSR has caused overflow. Fix trouble. Hit start to try again.
0003	MC1	No Label card (type 1). Prepare Label card. Reposition input deck. Hit start to begin again.
0004	PSE	Too many specs for current library routine. Hit start to proceed. Error code F will appear later.
0005	PRN	Malfunction in printer has caused overflow. Fix trouble. Hit start to print current line. (IT WAS PRN ORDER THAT CAUSED IT)
0006	PUN	Malfunction in RPU. Fix trouble. Hit start to execute punch order.
0007	MC9	Card type sequence error. Check last card read. If it is a type 7 card, hit start to get to next stop order. Go to c to process card. If it is type 8, go to m of next stop order.

TSTPR	OP	CD	LOCA	OP	MMMM	CCCC	K	A	TAG	C	OP	M	TAG	C	TAG	TP	CD	COMMENTS	54059	PAGE	6
	ERR	1														7					
	ERR	2	0601	25	403	205		ERR1N			LDA				1N	8	0060	TEST ERROR PATH G AND H IN ACO			
	GERR	3	0405	30	207	209			2N		LDA					8	0061				
	HERR	4	0209	82	612	412					TEQ					8	0062	TEST ERROR PATH H IN AC5			
	CAP	5	0412	25	405	407					LDA		2N			8	0063	TEST WRG OP AND CARD NO			
	ERR	7	0407	30	205	405					LDL		1N	2N		8	0064				
	ERR	8	0605		OBE03	1MJ1	3			8		17AY4	D26J9			8	0065	TEST CON FOR CONSTANTS			
	ERR	9	0805		EM1D53J	10-	4			9		A62K5	4JD1T			8	0066	WITH UNDIGITS 1 3 5			
	ERR	10	1005	36	408						CAA					8	0067				
	ERR	11	0408	86	222						CAX	ERR2N				8	0068	TEST NEW ORDERS			
	ERR	12	0422	23	224			ERR3N			CTA					8	0069				
	ERR	13	0224	82	227	222					TEQ		ERR2N			8	0070	TEST LATENCY PATCH			
	ERR	14	0227	87	422	230					TGR	ERR3N				8	0071	FOR ACTION CODE FIVE			
	ERR	15	0230	42	234	433					HBT	ERR4N	ERR5N			8	0072				
	ERR	16	0434	22	238	422					RBT		ERR3N			8	0073				
	GERR	17						RA			LDA	TS22N	TS23N			8					
	ERR	18	0505	30	307	309					LDL		4N			8	0074	105 A			
	ERR	19	0309	70	511	K					ADD			RA		8	0075	109 A			
	ERR	20	0511	25	313	307					LDA		3N	4N		8	0076	113 A			
	ERR	21						RA			LDA		3N	4N		8					
	ERR	22	0507	60	509	711					STA		5N			8	0077	107 A			
	ERR	23	0711	70	513	K					ADD		8N	RA		8	0078	109 A			
	ERR	24						RA			STL		6N	7N		8		113 A			
	ERR	25	0520	50	313	307					STL		3N	4N		8	0079	117 A			
	ERR	26	CLOCK			0008		CLOCK			STL	TS22N	00015			8		15 WORDS TIMES AFTER			

CODE	ORIGINATES IN OP.	MEANS
0008	PDC	Operation number on detail card is incorrect. Hit start and machine will stop on 67 order. Go to <i>m</i> to process card. Go to <i>c</i> to get next card.
0009	PDC	Card number on detail card incorrect. Same action as 0008 STOP.
8888	MCK	Final successful stop. Follow normal operating instruction before hitting start if new assembly is wanted.
0010	MCK	Card being processed is not a type 7 or 10 card. <sup>9</sup> Depress run button. If card last read is to be processed as type 10 card, go to the <i>c</i> address of this order. If it is to be processed as a type 7 or 8 card, go to the <i>m</i> address. This will transfer control to another stop order. Now if the card to be processed is a type 7, go to the <i>c</i> address of this stop order. If it is to be processed as a type 8 card, to <i>m</i> address.

