

CINEMATRONICS
APPLICATIONS
PROGRAMMING
MANUAL

TABLE OF CONTENTS

- I. VECTOR PROCESSOR GENERAL INFORMATION

- II. PROCESSOR INSTRUCTION REPERTOIRE
 - A. Overview
 - B. Format Types
 - C. Instruction Categories
 - 1. Load/Storage Operations
 - 2. Add/Multiply Operations
 - 3. Subtract Operations
 - 4. Indirect Addressing Operations
 - 5. Logical Operations
 - 6. Load Memory Address Operations
 - 7. Branch Operations
 - 8. Shift Operations
 - 9. Table Look-Up Operations
 - 10. Input-Output Operations
 - 11. Vector Operations
 - 12. Control Operations
 - D. Macroses

- III. PROGRAMMING EXAMPLES
 - A. Physical Layout Of Working Storage
 - B. Examples Of Instruction Usage
 - C. Vector Line Drawing Techniques
 - D. Sample Program To Draw A Line
 - E. Program Copyright
 - F. Star Drawing Programming Example
 - G. 'Space Wars' - A Programming Implementation
 - H. Digit-Drawing Subroutine
 - I. Primal Primer

- IV. USING THE DEVELOPMENT SYSTEM

I. VECTOR PROCESSOR GENERAL INFORMATION

The CINEMATRONICS video game system consists of two (2) interdependent sections:

- 1) Computational Section
- 2) Display Section

(Block Diagram 1)

PROGRAM MEMORY

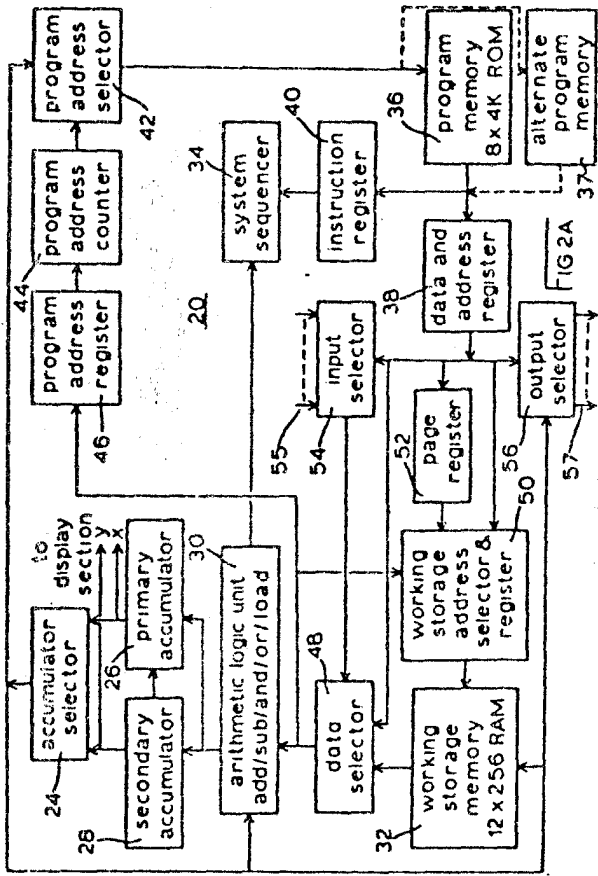
A 4096 word by 8-bit program memory containing all of the instructions and data necessary for the specification and operation of a particular game, functions as a read-only memory (ROM). The current configuration uses two (2) ROM's facilitating the development of a single application to 8912 words or allowing two games (1 per ROM) to be included in a system: the games can be changed completely by simply switching between different program memories.

WORKING STORAGE MEMORY

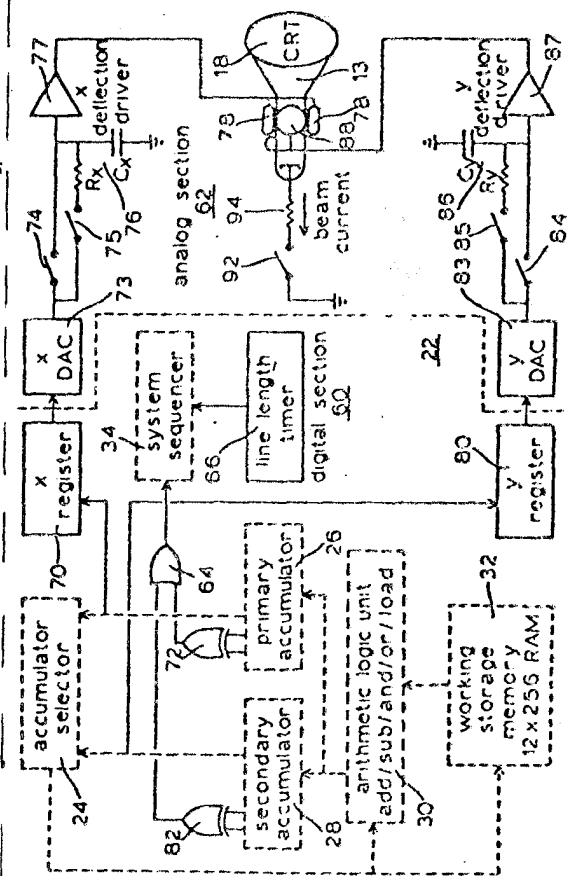
A 256 word by 12-bit RAM functioning as a scratch pad memory, used to temporarily store intermediate and final computational values necessary to the operation of the video game system.

Utilizing a working storage memory separate from the program memory speeds up operation since both memories can be accessed simultaneously.

The display section consists of both digital and analog subsections. This comprises the vector generator. The main function of this unit is to receive digital coordinate values of the initial point of a line segment to be drawn from the

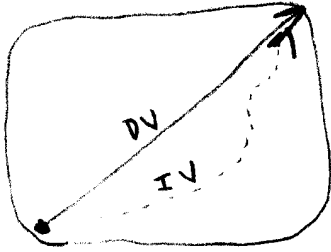


COMPUTATIONAL SECTION



DISPLAY SECTION

BLOCK DIAGRAM 1, VECTOR PROCESSOR



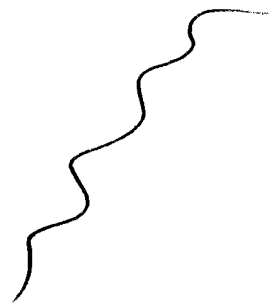
max. len IV takes \approx 100 μ s

amplifier is at max - beam gets non-linearly - but ok since it's off

max. len DV takes \approx 200 μ s

since amp. is at $\frac{1}{2}$ max. at most, so current in coils follows amplifier voltage more or less exactly.

\approx 1 mil
 \approx 2 mil



→ START FRAME (FRM FRM)

must be $<$ 3 mil

CST (inhibit automatic cold start/reset)

compute/draw

- Jmp

frame is 26.3 mil : 38 Frames/sec

if noise causes Jmp to a place which starts a tight loop, the program will freeze, screen goes blank. To prevent this, the machine will automatically do a Jmp to 0 (a CST) if no FRM is done within 26.3 mil of last FRM, unless a CST inst. is done within 3 mil of last FRM.

computational section and convert these values into voltages to thereof fix the initial position of the electron beam on the CRT display.

The system also includes a multiple operator control panel of output ports to enable further interaction with the operator. Sound effects; lights; coin boxes, ect. can be controlled from these ports.

MISCELLANEOUS INFORMATION

12 bits: address to 4095..

Clock time: 200 nanoseconds (5MHz).

Instruction cycle: 600 nanoseconds (average).

Number Of Instructions: 43

$$\text{clock cycle} = 200 \text{ ns}$$
$$\text{accum. inst (shift, Add/sub, imm.)} = \underline{300 \text{ ns}}$$

(no RAM wait)

RAM memory is even/odd interleaved, making
RAM Fetch time = 300 ns. So JMP commands
take longest, since interleaved lookahead
waited, & RAM must settle again.

$$\text{JMP inst} = \underline{1 \text{ ms}}$$

$$\text{Lookup} = \underline{1 \text{ ms}}$$

$$\text{Read/write RAM inst.} = \underline{600 \text{ n.s.}}$$

$$\text{w/ previous} = \underline{400 \text{ ns (save 1 cycle)}}$$

$$\text{NOP 400}$$

$$\text{LPIE 210}$$

The Central Processing Unit, contains circuitry to strobe and interpret all input functions including the player control panel switches and all coin and credit information and to create all the digital signals used in providing the visual display. It also contains all the software (i.e., machine language and game personality memory) needed to control the game operation and to generate the proper vectors needed to display.

* "Vectorbeam tm" is CINEMATRONICS service mark for video game educational services.

In fact, the CPU logic board contains a great portion of the vector generating system, which also includes the display unit. The CPU logic board also controls the switching (electrically) of the audio printed circuit board.

The Audio Board, as in many other video games, is comprised of a noise generator and the associated wave shaping circuits as well as a number of amplifiers. The various audio tones are simply switches to the output amplifier stages on command from the CPU logic board.

The Vectorbeam tm * Display Electronics is the final form of interpretation of the CPU's calculations. The CPU logic informs the display electronics unit of information regarding line length and line placement on the CRT. This is accomplished with two twelve-bit words, one each for horizontal and vertical deflection, and a number of other controlling signals for the cathode drive circuit and switching in the deflection circuits.

The major difference between the vector generator and raster scan type monitors is the means by which the cathode beam is directed (deflection) across the screen.

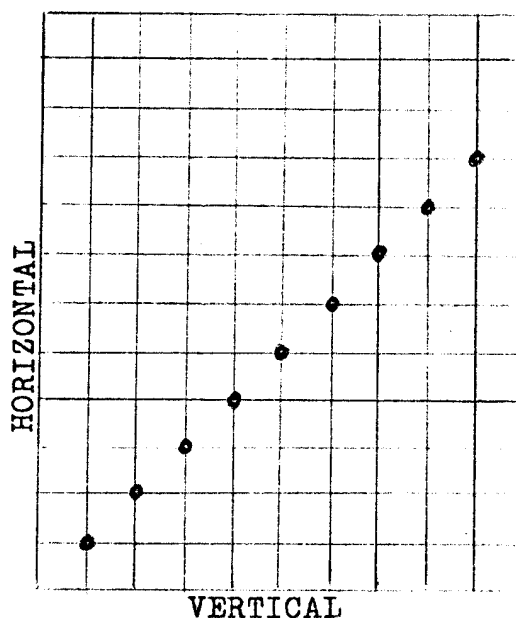
to accomodate two twelve-bit words of information, twelve each for vertical and horizontal deflection, and the fact that there is no background illumination from a constantly scanning beam when brightness is turned up. The higher degree of resolution combines with the totally blackened background creating an appearance of depth not found in a raster scan system.

Another major design difference is the fact that no sync. singals are needed to produce vectors on the CRT. This greatly simplifies the hardware design of the system, and therefore the understanding of the theory of operation, of the CPU logic as well as the display electronics.

VECTOR THEORY

In order to understand the basic concept behind a vector generated display, it is important to have a basic knowledge of vector theory.

The raster scan display uses a matrix display system. A graphical representation of a matrix is shown below.



For example, to produce a line on the CRT with a matrix-type pattern, the appropriate intersection points of horizontal and vertical lines are illuminated. The calculations which select these points are made on the logic board, and converted into video information for the monitor to digest. Although there are spaces between the illuminated points, the illusion of a solid line is made by your eyes, and the resolution is determined by the number of available horizontal and vertical lines in the system, and the speed of the sweep.

In the vector display system, there are no horizontal and vertical lines (no sweep) or sync. A line generated using a vector system is shown in Figure 2.

A line is drawn by programming a beginning and ending point of the line to be drawn, and forcing the cathode beam to travel between these two points, illuminating the entire path of phosphorous on the CRT. The angle of the line, the position of the line, and the length of the line are determined simultaneously, and simply, by selecting the proper voltage levels for the beginning and ending points of the line. This is accomplished by the two twelve-bit words applied to the d/a converters on the display board. The d/a will produce a different voltage level at its output for each possible combination of input levels (of which there are 4096 possibilities for each 12-bit word).

The end result of using the vector generator is an immensely increased number of programable point, which is in direct proportion the the word size and the capabilities of the DAC-80 (i.e., greater resolution, definition and smoother motion using minimum of hardware).

