SOFTWARE FOR A COMPUTER
BASED VIDEO SYNTHESIZER

WALTER WRIGHT

JULY 1977

EXPERIMENTAL TELEVISION CENTER LTD
164 COURT ST, BINGHAMTON
NEW YORK, 13901
607 723-9585

SUPPORTED BY NYSCA AND NEA
INDEX

CHAPTER 1 - INTRODUCTION AND PROGRAM GOALS

CHAPTER 2 - DESCRIPTION OF MAIN PROGRAM

2.1 HARDWARE CONFIGURATION

2.2 INITIALIZATION

2.2.1GLOBALS AND SYSTEM MACROS

2.2.2 DIGITAL TO ANALOG CONVERTERS

2.2.3 BUFFER MEMORY

2.2.4 DATA BUFFER CONTROL

2.3 TIMING ROUTINE

2.3.1 INTERRUPT SERVICING

2.3.2 POLLING THE DATA BUFFERS

2.4 THE INTERPRETER

2.4.1 SUBROUTINE CROSS-REFERENCING

2.5 TIMING CONTROL SUBROUTINES

2.5.1 SET THE TIMING INTERVAL

2.5.2 ADD TO THE TIMING INTERVAL

2.5.3 SUBTRACT FROM THE TIMING INTERVAL

2.5.4 COMPLEMENT THE TIMING INTERVAL

2.5.5 SHIFT THE TIMING INTERVAL RIGHT

2.5.6 SHIFT THE TIMING INTERVAL LEFT

2.6 DATA OUT SUBROUTINES

2.6.1 SET THE DATA WORD

2.6.2 INCREMENT THE DATA WORD

2.6.3 DECREMENT THE DATA WORD

2.6.4 ADD TO THE DATA WORD

2.6.5 SUBTRACT FROM THE DATA WORD

2.6.6 COMPLEMENT THE DATA WORD

2.6.7 SHIFT THE DATA WORD RIGHT

2.6.8 SHIFT THE DATA WORD LEFT

2.6.9 ROTATE THE DATA WORD RIGHT

2.6.10 ROTATE THE DATA WORD LEFT

2.6.11 BIT CLEAR WITH DATA WORD

2.6.12 BIT SET WITH DATA WORD

2.6.13 XOR WITH DATA WORD
2.7 DATA IN SUBROUTINES
2.7.1 INPUT DATA WORD 23
2.7.2 ADD INPUT TO DATA WORD 23
2.7.3 SUBTRACT INPUT FROM DATA WORD 23
2.7.4 BIT CLEAR INPUT WITH DATA WORD 24
2.7.5 BIT SET INPUT WITH DATA WORD 24
2.7.6 XOR INPUT WITH DATA WORD 24

2.8 BUFFER CONTROL SUBROUTINES 25
2.8.1 LOOP ROUTINE 25

2.9 PROGRAM CONTROL SUBROUTINES 26
2.9.1 INPUT ROUTINE 26
2.9.2 OUTPUT ROUTINE 32
2.9.3 EXIT ROUTINE 32

CHAPTER 3 - DESCRIPTION OF DATA BUFFERS 33

3.1 SOFTWARE CONFIGURATION 33

3.2 TABLES 35

3.3 THE ENABLE BUFFER 36

3.4 DATA BUFFERS 37

CHAPTER 4 - PROGRAMMING TECHNIQUES 39

4.1 CREATING TABLES 39
4.1.1 A SAMPLE TABLE 39

4.2 CREATING DATA BUFFERS 41
4.2.1 A SAMPLE BUFFER 41

4.3 PROGRAMMING THE DIGITAL TO ANALOG CONVERTERS 42
4.3.1 PROTOCOL 42
4.3.2 A SIMPLE RAMP 42
4.3.3 A REPEATING SAWTOOTH 44
4.3.4 A REPEATING TRIANGLE 46
4.3.5 MAKING A SINE WAVE 48
CHAPTER I - INTRODUCTION AND PROGRAM GOALS

THE SOFTWARE FOR THE EXPERIMENTAL TELEVISION CENTER COMPUTER BASED VIDEO SYNTHESIZER IS DESIGNED TO SATISFY THE FOLLOWING CRITERIA.

FIRSTLY, THE SOFTWARE IS CONCERNED WITH GRAPHIC DESIGN AND COMPOSITION.

SECONDLY, THE SOFTWARE WILL BE ABLE TO ANALYZE AND SYNTHESIZE IMAGES.

AND FINALLY, THE SOFTWARE PROGRAM WILL REPROGRAM ITSELF IN RESPONSE TO EXTERNAL STIMULAE. IN ORDER TO MEET THESE CRITERIA THE PROGRAM MUST BE REAL-TIME AND INTERACTIVE. THE ARTIST WILL CREATE IMAGES AND SEQUENCES OF IMAGES IN DIALOGUE WITH THE PROGRAM.

IN WRITING THE SOFTWARE I HAVE WORKED FROM THESE DEFINITIONS.

THE VIDEO SYNTHESIZER IS A GROUP OF PROGRAMMABLE MODULES FOR CREATING IMAGES. THE COMPUTER PROGRAMS THE MODULES COMPRISING THE SYNTHESIZER.

THE IMAGE CONTAINS BOTH TEMPORAL AND SPATIAL INFORMATION WHICH CONCERNS THE ARTIST AND THE PROGRAMMER. THE IMAGE IS RESURRECTED EVERY FIELD (1/60 SEC) AND THIS BECOMES THE TIME-BASE FOR THE PROGRAM. NEW CONTROL PARAMETERS ARE TRANSFERRED TO THE SYNTHESIZER MODULES EVERY FIELD.

A COMMON MISTAKE IN DEVELOPING NEW PROGRAMS IS TO BORROW FROM AND TO IMITATE RELATED MEDIA SUCH AS ELECTRONIC MUSIC. I AM INCLUDING IN THE PROGRAM COMMANDS TO EFFECT THE ELEMENTS AND ATTRIBUTES OF GRAPHIC DESIGN SUCH AS:

1. CREATING POINTS, LINES AND BASIC SHAPES
2. CREATING TEXTURES
3. DEFINING AREAS AND BOUNDARIES
4. DEFINING OBJECT/FIELD RELATIONSHIPS
5. CONTROLLING VALUE, LUMINENCE AND CONTRAST
6. CONTROLLING CHROMA, SATURATION AND HUE
7. CREATING SEQUENCES OF IMAGES, TIMING PATTERNS
8. CONTROLLING DENSITY
9. CONTROLLING BALANCE AND SYMMETRY
10. CONTROLLING DEPTH, SCALE AND PROPORTION
11. CREATING FOCAL POINTS
12. CREATING HARMONY, RHYTHM AND COUNTERPOINT
13. CREATING MOTION: TRANSLATION, ROTATION, WARPS, ETC.

THIS PECULIAR APPROACH TO DESIGNING SOFTWARE IS NECESSARY IN ORDER TO DEVELOP A PROGRAM USEFUL TO THE ARTIST; A PROGRAM THAT SPEAKS THE ARTIST'S LANGUAGE. THE TASK IS NOT AS HOPELESS AS IT APPEARS; THE SOFTWARE DESCRIBED SO FAR RUNS ON HIGH SCHOOL MATHEMATICS. IT DEPENDS ON THE DEVELOPMENT OF SPECIALIZED HARDWARE TO CONTROL VARIOUS ASPECTS OF THE IMAGING PROCESS AND TO ANALYZE REAL AND PRERECORDED IMAGES.

PAGE 1
USING SPECIAL PROGRAMS AND PROGRAMMING TECHNIQUES, THE COMPUTER WILL BE ENDOWED WITH A MINIMAL I.Q. ON THE ARTIFICIAL INTELLIGENCE SCALE. THE COMPUTER WILL NOT RESPOND IN A TOTALLY PREDICTABLE WAY. THE DEGREE OF UNPREDICTABILITY IS DETERMINED BY THE ARTIST.

ENCLOSED WITH THIS REPORT IS A FIRST ATTEMPT AT A PROGRAM OF THIS TYPE. THE IMAGING PROCESS IS CONTROLLED EITHER NUMERICALLY AS IN DON McARTHUR'S XY GENERATOR, OR WITH DIGITAL TO ANALOG CONVERTERS. IMAGES ARE ANALYZED USING THE ANALOG TO DIGITAL CONVERTERS. FINALLY, THE ARTIST AND THE COMPUTER CONVERSE USING THE TELETYPE AND THE REAL-TIME INTERFACE.

THE PROGRAM POLLS A SET OF DATA BUFFERS (RESERVED AREAS OF COMPUTER MEMORY) EVERY FIELD. EACH DATA BUFFER CONTROLS A PARTICULAR HARDWARE MODULE. THE DATA IN THE BUFFERS IS TIME DEPENDENT ALLOWING FOR THE CREATION OF COMPLEX TIMING PATTERNS USING THE FIELD AS THE BASIC TIME UNIT.