---

<sup>9</sup> This stop was inserted to remedy the error of not having a type 9 card followed by a type 7 or 10 card.

## Appendix A.

### Summary of Instruction Codes

A summary of instruction codes on the following pages lists X-6 mnemonic codes, their computer equivalents, and descriptions of each instruction.

A hyphen in an *m* or *c* portion of instruction indicates that the computer ignores that portion when the instruction is executed.

COMPUTER CODE	X-6 CODE	FUNCTION	MINIMUM WORD TIMES
<b>ARITHMETIC</b>			
70	ADD <i>m c</i>	Add ( <i>m</i> ) to ( <i>rA</i> ).	5
75	SUB <i>m c</i>	Subtract ( <i>m</i> ) from ( <i>rA</i> ).	5
85	MUL <i>m c</i>	Multiply ( <i>rL</i> ) by ( <i>m</i> ).	105
55	DIV <i>m c</i>	Divide ( <i>m</i> ) by ( <i>rL</i> ).	115
<b>TRANSFER</b>			
25	LDA <i>m c</i>	Load <i>rA</i> .	4
05	LDX <i>m c</i>	Load <i>rX</i> .	4
30	LDL <i>m c</i>	Load <i>rL</i> .	4
60	STA <i>m c</i>	Store <i>rA</i> .	4
65	STX <i>m c</i>	Store <i>rX</i> .	4
50	STL <i>m c</i>	Store <i>rL</i> .	4
77	ATL - <i>c</i>	( <i>rA</i> ) → <i>rL</i> .	3
23	CTA <i>m</i> -	( <i>rC</i> ) → <i>rA</i> .	3
26	CLA <i>m</i> -	Clear <i>rA</i> to zeros.	3
06	CLX <i>m</i> -	Clear <i>rX</i> to zeros.	3
31	CLL <i>m</i> -	Clear <i>rL</i> to zeros.	3
86	CAX <i>m</i> -	Clear <i>rA</i> and <i>rX</i> to zeros. (sign of <i>rL</i> goes to <i>rA</i> and <i>rX</i> ).	14
36	CAA <i>m</i> -	Clear <i>rA</i> to zeros (original sign remains).	3

**TRANSLATE**

12	CTM - c	Translate card-to-machine (computer) code.	3
17	MTC - c	Translate machine (computer)-to-card code.	3
3/4 3	TXM - c	Translate UNIVAC XS-3 code to machine (computer) code.	3
3/4 1	TMX - c	Translate machine (computer) code to UNIVAC XS-3 code.	3

**COMPARISON**

82	TEQ m c	Test (rA) and (rL) for equality. Next instruction is at m if (rA) = (rL).	3
87	TGR m c	Test (rA) and (rL) for magnitude. Next instruction is at m if (rA) > (rL).	3

**EDIT**

20	BUF m c	Buff (m) onto (rA).	4
35	ERS m c	Erase: (rA) controlled by (m).	4
32	SHR m c	Shift right (rA) and (rX) Here nn = number of places to shift (0 to 10).	3 + n
	$\overbrace{\Delta\Delta\Delta nn}$		
37	SHL m c	Shift (rA) left. Here nn = number of places to shift (0 to 10).	3 + n
	$\overbrace{\Delta\Delta\Delta nn}$		
62	ZUP - c	Zero-suppress in rA and rX.	4

**CONTROLS**

00	JMP m -	Jump to m.	2
67	STP m c	Stop. m is alternative next instruction (requires manual intervention).	Ind.

**INDEX REGISTERS**

02	L I R m c	Load index register.	3
07	I I R m c	Increment index register.	4

**HIGH-SPEED READER (INPUT)**

42	HBT m c	HSR buffer test. Next instruction is at m if buffer is loaded.	3 (4 if m)
----	---------	--	------------



96	<b>HBU</b> <i>m c</i> $\underbrace{\hspace{1.5cm}}$ <i>Hn00d</i>	HSR buffer unload. Here: <i>n</i> = HSR interlace (0 through 9). <i>d</i> = 0 for normal translation. <i>d</i> = 1 for translation “on the fly.”	203 <sup>10</sup>
72	<b>HCC</b> <i>m c</i>	HSR card cycle. Next instruction is at <i>m</i> if HSR is busy.	3 (4 if <i>m</i> )
47	<b>HSS</b> <i>m c</i> $\underbrace{\hspace{1.5cm}}$ $\Delta\Delta n00$	HSR stacker selection. Here: <i>n</i> = stacker 0, 1, or 2.	3