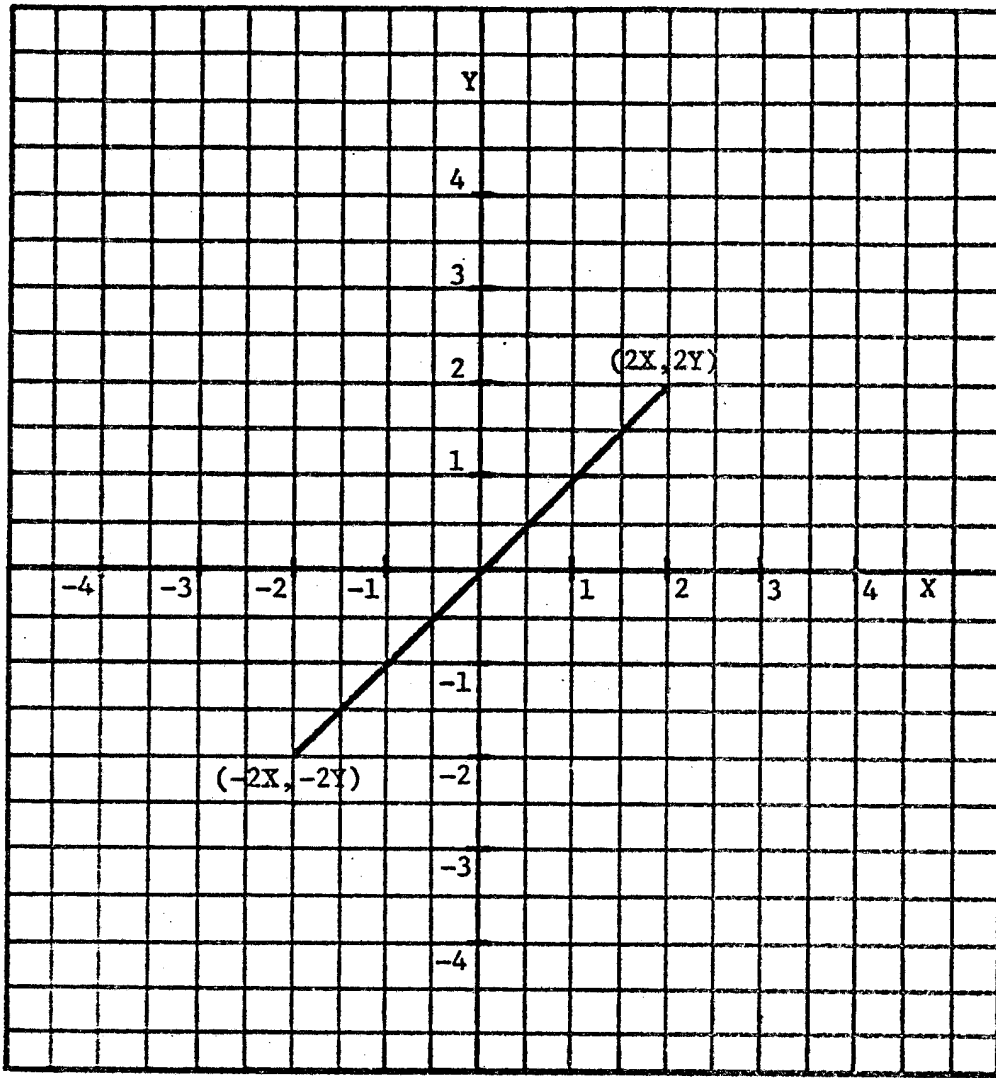


Figure 2. Line using Vector System

II. VECTOR PROCESSOR INSTRUCTION REPERTOIRE

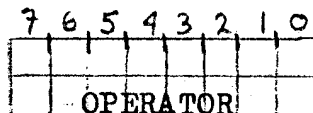
A. OVERVIEW

The instructions for use with the video game system are arranged in four (4) formats. The formats in which an instruction is stored in program memory is determined by the length of both the operator and operand. Being limited to the use of a maximum of eight (8) bits per word, the double-word instruction formats illustrated below are necessary for instructions wherein the combined length of the operator and operand exceeds 8-bits.

B. FORMAT TYPES

1. FORMAT 1

Single-Word/8-Bit Operator/ No Operand



When the instruction is decoded, (bits 0-3) and classified as a FORMAT 1 instruction, the contents of the data and address registers are ignored and the contents of the 8-bit operator instruction register are executed.

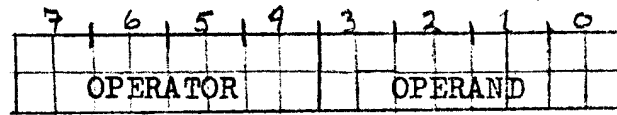
Instructions falling in this category are:

SSA Select Secondary Accumulator
LDAP Load Previously Selected
STAP Store Previously Selected
ADDP Add Previously Selected
SUBP Subtract Previously Selected
WSP Indirect Address Previously Selected

LPAP	Load Program Address Previous
JMP	Jump (unconditional)
JMI	Jump On Minus
JVN	Jump On Vector Not Finished
JCZ	Jump On Carry Zero
JLT	Jump On Less Than
JEQ	Jump On Equal
JOS	Jump On One's Shifted
T4K	Toggle 4-K
SHR	Shift Right
SHRB	Shift Right Both
ASR	Arithmetic Right Shift
SHL	Shift Left
SHLB	Shift Left Both
MUL	Multiply
LKP	Look Up
ANDP	And Previously Selected
IV	Initialize Vector
NV	Normalize Vector
DV	Draw Vector
FRM	Frame
CST	Cold Start <i>Inhibit</i>
NOP	No Operation

2. FORMAT 2

Single-Word/4-Bit/Operator/4-Bit Operand



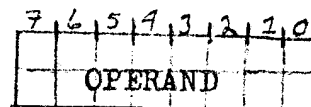
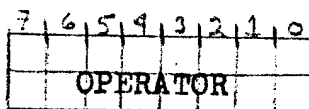
Classification as FORMAT 2 instruction causes the contents of the upper 4-bits of data and address register to be used as an operand (data or address) and the operation specified by the 4-bit operator to be executed.

Instructions falling in this category are:

LDA Load (accumulator)
LDI Load Immediate
STA Store
SUB Subtract
S4I Subtract 4 Immediate
WS Indirect Working Storage
SETP Load Page Register
INP Input
ADD Add
TST Test
OUT Output

3. FORMAT 3

Double-Word/8-Bit Operator/8-Bit Operand



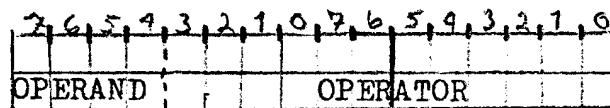
Causes the next word in program memory to be read and the contents loaded into the data and address registers. After the data and address register is loaded with the second word of the instruction, the 8-bit content of the data and address register is specified for use as an operand and the operation specified by the 8-bit operator in the instruction register is executed.

Instructions falling in this category are:

- ABI Add 8 Immediate
- SSI Subtract 8 Immediate

4. FORMAT 4

Double-Word/4-Bit Operator/12-Bit Operand

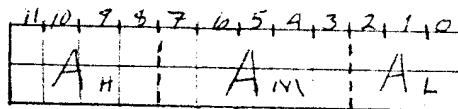


Causes contents of upper 4-bits (bits 4-7) of data and address register to be used as an operand (data or address) and the operation specified by the lower 4-bits (B0-B3) to be partially executed. Then the next word in memory is loaded into the data and address register. The second word of the instruction contained in the data and address register is specified for use as an operand and the remainder of the operation specified by the 4-bit operator in the lower 4-bits of the instruction is executed.

Instructions falling in this category are:

- LPAI Load Program Address Immediate

C. INSTRUCTION CATEGORIES AND DEFINITIONS



The description of the operand for some FORMAT 2 instructions (and the single FORMAT 4 instruction) uses the nomenclature shown above. This refers to the 4-bit nibble being referenced in working storage RAM.

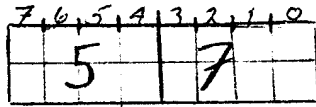
CONDITION CODES

Arithmetic operations will cause certain flags to be set depending on the result of the operation.

LT	ACC	Specified Value (JLT instruction)
EQ	ACC =	Specified Value (JEQ instruction)
CY	Carry Flag	Set (JCZ instruction)
NG	MSB Set in ACC	after Arithmetic Operation (JMI instruction)
RS	Right Shift Flag:	Set after one shifted out of position 0 (JOS instruction)
VT	Vector in Process	- after DV (JVN instruction)
A _H	Upper 4-Bits of 12-Bit Word	(Bits 11-8)
A _M	Middle 4-Bits of 12-Bit Word	(Bits 7-4)
A _L	Lower 4-Bits of 12-Bit Word	(Bits 3-0)

1. LOAD/STORE OPERATIONS

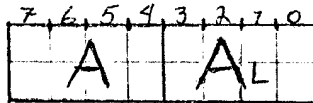
SSA SELECT SECONDARY ACCUMULATOR



Description: Causes the secondary accumulator to be selected during the execution of the next instruction. The absence of the SSA instruction preceding an instruction causes the primary AC to be selected.

Condition Codes: Unaffected

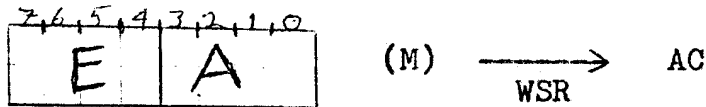
LDA LOAD (ACCUMULATOR)



Description: Loads the word selected by the contents of the page register and the 4-bit address carried with the instruction (A_L) from working storage memory into the selected accumulator.

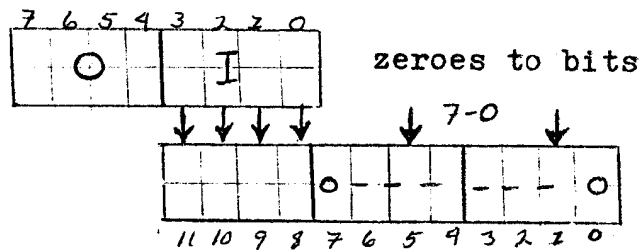
Condition Codes: NG: Set if result negative (MSB set)

LDAP LOAD PREVIOUSLY SELECTED



Description: Loads the word previously selected by the contents of the working storage address register from working storage memory into the selected accumulator.

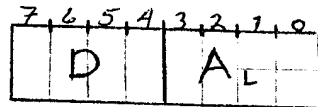
LDAI LOAD IMMEDIATE



Definition: Loads the 4-bits of data carried along with the instructions into the upper 4-bits of the selected accumulator while loading zeroes into the lower 8-bits.

Condition Codes: NG: Set if result negative (sets MSB)

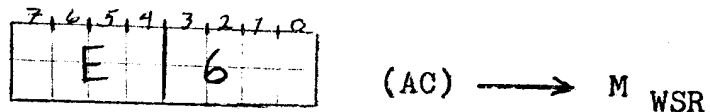
STA STORE



Description: Stores the contents of the selected accumulator into the working storage memory specified with the address carried with the instruction.

Condition Codes: NG: Retains previous state of ACC.

STAP STORE PREVIOUSLY SELECTED

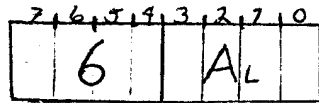


Description: Stores the contents of the selected accumulator into the word previously selected by the contents of the working storage address register into working storage memory.

Condition Codes: Unaffected

2. ADD/MULTIPLY OPERATIONS

ADD ADDITION



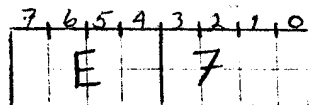
(M) WSPAR, AL + (AC) → AC

Description: Adds the word selected by the contents of the page register and the 4-bit address carried with the instruction from working storage memory to the selected accumulator.

Condition Codes: NG: Set if result negative (MSB)

CY:

ADDP ADD PREVIOUSLY SELECTED

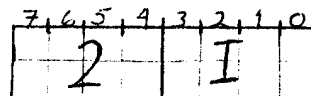


Description: Adds the previously selected contents of working storage to the selected accumulator.

Condition Codes: NG: Set if result negative (MSB)

CY:

A4I ADD 4-BIT DATA IMMEDIATE



$I \neq 0$ (AC) + I → AC

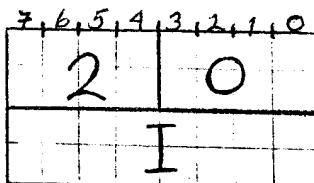
Description: Adds the 4-bits of data carried with the instruction to the selected accumulator.

Condition Codes: NG: Set if result negative (MSB)

CY:

****NOTE**** DATA MUST NOT BE ZERO - A ZERO IN THE UPPER 4-BITS OF THE FIRST WORD IS USED TO SIGNIFY THAT AN OPERAND IS CONTAINED IN THE FOLLOWING WORD.

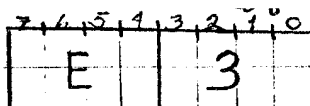
A8I ADD 8-BIT DATA IMMEDIATE



Description: Adds the second byte of the instructions to the selected accumulator.

Condition Code: NG: Set if result negative (MSB)
 CY:

MUL MULTIPLY

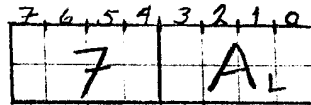


Description: Causes both the primary and secondary accumulator to be shifted right one place simultaneously and the contents of the selected word from WS memory to be added to the secondary AC if a one was shifted out of the primary AC.

Condition Codes: NG: Set if product is negative; cleared otherwise.
 RS: Set if one shifted out of bit (primary ACC).
 CY:

3. SUBTRACT OPERATIONS

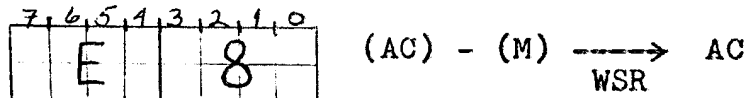
SUB SUBTRACT



Description: Subtracts the word selected by the contents of the page register and the 4-bit address carried with the instruction in working storage memory from the selected accumulator.

Condition Codes: NG: Set if result negative (MSB)
 CY:

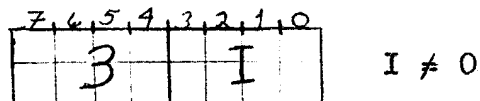
SUBP SUBTRACT PREVIOUSLY SELECTED



Description: Subtracts the word previously selected by the contents of the working storage address register in working storage memory from the contents of the selected accumulator.

Condition Codes: NG: Set if result negative (MSB)
 CY:

S4I SUBTRACT 4 IMMEDIATE

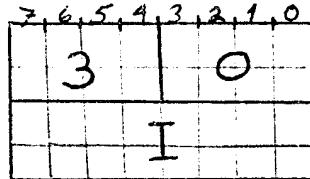


Description: Subtracts the 4-bits of data carried along with the instruction from the contents of the selected accumulator.

****NOTE**** DATA MUST NOT BE ZERO ADDRESS SELECTOR AND REGISTER.
ANY OF THE 'PREVIOUSLY SELECTED' INSTRUCTIONS DO THIS.

Condition Codes: NG: Set if result negative (MSB set)

S8I SUBTRACT 8 IMMEDIATE



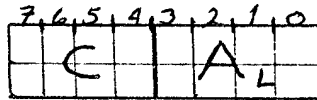
Description: Subtracts the 8-bits of data carried along with the instruction from the contents of the selected accumulator.

