SOFTWARE FOR A COMPUTER BASED VIDEO SYTHESIZER

WALTER WRIGHT

JULY 1977

EXPERIMENTAL TELEVISION CENTER LTD
164 COURT ST, BINGHAMTON
NEW YORK, 13901
607 723-9509

SUPPORTED BY NYSCA AND NEA

+--+

INDEX -

****

+--+

+---+

CHAPTER 1 - 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.

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 TELETYPE 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.

CHAPTER 2 - DESCRIPTION OF MAIN PROGRAM

****

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:

1.	LEWSTA	STATUS WORD	167770
2.	LEWOUT	OUTPUT WORD	167772
3.	LEWIN	INPUT WORD	167774
4.	LEWCHA	CHANNEL ADDRESS	167776

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 BESIDES IT'S QUICK; IMPORTANT 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:

1.	DONOUT	RED 16:1 SELECT	171040
2.	DONOUT+2	GREEN 16:1 SELECT	171042
3.	DONOUT+4	BLUE 16:1 SELECT	171044
4.	DONOUT+6	INVERSION REGISTER	171046
5.	LEDS	LED DISPLAY	171510
6.	DONIN	REAL TIME INPUT	171620
7.	DONSTA	STAUS REGISTER	171776
8.	JEFOUT	RED ALU	171100
9.	JEFOUT+2	GREEN ALU	171102
10.	JEFOUT+4	BLUE ALU	171104

2.2 INITIALIZATION
****

2.2.1 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 SEPERATELY 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-11. THE .REGDEF MACRO DEFINES THE LSI-11'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

****

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

```
1)              MOV     #1ØØØØØ,@#LEWOUT

2)              MOV     #1Ø,RØ

3)      BGN1:   DEC     RØ

4)              MOV     RØ,@#LEWCHA

5)              TST     RØ

6)              BEQ     BGN1
```

THE FIRST LINE OF CODE MOVES THE OCTAL NUMBER 1ØØØØØ TO THE OUTPUT WORD IN MEMORY WHICH CONTROLS THE D/A'S. THIS CAUSES THE D/A TO OUTPUT A CONSTANT ZERO VOLTS (+1ØV= 1777ØØ AND -1ØV= Ø). 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 Ø-7. THEREFORE WE SET REGISTER Ø EQUAL TO 8, OR OCTAL 1Ø (LINE 2). THEN WE COUNT DOWN REGISTER Ø WITH A LOOP (LINES 3,5 AND 6) AND AT THE SAME TIME ENABLE THE D/A'S BY MOVING THE CONTENTS OF REGISTER Ø TO THE CHANNEL ADDRESS WORD (LINE 4).

2.2.3 BUFFER MEMORY
****

AND WE INITIALIZE THE BUFFER MEMORY AS FOLLOWS:

```
1)              MOV     #DONOUT,RØ

2)              CLR     (RØ)+

3)              CLR     (RØ)+

4)              CLR     (RØ)+

5)              CLR     (RØ)+

6)              MOV     #DEFOUT,RØ

7)              CLR     (RØ)+

8)              CLR     (RØ)+

9)              CLR     (RØ)+
```

THIS CODE USES THE AUTO-INCREMENT MODE OF ADDRESSING (R)+ . LINE 1 MOVES #DONOUT (171Ø4Ø) INTO REGISTER Ø. THEN WE CLEAR THAT MEMORY LOCATION AND ADD +2 TO REGISTER Ø 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:

```
        MOVB    #20,TMRY
```

THAT IS, THE PROGRAM TOLERATES NO GREATER THAN 16 BUFFERS (OCTAL 20). 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     #10,R0
3)      BGN2:   CMPB    TMRX,TMRY
4)              BPL     TMR
5)              INCB    TMRX
6)              ADD     #10,R0
7)              CLR     (R0)
8)              MOV     #1,2(R0)
9)              MOVB    TMRX,R1
10)             DEC     R1
11)             SWAB    R1
12)             ADD     #DBUF,R1
13)             MOV     R1,4(R0)
14)             CLR     6(R0)
```

```
15)               BR       BGN2

16)      PBUF:    .=.+200
```

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. THEREFOR THE POINTER IS SET INITIALLY AS FOLLOWS:

POINTER= #DBUF+(256*(TMRX-1))

THIS FORMULA IS CODED FROM RIGHT TO LEFT. IN LINE 9 TMRX IS MOVED INTO REGISTER 1; THE DECREMENT INSTRUCTION IN LINE 10 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 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     #TMR1,@#170

2)              CLRB    TMRX

3)              INC     @#DONSTA

4)              BR      .

5)      TMR1:   CLR     @#DONSTA

6)              ADD     4,SP
```

IN LINE 1 WE PREPARE FOR THE INEVITALBLE INTERRUPT BY LOADING LOCATION 170 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:

```
        MOV     #PBUF,R0
        SUB     #10,R0
TMR2:   CMPB    TMRX,TMRY
        BPL     TMR
        INCB    TMRX
        ADD     #10,R0
        MOVB    TMRX,R2
        DEC     R2
        ADD     #EBUF,R2
        TSTB    (R2)
        BEQ     TMR2
        INC     (R0)
        CMP     (R0)
        BLE     TMR2
TMR3:   CLR     (R0)
        JSR     PC,INT
        BR      TMR2
TMRX:   .BYTE   0
TMRY:   .BYTE   0
```