**READ-PUNCH UNIT (INPUT-OUTPUT)**

22	<b>RBT</b> <i>m c</i>	RPU buffer test. Next instruction is at <i>m</i> if the buffer is loaded.	3 (4 if <i>m</i> )
46	<b>RBU</b> <i>m c</i> $\underbrace{\hspace{1.5cm}}$ <i>Rn00d</i>	RPU buffer unload. Here: <i>n</i> = RPU input inter- lace (0 through 9). <i>d</i> = 0 for normal trans- lation <i>d</i> = 1 for translation “on the fly.”	203 <sup>10</sup>
81	<b>RCC</b> <i>m c</i> $\underbrace{\hspace{1.5cm}}$ <i>0n00d</i>	RPU card cycle. Here: <i>n</i> = RPU output inter- lace (0 through 9). <i>d</i> = 0 for normal translation. <i>d</i> = 1 for translation “on the fly.”	103 <sup>10</sup>
57	<b>RSS</b> - <i>c</i>	RPU select stacker 1.	3

**HIGH-SPEED PRINTER (OUTPUT)**

27	<b>PBT</b> <i>m c</i>	Printer test. Next instruc- tion is at <i>m</i> if Printer is free for use.	3 (4 if <i>m</i> )
16	<b>PFD</b> <i>m c</i> $\underbrace{\hspace{1.5cm}}$ $\Delta\Delta\Delta yy$	Printer advance. Here: <i>yy</i> = number of lines to advance (00 to 79).	4
11	<b>PRN</b> <i>m c</i> $\underbrace{\hspace{1.5cm}}$ <i>Pn0yy</i>	Advance and print. Here: <i>n</i> = Print interlace (0 through 9). <i>yy</i> = number of lines to advance (00 to 79).	592

<sup>10</sup> Word-time applicable only when *d* = 0. Word time is 207 when *d* = 1 for HBU; is 208 when *d* = 1 for RCC and RBU.

TAPE SYNCHRONIZER (INPUT-OUTPUT)

7/4 2	TRD <i>m c</i> $\Delta\Delta xyz$	Read 1 block from tape buffer band. Here: <i>x</i> = Servo number (0 through 9). <i>y</i> = Mode – 0 if USS. 5 if UNIVAC. <i>z</i> = Direction and gain- 0 forward normal. 1 forward low. 2 forward high. 5 backward normal. 6 backward low. 7 backward high.	17
8/4 2	TWR <i>m c</i> $\Delta\Delta xy0$	Write 1 block from the tape buffer band onto the tape. Here: <i>x</i> = Servo number (0 through 9). <i>y</i> = Mode and density– 0 = USS       250 cpi. 5 = UNIVAC 250 cpi. 6 = UNIVAC 125 cpi.	17
3/4 7	TBT <i>m c</i>	Test tape buffer. If it is loaded, the next instruction is at <i>m</i> .	3 (5 if <i>m</i> )
3/4 2	TST <i>m c</i>	Test for servo availability, If available, the next instruction is at <i>m</i> .	3 (4 if <i>m</i> )
6/4 6	TBU <i>m c</i> $Tn000$ $Zn000$	Tape buffer unload. Here: <i>n</i> = the tape interlace (0 through 9).	205
3/4 6	TBL <i>m c</i> $Tn000$ $Zn000$	Tape buffer load. Here: <i>n</i> = the tape interlace (0 through 9).	205
6/4 2	TRW <i>m c</i> $\Delta\Delta xy0$	Rewind tape to first block condition. Here: <i>x</i> = Servo number. <i>y</i> = Type of rewind– 0 without interlock. 2 with interlock.	600 ms