Condition Codes: NG: Set if result negative (MSB)

CY:

4. INDIRECT ADDRESSING OPERATIONS

WS INDIRECT ADDRESS WORKING STORAGE



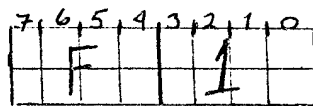
Description: Load the word specified by the contents of the page register and the 4-bit address carried with the instruction from the working storage memory to the working storage address register.

Condition Codes: NG: Reflects previous state of ACC

RS: Reflects previous state of ACC

****NOTE**** AFTER A WS INSTRUCTION IS EXECUTED, IT IS NECESSARY TO EXECUTE A MEMORY ACCESS INSTRUCTION WITHOUT MODIFYING THE CONTENTS OF WORKING STORAGE ADDRESS SELECTOR AND REGISTER. ANY OF THE 'PREVIOUSLY SELECTED' INSTRUCTIONS DO THIS.

WSP INDIRECT ADDRESS PREVIOUSLY SELECTED

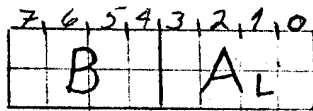


Description: Loads the word previously selected by the contents of the working storage address register from working storage address register.

Condition Codes: Unaffected

5. LOGICAL OPERATIONS

TST TEST



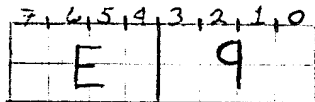
(AC) - (M)

$A_m, A_L \longrightarrow$ sets flag

Description: Subtracts the word selected by the contents of the page register and the 4-bit address carried with the instruction in working storage memory from the contents of the selected accumulator without modifying the contents of the accumulator.

Condition Codes: Flags are set: EQ: Set if value (WS) = ACC
 J EQ, JLT LT: Set if value (WS) < ACC
MAGNITUDE COMPARE NG: Reflects previous state of ACC.

ANDP AND PREVIOUSLY SELECTED



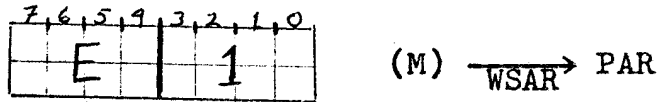
$(AC) \cdot (M)_{WSR} \longrightarrow AC$

Description: AND's the word previously selected by the contents of the working storage address register in working storage memory with the contents of the selected accumulator.

Condition Codes: NG: Set if result negative (MSB set)
 CY:

6. LOAD MEMORY ADDRESS OPERATIONS

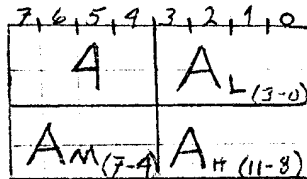
LPAP LOAD PROGRAM ADDRESS PREVIOUS



Description: Loads the word previously selected by the contents of the working storage address register from working storage memory into the program address register.

Condition Codes: Unaffected

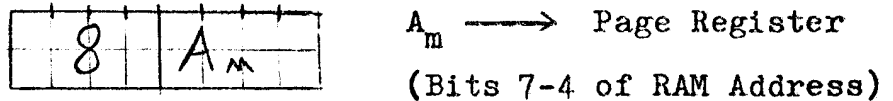
LP AI LOAD PROGRAM ADDRESS IMMEDIATE



Description: Loads the 12-bit address carried with the instruction into the program address register.

Condition Codes: Unaffected

SETP LOAD PAGE



Description: Loads the 4-bit address carried with the instruction into the page register.

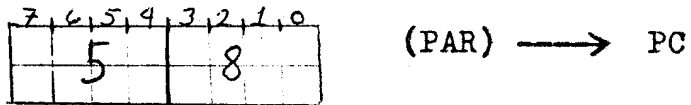
Condition Codes: Unaffected

SETP 1 Four high-order bits of address remain unchanged until next SETP instruction is executed.

SETP 0 Storing half the bits necessary to the selection of a word from working storage memory in a separately loaded page register reduces the storage requirements for memory access instructions from two (2) words to 1 (one) word. If the entire 8-bit address were carried along with the specification of the operation, a double-word instruction would be necessary. Thus: SETP 1 Loads 4 high-order bits, LDA XYZ loads 4 low-order bits, (Up to 16 words can be addressed in a working storage page: 4 bits).

7. BRANCH OPERATIONS

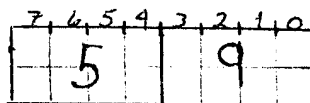
JMP JUMP (UNCONDITIONAL)



Description: Causes the contents of the program address register to be loaded into the program address counter.

Condition Codes: Unaffected

JMI JUMP ON MINUS



MSB is det (i.e. 5H)

Description: Causes a branch to be executed if ~~the specified value is greater than the value contained in the selected accumulator. Otherwise the program counter is incremented by one.~~

Condition Codes: Unaffected

****NOTE**** DUE TO A TIMING CONSTAINT, THE JMI INSTRUCTION MUST BE PRECEDED BY A 'NOP'.

NOP

JMI

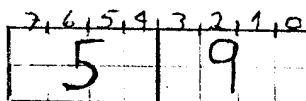
The following instruction has the same operation code as the preceding JMI instruction. Currently a jumper determines the hardware configuration;

if jumper in: JMI

if jumper out: JEH

↙

JEH JUMP ON EXTERNAL HIGH

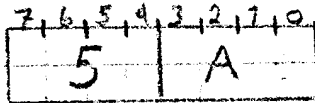


Description: Causes the program address register to be loaded into the program address counter if the level on an external input line is high. The program counter is incremented by a single address if the condition is not met.

Condition Codes: Unaffected

JVN

JUMP ON VECTOR NOT FINISHED

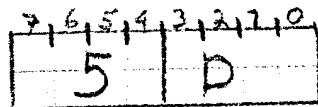


Description: Causes a branch to be executed if a line segment is in the process of being drawn. Otherwise the program counter is incremented by one.

Condition Codes: Unaffected

JCZ

JUMP ON CARRY EQUAL ZERO

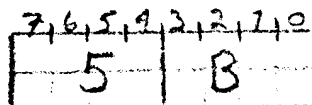


Description: Causes a branch if the result of the preceding arithmetic operation did not generate a carry. Otherwise the program counter is incremented by one.

Condition Codes: Unaffected

JLT

JUMP ON LESS THAN



(TST)

Description: Causes a branch to be executed if the specified address is less than the value contained in the selected accumulator. Otherwise the program counter is incremented by one.

Condition Codes: Unaffected

****NOTE****

JEQ/JLT operate on magnitude, not 2's complement.

JEQ: looks at magnitude comparator equal to output at time of the last accumulator operation.

JLT: looks at magnitude comparator less than output at time of the last accumulator operation.

Magnitude comparator compares the selected ACC to either the contents of working storage, or the data bus. The data bus is only selected for these instructions: LKP

A4I

A8I

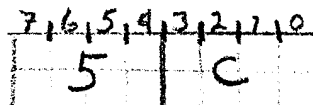
S4I

S8I

IPAI

JEQ

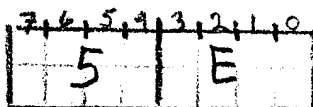
JUMP ON EQUAL



Description: Causes a branch to be executed if the specified value is equal to the selected accumulator. Otherwise the program counter is incremented by one.

Condition Code: Unaffected

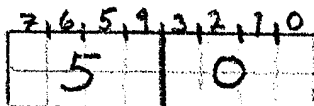
JOS JUMP ON ONE'S SHIFTED



Description: Causes a branch to be executed if a one was shifted from the least significant ^{bit} ~~byte~~ of the primary accumulator during a right shift operation. Otherwise the program counter is incremented by one.

Condition Codes: Unaffected

T4K TOGGLE 4-K BANK



Description: Bank selection of program memory. Selects the 4-K page according to the contents of the page register's two least significant bits. The two most significant bits are ignored. This instruction also causes a jump. The program address register should be loaded with the desired address before the instruction is executed.

Condition Codes: Unaffected

<u>Page Register</u>	<u>Function</u>
0	Not Implemented
1	Page 0 (0-4095)
2	Page 1 (4096-8192)
3	Toggle (not implemented)

****NOTE**** DUE TO TIMING, THE FIRST INSTRUCTION FOLLOWING A T4K
MUST BE A NOP.

EXAMPLE:

Lower 4-K	LPAI	UPPER
	SETP	2
	T4K	
LOWER:	NOP	(RETURN from upper)
UPPER:	NOP	
Upper 4-K	L:AI	LOWER
	SETP	1
	T4K	

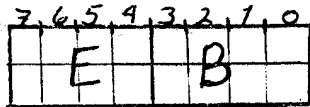
****NOTE**** DUE TO TIMING, THE FIRST INSTRUCTION FOLLOWING A T4K
MUST BE A NOP.

EXAMPLE:

Lower 4-K	LPAI	UPPER
	SETP	2
	T4K	
LOWER:	NOP	(RETURN from upper)
UPPER:	NOP	
Upper 4-K	L:AI	LOWER
	SETP	1
	T4K	

8. SHIFT OPERATIONS

SHR SHIFT RIGHT

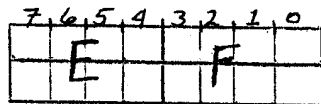


Description: Causes contents of the selected accumulator to be shifted right one place.

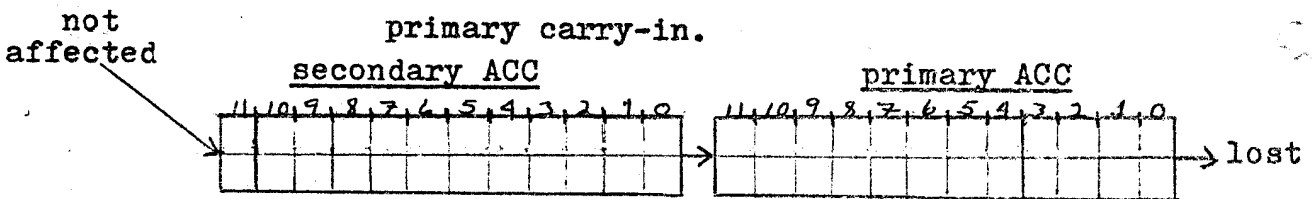
Condition Codes: RS: Right shift flag: set if one shifted out of 0.

NOTE: Subsequent JOs only works on primary acc. shifting a 1.

SHRB SHIFT RIGHT BOTH



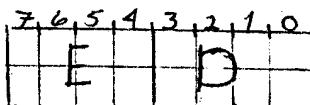
Description: Causes the contents of both primary and secondary accumulators to be shifted right one place simultaneously. The carry out from the secondary ACC is connected to the carry of the primary ACC i.e., the secondary carry-out is fed to the primary carry-in.



Condition Codes: RS: Set if one shifted out of Bit 0

****NOTE**** ANY RIGHT SHIFT OPERATION INVOLVING THE SECONDARY ACC IS AN ARITHMETIC SHIFT i.e. MOST SIGNIFICANT BIT IS NOT AFFECTED.

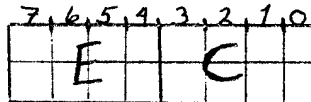
ASR ARITHMETIC RIGHT SHIFT



Description: Causes the contents of the selected ACC to be shifted right one place while forcing the most significant bit to remain unchanged.

Condition Codes: Set if one shifted out of Bit 0 of primary ACC

SHL SHIFT LEFT

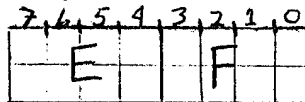


Description: Causes the contents of the selected accumulator to be shifted left one place. *NO instruction*

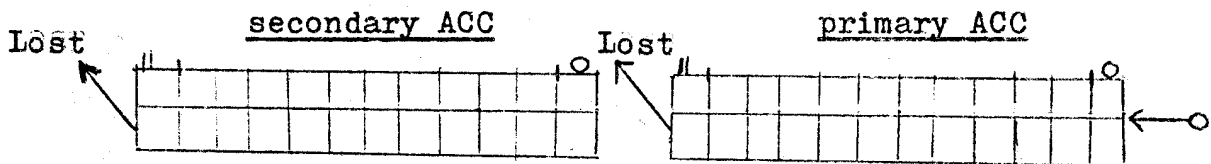
Condition Codes: NG: Set if result Negative (MSB set)

*MINUS SIGN BIT
cause succeeding JMI
to jump*

SHLB SHIFT LEFT BOTH



Description: Causes the contents of both the primary and secondary accumulators to be shifted left one place. There is no carry-in or carry-out i.e.



Condition Codes: NG: Set if result in primary ACC negative
(MSB set)

succeeding JMI

9. TABLE LOOK-UP OPERATIONS

LKP LOOK-UP

7	6	5	4	3	2	1	0
		E			2		

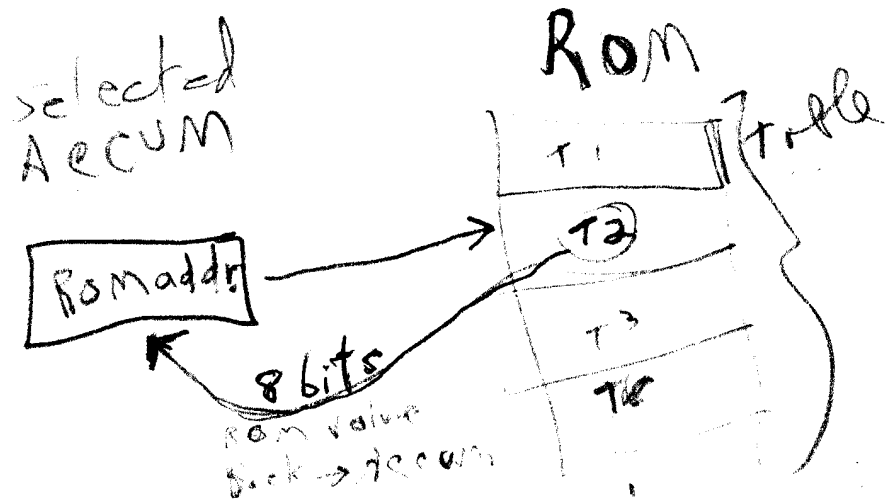
Description: Loads the word in program memory (ROM) addressed by the contents of the selected ACC into the selected ACC.