AGAIN WE HAVE A LOOP SIMILIAR TO THE LOOP USED TO INITIALIZE THE PARAMETER BUFFER. LINES 1 AND 2 LOAD REGISTER Ø WITH #PBUF-8. IN LINE 3 THE COUNTER TMRX (INITIALLY Ø) 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 Ø BY 8 (LINE 6) AND CHECK THE ENABLE BUFFER (LINES 7 TO 1Ø). IF THE BUFFER IS DISABLED (THE CONTENTS OF LOCATION #EBUF+(TMRX-1) EQUAL Ø) WE BRANCH BACK TO TMR2 (LINE 11). IF THE BUFFER IS ENABLED THE TIMING COUNTER IS INCREMENTED (LINE 12) AND COMPARED WITH THE TMING 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 ROUNTINE. 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.

```
INT:     MOV     4(R0),R1

         MOV     (R1)+,R2

         ASL     R2

         ADD     #JBUF,R2

         MOV     (R2),R2

         SUB     #INT1,R2

         MOV     R2,INT1-2

         CLR     R5

         JSR     PC,EXIT

INT1:    MOV     R1,4(R0)

         TST     R5

         BEQ     INT

         RTS     PC
```

REMEMBER THAT REGISTER Ø CONTAINS THE ADDRESS OF THE FIRST PARAMETER WORD CONTROLLING THE DATA BUFFER. IN LINE 1 THE DATA POINTER 4(RØ) IS MOVED TO REGISTER 1. THEN THE COMMAND WORD (R1)+ IS MOVED FROM THE DATA BUFFER TO REGISTER 2; AND THE DATA POINTER IN AUTO-INCREMENTED (LINE 2). THE JUMP SUBROUTINE THROUGH THE PROGRAM COUNTER INSTRUCTION (LINE 9) IS DECODED BY THE ASSSEMBLER AS TWO WORDS - ØØ4767, XXXXXX. THE FIRST THREE DIGITS OF THE FIRST WORD (ØØ4) INDICATE A JSR INSTRUCTION. THE FOURTH DIGIT (7) INDICATES THAT REGISTER 7 (PC) WILL BE THE LINKAGE POINTER. THE FIFTH AND SIXTH DIGIT REPRESENT THE DESTINATION, THE FIFTH DIGIT SPECIFIES THE INDEX ADDRESSING MODE AND THE SIXTH DIGIT INDICATES THAT THE INDEX VALUE FOLLOWS THE INSTRUCTION. THE INDEX VALUE PLUS THE PROGRAM COUNTER EQUALS THE DESTINATION ADDRESS. IN LINES 3 - 6 THE INDEX VALUE IS CALCULATED USING THESE FORMULAE:

INDEX = SUBROUTINE ENTRY PT-#INT1

SUBROUTINE ENTRY PT = #JBUF+(2*COMMAND WORD)

THE INDEX VALUE IS MOVED TO LOCATION INT-2 (LINE 7). REGISTER 5 IS A DONE FLAG SET FOLLOWING THE OUTPUT COMMAND, IT IS CLEARED INITIALLY (LINE 8). THE JUMP SUBROUTINE INSTRUCTION IS EXECUTED (LINE 9), THE PROGRAMEXECUTES THE APPROPRIATE SUBROUTINE, AND RETURNS TO RESTORE THE DATA BUFFER POINTER (LINE 1Ø). THE DONE FLAG (R5) IS TESTED (LINE 11); IF ZERO THE PROGRAM BRANCHES BACK AND READS THE NEXT COMMAND WORD (LINE 12), OR RETURNS TO THE TIMING ROUTINE (LINE 13).

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 TIMIMG INTERVAL
****

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

```
1)      SUBØØ:  MOV     (R1)+,2(RØ)

2)              RTS     PC
```

2.5.2 ADD TO THE TIMING INTERVAL
****

COMMAND WORD Ø1 ADDS THE NEXT WORD IN THE DATA BUFFER T O THE TIMING INTERVAL.

```
1)      SUBØ1:  ADD     (R1)+,2(RØ)

2)              RTS     PC
```

2.5.3 SUBTRACT FROM THE TIMING INTERVAL
****

COMMAND WORD Ø2 SUBTRACTS THE NEXT WORD IN THE DATA BUFFER FROM THE TIMING INTERVAL.

```
1)      SUBØ2:  SUB     (R1)+,2(RØ)

2)              RTS     PC
```

2.5.4 COMPLEMENT THE TIMING INTERVAL
****

COMMAND WORD Ø3 COMPLEMENTS THE TIMING INTERVAL, EQUIVALENT TO 177777- TIMING INTERVAL.

```
1)      SUBØ3:  COM     2(RØ)

2)              RTS     PC
```