## Appendix B. Summary of Card Types

### CARD TYPE 1 - LABEL

CARD TYPE											PROGRAM I.D.					DATE																																		
1											x	x	x	x	x	Δ	Δ	Δ	Δ	Δ	m	m	Δ	d	d	Δ	y	y	Δ	Δ	12																			
																															34																			
																															56																			
																															78																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	90				
Printed in U.S.A. REMINGTON RAND	COMMENTS - ANY DESCRIPTIVE ENGLISH																																												12					
																																																		34
																																																		56
																																																		78
																																																		90

Here,        **xxxxx**    is the five-digit program identification.  
               **mm**        is month.<sup>11</sup>  
               **dd**        is day.  
               **yy**        is year.

<sup>11</sup> Month, day, and year can be in any format.

CARD TYPE 2 - RESTRICTS

CARD TYPE											ENTRY 1										ENTRY 2										ENTRY 3														
											i n n n n n s s s s										i n n n n n s s s s										i n n n n n s s s s										12				
2																																									34				
																																									56				
																																									78				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
Printed in U.S.A. REMINGTON RAND	ENTRY 4										ENTRY 5										ENTRY 6										ENTRY 7														
	i n n n n n s s s s										i n n n n n s s s s										i n n n n n s s s s										i n n n n n s s s s										12				
																																									34				
																																									56				
																																									78				
																																									9				
	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90

Here,            i i    is the increment.  
                  n n n n    is the total number of restricted addresses.  
                  s s s s    is the absolute starting address.

CARD TYPE 3 - TAG EQUALS

CARD TYPE											ENTRY 1										ENTRY 2										ENTRY 3														
											t t t t t Δ n n n n n t										t t t t t Δ n n n n n t										t t t t t Δ n n n n n										12				
3																																									34				
																																									56				
																																									78				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
Printed in U.S.A. REMINGTON RAND	ENTRY 4										ENTRY 5										ENTRY 6										ENTRY 7														
	t t t t t Δ n n n n n t										t t t t t Δ n n n n n t										t t t t t Δ n n n n n t										t t t t t Δ n n n n n										12				
																																									34				
																																									56				
																																									78				
																																									9				
	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90

Here,            t t t t t    is the permanent tag or K or W address.  
                  n n n n n    is the absolute address.

CARD TYPE 4 - INTERLACES

CARD TYPE	ENTRY 1										ENTRY 2										ENTRY 3																							
	t n Δ Δ Δ x b b 0 0										t n Δ Δ Δ x b b 0 0										t n Δ Δ Δ x b b 0 0										12													
4																															34													
																															56													
																															78													
																															90													
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
Printed in U.S.A. REMINGTON RAND	ENTRY 4										ENTRY 5										ENTRY 6										ENTRY 7													
	t n Δ Δ Δ x b b 0 0										t n Δ Δ Δ x b b 0 0										t n Δ Δ Δ x b b 0 0										t n Δ Δ Δ x b b 0 0										12			
																																									34			
																																									56			
																																									78			
																																									90			
46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90

Here, t is the type of interlace; R, P, O, H, T, or Z.  
n is interlace number (0-9).  
x is 0 for a two part untranslated interlace (unprimed, primed).<sup>13</sup>  
1 for a two part translated interlace (zone, numeric).  
2 if both kinds are specified.  
bb00 is the absolute address of the band. (bb must be an even number).

CARD TYPE 5 - TABLES

CARD TYPE	ENTRY 1															ENTRY 2																												
	t n Δ Δ Δ Δ s s s s i i i Δ Δ Δ x x x x															t n Δ Δ Δ Δ s s s s															12													
5																															34													
																															56													
																															78													
																															90													
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
Printed in U.S.A. REMINGTON RAND	ENTRY 2 (cont.)															ENTRY 3																												
	i i i Δ Δ Δ x x x x															t n Δ Δ Δ Δ s s s s i i i Δ Δ Δ x x x x															12													
																															34													
																															56													
																															78													
																															90													
46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90

Here, t is the type of table; S, U, or V.  
n is table number (0-9).  
s s s s is the absolute starting address.  
i i i is the increment.  
x x x is the total number of entries in the table.