Condition Codes: Unaffected

****NOTE**** EXECUTION OF LKP INSTRUCTION MUST BE FOLLOWED BY NOP DUE TO TIMING CONSIDERATIONS.

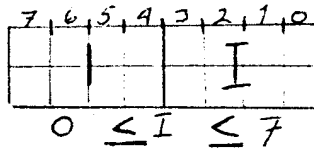
LKP

NOP



10. INPUT-OUTPUT OPERATIONS

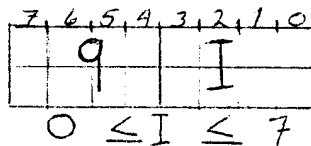
INP INPUT



Description: Loads the signal level from the primary input line specified by the 4-bit address carried with the instruction into the LSB of the primary ACC. When the secondary ACC is selected, the address specifies one of 8 secondary input lines and the destination become the secondary ACC.

Condition Codes: Unaffected

OUT OUTPUT

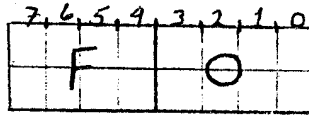


Description: Loads the contents of the least significant byte of the selected ACC into one of 8 external output latches selected by the 4-bit address carried along with the instruction.

Condition Codes: Unaffected

11. VECTOR OPERATIONS

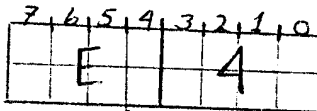
IV INITIALIZE VECTOR



Description: Initialize the vector generating sequence. Causes the X and Y coordinates of the initial point of the line segment to be drawn to be loaded into the X and Y registers from the primary and secondary accumulators respectively.

Condition Codes: Unaffected

NV NORMALIZE VECTOR



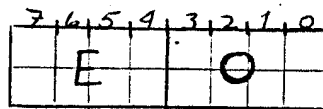
Description: Causes delta X and delta Y, the parameters describing the length and direction of the line segment to be drawn to be normalized in preparation for generating the vector. Delta X and delta Y are previously loaded into the primary and secondary accumulators respectively where the normalization by simultaneous left shifts occurs.

Condition Codes: Unaffected

****NOTE**** ATTEMPTING TO NORMALIZE DELTA VALUES OF 0, 0 WILL NOT WORK. CPU WILL HANG UP UNTIL FRAME TIMES OUT, THEN WILL DO A RESET. EITHER DELTA VALUE BEING NON-ZERO GETS AROUND THIS.

DV

START (DRAW) VECTOR

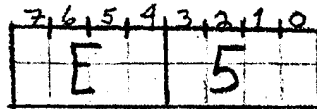


Description: Causes the drawing of the line segment to be started.

12. CONTROL OPERATIONS

FRM

FRAME



Description: Timing function----causes the computational section of the system to remain halted until a 26 millisecond period is completed.

To prevent the images on the screen from flickering and to create the illusion of continuous motion as an object moves, the line segments creating the display are redrawn 33 times per second. The FRM instruction causes the computational section to be halted until a 26 millisecond period is reached. For example, if the FRM instruction is reached at 23 milliseconds after completion of the previous 26 millisecond period the computational section will remain halted for 3 milliseconds. After a 26 MS period is completed, normal execution of the program instructions is again resumed starting with the instruction immediately following the FRM instruction.

Condition Codes: Unaffected

****NOTE**** THE INSTRUCTION IMMEDIATELY FOLLOWING THE FRM INSTRUCTION MUST BE CST (COLD-START).

CST COLD START

7	6	5	4	3	2	1	0
		F				F	

Description: Resets watch-dog timer----CST must be executed within 3MS of the FRM instruction otherwise a hardware timeout will occur forcing a reset and a jump to location 0.

Condition Codes: Unaffected

NOP NO-OPERATION

7	6	5	4	3	2	1	0
		5				F	

Description: Pseudo operation-----requires 400 nanoseconds to execute and is often useful in conjunction with those functions which cannot complete normal execution without timeout conditions occurring.

Condition Codes: Unaffected

D. MACROES

Following is a list of some commonly used macros. The list is by no means comprehensive: rather, it is an attempt to illustrate macro construction and use.

1. BLD BIG LOAD

Description: Loads primary ACC with 12-bit value
Loads upper 4-bits LDAI %
Loads lower 8-bits ABI %

2. SBLD SECONDARY BIG LOAD

Description: Loads secondary ACC with 12-bit value
SSA
LDAI %
SSA
ABI %

3. TLD TINY LOAD

Description: Loads primary ACC with 4-bit value
LDAI 0
A4I %

4. JSR JUMP TO SUBROUTINE

Description: .MACRO JSR

BLD * + 7

STA % 1

LP AI % 2

JMP

.ENDM

*JSR RET add \$UBR-Add
holder*

↓ call SUBR

TST RET add holder

LPAP

JMP

5. SJSR SHORT JSR

Description: Short form of JSR which allows routine to store return address.

```
.MACRO SJSR  
BLD * + 6  
LPAI % 1  
JMP  
.ENDM
```

SJSR SUBRaddV
↓ call Rout
STA RETadd.

TST RETadd.
LPAI
JMP

6. JSRUP JUMP SUBROUTINE UPPER BANK

Description: Jump and link to subroutine in upper 4-k from lower 4-k.

```
.MACRO JSRUP  
LPAI % 1  
BLD * + 6  
STA RTN  
SETP 2  
T4K  
.ENDM
```

7. JSRLO JUMP SUBROUTINE LOWER BANK

Description: Jump and link to subroutine in lower 4-k from upper 4-k.

```
.MACRO JSRLO  
LPAI % 1  
BLD * + 6  
STA RTN
```

SETP 1

T4K

.ENDM

8. MUL8 MULTIPLY 8-BIT VALUE

Description: .MACRO MUL8

MUL

MUL

MUL

MUL

MUL

MUL

MUL

MUL

.ENDM

III. PROGRAMMING EXAMPLES

A. PHYSICAL LAYOUT OF WORKING STORAGE

<u>WORD</u>	<u>PAGE</u>
0	
15	0
16	
31	1
32	
47	2
48	
63	3
64	
79	4
80	
95	5
96	
111	6
112	
127	7
128	
143	8
144	
159	9
160	
175	10
176	
191	11
192	
207	12
208	
223	13
224	
239	14
240	
255	15

B. EXAMPLES OF INSTRUCTION USAGE

1. SOFTWARE TIMER IMPLEMENTATIONS

a. Using a Positive Delay

```
TLD 10      ;Delay Count
LPAI TIMER  ;Delay Loop Address
TIMER: S41 1      ;Delay ←—— Delay - 1
TST ZERO    ;Test Count
JLT         ;ACC > 0
↓
(Timer has expired)
```

b. Using a Negative Count

```
DELAY: EQU - 50
START: BLD DELAY ;Get Count (12-bit value)
TIMER: LPAI CONT ;Load PAR with address
        ;of where to go when expired
A41     ;Delay ←—— Delay - 1
TST ZERO ;Load Condition Code -
        ;ACC not modified
JEQ     ;Timer Expired If True
LPAI TIMER ;Else Next Iteration
JMP     ;Do It
```

2. USING THE TEST ('TST') INSTRUCTION

One way of using the 'TST' instruction is to test a variable in working storage against a known value in the ACC for magnitude and conditionally branch to another location in the program depending on the results of that test:

```
LPAI ELSEWHR ;Exit address
```

```

LDAI      0          ;ACC←—0
SETP      7          ;'VAR' Defined on Page 7
TST       VAR        ;Test 'VAR' EQ. 0
JEQ       VAR VAR = 0 ;Jump if true

```

Another use for the 'TST' instruction is in conjunction with the LPAP instruction.

```

SETP      4          ;'Return' on Page 4
TST       RETURN    ;Set Up WS ADDR SEL/REG
LPAP                 ;Load PAR With Previously
                        ;Selected Address
JMP                  ;Take (Subroutine) Return

```

An example of a range test using the 'TST' instruction.

```

LPAI      EXIT      ;
LDAI      0          ;ACC←—0
TST       COUNT     ;If Count EQ. 0
JEQ                  ;Then Go To 'Exit'
ABI       25         ;Else ACC←—ACC + 25
TST       LIMIT     ;If 'Limit' < 25
JLT                  ;Then Go To 'Exit'

```

3. EXAMPLES USING PREVIOUSLY SELECTED INSTRUCTIONS

a) LDAP/STAP

```

LPAI      EXIT      ;Exit Address
LDA       COUNT     ;Load ACC With Something
SHR                  ;Right Shift One Place
JOS                  ;Jump If 1 Shifted From LSB
LDAP                 ;Else Load ACC With
                        ;Original Value From 'Count'
A41      5          ;Count←—Count + 5

```

STAP ;Store In 'Count'

b) ADDP

LDA XFIVE ;ACC ← 5
SHL ;ACC * (2) = 10
SHL ;ACC * (2) = 20
ADDP ;ACC + (5) = 25
STA TEMP

c) ANDP

LDA COIN ;Get # Coins Credited
;(1 or 2)
A4I 1 ;Coin ← Coin + 1
STAP ;Store In 'Coin'
TLD 1 ;ACC ← 1
ANDP ;And Coin Count To ACC

COIN: 010 or 011

ACC: 001 001

ACC: 000 001

A4I 1 001 001

1 2

STAP ;Store In 'Coin'

d) LPAP ET AL

LDA OBJNUM ;Object Number To Draw (From
;Data Tables Of N Objects)
SHL
SHL ;OBJNUM * 4
ADDP ;+ OBJNUM
STA TEMP ;To Set Up WS ADDR. REG.

```

BLD      GETIT      ;Beginning Addr Of Vector
ADDP                                ;(OBJNUM) + (OBJNUM) * 4
STAP                                ;Store It So We Can
LPAP                                ;Load PROGRAM ADDR. REG.
BLD      DOIT      ;ADDR Of Routine To Draw
                                ;Object After Finding In Table
STAP                                ;Set Up For Next LPAP
JMP                                            ;Go Into 'GETIT' Vector Table
GETIT:BLD  OBJ1     ;'OBJNUM' = 0: Pick Up Table Pointer
LPAP                                ;Load PAR With 'DOIT'
JMP                                            ;Do It
BLD      OBJ2     ;'OBJNUM' = 1
LPAP
JMP
BLD      OBJ3     ;OBJNUM = 2
LPAP
JMP
BLD      OBJn     ;OBJNUM = n; Last Entry
DOIT: STA  POINTER ;Store ADDRS Of Data Table
JSR      Draw     ;Draw It

```

4. EXAMPLE OF INDIRECT ADDRESS INSTRUCTION

The following example will illustrate one use of the WS instruction. The INDIRECT ADDRESS instruction is important since it enables the use of subscripted variables. This routine could be used to calculate how many remaining ships the current player has and cause that number to be displayed on the CRT.


```

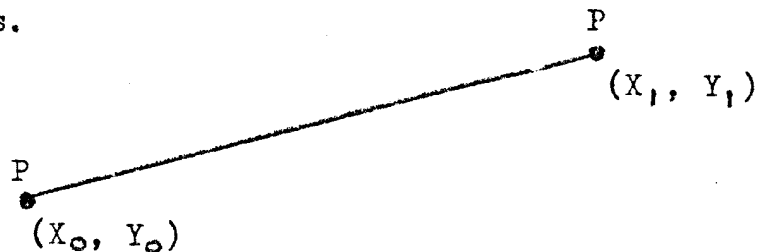
PAGE7 EQU 112      ;START OF PAGE 7
;WORKING STORAGE -- PAGE 7
SHIP1 EQU 0        ;# SHIPS LEFT - PLAYER 1
TEMP7 EQU 1        ;TEMPORARY STORAGE
SHIP2 EQU 2        ;# SHIPS LEFT - PLAYER 2
PLAYER EQU 3       ;CURRENT PLAYER (1 or 2)
;
START SETP 7       ;LOAD PAGE REGISTER
    LDA PLAYER     ;GET CURRENT PLAYER
    SHR            ;(BECOMES 0 OR 1)
    SHL            ;(BECOMES 0 OR 2)
    ASI PAGE7      ;FORM PAGE-WORD ADDRESS
                    ;OF EITHER 'SHIP 1' OR
                    ;'SHIP 2'
    STA TEMP7      ;SAVE IT
    WS TEMP7       ;LOAD WS ADDR REG. WITH
                    ;EITHER 112 OR 114
    LDAP          ;PICK UP CONTENTS OF
                    ;'SHIP1' OR 'SHIP2'
    STA NUMBER
    SJSR CONVERT   ;CONVERT BCD
    SJSR DRAW      ;DISPLAY ON SCREEN

```

C. VECTOR LINE DRAWING TECHNIQUES

A. GENERAL

The straight line, a basic element in many displays. If two (2) points are to be connected by a line segment, only the end points of the vector need be computed for a vector display. In most cases, operating on individual points is only a beginning. Generally, techniques are needed for dealing with line segments that connect points to define figures and regions.



B. TRANSFORMATIONS

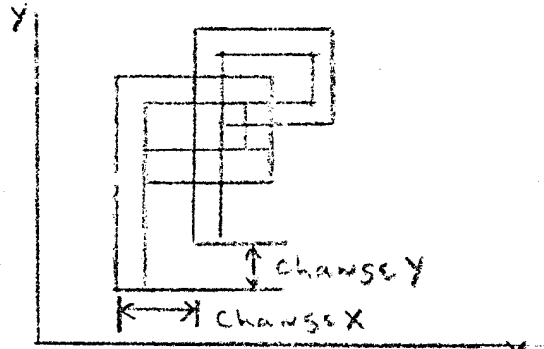
The operations necessary to manipulate points to perform useful tasks are called transformations. There are three (3) basic transformations in two (2) dimensions; 1) Translation
2) Rotation
3) Scaling

1. TRANSLATION

Translation, or positioning, is the movement of a point or points by an amount in X and an amount in Y. The motion is such that neither the shape, size, or orientation is changed. If all the points associated with a line or figure are translated by an equal amount, the graphic element is translated without change in size, shape, or

orientation. It may be expressed as: $X^1 = X + \text{CHANGE } X$
 $Y^1 = Y + \text{CHANGE } Y$

Where change X need not equal change y.