AT PRESENT ONLY THE SIMPLEST CONTROL PARAMETERS ARE PROGRAMMED. I AM MODIFYING THE PROGRAM TO ACCEPT TELETYPewriter INPUT IN REAL-TIME. THIS WILL ALLOW THE ARTIST TO TALK TO THE PROGRAM AND TO SYNTHESIZE AND MODIFY IMAGES AS THEY ARE BEING GENERATED.
2.1 HARDWARE CONFIGURATION

**THE FIRST PROGRAM WAS DEVELOPED FOR WOODY VASULKA WHO USES AN LSI-11 MICROCOMPUTER INTERFACED TO VIDEO SYNTHESIS MODULES INCLUDING DIGITAL TO ANALOG CONVERTERS (D/A'S), ANALOG TO DIGITAL CONVERTORS (A/D'S). DON MCARTHUR'S MODULES DESCRIBED ELSEWHERE IN THIS REPORT, JEFF SCHIER'S ALU MODULES AND GEORGE BROWN'S MULTIPLE LEVEL KEYER. THE D/A'S AND A/D'S ARE CONTROLLED THROUGH FOUR WORDS IN MEMORY AS FOLLOWS:**

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>LEWSTA STATUS WORD</td>
<td>167778</td>
</tr>
<tr>
<td>2.</td>
<td>LEWOUT OUTPUT WORD</td>
<td>167772</td>
</tr>
<tr>
<td>3.</td>
<td>LEWIN INPUT WORD</td>
<td>167774</td>
</tr>
<tr>
<td>4.</td>
<td>LEWCHA CHANNEL ADDRESS</td>
<td>167776</td>
</tr>
</tbody>
</table>

**MCARTHUR'S MODULES ARE CONTROLLED THROUGH THE BUFFER MEMORY WHICH APPEARS AS NORMAL MEMORY TO THE PROGRAM. ANY LOCATION IN BUFFER MEMORY CAN BE READ IN OR WRITTEN TO, AND ARITHMETIC AND LOGIC OPERATIONS CAN BE PERFORMED THEREUPON. THIS TECHNIQUE OF "MEMORY-MAPPED I/O" MAKES THE PROGRAMMER'S LIFE MUCH EASIER AND BECAUSE ALL MODULES MUST BE UPDATED IN LESS THAN 1/60 SEC. CONTROL WORDS FOR MCARTHUR'S AND SCHIER'S MODULES ARE LOCATED IN THE UPPER REACHES OF MEMORY AS FOLLOWS:**

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>DONOUT RED 16:1 SELECT</td>
<td>171047</td>
</tr>
<tr>
<td>2.</td>
<td>DONOUT+2 GREEN 16:1 SELECT</td>
<td>171042</td>
</tr>
<tr>
<td>3.</td>
<td>DONOUT+4 BLUE 16:1 SELECT</td>
<td>171044</td>
</tr>
<tr>
<td>4.</td>
<td>DONOUT+6 INVERSION REGISTER</td>
<td>171046</td>
</tr>
<tr>
<td>5.</td>
<td>LEWS LED DISPLAY</td>
<td>171512</td>
</tr>
<tr>
<td>6.</td>
<td>DONIN REAL TIME INPUT</td>
<td>171622</td>
</tr>
<tr>
<td>7.</td>
<td>DONSTA STATUS REGISTER</td>
<td>171776</td>
</tr>
<tr>
<td>8.</td>
<td>JEFOUT RED ALU</td>
<td>171112</td>
</tr>
<tr>
<td>9.</td>
<td>JEFOUT+2 GREEN ALU</td>
<td>171112</td>
</tr>
<tr>
<td>10.</td>
<td>JEFOUT+4 BLUE ALU</td>
<td>171114</td>
</tr>
</tbody>
</table>
2.2 INITIALIZATION

GLOBALS AND SYSTEM MACROS

The first step in the program is to initialize the modules one by one setting each to its normal default condition. However there's a little housekeeping to be done. The tables and data buffers are declared as global variables which allows them to be assembled separately from the main program. This is done with the following statement:

```
GLOBL TABLES, EBUF, DBUF
```

More about these tables and data buffers in Chapter 3. Next the system macros are invoked with the following statements:

1) BEGIN: MCALL \*V2 \*REGDEF \*EXIT

2) \*V2

3) \*REGDEF

The label BEGIN is used by the linking loader to identify the entry point to the main program. This is done using this statement at the end of the program:

```
\*END BEGIN
```

The \*V2 macro identifies the monitor system used by the LSI-

II. The \*REGDEF macro defines the LSI-II's internal registers using two character mnemonics as follows:

1. R0 GENERAL PURPOSE REGISTER 0
2. R1 GENERAL PURPOSE REGISTER 1
3. R2 GENERAL PURPOSE REGISTER 2
4. R3 GENERAL PURPOSE REGISTER 3
5. R4 GENERAL PURPOSE REGISTER 4
6. R5 GENERAL PURPOSE REGISTER 5
7. SP STACK POINTER REGISTER 6
8. PC PROGRAM COUNTER REGISTER 7
2.2.2 DIGITAL TO ANALOG CONVERTERS

PENDICISS--------------------

NOW WE'RE READY TO INITIALIZE THE D/A'S WHICH IS ACCOMPLISHED

THUS:

1) MOV $1000000LEWOUT
2) MOV $10,R0
3) BGN1: DEC R0
4) MOV R0,0 #LEWCHA
5) TST R0
6) BEQ BGN1

THE FIRST LINE OF CODE MOVES THE OCTAL NUMBER 100000 TO THE
OUTPUT WORD IN MEMORY WHICH CONTROLS THE D/A'S. THIS CAUSES THE D/A
TO OUTPUT A CONSTANT ZERO VOLTS ( +10V = 17777777 AND -10V = 0 ). THE
PREFIX $ DEFINES A REAL NUMBER, AND THE PREFIX # DEFINES A LOCATION
IN MEMORY. HOWEVER THE DATA TRANSFER IS NOT CONSUMMATED UNTIL THE D/A
CHANNEL IS ADDRESSED THROUGH THE CHANNEL ADDRESS WORD. THERE ARE 8 D/A
CHANNELS NUMBERED 0-7. THEREFORE WE SET REGISTER 2 EQUAL TO 8, OR OCTAL
10 (LINE 2). THEN WE COUNT DOWN REGISTER 2 WITH A LOOP (LINES 3, 5 AND
6) AND AT THE SAME TIME ENABLE THE D/A'S BY MOVING THE CONTENTS OF
REGISTER 2 TO THE CHANNEL ADDRESS WORD (LINE 4).

PAGE 5
2.2.3 BUFFER MEMORY

***************

AND WE INITIALIZE THE BUFFER MEMORY AS FOLLOWS:

1) MOV #DONOUT,RZ
2) CLR (RZ)+
3) CLR (RZ)+
4) CLR (RZ)+
5) CLR (RZ)+
6) MOV #DEFOUT,RZ
7) CLR (RZ)+
8) CLR (RZ)+
9) CLR (RZ)+

THIS CODE USES THE AUTO-INCREMENT MODE OF ADDRESSING (R)+.
LINE 1 MOVES #DONOUT (171740) INTO REGISTER R. THEN WE CLEAR THAT
MEMORY LOCATION AND ADD +2 TO REGISTER R WHICH NOW POINTS TO THE NEXT
WORD IN MEMORY (LINES 2-5). THIS SETS THE RED, GREEN AND BLUE 16:1
SELECT CHANNELS TO BLACK AND THE INVERSION REGISTER TO NORMAL OR NON-
INVERTING. SIMILARLY THE ALU'S ARE SET TO PASS RED, GREEN AND BLUE
RESPECTIVELY (LINES 6-9).
2.2.4 DATA BUFFER CONTROL

THE MAXIMUM NUMBER OF DATA BUFFERS IS SET:

```
MOV B 128, TMRY
```

THAT IS, THE PROGRAM TOLERATES NO GREATER THAN 16 BUFFERS (OCTAL 2*). THIS FACT IS RECORDED IN THE BYTE LABELLED TMRY.

EACH DATA BUFFER IS ASSOCIATED WITH FOUR PARAMETER WORDS AND THESE 64 WORDS (4*64) ARE KEPT IN THE PARAMETER BUFFER PBUF. WE INITIALIZE THIS BUFFER AS FOLLOWS:

1)  MOV #PBUF, R0
2)  SUB #12, R0
3)  BGN2: CMPB TMRX, TMRY
4)  BPL TMR
5)  INCB TMRX
6)  ADD #12, R0
7)  CLR (R0)
8)  MOV #1-2(R0)
9)  MOV B TMRX, R1
10) DEC R1
11) SWAB R1
12) ADD #DBUF, R1
13) MOV R1, 4(R0)
14) CLR 6(R0)

PAGE 7
Again we use a loop; we set register 0 to the location of PBUF (lines 1 and 2). Note PBUF is created by causing the program counter (•) to skip over 64 words of memory (line 16). The loop is controlled by TMRX and TMRY. TMRX counts up to the maximum number of data buffers; then a branch to the next block of code is executed (lines 3, 4, 5 and 15). The four parameter words are:

1. Timing counter
2. Timing interval
3. Pointer to DBUF
4. Data word

The first word is cleared (line 7). The timing interval is set to a single field (line 8). Next address of the data buffer is calculated and put in the third word (lines 9-13). There are 16 data buffers each containing 128 words. Therefore the pointer is set initially as follows:

\[ \text{POINTER} = (256 \times \text{TMRX}) + \text{DBUF} \]

This formula is coded from right to left. In line 9 TMRX is moved into register 1; the decrement instruction in line 16 subtracts 1 from the register; the swap byte instruction in line 11 effectively multiplies the register by 256 (equivalent to 8 left shifts); DBUF is added to register 1 in line 12 and finally in line 13 the result is stored in the parameter buffer using the indexed addressing mode \( \text{X(R)} \) the contents of the register plus the index produce the effective address.
2.3 TIMING ROUTINE

***************

2.3.1 INTERRUPT SERVICING

***************

FROM HERE WE GO TO THE TIMING ROUTINE (TMR). THIS ROUTINE ENABLES THE 1/60 SEC INTERRUPT AND EVERY 1/60 SEC POLLS THE PARAMETER BUFFER CHECKING FOR TIME OUTS (TIMING COUNTER EQUAL TIMING INTERVAL). IF A DATA BUFFER TIMES OUT A BRANCH TO THE NEXT BLOCK OF CODE IS EXECUTED.

THE BUFFER MEMORY TRANSFERS DATA TO THE MODULES DURING THE VERTICAL INTERVAL BETWEEN EACH FIELD OF VIDEO. THEN THE BUFFER MEMORY GENERATES AN INTERRUPT TELLING THE COMPUTER TO GET WORKING ON DATA FOR THE NEXT FIELD. THIS INTERRUPT IS ENABLED OR DISABLED WITH THE STATUS WORD DONSTA. IF THE STATUS WORD EQUALS 1 THE INTERRUPT IS ENABLED; IF 0 THE INTERRUPT IS DISABLED. SO MUCH FOR THE BUFFER MEMORY; THE LSI-11 HANDLES INTERRUPTS THUS. THE COMPUTER INTERRUPTS ITS NORMAL FLOW OF OPERATIONS AND AS A PRECAUTION PUSHES THE CURRENT PROGRAM COUNTER (PC OR REGISTER 6) AND THE PROGRAM STATUS WORD (PSW) ONTO THE STACK. THE STACK POINTER (SP) IS DECREMENTED BY 4. THEN THE COMPUTER GOES TO A PREDETERMINED LOCATION IN MEMORY (IN THIS CASE LOCATION 170) AND USES THE CONTENTS AS THE NEW PROGRAM COUNTER (PC). EXECUTION BEGINS ANEW FROM THE LOCATION POINTED TO BY #170. USUALLY THIS IS AN INTERRUPT SERVICE ROUTINE, HOWEVER I HAVE TAKEN A SHORTCUT AS EXPLAINED BELOW.