<sup>13</sup> Notations here are not applicable to print or tape interlaces as x will always equal 0.

**CARD TYPE 6 - SPECIFICATIONS**

CARD TYPE	OPER. NO.	CARD NO.	ENTRY 1																		ENTRY 2																		ENTRY 3																		12 34 56 78																
			6	h	h	h	y	y	y	X	Δ	Δ	n	n	e	e	e	e	e	e	e	X	Δ	Δ	n	n	e	e	e	e	e	e	X	Δ	Δ	n	n	e	e	e	e																																
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45																													
Printed in U.S.A. REMINGTON RAND	ENTRY 4																		ENTRY 5																		ENTRY 6																		ENTRY 7																		12 34 56 78 9
	X	Δ	Δ	n	n	e	e	e	e	e	e	e	e	e	e	e	e	e	e	X	Δ	Δ	n	n	e	e	e	e	e	e	e	e	e	e	e	e	e	e	X	Δ	Δ	n	n	e	e	e	e																										
46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90																													

Here,            h h h    is the operation number.  
                   y y y    is the card number.  
                   X Δ Δ n n    is the nth variable, as X Δ Δ 01.  
                   e e e e e    is the X-6 equivalent address.

**CARD TYPE 7 - HEADER**

CARD TYPE	OPER. NO.	CARD NO.																																													12 34 56 78
			7	h	h	h	y	y	y																																						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45			
Printed in U.S.A. REMINGTON RAND	COMMENTS - ANY DESCRIPTIVE ENGLISH																																												12 34 56 78 9		
	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89		90	

Here,            h h h    is the operation number.  
                   y y y    is the card number.



CARD TYPE 9 - END-OPERATION SENTINEL

Printed in U.S.A. - REMINGTON RAND	CARD TYPE	OPER. NO.	CARD NO.																																										12	
	9	h h h	y y y																																										34	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	56
	COMMENTS - ANY DESCRIPTIVE ENGLISH																																											78		
																																												9		
	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	90

Here,            h h h    is the operation number.  
                   y y y    is the card number.

CARD TYPE 10 - END INPUT

Printed in U.S.A. - REMINGTON RAND	CARD TYPE																INST. CODE	<i>m</i>					<i>C</i>									12														
	10																I I I	m m m m m					c c c c c									34														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	56
	COMMENTS - ANY DESCRIPTIVE ENGLISH																																											78		
																																												9		
	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	90

Here, fields III, mmmmm, and ccccc indicate the first instruction to be executed after the program is fully loaded.



## Appendix C.

### X-6 Storage Layout

A memory dump at the end of a successful assembly is desirable for debugging and patching of the object program.

LOCATION	NAME	USE
0800	Table S8	Valid mnemonic codes stored 20 words apart.
0816	Table S9	Information words for each mnemonic code stored 20 words apart.
2100-2109	Table S5	Interlace origins (from card type 4).
2110-2119	Table V3	Two part translated or untranslated interlace word positions for O.
2120-2132	Table V4	Two part interlace word positions for P.
2200-2399	O2 Interlace	Repunching of output cards which fail read check.
2450-2465	Table V2	Translated and untranslated word positions for the H and R interlaces.
2470-2479	Table S6	Interlace origins (from card type 4).
2480-2509	Table S7	Table origins and increments (from card type 5).
2520-2539	Table V1	X-6 equivalents for last set of specifications.
2540-2559	Table V0	Specifications. Cleared after every operation. No value after complete assembly.
2800-3099	Table S4	K and W addresses and absolute addresses stored as follows: 2800 - K0 and W0 as OKKKK0WWWW 2801 - K1 and W1 as OKKKK0WWWW
3100-3249	Table S2	Addresses of permanent tags in same order as table S1. Stores as 0aaaa0aaaa. Left half-words used for first 150 tag addresses, then right half-words are filled.