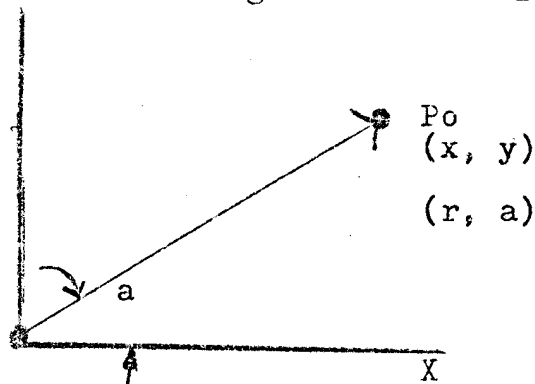


2. ROTATION

Rotation involves a computation which maintains shape but changes orientation. A rotation will generally leave only one point in the two-dimensional space with its position unchanged: The center of rotation.

For example, assume the object to display is a space ship. The information defining it's appearance is contained in program memory as a series of coordinate points, each coordinate point being an end point of a line segment defining the outline of a ship. To rotate the ship, the stored end points are defined in terms of polar coordinates. Rotating the entire set of polar coordinate end points is achieved by incrementing every angle by the value of the angular rotation variable. The value of the angular rotation variable is determined by the rotation routine, a software counter which either increments the value, decrements the value, or leaves it unchanged, depending on the external operator action (tilt left; tilt right; don't touch). After the rotation of the set of end points, the points are converted into Cartesian

coordinates using a sine table stored in program memory. Those points are then used to determine the parameters necessary to draw the line segments from which a ship is constructed.

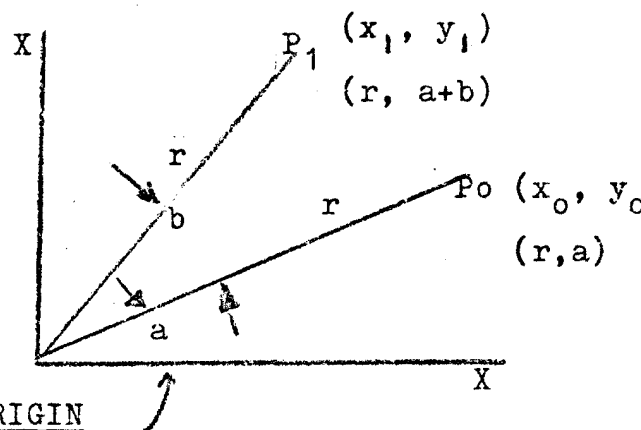


POLAR COORDINATE REPRESENTATION OF A POINT IN THE XY PLANE

if P_0 is rotated about $(0,0)$ by an angle of b to become

P_1 then; $X_1 = r \cos (a + b)$

$Y_1 = r \sin (a + b)$



ROTATION OF VECTOR ABOUT THE ORIGIN

The Trigonometric equation for rotation of vector about

the origin is: $X_1 = X_0 \cos (b) - Y_0 \sin (b)$

$Y_1 = X_0 \sin (b) - Y_0 \cos (b)$

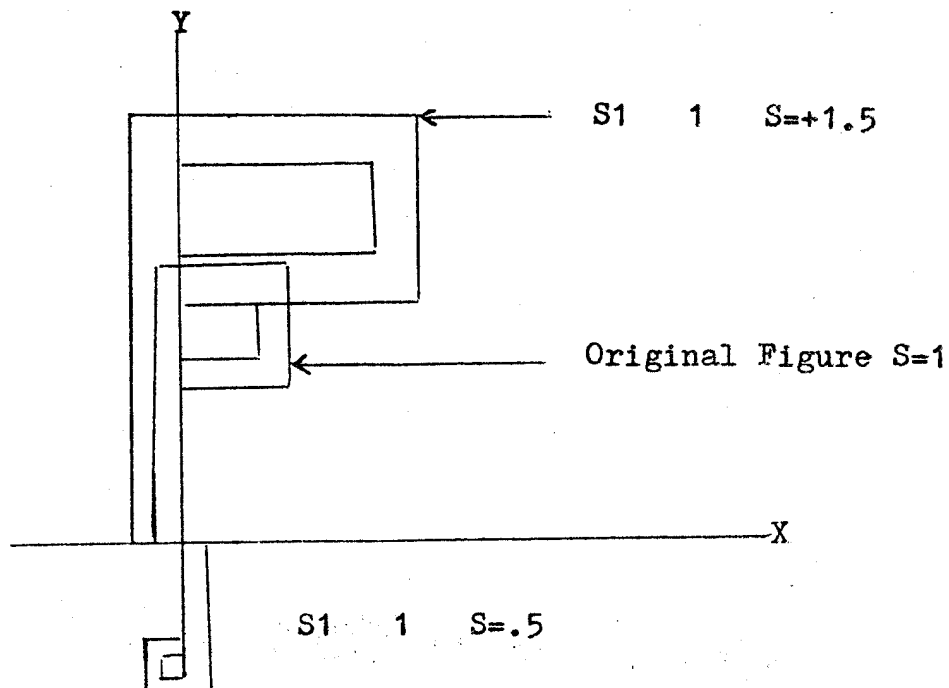
3. SCALING

Scaling, or magnification, involves a change in size without change in orientation. Depending on the definition of shape, it is either unchanged or changed "without distortion."

The equations; $x_1 = sx_0$

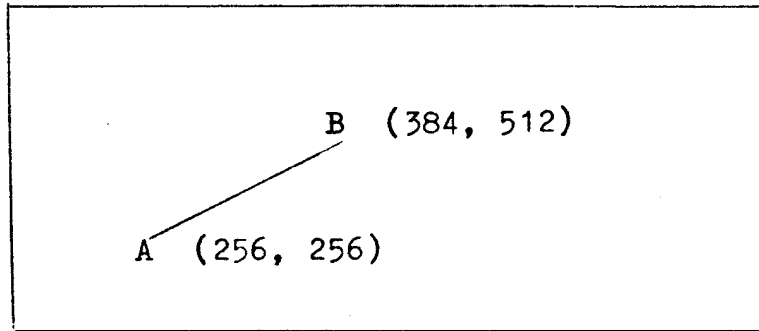
$y_1 = sy_0$

will scale X and Y by a factor of S. The factor may be greater than or less than 1. If a negative value is used for S, then reflection about the origin is performed. If the scale factors for X and Y are different, then stretching is accomplished.



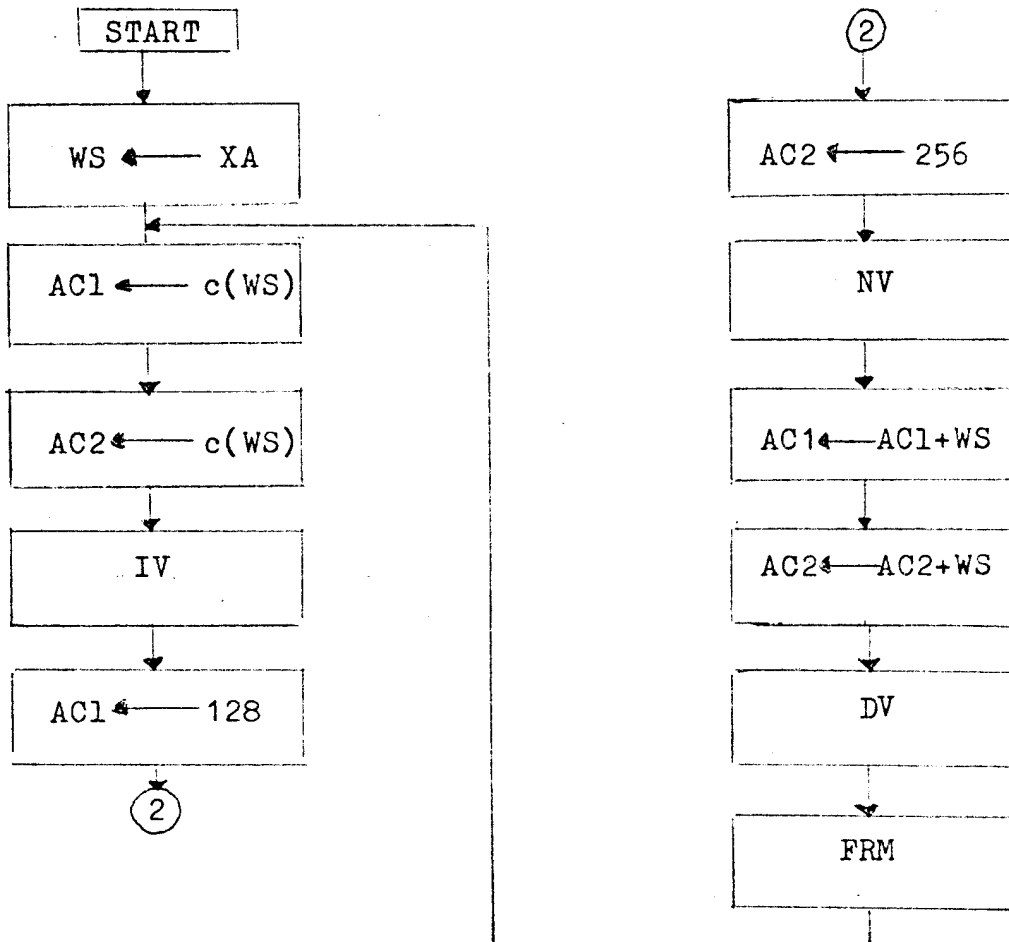
SCALING AN ARBITRARY FIGURE IN THE XY PLANE

D. SAMPLE PROGRAM TO DRAW A LINE



LINE SEGMENT AB Point A: X = 256 Y = 256
Point B: X = 384 Y = 512

Parameters necessary for specifying line segment AB are Cartesian coordinates of point A and delta X and Y values which define length and direction of line segment relative to point A.



AC1 = Primary ACC
AC2 = Secondary ACC
WS = Working Storage

SAMPLE LINE DRAWING PROGRAM

```
.PROG
;*** EQUATES
XA      .EQU      256      ;X CO-ORDINATE FOR A
YA      .EQU      256      ;Y CO-ORDINATE FOR A
XB      .EQU      384      ;X CO-ORDINATE FOR B
YB      .EQU      512      ;Y CO-ORDINATE FOR B
DELTAX  .EQU      XB-XA    ;X DELTA (AB SEGMENT)
DELTAY  .EQU      YB-YA    ;Y DELTA (AB SEGMENT)
DELAY   .EQU      -50

;*** WORKING STORAGE - PAGE 0
XO      .EQU      0        ;INITIAL SEGMENT - X CO-ORDINATE
YO      .EQU      1        ;INITIAL SEGMENT - Y CO-ORDINATE
ZERO    .EQU      2

;*** START OF EXECUTABLE CODE
        .ORG      0        ;PROGRAM ORIGIN - LOWER PAGE
        .SETP     0        ;SET TO WORKING STORAGE - 0
        BLD      XA init yo? ;LOAD INITIAL X
        STA      XO ↙      ;STORE IN PAGE 0
LOOP:   LDA      XO        ;LOAD X-COORD TO PRIMARY AC
        SSA                      ;SELECT SECONDARY AC
        LDA      YO        ;LOAD Y-COORD TO SECONDARY AC
        IV                      ;POSITION ELECTRON BEAM (POINT A)
        LDAI     0
        STA      ZERO      ;DELAY TO ALLOW
        ABI      DELAY     ;D/A CONVERTERS TO SETTLE
```

```

TIMER:  LPAI      CONT
        A4I        1      ;WAIT FOR TIME-OUT
        TST        0
        JEQ                ;TIMER HAS EXPIRED
        LPAI      TIMER
        JMP
CONT:   BLD      DELTAX      ;LOAD SEGMENT DISP. FOR X-AXIS
        SSA
        BLD      DELTAX      ;LOAD SEGMENT DISP. FOR Y-AXIS
        NV
        ADD      XO          ;ADD OFFSET TO NORMALIZED
        SSA
        ADD      YO          ;CONTENTS IN BOTH ACs
        DV                ;DRAW LINE SEGMENT FROM A-B
DONE:   LPAI      DONE
        JVN                ;WAIT UNTIL LINE COMPLETE
        FRM                ;CONTROL LOOP - WAIT 30 MS
        CST                ;SO WE WON'T RESET
        LPAI      LOOP      ;CONTINUE
        JMP
        .END

```


E. PROGRAM COPYRIGHT

Usually the first function to be performed upon powering up the ROM, is to checksum the manufacturer's copyright. This provides (at least) a degree of security if a competitor were to 'bootleg' a CINEMATRONICS property and attempt to delete or alter the copyright. The checksum code should be imbedded between non executable code to make decoding of the algorithm more difficult and program execution fail in the event that the checksum fails to match the expected value. In addition, the checksum should be computed so as to be an instruction (operator-operand) in program memory; if the checksum is altered, the program will not execute properly.

1) Define the ASC11 character string

1981 CINEMATRONICS

at the end of the lower 4-K prom where the last character would end on FFF_{16} (4095_{10}).