1) TMR: MOV #TMR,#170
2) CLR6 TMRX
3) INC #DONSTA
4) BR .
5) TMRI: CLR #DONSTA
6) ADD 4, SP

PAGE 5
IN LINE 1 WE PREPARE FOR THE INEVITABLE INTERRUPT BY LOADING LOCATION 176 WITH THE LOCATION #TMRI; THE LOCATION WHERE WE WILL RESUME EXECUTION. NEXT THE BUFFER COUNTER ( TMRX ) IS CLEARED AND THE INTERRUPT IS ENABLED ( LINES 2 AND 3 ). WE WAIT FOR THE INTERRUPT BY EXECUTING THE BRANCH INSTRUCTION ON LINE 4. FOLLOWING THE INTERRUPT WE RETURN TO LINE 5 AND DISABLE FURTHER INTERRUPTS BY CLEARING THE STATUS WORD IN THE BUFFER MEMORY. THEN IN LINE 6 WE DO SOME HOUSEKEEPING, RESTORING THE STACK POINTER ( SP ).
2.3.2 POLLING THE DATA BUFFERS

WE ARE NOW READY TO POLL THE DATA BUFFERS:

1) MOV #PBUR, R0
2) SUB #I8, R0
3) TMR2: CMPB TMRX, TMRY
4) BPL TMR
5) INCB TMRX
6) ADD #I8, R0
7) MOVB TMRX, R2
8) DEC R2
9) ADD #EBUF, R2
10) TSTB (R2)
11) BEQ TMR2
12) INC (R2)
13) CMP (R2)
14) BLE TMR2
15) TMR3: CLR (R2)
16) JSR PC, INT
17) BR TMR2
18) TMRX: .BYTE 0
19) TMRY: .BYTE 0

PAGE 11
AGAIN WE HAVE A LOOP SIMILAR TO THE LOOP USED TO INITIALIZE THE PARAMETER BUFFER. LINES 1 AND 2 LOAD REGISTER $0$ WITH #PBUF-8. IN LINE 3 THE COUNTER TMRX (INITIALLY $0$) AND THE NUMBER OF BUFFERS TMRY ARE COMPARED. ASSUMING ALL THE BUFFERS WERE CHECKED WE BRANCH BACK TO WAIT FOR THE NEXT INTERRUPT (LINE 4). OTHERWISE WE INCREMENT REGISTER $0$ BY 8 (LINE 6) AND CHECK THE ENABLE BUFFER (LINES 7 TO 10). IF THE BUFFER IS DISABLED (THE CONTENTS OF LOCATION #EBUF*(TMRX-1) EQUAL 0) WE BRANCH BACK TO TMR2 (LINE 11). IF THE BUFFER IS ENABLED THE TIMING COUNTER IS INCREMENTED (LINE 12) AND COMPARED WITH THE TIMING INTERVAL (LINE 13). IF THE COUNTER IS LESS THAN OR EQUAL TO THE INTERVAL WE BRANCH BACK TO TMR2 (LINE 14). OTHERWISE WE CLEAR THE TIMING COUNTER AND JUMP TO THE INTERPRETER ROUTINE (LINES 15 AND 16). UPON RETURNING FROM THE INTERPRETER (LINE 17) WE BRANCH BACK TO TMR2 COMPLETING THE TIMING ROUTINE. LINES 18 AND 19 RESERVE SPACE IN MEMORY FOR THE BUFFER COUNTER TMRX AND THE NUMBER OF BUFFERS TMRY.
2.4 THE INTERPRETER

*********************************

2.4.1 SUBROUTINE CROSS-REFERENCING

*********************************

The interpreter reads a command word from the data buffer and uses this word to create a special jump subroutine instruction. The subroutine in turn executes the command reading additional data words from the buffer as required.

1) INT: MOV 4(R2),R1
2) MOV (R1)+,R2
3) ASL R2
4) ADD #JBUF,R2
5) MOV (R2),R2
6) SUB #INT1,R2
7) MOV R2,INT1-2
8) CLR R5
9) JSR PC,EXIT
10) INT1: MOV R1,4(R2)
11) TST R5
12) BEQ INT
13) RTS PC

\[
\text{INDEX} = \text{SUBROUTINE ENTRY PT} - \text{INTI}
\]

\[
\text{SUBROUTINE ENTRY PT} = \text{JBUF} + (3 \times \text{COMMAND WORD})
\]


A CROSS-REFERENCE TABLE JBUF FOLLOWS THE INTERPRETER. THE ENTRY POINTS FOR THE SUBROUTINES ARE STORED SEQUENTIALLY AND ARE ACCESSED WITH THE COMMAND WORD.
2.5 TIMING CONTROL SUBROUTINES

2.5.1 SET THE TIMING INTERVAL

COMMAND WORD V2 SETS THE TIMING INTERVAL (SECOND WORD ON THE PARAMETER LIST) EQUAL TO THE NEXT WORD IN THE BUFFER.

1) SUBV2: MOV (R1)+,2(R2)
2) RTS PC

2.5.2 ADD TO THE TIMING INTERVAL

COMMAND WORD V1 ADDS THE NEXT WORD IN THE DATA BUFFER TO THE TIMING INTERVAL.

1) SUBV1: ADD (R1)+,2(R2)
2) RTS PC

2.5.3 SUBTRACT FROM THE TIMING INTERVAL

COMMAND WORD V2 SUBTRACTS THE NEXT WORD IN THE DATA BUFFER FROM THE TIMING INTERVAL.

1) SUBV2: SUB (R1)+,2(R2)
2) RTS PC

2.5.4 COMPLEMENT THE TIMING INTERVAL

COMMAND WORD V3 COMPLEMENTS THE TIMING INTERVAL, EQUIVALENT TO 177777 - TIMING INTERVAL.

1) SUBV3: COM 2(R2)
2) RTS PC
2.5.5 SHIFT THE TIMING INTERVAL RIGHT

**COMMAND WORD 74 SHIFTS THE TIMING INTERVAL TO THE RIGHT. THE MOST SIGNIFICANT BIT (BIT 15) IS CLEARED, EQUIVALENT TO TIMING INTERVAL/2.**

1) SUBQ4: CLC
2) ROR 2(R0)
3) RTS PC

2.5.6 SHIFT THE TIMING INTERVAL LEFT

**COMMAND WORD 75 SHIFTS THE TIMING INTERVAL TO THE LEFT. THE LEAST SIGNIFICANT BIT (BIT 2') IS CLEARED, EQUIVALENT TO 2\(^*\) TIMING INTERVAL.**

1) SUBQ5: CLC
2) ROL 2(R0)
3) RTS PC

COMMAND WORDS 26 AND 27 ARE NOT USED, THEREFORE THEY ARE CROSS-REFERENCED TO THE ERROR ROUTINE ERR IN JBUF.
2.6 DATA OUT SUBROUTINES

2.6.1 SET THE DATA WORD

COMMAND WORD 10 SETS THE DATA WORD (FOURTH WORD IN THE PARAMETER LIST) EQUAL TO THE NEXT WORD IN THE DATA BUFFER.

1) SUB10: MOV (R1)+, 6(R0)
2) RTS PC

2.6.2 INCREMENT THE DATA WORD

COMMAND WORD 11 INCREMENTS THE DATA WORD, EQUIVALENT TO DATA WORD+1.

1) SUB11: INC 6(R0)
2) RTS PC

2.6.3 DECREMENT THE DATA WORD

COMMAND WORD 12 DECREMENTS THE DATA WORD, EQUIVALENT TO DATA WORD-1.

1) SUB12: DEC 6(R0)
2) RTS PC
2.6.4 ADD TO THE DATA WORD

COMMAND WORD 13 ADDS THE NEXT WORD IN THE DATA BUFFER TO THE DATA WORD.

1) SUB13: ADD (R1)+,6(R0)
2) RTS PC

2.6.5 SUBTRACT FROM THE DATA WORD

COMMAND WORD 14 SUBTRACTS THE NEXT WORD IN THE DATA BUFFER FROM THE DATA WORD.

1) SUB14: SUB (R1)+,6(R0)
2) RTS PC

2.6.6 COMPLEMENT THE DATA WORD

COMMAND WORD 15 COMPLEMENTS THE DATA WORD, EQUIVALENT TO 177777-DATA WORD.

1) SUB15: COM 6(R0)
2) RTS PC
2.6.7 SHIFT THE DATA WORD RIGHT

************************************************

COMMAND WORD 16 SHIFTS THE DATA WORD TO THE RIGHT, THE MOST SIGNIFICANT BIT (BIT 15) IS CLEARED, EQUIVALENT TO DATA WORD/2.

BIT N BECOMES BIT N-1

\[
\begin{array}{c}
\text{15} \\
\hline \\
\text{15} \\
\end{array}
\]

\[\text{\textbf{--> BIT 0 DROPPED}}\]

1) \text{SUB16: CLC}
2) \text{ROR 6(R0)}
3) \text{RTS PC}

2.6.8 SHIFT THE DATA WORD LEFT

************************************************

COMMAND WORD 17 SHIFTS THE DATA WORD TO THE LEFT, THE LEAST SIGNIFICANT BIT (BIT 0) IS CLEARED, EQUIVALENT TO 2* DATA WORD.