LOCATION	NAME	USE
3250-3299	Table S3	Temporary tags with absolute addresses. Cleared after every operation. No value after complete assembly.
3300-3599	Table S1	Permanent tags. The five-character alphanumeric tag is stored as zzzzznnnnn. One tag per word.
3600-3799	Table S0	Storage availability. Each word of table represents a band-relative address, 000-199. The twenty bits in the left half-word are zero (0) for unused or one (1) for used, representing the twenty standard access bands. The twenty bits in the right half of words 3600-3649 represent high-speed access storage. Addresses 4000, 4050, 4100, and 4150 are included in first digit of right half-words. Right half of words 3650-3799 are unused.
3800-3999	P0 Interlace	Header for X-6 listing.
4000-4199	H0 Interlace	High-Speed Reader read-in area.
4200-4399	O1 Interlace	Output punching area.
4200-4399	R0 Interlace	Read-Punch Unit read-in area.
4400-4599	P1 Interlace	Detail lines for X-6 listing.
0000-0199	Restricted	Used to load X-6 and later filled with memory dump routine.





# UNIVAC® Solid-State Computer

PAGE \_\_\_\_ OF PROGRAM NO. \_\_\_\_\_

## X-6 ASSEMBLY SYSTEM SUMMARY CARD LAYOUT SHEET

APPLICATION \_\_\_\_\_

CARD TYPE									
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

CARD TYPE									
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

CARD TYPE									
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

CARD TYPE									
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

CARD TYPE									
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

CARD TYPE									
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

CARD TYPE									
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

CARD TYPE									
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

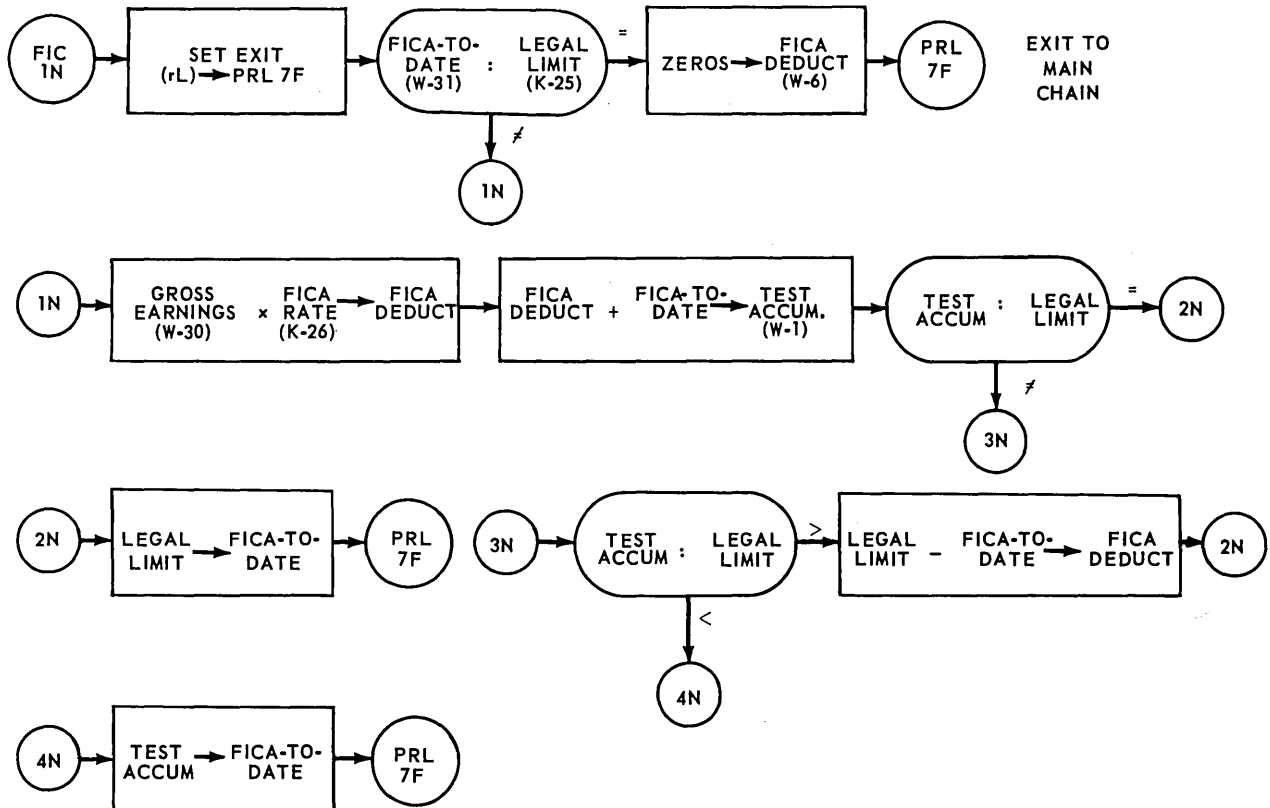
U-2290.1 PRINTED IN U.S.A.