.ORG FEC
COPY: .ASC11 "(C) 1981 CINEMATRONICS"

FEC	28 43 29 31
FF0	39 38 31 43
FF4	49 4E 45 4D
FF8	41 54 52 4F
FFC	4E 49 43 53

2) Compute the checksum i.e. add all hex characters and divide by 16 to form 8 bit value; $536_{16} \div 16 \longrightarrow 3C$ (decodes as S4I 12)

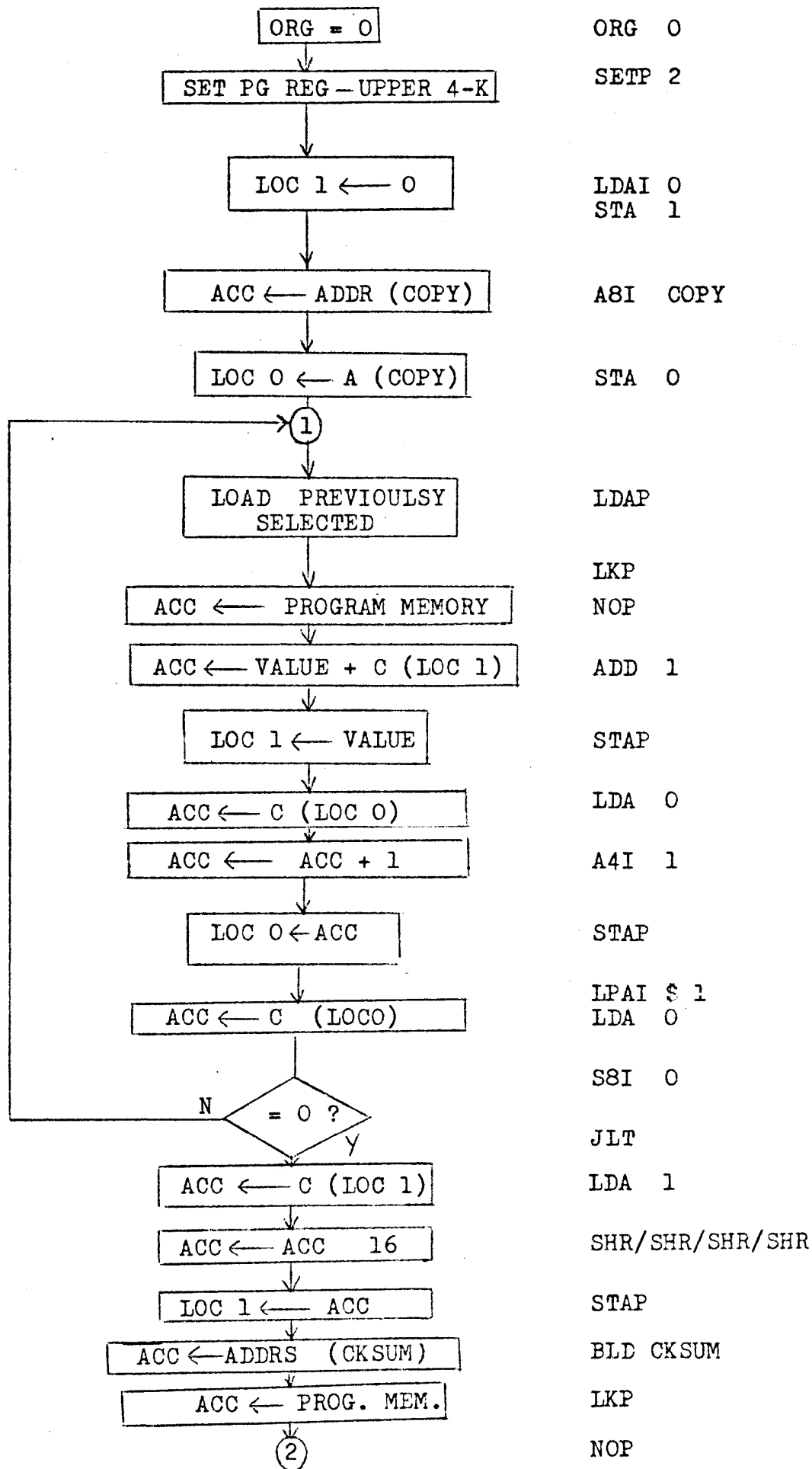
CKSUM: S4I 12

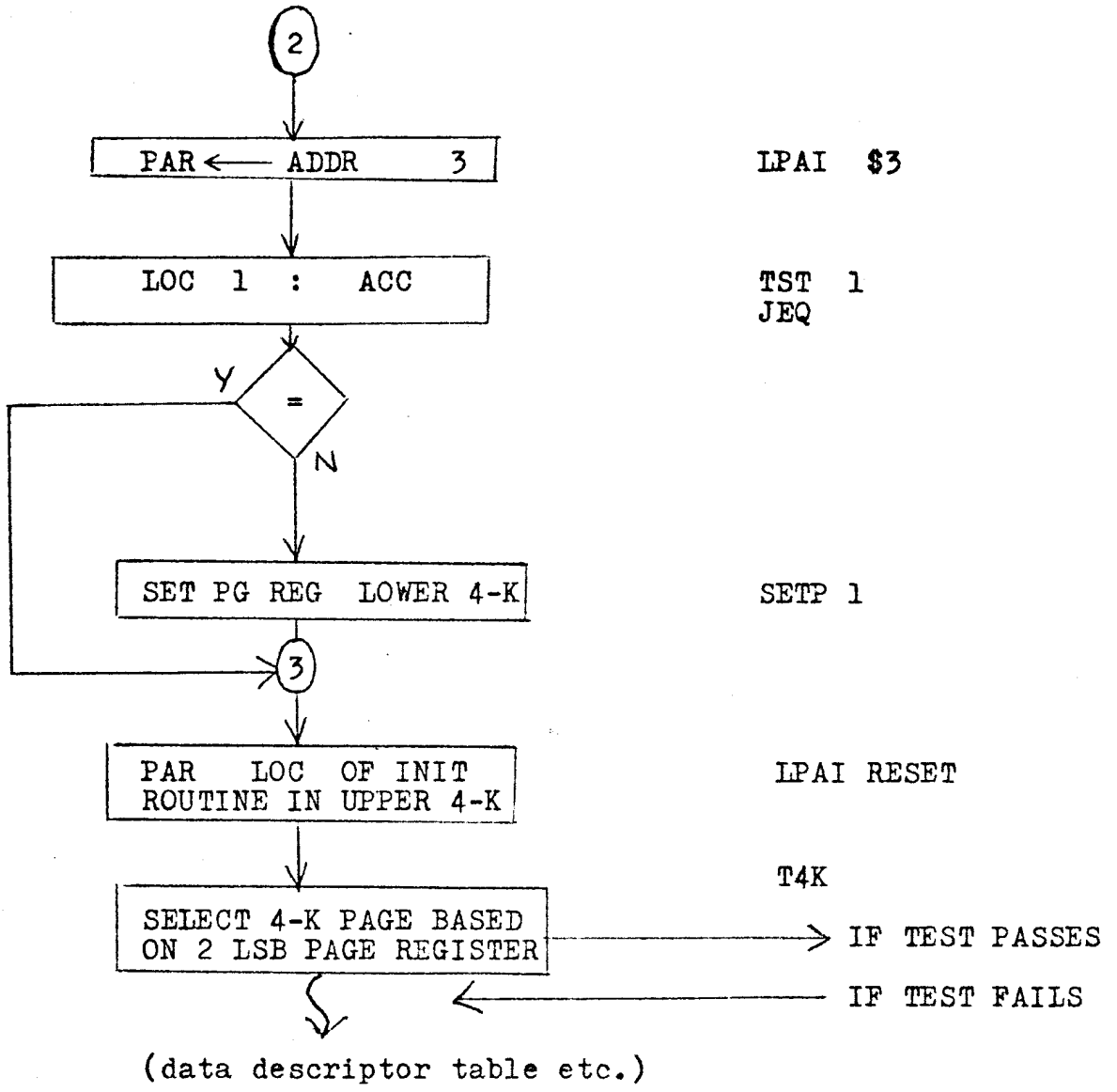
3) Following is a flow diagram of the checksum process:

LOC 0/LOC 1 refer to working storage memory locations 0 and 1 respectively.

ACC refers to the primary accumulator.

PAR is program address register.






```

SHL
SHL                ;X4
STA                Y-AX        ;CURRENT Y-COORDINATE
LDA                ADDRESS
A4I                1
STAP               ;POINTER ← POINTER + 1
LPAI               S+2
JVN                ;WAIT ON DRAWING NOT FINISHED
SSA
LDA                Y-XA        ;Y TO SECONDARY AC
LDA                X-AX        ;X TO PRIMARY AC
IV                 ;POSITION BEAM
(DELAY)
LDAI               0
A4I                4          ;SEGMENT DISP. FOR X
SSA
LDAI               0          ;SEGMENT DISP FOR Y
NV                 ;NORMALIZE
                   ;ΔX AND ΔY
                   ;VALUES
SSA
LDA                Y-AX
LDA                X-AX
DV                 ;DRAW POINT
LPAI               START
JMP                ;GET NEXT X-Y POINT

```

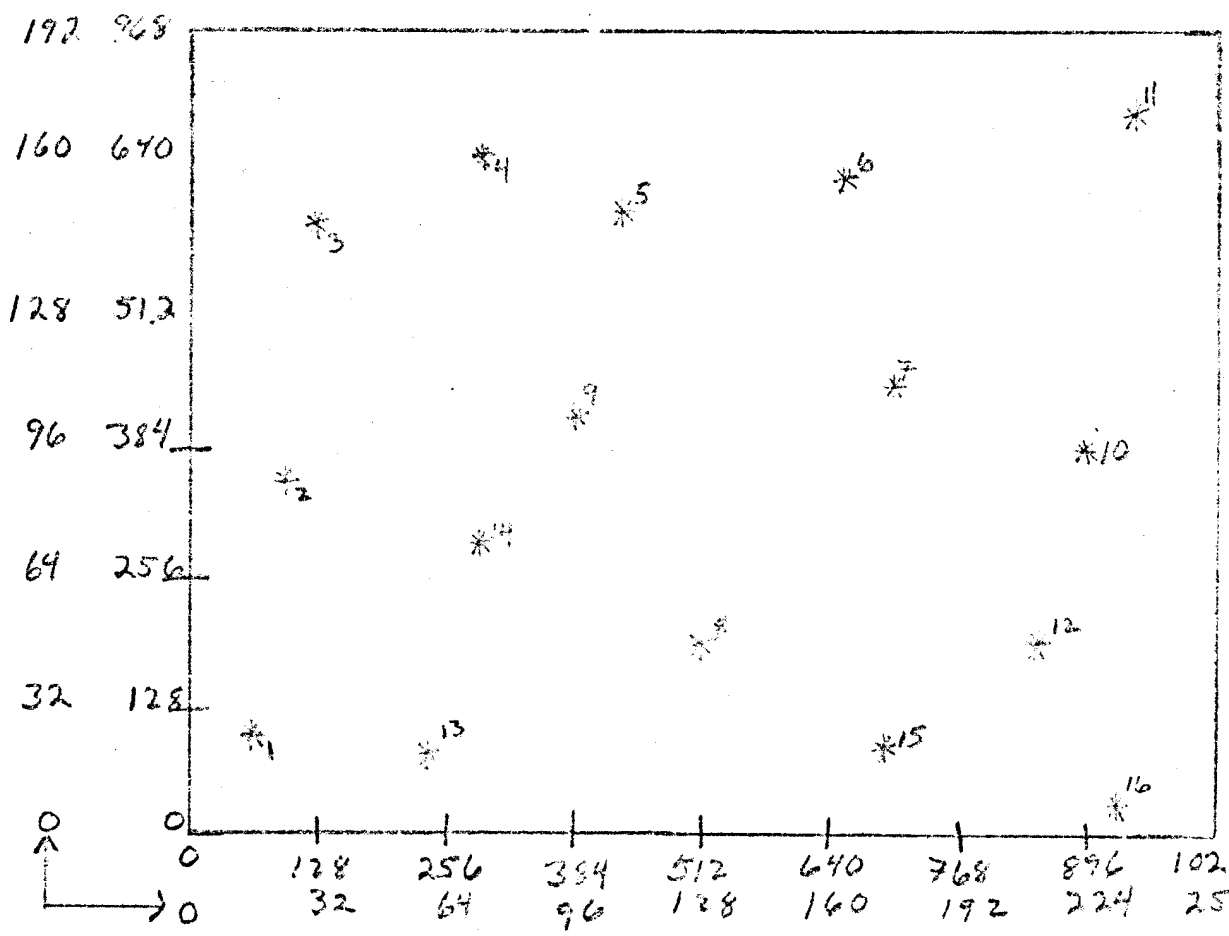
DONE:

STAR: DATA TABLE FOR STARS

BYTE	16, 24, 24, 88, 32, 152, 72, 160
BYTE	112, 156, 166, 155, 176, 108, 128, 48
BYTE	96, 104, 224, 96, 234, 172, 212, 48
BYTE	60, 20, 72, 72, 172, 22, 222, 8
BYTE	255, 255,