BIT N BECOMES BIT N+1

\[
\begin{array}{c}
\text{15} \\
\hline \\
\text{15} \\
\end{array}
\]

\[\text{\textbf{BIT 15 DROPPED}}\]

1) \text{SUB17: CLC}
2) \text{ROL 6(R0)}
3) \text{RTS PC}

---

PAGE 15
2.6.9 ROTATE THE DATA WORD RIGHT

COMMAND WORD 20 ROTATES THE DATA WORD TO THE RIGHT, SHIFTS THE BITS RIGHT AND THE LEAST SIGNIFICANT BIT ( BIT 0 ) IS ROTATED AROUND TO BECOME THE MOST SIGNIFICANT BIT ( BIT 15 ).

\[ 15 \rightarrow 0 \]

\[ \rightarrow \text{BIT 0 BECOMES BIT 15} \]

1) SUB20: MOV 6(R0).R2
2) ROR R2
3) ROR 6(R0)
4) RTS PC

2.6.10 ROTATE THE DATA WORD LEFT

COMMAND WORD 21 ROTATES THE DATA WORD TO THE LEFT, SHIFTS THE BITS LEFT AND THE MOST SIGNIFICANT BIT ( BIT 15 ) BECOMES THE LEAST SIGNIFICANT BIT ( BIT 0 ).

\[ 15 \leftarrow 0 \]

\[ \leftarrow \text{BIT 15 BECOMES BIT 0} \]

1) SUB21: MOV 6(R0).R2
2) ROL R2
3) ROL 6(R0)
4) RTS PC
2.6.11 BIT CLEAR WITH DATA WORD

********************************************************************************

COMMAND WORD E2 TAKES THE NEXT WORD IN THE DATA BUFFER AND CLEAR EACH BIT IN THE DATA WORD WHICH CORRESPONDS TO A SET BIT IN THE FORMER, EQUIVALENT TO:

DATA WORD = NEXT WORD IN BUFFER DATA WORD

NEXT WORD IN BUFFER       e e e e e e e e e e e e
DATA WORD                 e e e e e e e e e e e e
DATA WORD                 e e e e e e e e e e e e

1)   SUB22:  BIC (R1)+, 6(R2)

2)   RTS  PC

2.6.12 BIT SET WITH DATA WORD

********************************************************************************

COMMAND WORD E3 TAKES THE NEXT WORD IN THE DATA BUFFER AND SETS THE CORRESPONDING BITS IN THE DATA WORD, EQUIVALENT TO:

DATA WORD = NEXT WORD IN BUFFER DATA WORD

NEXT WORD IN BUFFER       e e e e e e e e e e e e
DATA WORD                 e e e e e e e e e e e e
DATA WORD                 e e e e e e e e e e e e

DATA WORD                 e e e e e e e e e e e e

1)   SUB23:  BIS (R1)+, 6(R2)

2)   RTS  PC
2.6.13 XOR WITH DATA WORD

***************

COMMAND WORD 24 TAKES THE NEXT WORD IN THE DATA BUFFER AND EXCLUSIVE OR'S IT WITH THE DATA WORD.

NEXT WORD IN BUFFER: 0 0 0 0 1 0 1 1 0 0
DATA WORD: 0 0 0 0 1 0 1 1 1 0
DATA WORD: 0 0 0 0 1 1 1 1 0 1

1) SUB24: MOV (R1)+,R2
2) XOR R2,6(R0)
3) RTS PC

COMMAND WORDS 25, 26 AND 27 ARE NOT USED. THEREFORE THEY ARE CROSS-REFERENCED TO THE ERROR ROUTINE ERR IN JBUF.
2.7 DATA IN SUBROUTINES

2.7.1 INPUT DATA WORD

COMMAND WORD 30 CALLS THE INPUT ROUTINE AND SETS THE DATA WORD EQUAL TO INPUT DATA (IN REGISTER 2).

1) SUB30: JSR PC, IN
2) MOV R2, 6(R2)
3) RTS PC

2.7.2 ADD INPUT TO DATA WORD

COMMAND WORD 31 CALLS THE INPUT ROUTINE AND ADDS THE INPUT DATA TO THE DATA WORD.

1) SUB31: JSR PC, IN
2) ADD R2, 6(R2)
3) RTS PC

2.7.3 SUBTRACT INPUT FROM DATA WORD

COMMAND WORD 32 CALLS THE INPUT ROUTINE AND SUBTRACTS THE INPUT DATA FROM THE DATA WORD.

1) SUB32: JSR PC, IN
2) SUB R2, 6(R2)
3) RTS PC
2.7.4 BIT CLEAR INPUT WITH DATA WORD

COMMAND WORD 33 CALLS THE INPUT ROUTINE AND CLEARS EACH BIT IN THE DATA WORD AS IN SUB22.

1) SUB33: JSR PC, IN
2) BIC R2, 6(RZ)
3) RTS PC

2.7.5 BIT SET INPUT WITH DATA WORD

COMMAND WORD 34 CALLS THE INPUT ROUTINE AND SETS EACH BIT IN THE DATA WORD AS IN SUB23.

1) SUB34: JSR PC, IN
2) BIS R2, 6(RZ)
3) RTS PC

2.7.6 XOR INPUT WITH DATA WORD

COMMAND WORD 35 CALLS THE INPUT ROUTINE AND EXCLUSIVE OR'S THE INPUT DATA WITH THE DATA WORD AS IN SUB24.

1) SUB35: JSR PC, IN
2) XOR R2, 6(RZ)
3) RTS PC

COMMAND WORDS 36 AND 37 ARE NOT USED. THEREFORE THEY ARE CROSS-REFERENCED TO THE ERROR ROUTINE ERR IN JBUF.
2.8 BUFFER CONTROL SUBROUTINES

2.8.1 LOOP ROUTINE

COMMAND WORD 40, THIS SUBROUTINE USES THE NEXT THREE WORDS IN THE DATA BUFFER TO CREATE A REPEATING LOOP IN THE DATA BUFFER. THE THREE WORDS ARE:

1. A COUNTER, INCREMENTED EACH REPETITION
2. MAXIMUM NUMBER OF REPETITIONS
3. POINTER TO THE TOP OF THE LOOP


1) LOOP: CMP (R1),#2(R1)
2) BPL LOOP1
3) INC (R1)
4) MOV 4(R1),R1
5) RTS PC
6) LOOP1: CLR (R1)
7) ADD #6,R1
8) RTS PC

COMMAND WORDS 41-45 ARE NOT USED, THEREFORE THEY ARE CROSS-REFERENCED TO THE ERROR ROUTINE ERR IN JBUF. THE ERROR ROUTINE IS IN REALITY THE EXIT ROUTINE (SEE SECTION 2.9.3).
2.9 PROGRAM CONTROL SUBROUTINES

************

2.9.1 INPUT ROUTINE

************

THE INPUT SUBROUTINE SERVICES THESE FOURTEEN INPUT DEVICES:

1-8. DATA TABLES DEFINED BY USER
9-12. ANALOG TO DIGITAL CONVERTERS
13. REAL TIME INTERFACE
14. RANDOM NUMBER GENERATOR

THE FIRST PART OF THE INPUT ROUTINE RETRIEVES DATA FROM THE TABLES (INPUT DEVICES 1-8):

1) IN: MOV (R1)+, R2
2) CMP R2, #11
3) BPL IN1
4) MOV (R1)+, R3
5) DEC R2
6) ASL R2
7) ASL R2
8) ASL R2
9) ASL R2
10) DEC R3
11) ASL R3
12) ADD R3, R2

PAGE 26
IN LINE 1 THE INPUT DEVICE NUMBER IS TRANSFERRED FROM THE DATA BUFFER TO REGISTER 1, AND THE BUFFER POINTER INCREMENTED. IF THE DEVICE NUMBER IS GREATER THAN 8 BRANCH TO INI (LINES 2 AND 3), IF NOT MOVE THE TABLE ENTRY NUMBER TO REGISTER 2 AND CALCULATE THE LOCATION OF THE DATA (LINES 4 TO 13) AS FOLLOWS:

LOCATION = $TABLES + 2*(ENTRY NUMBER - 1) + 16*(DEVICE NUMBER - 1)

FINALLY REGISTER 2 TRANSFORMS ITSELF INTO THE REQUESTED DATA (LINE 14) AND WE RETURN TO THE CALLING SUBROUTINE (LINE 15).

THE SECOND PART OF THE INPUT ROUTINE SERVICES THE ANALOG TO DIGITAL CONVERTERS (INPUT DEVICES 5-12):

1) INI: CMP R2,#15
2) SPL IN2
3) SUB #11,R2
4) MOV R2,#LEWINA
5) MOV #LEWIN,R2
6) RTS PC

AGAIN WE TEST THE DEVICE NUMBER. IF GREATER THAN 12 BRANCH TO IN2 (LINES 1 AND 2). THE CHANNEL ADDRESS IS CALCULATED AND MOVED TO THE CONTROL WORD LEWINA (LINES 3 AND 4). THE DATA APPEARS AT THE INPUT WORD LEWIN AND IS TRANSFERRED TO REGISTER 2 (LINE 5). WE RETURN TO THE CALLING SUBROUTINE (LINE 6).
THE THIRD PART OF THE INPUT ROUTINE SERVICES DON McARTHUR’S REAL
TIME INTERFACE ( A REGISTER LOADED FROM THE OUTSIDE WORLD USING TOGGLE
SWITCHES, INPUT DEVICE NUMBER 13 ):  

1)  IN2:  CMP  R2,#16  
2)  BPL  IN3  
3)  MOV  #DONIN,R2  
4)  RTS  PC  

A MODEL OF THE EFFICIENCY OF MEMORY MAPPED I/O, BUT FIRST WE TEST THE DEVICE NUMBER. IF GREATER THAN 13 BRANCH TO IN3 ( LINES 1 AND 2 ). IN A SINGLE LINE OF CODE THE DATA IS TRANSFERRED TO REGISTER 2 ( LINES 3 ) AND WE RETURN TO THE CALLING SUBROUTINE ( LINE 4 ). GOOD WORK DON!  