## Appendix F. Sample Problem

The following is a FICA routine that may be employed in a larger payroll application. The routine is coded to achieve the following results:

1. Compute FICA deduction.
2. Compute amount of FICA paid to date.
3. Test to see if the legal FICA limit has been reached.

### FLOW CHART



PAYRL	OP	CD	LOCA	OP	MMMM	CCCC	K	A TAG	C	OP	M TAG	C TAG	TP	CD	COMMENTS	043982	PAGE 6
FIC	1												7		FICA ROUTINE		
FIC	2	0600	50	4202	0604			FIC1N		STL	PRL7F	FIC2N	8	0001	SET EXIT		
FIC	3	0604	30	4206	0608			FIC2N		LDL	K 25		8	0002	LEGAL LIMIT TO RL		
FIC	4	0608	25	4010	0612					LDA	W 31		8	0003	FICA-TO-DATE TO RA		
FIC	5	0612	82	0615	0815					TEQ		1N	8	0004	TEST EQUALITY.FICA-TO-DATE: LEGAL LIMIT		
FIC	6	0615	06	0618						CLX			8	0005			
FIC	7	0618	65	4020	4202					STX	W 0	PRL7F	8	0006	ZEROS TO FICA DEDUCTION		
FIC	8	0815	30	4017	0819			1N		LDL	W 30		8	0007	GROSS EARNINGS TO RL		
FIC	9	0819	85	4021	0924					MUL	K 26		8	0008	MULT. FICA RATE		
FIC	10	0924	65	4020	0972					STX	W 0		8	0009	DEDUCTION TO W 0		
FIC	11	0972	25	4010	0814					LDA	W 31		8	0010	FICA-TO-DATE TO RA		
FIC	12	0814	70	000C	0619					ADD	RX		8	0011	ADD DEDUCTION		
FIC	13	0619	60	4221	0623					STA	W 1		8	0012	TEST ACCUMULATION TO W 1		
FIC	14	0623	30	4206	0658					LDL	K 25		8	0013	LEGAL LIMIT TO RL		
FIC	15	0658	82	0660	0860					TEQ	2N	3N	8	0014	TEST EQUALITY.TEST ACCUM: LEGAL LIMIT		
FIC	16	0660	50	4010	4202			2N		STL	W 31	PRL7F	8	0015	LEGAL LIMIT TO FICA-TO-DATE		
FIC	17	0860	87	0862	0662			3N		TGR		4N	8	0016	TEST MAGNITUDE, TEST ACCUM: LEGAL LIMIT		
FIC	18	0862	25	000B	0866					LDA	RL		8	0017	LEGAL LIMIT TO RA		
FIC	19	0866	75	4010	0913					SUB	W 31		8	0018	SUBTRACT FICA-TO-DATE		
FIC	20	0913	60	4020	0660					STA	W 0	2N	8	0019	RESULT TO FICA DEDUCT		
FIC	21	0662	25	4221	0673			4N		LDA	W 1		8	0020	TEST ACCUM. TO RA		
FIC	22	0673	60	4010	4202					STA	W 31	PRL7F	8	0021	TEST ACCUM. TO FICA-TO-DATE, EXIT		
FIC	23												9				

***Remington Rand Univac***  
DIVISION OF SPERRY RAND CORPORATION