X

Y

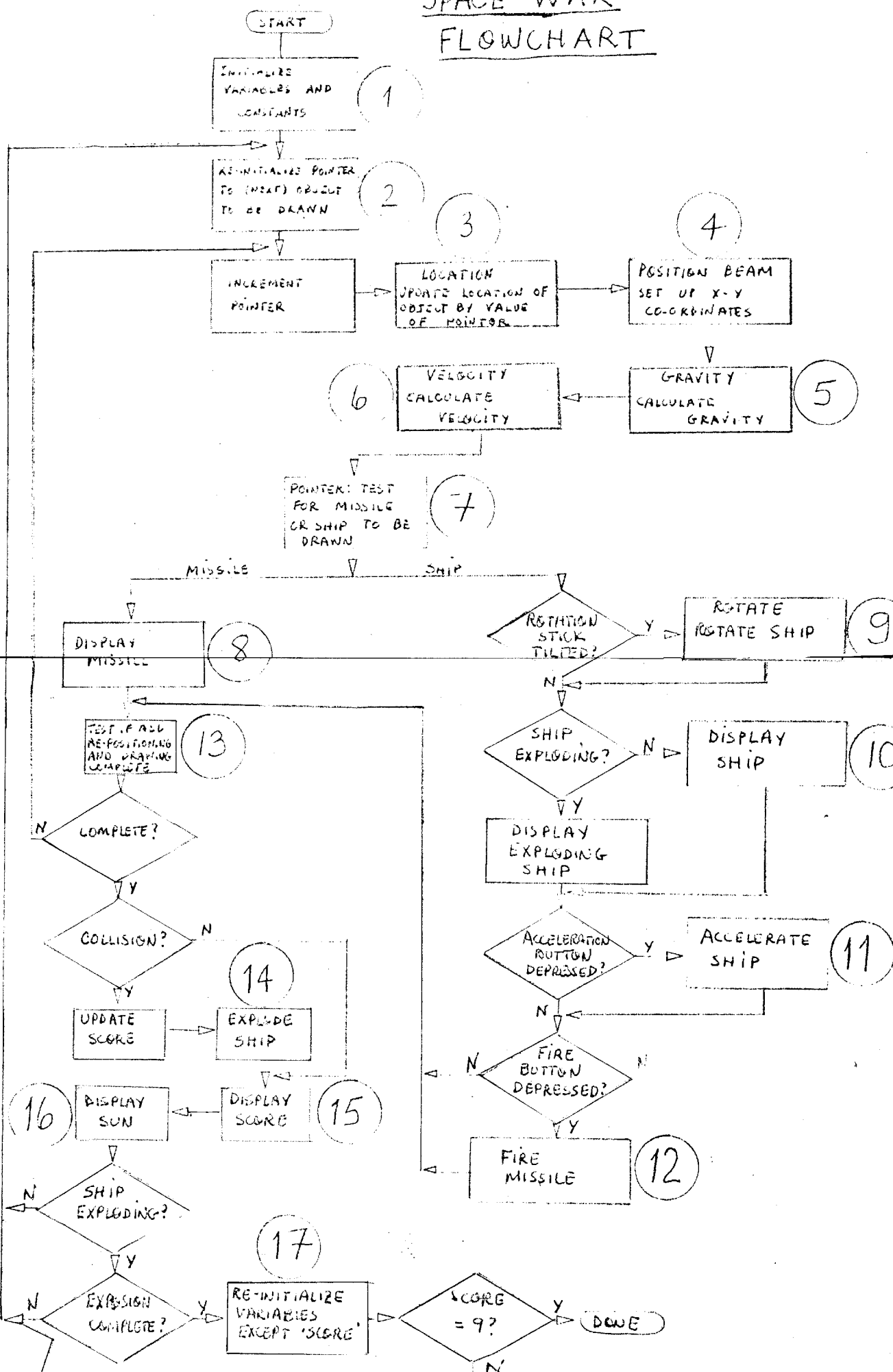


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
X	16	24	32	72	112	160	176	128	96	224	234	212	60	72	172	292
Y	24	88	152	160	156	155	108	48	104	96	172	48	20	72	22	8

G. 'SPACE WARS' - A PROGRAMMING IMPLEMENTATION

The accompanying flow chart and general discussion of the functions employed in the design of 'SPACE WARS' may aid the reader in gaining a more in-depth understanding in the design of video game programming techniques. The general flow of events may vary from game to game, but functionally the techniques are essentially the same.

SPACE WAR FLOWCHART



1) INITIALIZATION OF WORKING STORAGE VARIABLES

When a game is first started, variables and constants which have been assigned locations in the working storage memory are loaded with initial values from the program memory. The values loaded into the variables at this time determine the initial positions and velocities of the ships and any other objects appearing on the screen.

Variables associated with a particular routine should all be contained in the same page of working storage to minimize the number of times that the page register has to be reloaded.

2) OBJECT POINTER

A variable referred to as a pointer is initially loaded with a value that both indicates the first object to be displayed and points to the parameters describing the positioning and trajectory of the object. When the value of the pointer is added to a position offset, a velocity offset or any other offset used in the program, the sum will yield the working storage memory address a F the X component parameter required by the particular routine associated with the offset. When the value of the pointer plus 1 is added to an offset the sum yields the memory address of the Y component parameter. Parameters associated with the same function but determining X and Y components are stored in adjacent memory locations.

3) LOCATION ROUTINE

This routine will update the location of the ship or missile (depending on pointer). The X and Y velocities of the ship or missile being relocated (moved across the display screen) are added to the previous location yielding a new location on the screen. Also a test is made to ensure that the ship or missile remains on the CRT display. If the ship or missile is about to go off the screen at the top, it is assigned the coordinates of the bottom of the screen, or if it is about to go off the left side it is assigned the location of the right side and so forth to create a wrap around effect.

4) POSITION BEAM ROUTINE

After the location of the ship or missile has been determined, the display section is given the coordinates of the center point of the ship or missile so that the beam can start moving towards that position.

Previously the beam was at the position of another ship or missile that could have been located any place on the screen, near or far from the ship or missile being positioned. The beam is repositioned at this time so that it will have settled at the designated position by the time that the system is ready to start drawing the ship or missile.

5) GRAVITY ROUTINE

A gravitational sun is located in the center of the screen.

The gravitational forces on the ships and missiles are calculated using Newton's equations of motion. In solving for X and Y component gravitational accelerations acting upon a ship or missile, first the radius from the sun to the ship or missile is calculated. Next the radius offset by a constant is used to address a location in a look-up table contained in the program memory. The value returned from the addressed location in the look-up table is proportional to $1/\text{radius}^3$. Finally, a constant, the returned value for $1/\text{radius}^3$, and the X component of the position are multiplied together yielding the X component gravitational acceleration. The same multiplication is then repeated using the Y component gravitational acceleration.

6) VELOCITY ROUTINE

The velocity routine adds the current X and Y velocity values of the ship or missile being repositioned. An additional function of the velocity routine is to limit the maximum velocity of the ship so as not to frustrate inexperienced players who would lose control of their ships.

7) POINTER TEST

At this point in the execution of the program, the CRT beam is at the desired location having been allowed to reposition itself and settle while the gravity and velocity calculations needed to update the next position of the ship or missile were being executed. First the value of the pointer is examined to

determine whether a ship or missile is to be drawn.

8) MISSILE DRAWING ROUTINE

To draw a missile, the display section is simply instructed to plot a point at the location that was already loaded for purposes of positioning the beam.

9) SHIP ROTATION ROUTINE

The methodology for rotating an object about an origin is covered in "VECTOR LINE DRAWING TECHNIQUES!"

10) SHIP DISPLAY ROUTINE

The information defining the appearance of a ship is contained in the program memory as a series of coordinate points, each coordinate point being an end point of a line segment defining the outline of the ship. This is done so that if a ship were to be rotated, the stored end points are defined in terms of polar coordinates. If no rotation were necessary, to draw a ship would simply involve retrieving the parameters defining the line segments to construct the ship from a table in program memory and passing these (coordinates) to the display system.

11) ACCELERATION ROUTINE

The acceleration routine causes a ship to be accelerated in the forward direction when the associated thrust button is pressed by the player. When the associated thrust button is pressed, the X and Y coordinates of the front of the ship

relative to the ship's center of gravity are added to the value of the variable defining the ship's acceleration.

12) MISSILE FIRE ROUTINE

When the associated missile fire button is depressed a missile is assigned the coordinates of the front of the ship from which it is being fired. A button held down continuously is ignored; it must be released and pressed again to fire another missile. The velocity with which the missile leaves the ship is the vector sum of a fixed velocity in the direction the ship is pointed and the velocity of the ship itself. Also, when a missile is fired, a variable which indicates that the missile is active is loaded with a value that is repetitively decremented. If the decremented value reaches zero before the missile hits a ship, it disappears from the screen. Because it takes a finite amount of time to locate and display a missile, each ship is limited to a specified number of missiles on the screen at a time.

13) COMPLETION TEST

After a ship or missile is drawn the value of the pointer is again examined to determine if all of the ships and missiles have been repositioned and drawn. If the value of the pointer indicates that everything has not been repositioned and drawn, the value is incremented to point at the parameters defining the next ship, or missile, and the loop just described is repeated. Otherwise, if the value of the pointer indicates

that everything is complete, the collision and scoring routines are executed.

14) COLLISION ROUTINE

The collision routine checks for a collision between (a) Two Ships, (b) A Ship and a Missile, (c) A Ship and the Sun. A hit is detected when the center point of a ship falls within a specified \pm AX and \pm AY of the center of another object. If a ship is hit, flags are set that cause the doomed ship to be drawn from points in an exploding ship table depicting the scattering of pieces of a ship. Also if a ship is destroyed, a variable assigned to the other ship is incremented and set for keeping score.

15) DISPLAY SCORE

Seven segment numbers are used for displaying the score with the decoding of the segments done through a look-up table in program memory.

16) DISPLAY SUN

After the collision routine has been executed the sun is displayed. The sun is made up of a cluster of radial line segments, half of which are displayed every other frame to give a flickering effect.

17) REINITIALIZE VARIABLES

When no ships have been hit or when a hit ship is in the process

of exploding, the value of the pointer is reinitialized so as to point to the first ship in preparation for the program to begin again the program loop just described. If a ship has been hit and the explosion has been completed, all the variables except the score variables are reinitialized enabling another contest to be played. Also, at this point in the program if a player has attained a score of 9 hits the game is ended.

H. SUBROUTINE TO DRAW A NUMERIC DIGIT

Pass the binary number 0-9 in 'INDEX' 'DRAW' will draw non-scaled digit on the screen at coordinates specified by 'XPOS' and 'YPOS' using data tables located at start of second program memory bank (1000 H) following are parameters required by subroutine (Page 0 is arbitrary)

*PAGE 0 WORKING STORAGE

RETURN	EQU	0	;return address
XPOS	EQU	1	;initial X-coordinate
YPOS	EEQU	2	;initial Y-coordinate
INDEX	EQU	3	;binary number 0-9
IMPX1	EQU	4	;intermediate X1
IMPX2	EQU	5	;intermediate X2
IMPY1	EQU	6	;intermediate Y1
IMPY2	EQU	7	;intermediate Y2
K255	EQU	8	
ZERO	EQU	9	

DRAW:

SETP		0	
STA	RETURN		;SAVE RETURN
LDA	XPOS		
SBI		42	
STAP			
SSA			
LDA	YPOS		
IV			;INITIALIZE BEAM

```
LOOP:  ADD          TEMP          ;ADDR OF NEXT MULTIPLE
      ;OF PRIME
      STA          WS
      TST          N255          ;TEST FOR END OF ARRAY
      JLT
      SSA
      LDAI         0            ;SET MULTIPLES TO ZERO
      SSA
      STAP
      LPAI         LOOP
      JMP
DONE:  LPAI         DONE
      JMP
```

```

LDA          INDEX          ;GET BINARY NUMBER
LKP          ;GET INDEX INRO
NOP          ;TABLE FROM 'DISP'
STAP        ;STORE IN 'INDEX'

DLOOP:

LDA          INDEX          ;BUMP TO FIRST BYTE
A4I         1              ;OF RESPECTIVE TABLE
LKP          ;i.e. DATA0.....DATA9
NOP          ;AND GET INCREMENT
STA         TMPY1         ;SAVE Y1 INCREMENT
SSA
LDAP        ;SAME TO SECONDARY
SSA
ADD         YPOS          ;ADD IN CURRENT Y
LDA         INDEX
LKP          ;GET X1 INCREMENT
NOP
LPAI        DEXIT
TST         K255
JEQ         ;IF LAST ENTRY IN TABLE
STA         IMPX1         ;SAVE X1 INCREMENT
ADD         XPOS          ;ADD IN CURRENT X
LPAI        *+2
JVN         ;WAIT ON VECTOR
IV          ;INIT. VECTOR WITH COORDS.
TLD         15           ;WAIT A FEW TICKS
S4I         1
TST         ZERO

```

JLT		
LDA	INDEX	;GET POINTER
A4I	2	
LKP		;GET NEXT INCREMENT
NOP		
SUB	IMPX 1	;LESS INITIAL X1 INCREMENT
STA	IMPX 2	;BECOMES X 2
LDA	INDEX	
A4I	4	;DO Y
STAP		
S4I	1	
LKP		;GET Y INCREMENT
NOP		
SUB	IMPY1	;LESS INITIAL Y1 INCREMENT
SSA		
STAP		;RESET INITIAL Y COORD.
STA	IMPY2	;BECOMES Y2
SSA		
LDAP		;SECONDARY ACC = Y2
LDA	IMPX 2	;PRIMARY ACC = X 2
NV		;NORMALIZE
ADD	IMPX 1	;ADD X OFFSET
ADD	XPOS	;ADD BASE
SSA		
ADD	IMPY1	;ADD Y OFFSET
DV		;DRAW LINE SEGMENT
LDAI	DLOOP	
JMP		;FINISH REST OF DIGIT

DEXIT:

```
JVN                                ;WAIT FOR SEGMENT TO FINISH
TST                                RETURN
LDAP                               ;RETURN TO CALLER
JMP
```

ORG 1000H

;UPPER 4 K DATA TABLES

;THESE NEXT 11 DATA TABLES MUST NOT MOVE FROM HERE!!