THE FINAL SECTION OF THE INPUT ROUTINE IS A RANDOM NUMBER GENERATOR OF SORTS ( INPUT DEVICE 14 ):  

1)  IN3:  CMP  R2,#17  
2)  BPL  IN4  
3)  MOV  TEMP,R2  
4)  CLC  
5)  ROL  TEMP  
6)  BCC  RND1  
7)  INC  R2  
8)  RND1:  ROL  TEMP+2  
9)  BCC  RND2  
10) INC  R2  

PAGE 28
TEST THE DEVICE NUMBER. IF GREATER THAN 14 RETURN TO THE CALLING PROGRAM VIA IN4 (LINES 1, 2 AND 20). NOW WE PERFORM A LEFT SHIFT ON TEMP (A GIANT 64 BIT WORD). THIS IS DONE IN FOUR STEPS OF SIXTEEN BITS EACH THROUGH THE CARRY REGISTER (1 BIT).

\[
\begin{array}{cccccccccccc}
64 & 48 & 47 & 32 & 31 & 16 & 15 & 0 \\
+--------+--------+--------+--------+
\end{array}
\]

\[
\begin{array}{cccc}
<TEMP+6 & <TEMP+4 & <TEMP+2 & <TEMP \\
+--------+--------+--------+--------+
\end{array}
\]

\[
\begin{array}{cccc}
C4 & C3 & C2 & C1 \\
\end{array}
\]

\[
\text{TEMP} = \text{TEMP} + (-1) \times (\text{TEMP+C4+G3+G2+G1})
\]

THE INITIAL VALUE OF TEMP IS STORED IN REGISTER 2 AND THE CARRY REGISTER CLEARED (LINES 3 AND 4). NOW THE SHIFTS ARE EXECUTED AND THE RESULTANT CARRYS ADDED TO REGISTER 2 (LINES 5 - 16). WE WRAP IT UP (LINES 17 AND 18), MOVE THE LOW ORDER BITS TO REGISTER 2 (LINE 19), AND RETURN TO WHERE WE CAME FROM (LINE 20). SPACE FOR TEMP IS CREATED WITH THE *WORD MACRO (LINE 21).
2.5.2 OUTPUT ROUTINE

COMMAND WORD 46 - THE OUTPUT SUBROUTINE SERVICES THESE FIFTEEN DEVICES:

1-8. DIGITAL TO ANALOG CONVERTERS
9. RED 16:1 SELECT CHANNELS
10. GREEN 16:1 SELECT CHANNELS
11. BLUE 16:1 SELECT CHANNELS
12. INVERSION REGISTER
13. RED ALU ( ARITHMETIC LOGIC UNIT )
14. GREEN ALU
15. BLUE ALU


1) OUT: CMPB TMRX, #11
2) BPL OUT1
3) MOVB TMRX, R2
4) DEC R2
5) MOV R2, @LEWCHA
6) MOV 6(R2), @LEWOUT
7) INC R5
8) RTS PC

IF THE BUFFER NUMBER IS GREATER THAN 8 BRANCH TO OUT1 ( LINES 1 AND 2 ). IF NOT CALCULATE THE CHANNEL ADDRESS AND MOVE IT TO THE CONTROL WORD LEWCHA ( LINES 3 AND 5 ). NEXT MOVE THE DATA TO THE OUTPUT WORD LEWOUT, SET THE DONE FLAG ( REGISTER 5 ), AND RETURN TO THE CALLING PROGRAM ( LINES 6 - 8 ).
THE SECOND PART OF THE ROUTINE CONTROLS McARTHUR'S 16:1 SELECTS AND INVERSION REGISTER:

1) OUT1: CMPB TMRX,15
2) BPL OUTF
3) MOVB TMRX,R2
4) SUB II,R2
5) ASL R2
6) ADD DOUT,R2
7) MOV 6(RR),(R2)
8) INC R5
9) RTS PC

IF THE BUFFER NUMBER IS GREATER THAN 12 BRANCH TO OUTF (LINES 1 AND 2). IF NOT CALCULATE THE OUTPUT ADDRESS (LINES 3 - 6):

OUTPUT ADDRESS = DONOUT+2*(TMRX-9)

FINALLY WE TRANSFER THE DATA WORD TO THE OUTPUT ADDRESS, SET THE DONE FLAG, AND RETURN TO THE CALLING PROGRAM (LINES 7 - 9).

PART THREE OF THE ROUTINE IS SIMILAR; IT CONTROLS JEFF SCHIER'S ARITHMETIC LOGIC UNITS:

1) OUT2: CMPB TMRX,12
2) BPL OUT3
3) MOVB TMRX,R2
4) SUB II,R2
IF THE BUFFER NUMBER IS GREATER THAN 15, GAME OVER, WE RETURN TO THE CALLING PROGRAM VIA OUT3 (LINES 1, 2, 8 AND 9). IF NOT CALCULATE THE OUTPUT ADDRESS (LINES 3 AND 4):

\[
\text{OUTPUT ADDRESS} = \#\text{JEFOU} + 2 \times (\text{TMRX} - 13)
\]

FINALLY WE OUTPUT THE DATA WORD, SET THE DONE FLAG, AND RETURN (LINES 7 - 9).

2.5.3 EXIT ROUTINE

*******************************************************************************

COMMAND WORD 47 - THIS SUBROUTINE IS INVOKED OVERTLY BY COMMAND WORD 47 AND COVERTLY BY 26, 27, 28, 36, 37, 41, 42, 43, 44 AND 45. IT ENDS THE PROGRAM IN A RELATIVELY PAINLESS MANNER AND RETURNS CONTROL TO THE SYSTEM MONITOR:

1) ERR:

2) EXIT: *EXIT
CHAPTER 3 - DESCRIPTION OF DATA BUFFERS

3.1 SOFTWARE CONFIGURATION

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>LABEL</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>170</td>
<td>INTERRUPT VECTOR</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>BEGIN: INITIALIZATION</td>
<td></td>
</tr>
<tr>
<td>1146</td>
<td>PBUF: CONTROL WORDS FOR DATA BUFFERS</td>
<td></td>
</tr>
<tr>
<td>1346</td>
<td>TMR: TIMING ROUTINE</td>
<td></td>
</tr>
<tr>
<td>1472</td>
<td>INT: INTERPRETER</td>
<td></td>
</tr>
<tr>
<td>1536</td>
<td>JBUF: COMMAND WORD TO SUBROUTINE CROSS-REFERENCE</td>
<td></td>
</tr>
<tr>
<td>1656</td>
<td>SUBEC: TIMING CONTROL SUBROUTINES</td>
<td></td>
</tr>
<tr>
<td>1664</td>
<td>SUBE1:</td>
<td></td>
</tr>
<tr>
<td>1672</td>
<td>SUBE2:</td>
<td></td>
</tr>
<tr>
<td>1707</td>
<td>SUBE3:</td>
<td></td>
</tr>
<tr>
<td>1706</td>
<td>SUBE4:</td>
<td></td>
</tr>
<tr>
<td>1716</td>
<td>SUBE5:</td>
<td></td>
</tr>
<tr>
<td>1726</td>
<td>SUBE6: DATA OUT SUBROUTINES</td>
<td></td>
</tr>
<tr>
<td>1734</td>
<td>SUB11:</td>
<td></td>
</tr>
<tr>
<td>1742</td>
<td>SUB12:</td>
<td></td>
</tr>
<tr>
<td>1750</td>
<td>SUB13:</td>
<td></td>
</tr>
<tr>
<td>1756</td>
<td>SUB14:</td>
<td></td>
</tr>
<tr>
<td>1764</td>
<td>SUB15:</td>
<td></td>
</tr>
<tr>
<td>1772</td>
<td>SUB16:</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>SUB17:</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>SUB20:</td>
<td></td>
</tr>
<tr>
<td>2026</td>
<td>SUB21:</td>
<td></td>
</tr>
<tr>
<td>2042</td>
<td>SUB22:</td>
<td></td>
</tr>
<tr>
<td>2056</td>
<td>SUB23:</td>
<td></td>
</tr>
<tr>
<td>2066</td>
<td>SUB24:</td>
<td></td>
</tr>
<tr>
<td>2266</td>
<td>OUT: OUTPUT ROUTINE</td>
<td></td>
</tr>
<tr>
<td>2216</td>
<td>SUB37: DATA IN SUBROUTINES</td>
<td></td>
</tr>
<tr>
<td>2232</td>
<td>SUB31:</td>
<td></td>
</tr>
<tr>
<td>2242</td>
<td>SUB32:</td>
<td></td>
</tr>
<tr>
<td>2254</td>
<td>SUB33:</td>
<td></td>
</tr>
<tr>
<td>2266</td>
<td>SUB34:</td>
<td></td>
</tr>
<tr>
<td>2282</td>
<td>SUB35:</td>
<td></td>
</tr>
<tr>
<td>2312</td>
<td>IN: INPUT ROUTINE</td>
<td></td>
</tr>
<tr>
<td>2512</td>
<td>LOOP: LOOP ROUTINE</td>
<td></td>
</tr>
<tr>
<td>2540</td>
<td>EXIT: EXIT ROUTINE</td>
<td></td>
</tr>
</tbody>
</table>
THE DATA BUFFERS, BEGINNING AT LOCATION 2541, BECOME A SEPARATE PROGRAM WHICH IS LINKED TO THE MAIN PROGRAM BY THE SYSTEM LOADER BEFORE EXECUTION. FIRST WE ESTABLISH THE GLOBALS IDENTIFYING THE LABELS COMMON TO BOTH THE MAIN PROGRAM AND THE DATA PROGRAM:

*GLOBAL TABLES, EBUF, DBUF*
3.2 TABLES

************

THERE ARE EIGHT TABLES OF SIXTEEN WORDS (8*16 = 128). THE FOLLOWING SEQUENCE OF CODE WILL RESERVE MEMORY FOR THE TABLES:

1) TABLES:
2) TBL1:
3) =TABLES+20
4) TBL2:
5) =TABLES+40
6) TBL3:
7) =TABLES+60
8) TBL4:
9) =TABLES+100
10) TBL5:
11) =TABLES+120
12) TBL6:
13) =TABLES+140
14) TBL7:
15) =TABLES+160
16) TBL8:
17) =TABLES+200

NOTE THE FIRST TWO LABELS ARE SYNONYMOUS (TABLES AND TBL1, LINES 1 AND 2) FOR CONVENIENCE. AFTER EACH TABLE HEADING (TBL1, TBL2, ETC.) A BLOCK OF SIXTEEN WORDS IS RESERVED BY SETTING THE PROGRAM COUNTER (•) TO THE NEXT HEADING OR LABEL (LINE 3, ETC.).

PAGE 35
3.3 THE ENABLE BUFFER

************

FOLLOWING THE TABLES IS THE ENABLE BUFFER (EBUF), A SHORT BUFFER OF SIXTEEN BYTES (8 WORDS) SET 0 FOR AN INACTIVE BUFFER, AND 1 FOR AN ACTIVE BUFFER.

1) EBUF: BYTE 0,0,0,0,0,0,0,0

2) BYTE 1,1,1,1,0,0,0,0

3) =EBUF+10

IN THE EXAMPLE ONLY BUFFERS 9, 10, 11 AND 12 ARE ACTIVE AND THE REMAINDER INACTIVE. THE BLOCK OF EIGHT WORDS IS CREATED (LINES 1 AND 2) AND THE PROGRAM COUNTER SET TO THE NEXT LABEL (LINE 3).
3.4 DATA BUFFERS

NOW WE RESERVE MEMORY FOR THE SIXTEEN DATA BUFFERS AS FOLLOWS:

1) DBUF:

2) DBUF1:

3) DBUF2:
   *=DBUF+4000

4) DBUF3:
   *=DBUF+1000

5) DBUF4:
   *=DBUF+14000

6) DBUF5:
   *=DBUF+2000

7) DBUF6:
   *=DBUF+24000

8) DBUF7:
   *=DBUF+3000

9) DBUF8:
   *=DBUF+34000

10) DBUF9:
    *=DBUF+4000

11) DBUF10:
    *=DBUF+44000
AGAIN THE FIRST TWO LABELS (DBUF AND DBUF1, LINES 1 AND 2) ARE SYNONYMOUS. AFTER EACH BUFFER HEADING (DBUF1, DBUF2, ETC) A BLOCK OF ONE HUNDRED AND TWENTY-EIGHT WORDS IS RESERVED.

+16V

+------------------
| ** | ** | ** |
| *  | *  | *  |
| *  | *  | *  |

0V

+------------------
| *  | *  | *  | *  | *  |
| *  | *  | *  | *  | *  |
| *  | *  | *  | *  | *  |

-16V

+------------------
| *  | ** | ** |
| *  | *  | *  |
| *  | *  | *  |

T = 0 T = 16 FIELDS
0 1

DELTA T = 1 FIELD
### 4.1 CREATING TABLES

#### 4.1.1 A SAMPLE TABLE

Tables are filled in as illustrated in this example:

<table>
<thead>
<tr>
<th></th>
<th>TABLE:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.WORD 174217</td>
</tr>
<tr>
<td>2</td>
<td>.WORD 177777</td>
</tr>
<tr>
<td>3</td>
<td>.WORD 167356</td>
</tr>
<tr>
<td>4</td>
<td>.WORD 156735</td>
</tr>
<tr>
<td>5</td>
<td>.WORD 146314</td>
</tr>
<tr>
<td>6</td>
<td>.WORD 135673</td>
</tr>
<tr>
<td>7</td>
<td>.WORD 125252</td>
</tr>
<tr>
<td>8</td>
<td>.WORD 114631</td>
</tr>
<tr>
<td>9</td>
<td>.WORD 73567</td>
</tr>
<tr>
<td>10</td>
<td>.WORD 63146</td>
</tr>
<tr>
<td>11</td>
<td>.WORD 52525</td>
</tr>
<tr>
<td>12</td>
<td>.WORD 42124</td>
</tr>
<tr>
<td>13</td>
<td>.WORD 31463</td>
</tr>
<tr>
<td>14</td>
<td>.WORD 21242</td>
</tr>
<tr>
<td>15</td>
<td>.WORD 10421</td>
</tr>
<tr>
<td>16</td>
<td>.WORD 0</td>
</tr>
</tbody>
</table>
This table contains the simplest bar patterns available on Don McArthur's 16:1 Select Modules.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Represents a solid field</td>
</tr>
<tr>
<td>2</td>
<td>Two horizontal bars</td>
</tr>
<tr>
<td>3</td>
<td>Four horizontal bars</td>
</tr>
<tr>
<td>4</td>
<td>Eight horizontal bars</td>
</tr>
<tr>
<td>5</td>
<td>Sixteen horizontal bars</td>
</tr>
<tr>
<td>6</td>
<td>Thirty-two horizontal bars</td>
</tr>
<tr>
<td>7</td>
<td>Sixty-four horizontal bars</td>
</tr>
<tr>
<td>8</td>
<td>One hundred and twenty-eight hours</td>
</tr>
<tr>
<td>9</td>
<td>Two vertical bars</td>
</tr>
<tr>
<td>10</td>
<td>Four vertical bars</td>
</tr>
<tr>
<td>11</td>
<td>Eight vertical bars</td>
</tr>
<tr>
<td>12</td>
<td>Sixteen vertical bars</td>
</tr>
<tr>
<td>13</td>
<td>Thirty-two vertical bars</td>
</tr>
<tr>
<td>14</td>
<td>Sixty-four vertical bars</td>
</tr>
<tr>
<td>15</td>
<td>One hundred and twenty-eight vertical bars</td>
</tr>
<tr>
<td>16</td>
<td>Two hundred and fifty-six vertical bars</td>
</tr>
</tbody>
</table>

Other tables are useful, shaded bar patterns, crosshatch patterns and masks for example.
4.2 CREATING A DATA BUFFER

4.2.1 A SAMPLE BUFFER

AN EXAMPLE OF A REAL DATA BUFFER FOLLOWS:

1) DBUF9:  +WORD 262
2) +WORD 123122
3) +WORD 46
4) L901:  +WORD 1310421
5) +WORD 46
6) +WORD 4647777L901
7) +WORD 47

THE DATA BUFFER IS FILLED WITH A SEQUENCE OF COMMAND WORDS USED BY THE MAIN PROGRAM TO CONTROL. IN THIS EXAMPLE, THE MCGARTHUR RED 16:1 SELECT MODULE. FIRST THE TIMING INTERVAL IS SET TO 1 SEC (60 FIELDS, LINE 1). THE COMMAND WORD IS 6, THE INTERVAL IS 60, THE PERIOD INDICATING A DECIMAL RATHER THAN AN OCTAL NUMBER. THE COMMAND 10 SETS THE DATA EQUAL TO THE OCTAL NUMBER 31222 (LINE 2). FINALLY A 46 CAUSES THE DATA TO BE TRANSFERRED TO THE BUFFER MEMORY. THE MAIN PROGRAM GOES ON TO THE NEXT BUFFER AND WILL NOT RETURN TO THIS BUFFER FOR ANOTHER 60 INTERRUPTS OR 1 SEC. WHEN IT DOES RETURN (TO LINE 4) IT ADDS THE OCTAL NUMBER 12421 TO THE DATA AND TRANSFERS THE SUM TO THE BUFFER MEMORY (LINE 5). AGAIN THE MAIN PROGRAM RETURNS AFTER 1 SEC. IT RETURNS TO LINE 6 AND FINDS THE LOOP COMMAND 47. INITIALLY THE COUNTER IS 0, THE NUMBER OF TIMES THROUGH THE LOOP WILL BE 777 OCTAL, AND THE DATA BUFFER POINTER WILL BE SET BACK TO L901. THE MAIN PROGRAM WILL REPEAT LINES 4-6 777 OCTAL TIMES AND THEN EXPIRES (LINE 7).
4.3 PROGRAMMING THE DIGITAL TO ANALOG CONVERTERS

4.3.1 PROTOCOL

NOW FOR SOME SIMPLE (MINDED) EXAMPLES OF PROGRAMMING TECHNIQUES. THE EASIEST DEVICES TO PROGRAM ARE THE D/A CONVERTERS (OUTPUT DEVICES 1-8) WHICH TRANSLATE A NUMBER INTO A CONTROL VOLTAGE:

- 1277*** = +12V
- 1208** = 0V
- 0** = -12V

** - LOW ORDER BITS 0-5 NOT USED

4.3.2 A SIMPLE RAMP

1) 0, 66
2) 18, 0
3) 46
4) LI91: 13, 100
5) 46
6) 46, 0, 1776, LI91
IN LINE 1 WE SET THE TIMING INTERVAL TO 627 FIELDS OR 1 SEC. WE SET THE D/A TO -10V (LINE 2) AND OUTPUT THIS VALUE TO THE D/A (LINE 3). NOW WE CONSTRUCT A LOOP (LINES 4-6). THE LABEL L1271 SETS THE TOP OF THE LOOP. THE COMMANDS TO BE REPEATED ARE ADD 192 OCTAL TO THE DATA AND OUTPUT THE NEW VALUE TO THE D/A. THIS IS REPEATED 1776 TIMES.

A SIMPLE METHOD FOR UNDERSTANDING A LOOP IS A TABLE:

<table>
<thead>
<tr>
<th># REPEATS</th>
<th>OLD DATA</th>
<th>NEW DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0 +1024 = 1024</td>
</tr>
<tr>
<td>2</td>
<td>1024</td>
<td>1024 +1024 = 2048</td>
</tr>
<tr>
<td>3</td>
<td>2048</td>
<td>2048 +1024 = 3072</td>
</tr>
<tr>
<td>4</td>
<td>3072</td>
<td>3072 +1024 = 4096</td>
</tr>
<tr>
<td>5</td>
<td>4096</td>
<td>4096 +1024 = 5120</td>
</tr>
<tr>
<td>6</td>
<td>5120</td>
<td>5120 +1024 = 6144</td>
</tr>
<tr>
<td>7</td>
<td>6144</td>
<td>6144 +1024 = 7168</td>
</tr>
<tr>
<td>8</td>
<td>7168</td>
<td>7168 +1024 = 8192</td>
</tr>
</tbody>
</table>
4.3.3 A REPEATING SAWTOOTH

************

1) \( T = 1 \)
2) \( Q_1 : 18 \, \Omega \)
3) \( T = 46 \)
4) \( Q_2 : 13 \, \Omega \)
5) \( T = 46 \)
6) \( Q_3 : 17 \, \Omega \)
7) \( Q_1 \)

\\begin{tabular}{c}
\hline
+10V & - &  \\
\hline
0V & - &  \\
-10V & - &  \\
\hline
\end{tabular}

\( T = 0 \) \quad \text{T = 16 FIELDS}
\( T = 1 \) \quad \text{T = 16 FIELDS}
\( T = 1 \) \quad \text{DELTA T = 1 FIELD}
\( T = 1 \) \quad \text{DELTA V = 1.25V}
\( T = 1 \) \quad \text{FREQUENCY = APPROX 4 HZ}
\( T = 1 \) \quad \text{AMPLITUDE = 20V PP}

PAGE 44
THIS COULD BE A NEGATIVE GOING SAWTOOTH:

1) \( Q \cdot 1 \)
2) \( L101: \quad 12 \cdot 177722 \)
3) \( 46 \)
4) \( L102: \quad 14 \cdot 1000 \)
5) \( 46 \)
6) \( 40 \cdot 2 \cdot 17 \cdot L102 \)
7) \( 14 \cdot 77722 \)
8) \( 46 \)
9) \( 40 \cdot 2 \cdot 10000 \cdot L101 \)

IN BOTH EXAMPLES A PAIR OF NESTED LOOPS IS USED. A LOOP L101 REPEATS THE BASIC WAVE FORM 1000 TIMES (LINES 2-9) AND LOOP L102 BUILDS THE WAVEFORM (LINES 4-6).