DATDIS: BYTE 10, 27, 32, 53, 70, 83, 100, 121, 130, 147

DATA0: BYTE 0, 0, 30, 30, 30, 30, 30, 54
 BYTE 30, 54, 0, 24, 0, 24, 0, 0, 255

DATA1: BYTE 0, 0, 30, 54, 255

DATA2: BYTE 0, 0, 30, 12, 0, 0, 0, 12
 BYTE 0, 12, 30, 48, 30, 48, 24, 54
 BYTE 24, 54, 0, 42, 255

DATA3: BYTE 0, 0, 30, 42, 30, 42, 12, 30
 BYTE 12, 30, 30, 54, 30, 54, 0, 42, 255

DATA4: BYTE 0, 0, 30, 54, 30, 42, 0, 24
 BYTE 0, 24, 12, 48, 255

DATA5: BYTE 0, 0, 22, 32, 22, 32, 0, 24
 BYTE 0, 24, 12, 45, 12, 45, 30, 54, 255

DATA6: BYTE 0, 0, 0, 12, 0, 0, 18, 7
 BYTE 18, 7, 30, 30, 30, 30, 0, 12
 BYTE 0, 12, 30, 54, 255

DATA7: BYTE 0, 0, 30, 54, 30, 54, 0, 42, 255

DATA8: BYTE 0, 0, 30, 30, 30, 30, 0, 30
BYTE 0, 30, 30, 54, 30, 54, 0, 0, 255

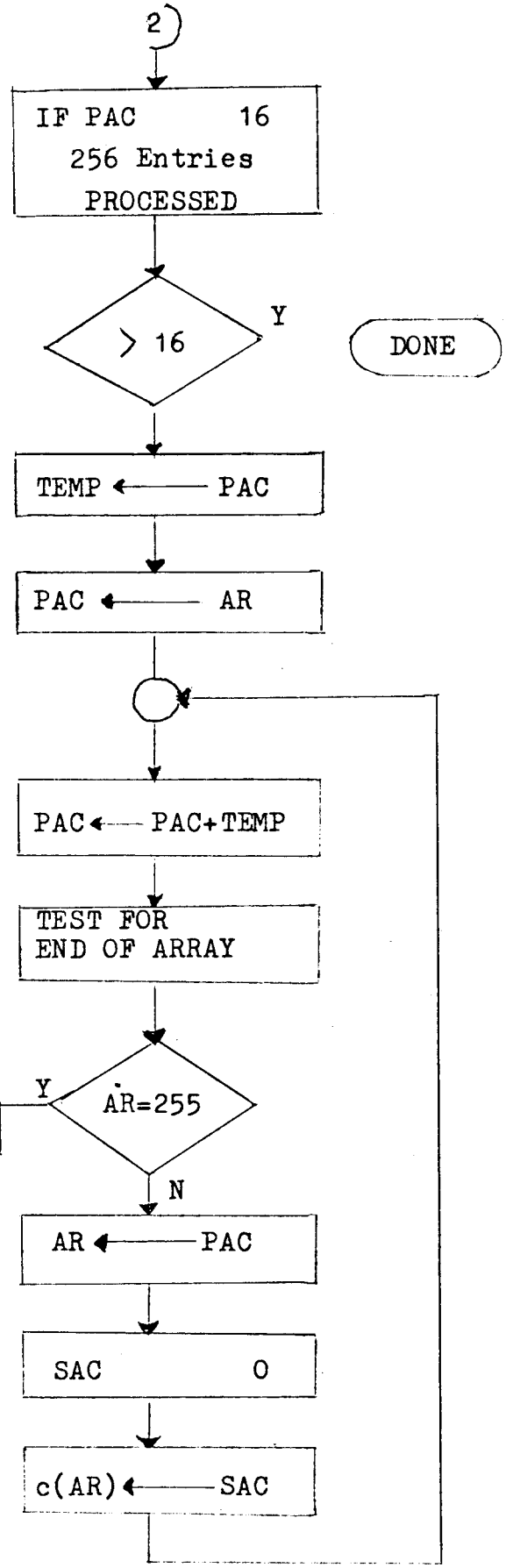
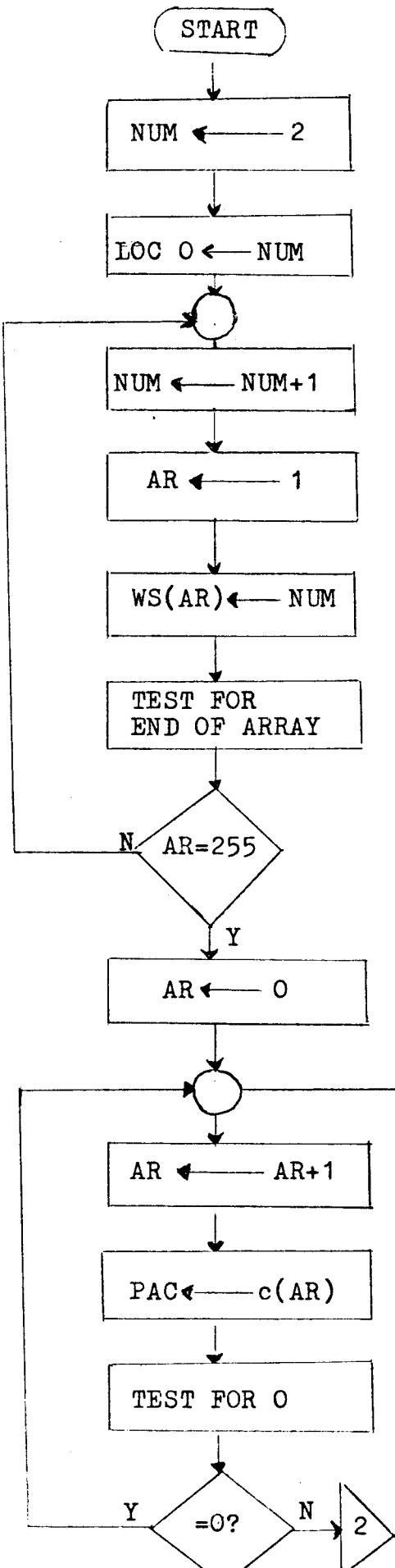
DATA9: BYTE 0, 0, 30, 42, 30, 42, 0, 24
BYTE 0, 24, 12, 45, 12, 45, 12, 45, 30, 54
BYTE 30, 54, 30, 42, 255

I. PRIMAL PRIMER

The attached program shows a relatively simple implementation of an algorithm to derive all prime numbers less than 256. Due to the fact that the Vector Processor is not a register-oriented machine, it becomes rather cumbersome to solve the problem without register-register operations. Consider that on the Intel 8080 the same routine could be coded with half as many statements and memory requirements (no variables need be carried in RAM). Or note the straight forward FORTRAN IV approach:

```
      DIMENSION  IPRIME (128)
      DATA IPRIME (1) /2/, IPRIME (2) /3/
      DO 100  I = 3,127
100  IPRIME (I) = IPRIME (I-1) + 2
      DO 200  I = 2,8
      IF (IPRIME (I). EQ.0) GO TO 200
      INC = IPRIME (I)
      DO 200 IADD = I, 127, INC
200  IPRIME (IADD) = 0
      STOP
```

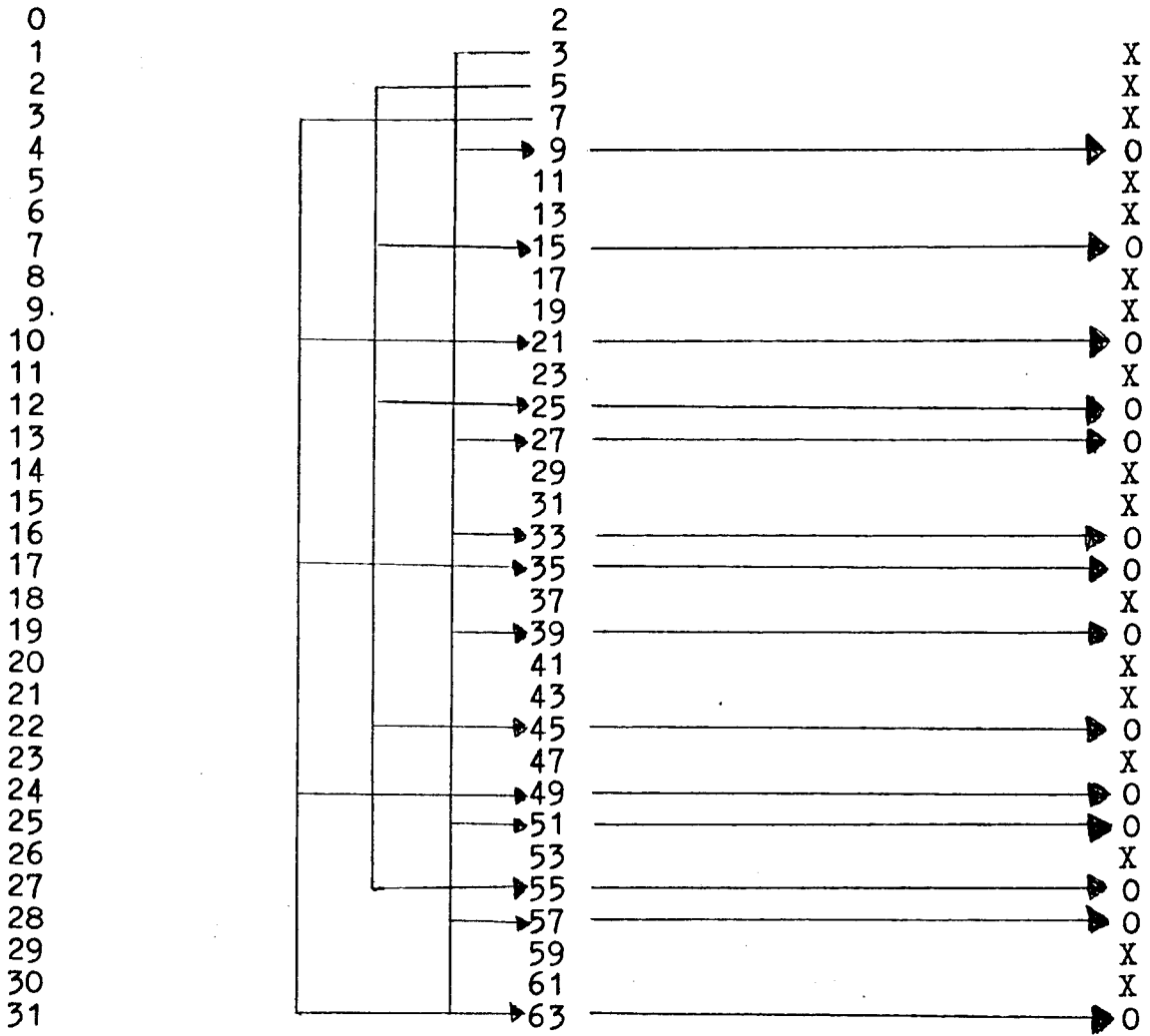
AR: WORKING STORAGE ADDR REG
 PAC: PRIMARY ACC
 SAC: SECONDARY ACC



LOC

VALUE

PRIME



**CALCULATE ALL PRIMES LESS THAN 256

*PRIMES: PAGES 0-7

*PAGE 15 WORKING STORAGE

WS	EQU	0
ZERO	EQU	1
N16	EQU	2
TEMP	EQU	3
N255	EQU	4

*PROGRAM BEGINS EXECUTION HERE

```
.ORG      0
SETP     15
```

*INITIALIZATION

```
TLD      2
STA      0      K      ;FIRST PRIME IS 2
LDAI     0
STA      WS     ;WS ADDR PTR
STA      ZERO
LDAI     1
STA      N255
SSA
TLD      1      ;CARRY BASE OF FIRST ODD#
```

← init N16

*FILL ARRAY WITH ODD NUMBERS

```
FILL:    LPAI    PSLEEV
LDA      WS     ;INCREMENT ADDR PTR
A41      1
STAP
```

```

TST          N255          ;TEST FOR FULL ARRAY
JLT          ;( 128 ENTRIES)
SSA
A4I          2             ;GENERATE NEXT ODD #
WS          WS             ;LOAD WS ADDR REG
SSA
STAP        ;STORE # IN ARRAY
LPAI        FILL
JMP

```

*ZERO OUT NON-PRIMES BY FINDING MULTIPLES

```

PSLEEV: LDAI          0
STA          WS          ;RESET WS ADDR PTR
LPAI        SIEVE
SIEVE: LDA          WS
A4I          1           ;UPDATE ADDR PTR
STAP
WS          WS          ;LOAD WS ADDR REG
LDAP        ;GET ENTRY FROM TABLE
TST          ZERO
JEQ          ;IF ZERO, IT WASN'T PRIME
LPAI        DONE
TST          N16
JLT          ;IF OVER 16 WE'RE DONE
LPAI        SIEVE
STA          TEMP        ;SAVE CURRENT VALUE
LDA          WS

```

IV. BASIC INSTRUCTIONS FOR CREATING AND EXECUTING A PROGRAM
USING THE DEVELOPMENT SYSTEM

1. Turn on CRT terminal.
2. Turn on LSI11 CPU (rear switch)
3. Place bootable floppy disc into left drive.
4. Place work disc in right drive.
5. Ensure front 2 left switches in ON position.
6. When S appears on screen, enter DX (CR)
7. When primary selection menu appears, enter EDIT function (E CR)
8. Subnote on EDIT menu will state: "IF NO ASSIGNED FILE, ENTER CR" (File will open to SYSTEM. WRK.TEXT (default)
9. Select INSERT mode (I CR)
10. Enter source program using various available editing features as listed on menu. If mistake is made in entering source file text, use CONTROL C (CTLC) to get to level of (re) selecting desired function, then return to INSERT mode (I CR)
11. Enter QUIT function (Q CR) when completed.
12. Then, three (3) choices will be displayed:
 - E (Exit File) :No Changes
 - U (Update File) :Current File
 - R (Return to Editor) :No Changes
13. Select update function (U CR)
14. System will go back to primary command menu.
15. to assemble program, enter A CR CR
Enter (PRINTER : CR) if output is desired
(PRINTER MUST BE ON)

17. At DEBUG? (at PASCAL level) enter (Y CR) if desired. (CR) if not desired. Note: if both PRINTER and DEBUG options are selected, output listing will be inordinately long (length)
18. After assembly has completed, (with no errors) system displays primary command menu.
19. To execute program ie. translate PASCAL file to object code, enter (X CR).
20. System will then ask : File?
Default is NEWROMEMU
21. At the next menu, select LOAD function (L CR)- brings file into buffer.
22. When system asks for name of current file, enter:
(SYSTEM. WRK CR)
23. Ensure EMULATOR power on (main power also)
24. When next response from system appears enter (D CR) for DOWN-LOAD function (causes object file to be loaded into ROM and executed.
25. ROM can be modified using EMULATOR edit functions through terminal keyboard.