THERE IS A SIMPLER WAY OF BUILDING A SAWTOOTH WHICH USES THE CPU’S WRAP-AROUND FEATURE:

1) \( Q \cdot 1 \)
2) \( 12 \cdot 2 \)
3) \( 46 \)
4) \( L101: \quad 13 \cdot 1000 \)
5) \( 46 \)
6) \( 40 \cdot 2 \cdot 20 \cdot L101 \)
7) \( 40 \cdot 2 \cdot 10000 \cdot L101 \)

THIS PRODUCES EXACTLY THE SAME WAVEFORM AS THE FIRST EXAMPLE. ON THE SIXTEENTH REPETITION WE GET 177722 + 1000 = 0, WHICH COMPLETES THE INSIDE LOOP. THE OUTSIDE LOOP REMAINS THE SAME.
4.3.4 A REPEATING TRIANGLE

1) 0,1
2) 12.2
3) 46
4) L1E1: 13,1000
5) 46
6) 48,0,17,L1E1
7) 13,7722
8) 46
9) 14,7722
10) 46
11) L1E2: 14,1022
12) 46
13) 48,0,17,L1E2
14) 48,0,1000,L1E1
Again the timing interval is set to 1 field and the D/A converter set to 0V (lines 1-3). The outside loop (lines 4-14) repeats the waveform 1000 times. The first inside loop builds the positive going slope of the triangle (lines 4-6). Then the peak of the triangle is formed (lines 7-10). The second inside loop builds the negative slope (lines 11-13).
4.3.5 MAKING A SINE WAVE

FIRST EXAMINE THIS TABLE:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>+100</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>+200</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
<td>+400</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>700</td>
<td>+1000</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>1700</td>
<td>+2000</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>3700</td>
<td>+4000</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>7700</td>
<td>+10000</td>
<td>21</td>
</tr>
<tr>
<td>8</td>
<td>177772</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>9</td>
<td>277772</td>
<td></td>
<td>23</td>
</tr>
<tr>
<td>10</td>
<td>377772</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>11</td>
<td>477772</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>577772</td>
<td></td>
<td>26</td>
</tr>
<tr>
<td>13</td>
<td>677772</td>
<td></td>
<td>27</td>
</tr>
<tr>
<td>14</td>
<td>777772</td>
<td></td>
<td>28</td>
</tr>
</tbody>
</table>
NEXT THE TABLE IS CODED AS FOLLOWS:

1) $2.6$
2) $12.7$
3) $46$
4) $13.16$
5) $46$
6) $13.26$
7) $46$
8) $13.46$
9) $46$
10) $13.16$
11) $46$
12) $13.26$
13) $46$
14) $13.46$
15) $46$
16) L101: $13.16$
17) $46$
18) $46.14.1101$
19) $13.46$
20) $46$
\[ T = \theta \]
\[ T = 156 \text{ fields} \]
\[ \Delta T = 6 \text{ fields} \]
\[ \Delta V \text{ varies} \]

This is too much work for a sine wave. Improvements will be made. At this point development stops.
5.1 LIMITATIONS OF PRESENT SOFTWARE

AS OBVIOUS THE PROGRAM FAILS TO SATISFY THE ORIGINAL DESIGN CRITERIA. THE PROGRAM IS NOT INTERACTIVE. IT IS NOT CONCERNED WITH GRAPHIC DESIGN OR COMPOSITION. IT CANNOT REPROGRAM ITSELF IN RESPONSE TO EXTERNAL STIMULAE. HOWEVER IT'S NOT A TOTAL LOSS; THE BASIC GROUNDWORK IS COMPLETE. THE ELEMENTS OF THE LANGUAGE OUTLINED IN APPENDICES A AND B ARE STILL BEYOND THE UNINITIATED. BUT, FROM THESE ELEMENTS A HIGHER LEVEL LANGUAGE WILL BE CREATED. THIS NEW LANGUAGE WILL FACILITATE THE DIALOGUE BETWEEN THE ARTIST AND THE PROGRAM ALLOWING HIM TO CREATE THE IMAGES AND SEQUENCES OF IMAGES IN A LANGUAGE HE UNDERSTANDS; A GRAPHIC DESIGN LANGUAGE.

THE PRESENT PROGRAM RUNS IN BATCH MODE. THAT IS, THE DATA MUST BE PREPARED BEFORE THE PROGRAM IS RUN. THEN THE MAIN PROGRAM AND THE DATA ARE LINKED, LOADED AND FINALLY PROCESSED. IF THE RESULTS ARE NOT QUITE AS EXPECTED ( THE NORM RATHER THAN THE EXCEPTION ) THEN THE WHOLE PROCESS MUST BE REPEATED; HARDLY INSTANT GRATIFICATION.

AGAIN, THIS MODE OF OPERATION IS ONLY TEMPORARY; REAL-TIME INTERACTION WILL BE ADDED BY EXPANDING THE INTERPRETER ROUTINE TO INCLUDE THE ABILITY TO LISTEN AND TALK BACK.

IF THE PROGRAM LISTENS AND TALKS THEN IT CAN LEARN. COMBINING THE RANDOM NUMBER GENERATOR WITH A SIMPLE ALGORITHM FOR ANALYZING IMAGES WE CAN ENDOW THE PROGRAM WITH A PERSONALITY ( OR SEVERAL PERSONALITIES ).

BUT WHAT IS THE LANGUAGE SPOKEN BY THE ARTIST AND THE PROGRAM? THAT'S A QUESTION FOR CONTINUING RESEARCH.
5.2 PROPOSED SOFTWARE DEVELOPMENT

PROPOSED PROGRAM DEVELOPMENT INCLUDES:

1. ADDING A TERMINAL INPUT AND OUTPUT ROUTINE TO THE INTERPRETER.
2. ADDING MACRO COMMANDS INVOKING COMMAND WORD SEQUENCES.
3. ADDING A DATA BUFFER TO OUTPUT DEVICE CROSS-REFERENCE TABLE.
4. ADDING EDITING COMMANDS TO MODIFY DATA BUFFER CONTENTS IN REAL-TIME.
5. ADDING CONDITIONAL BRANCH COMMANDS.
6. DESIGNING A HIGHER LEVEL LANGUAGE BASED ON THE ELEMENTS AND ATTRIBUTES OF GRAPHIC DESIGN.
7. EXPANDING THE MANUAL OF PROGRAMMING TECHNIQUES.
8. CREATING A PERSONALITY FOR THE PROGRAM: ANTHROPOMORPHIZATION OF THE PROGRAM.

AND FINALLY I WILL ATTEMPT TO KEEP UP WITH THE BREAK-NECK PACE OF HARDWARE DEVELOPMENT.
APPENDIX A - COMMAND WORDS

Q+N ;SET THE TIMING INTERVAL

DIV $71RSS DIVIDE <R> <R/S>
ASH $72RSS ARITHMETIC SHIFT
ASHC $73RSS SHIFT COMBINED

FADD $7500R FLOATING ADD
FSUB $7501R FLOATING SUBTRACT
FMUL $7502R FLOATING MULTIPLY
FDIV $7503R FLOATING DIVIDE

BR $000402 BRANCH UNCONDITIONAL
BNE $000402 BRANCH IF $, Z = 0
BEQ $000402 BRANCH IF $, Z = 1
BPL $100002 BRANCH IF PLUS, N = 0
BMI $100002 BRANCH IF MINUS, N = 1
BVC $100002 BRANCH IF OVERFLOW CLEAR, V = 0
BVS $100002 BRANCH IF OVERFLOW SET, V = 1
BCC $100002 BRANCH IF CARRY CLEAR, C = 0
BGS $100002 BRANCH IF CARRY SET, C = 1

BEQ $000402 BRANCH IF = Vi Z = 1
BPL $100002 BRANCH IF PLUS, N = 0
BMI $100002 BRANCH IF MINUS, N = 1
BVC $100002 BRANCH IF OVERFLOW CLEAR, V = 0
BVS $100002 BRANCH IF OVERFLOW SET, V = 1

BEQ $000402 BRANCH IF = Vi Z = 1
BPL $100002 BRANCH IF PLUS, N = 0
BMI $100002 BRANCH IF MINUS, N = 1
BVC $100002 BRANCH IF OVERFLOW CLEAR, V = 0
BVS $100002 BRANCH IF OVERFLOW SET, V = 1

BR $101000 BRANCH IF HIGHER, C Z = 9
BLOS $101000 BRANCH IF LOWER OR SAME, C Z = 1
BHIS $101000 BRANCH IF HIGHER OR SAME, C = 9
BL $000402 BRANCH IF LOWER, C = 1

JMP $0001DD JUMP PG <D>
JSR $0001DD JUMP SUBROUTINE
RTS $0002RR RETURN FROM SUBROUTINE
MARK $0064NN MARK
SGB $0077NN SUBTRACT 1 AND BRANCH IF $

EMT $104*** EMULATOR TRAP
TRAP $104*** TRAP
BPT $000003 BREAKPOINT TRAP
IOT $000004 INPUT/OUTPUT TRAP
RTI $000006 RETURN FROM INTERRUPT
RTT $000006 RETURN FROM INTERRUPT, INHIBIT TRAP

HALT $000000 HALT
WAIT $000001 WAIT FOR INTERRUPT
RESET $000005 RESET BUS
NOP $000240 NO OPERATION

CLC $000241 CLEAR C C 0
CLV $000242 CLEAR V V 0
CLZ $000244 CLEAR Z Z 0
CLN $000250 CLEAR N N 0
CCC $000257 CLEAR ALL

SEC $000261 SET C C 1
SEV $000262 SET V V 1
SEZ $000264 SET Z Z 1
SEN $000270 SET N N 1
SCC $000277 SET ALL

NEA REPORT
APPENDIX A - COMMAND WORDS

Q+N ;SET THE TIMING INTERVAL
THE INTERVAL IS THE NUMBER OF FIELDS THE MAIN PROGRAM WAITS BEFORE RETURNING TO THE DATA BUFFER FOR THE NEXT COMMAND WORD.

1. \( N \)  
   \[ \text{ADD TO THE TIMING INTERVAL} \]  
   \[ \text{INTERVAL} = \text{INTERVAL} + N \]

2. \( N \)  
   \[ \text{SUBTRACT FROM THE TIMING INTERVAL} \]  
   \[ \text{INTERVAL} = \text{INTERVAL} - N \]

3  
   \[ \text{COMPLEMENT THE TIMING INTERVAL} \]  
   \[ \text{INTERVAL} = \text{INTERVAL} \oplus 77777 \]

4. \( N \)  
   \[ \text{SHIFT THE TIMING INTERVAL RIGHT} \]  
   \[ \text{INTERVAL} = \text{INTERVAL} \div 2 \]

   AN INTERVAL OF 1 SEC BECOMES 2 SEC.

5  
   \[ \text{SHIFT THE TIMING INTERVAL LEFT} \]  
   \[ \text{INTERVAL} = \text{INTERVAL} \times 2 \]

   AN INTERVAL OF 1 SEC BECOMES 2 SEC.

10. \( N \)  
   \[ \text{SET THE DATA WORD} \]  
   \[ \text{DATA} = \text{N, WHERE -1<N<20000 OCTAL} \]

11  
   \[ \text{INCREMENT THE DATA WORD} \]  
   \[ \text{DATA} = \text{DATA} + 1, \ 77777 + 1 = 0 \]

12  
   \[ \text{DECREMENT THE DATA WORD} \]  
   \[ \text{DATA} = \text{DATA} - 1, \ 0 - 1 = 17777 \]

13. \( N \)  
   \[ \text{ADD TO THE DATA WORD} \]  
   \[ \text{DATA} = \text{DATA} + N \]

14. \( N \)  
   \[ \text{SUBTRACT FROM THE DATA WORD} \]  
   \[ \text{DATA} = \text{DATA} - N \]

15  
   \[ \text{COMPLEMENT THE DATA WORD} \]  
   \[ \text{DATA} = \text{DATA} \oplus 77777 \]

16  
   \[ \text{SHIFT THE DATA WORD RIGHT} \]  
   \[ \text{DATA} = \text{DATA} \div 2 \]

17  
   \[ \text{SHIFT THE DATA WORD LEFT} \]  
   \[ \text{DATA} = 2 \times \text{DATA} \]

20  
   \[ \text{ROTATE THE DATA WORD RIGHT} \]

15  
   \( \oplus \)  
   \[ \text{BIT N BECOMES BIT N-1} \]
BIT 2 BECOMES BIT 15
21; ROTATE THE DATA WORD LEFT
15

BIT N BECOMES BIT N + 1

BIT 15 BECOMES BIT 2

22.N; BIT CLEAR DATA WORD WITH N
DATA = DATA (N)
DATA = 0110 010 010 010 111 065327
N = 0111 101 101 101 101
DATA = 0110 010 010 010 111 041542

23.N; BIT SET DATA WORD WITH N
DATA = DATA N
DATA = 0110 010 010 010 111 065767
N = 0111 101 101 101 101
DATA = 0110 010 010 010 111 024665

24.N; XOR DATA WORD WITH N
DATA = DATA N
DATA = 0110 010 010 010 111 065767
N = 0111 101 101 101 101
DATA = 0110 010 010 010 111 024665

30.N1,N2; GET DATA

WITH N1 = 1 TO 8 AND N2 = 1 TO 16 REGISTER 2 BECOMES THE VALUE CONTAINED IN TABLE N1 ENTRY N2.

WITH N1 = 9 TO 12 REGISTER 2 BECOMES THE VALUE SENSED BY ANALOG TO DIGITAL CONVERTER N1.

WITH N1 = 13 REGISTER 2 BECOMES THE VALUE SENSED BY THE REAL-TIME INTERFACE.

WITH N1 = 14 REGISTER 2 IS SET BY THE RANDOM NUMBER GENERATOR.


31.N1,N2; GET NEW DATA AND ADD TO OLD DATA

COMBINES COMMANDS 30 AND 31.

32.N1,N2; GET NEW DATA AND SUBTRACT FROM OLD DATA

COMBINES COMMANDS 30 AND 14.

33.N1,N2; GET NEW DATA AND BIT CLEAR WITH OLD DATA

COMBINES COMMANDS 30 AND 22.

34.N1,N2; GET NEW DATA AND BIT SET WITH OLD DATA

COMBINES COMMANDS 30 AND 22.
COMBINES COMMANDS 32 AND 23.

35, N1, N2  GET NEW DATA AND XOR WITH OLD DATA

COMBINES COMMANDS 37 AND 24.

46, N1, N2, LABEL ; LOOP COMMAND

THE PROGRAM IS SET TO REPEAT A SEQUENCE OF COMMANDS WHERE:

N1  - @, USED AS A COUNTER BY PROGRAM
N2  - @ TO 177777, NUMBER OF REPETITIONS
LABEL - POINTER TO TOP OF LOOP

EXAMPLE OF A SINGLE LOOP:

1) LABEL1: COMMAND
2) COMMAND
3) 42,0,100,..,LABEL1

EXAMPLE OF A NESTED LOOPS:

1) LABEL1: COMMAND
2) LABEL2: COMMAND
3) COMMAND
4) 42,0,100,..,LABEL2
5) 42,0,100,..,LABEL1

EXAMPLE OF MULTIPLE LOOPS:

1) LABEL1: COMMAND
2) COMMAND
3) 42,0,100,..,LABEL1
4) LABEL2: COMMAND
5) COMMAND
6) 42,0,100,..,LABEL2
7) 42,0,100,..,LABEL1
46 ; OUTPUT COMMAND

THE DATA WORD CONTAINED IN THE PARAMETER LIST IS TRANSFERRED TO THE BUFFER MEMORY AND THE MAIN PROGRAM GOES ON TO THE NEXT DATA BUFFER.

47 ; THE EXIT COMMAND, THE END, FINISH
**APPENDIX A - LSI-11 OPERATION CODES**

- **B**: 0 for word, 1 for byte
- **SS**: source field 6 bits
- **DD**: destination field 6 bits
- **R**: general register 3 bits, 0-7

- AND
- INCLUSIVE OR
- EXCLUSIVE OR, XOR
- NOT

- Contents of source
- Contents of destination
- Contents of register
- Becomes

**N**: sign condition code
**Z**: zero condition code 1 bit
**V**: overflow condition code 1 bit
**C**: carry condition code 1 bit

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>_OPCODE</th>
<th>INSTRUCTION</th>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR(B)</td>
<td>B050DD</td>
<td>CLEAR</td>
<td>&lt;D&gt; 0</td>
</tr>
<tr>
<td>COM(B)</td>
<td>B051DD</td>
<td>COMPLEMENT</td>
<td>&lt;D&gt; &lt;D&gt;</td>
</tr>
<tr>
<td>INC(B)</td>
<td>B052DD</td>
<td>INCREMENT</td>
<td>&lt;D&gt; &lt;D&gt;+1</td>
</tr>
<tr>
<td>DEC(B)</td>
<td>B053DD</td>
<td>DECREMENT</td>
<td>&lt;D&gt; &lt;D&gt;-1</td>
</tr>
<tr>
<td>NEG(B)</td>
<td>B054DD</td>
<td>NEGATE</td>
<td>&lt;D&gt; -&lt;D&gt;</td>
</tr>
<tr>
<td>TST(B)</td>
<td>B057DD</td>
<td>TEST, SETS STATUS BITS</td>
<td></td>
</tr>
</tbody>
</table>

| ROR(B)   | B060DD | ROTATE RIGHT | <C,D> |
| ROL(B)   | B061DD | ROTATE LEFT  | <C,D> |
| ASR(B)   | B066DD | SHIFT RIGHT  | <D> 2 |
| ASL(B)   | B063DD | SHIFT LEFT   | 2*<D> |
| SWAB     | B067DD | SWAP BYTES   |
| ADC(B)   | B055DD | ADD CARRY    | <D> <D>+<C> |
| SBC(B)   | B056DD | SUBTRACT CARRY | <D> <D>-<C> |
| SXT      | B067DD | SIGN EXTEND  | 0 OR -1 |
| MFPS     | B067DD | MOVE BYTE FROM PS | <D> PS |
| MTPS     | B0645S | MOVE BYTE TO PS | PS <D> |
| MOV(B)   | B155DD | MOVE         | <D> <S> |
| CMP(B)   | B055DD | COMPARE      | <S-D>, SETS STATUS BITS |
| ADD      | B055DD | ADD          | <D> <S+D> |
| SUB      | B155DD | SUBTRACT     | <D> <D-S> |
| BIT(B)   | B055DD | BIT TEST     | <S D>, SETS STATUS BITS |
| BIG(B)   | B045DD | BIT SET      | <D> <(S) D> |
| BIS(B)   | B055DD | BIT SET      | <D> <S D> |
| XOR      | B074RDD | XOR         | <D> <R D> |
| MUL      | B070RSS | MULTIPLY     | <R> <R*S> |