2.5.5 SHIFT THE TIMING INTERVAL RIGHT
****

COMMAND WORD Ø4 SHIFTS THE TIMING INTERVAL TO THE RIGHT, THE MOST SIGNIFICANT BIT (BIT 15) IS CLEARED, EQUIVALENT TO TIMING INTERVAL/2.

```
1)      SUBØ4:  CLC

2)              ROR     2(RØ)

3)              RTS     PC
```

2.5.6 SHIFT THE TIMING INTERVAL LEFT
****

COMMAND WORD Ø5 SHIFTS THE TIMING INTERVAL TO THE LEFT, THE LEAST SIGNIFICANT BIT (BIT Ø) IS CLEARED, EQUIVALENT TO 2* TIMING INTERVAL.

```
1)      SUBØ5:  CLC

2)              ROL     2(RØ)

3)              RTS     PC
```

COMMAND WORDS Ø6 AND Ø7 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 1Ø SETS THE DATA WORD (FOURTH WORD IN THE PARAMETER LIST) EQUAL TO THE NEXT WORD IN THE DATA BUFFER.

```
1)      SUB1Ø:  MOV     (R1)+,6(RØ)

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(RØ)

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(RØ)

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(RØ)

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(RØ)

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(RØ)

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

                15                   Ø
               +---------------------+
                                       -> BIT Ø DROPPED
               +---------------------+
```

```
1)      SUB16:  CLC

2)              ROR     6(RØ)

3)              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 Ø) IS CLEARED, EQUIVALENT TO 2* DATA WORD.

```
                BIT N BECOMES BIT N+1

                15                   Ø
               +---------------------+
          <-                            BIT 15 DROPPED
               +---------------------+
```

```
1)      SUB17:  CLC

2)              ROL     6(RØ)

3)              RTS     PC
```


+--+

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                  0
                 +----------------------+
          ->                               -> 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                  0
                 +----------------------+
          <-                               <- 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 22 TAKES THE NEXT WORD IN THE DATA BUFFER AND CLEARS 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       0 000 001 010 011 100
        DATA WORD                 0 000 001 001 001 001

        DATA WORD                 0 000 000 001 010 101
```

```
1)      SUB22:  BIC     (R1)+,6(R0)

2)              RTS     PC
```

2.6.12 BIT SET WITH DATA WORD
****

COMMAND WORD 23 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       0 000 001 010 011 100
        DATA WORD                 0 000 001 001 001 001

        DATA WORD                 0 000 001 011 011 101
```

```
1)      SUB23:  BIS     (R1)+,6(R0)

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      0 000 001 010 011 100
DATA WORD                0 000 001 001 001 001

DATA WORD                0 000 000 011 010 101
```

```
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(R0)

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(R0)

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(R0)

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(RØ)

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(RØ)

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(RØ)

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

EACH TIME A LOOP COMMAND (40) IS ENCOUNTERED IN THE DATA BUFFER, THE LOOP SUBROUTINE FIRST COMPARES THE COUNTER WITH THE MAXIMUM NUMBER OF REPETITIONS (LINE 1). IF THE COUNTER IS LESS THAN THE MAXIMUM NUMBER THE COUNTER IS INCREMENTED, THE POINTER TO DBUF (THIRD WORD IN THE PARAMETER LIST) IS UPDATED WITH THE POINTER TO THE TOP OF THE LOOP, AND RETURN TO THE INTERPRETER (LINES 3 - 5). IF THE COUNTER IS EQUAL TO OR GREATER THAN THE COUNTER WE BRANCH TO LOOP 1 (LINE 2), CLEAR THE COUNTER (LINE 6), STEP THE DATA BUFFER POINTER (LINE 7), AND RETURN TO THE INTERPRETER (LINE 8).

```
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
```

```
13)             ADD     #TABLES,R2

14)             MOV     (R2),R2

15)             RTS     PC
```

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 IN1 (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 9- 12):

```
1)      IN1:    CMP     R2,#15

2)              BPL     IN2

3)              SUB     #11,R2

4)              MOV     R2,@#LEWCHA

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 LEWCHA (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):

```
IN3:    CMP     R2,#17

        BPL     IN4

        MOV     TEMP,R2

        CLC

        ROL     TEMP

        BCC     RND1

        INC     R2

RND1:   ROL     TEMP+2

        BCC     RND2

        INC     R2
```

```
11)     RND2:   ROL     TEMP+4
12)             BCC     RND3
13)             INC     R2
14)     RND3:   ROL     TEMP+6
15)             BCC     RND4
16)             INC     R2
17)     RND4:   COM     R2
18)             ADD     R2,TEMP
19)             MOV     TEMP,R2
20)     IN4:    RTS     PC
21)     TEMP:   .WORD   0,0,0,0
```

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).

```
     64     48  47     32  31     16  15      0
     +-------+  +-------+  +-------+  +-------+
  <-TEMP+6   <-TEMP+4   <-TEMP+2   <-TEMP
     +-------+  +-------+  +-------+  +-------+

  C4         C3         C2         C1
```

TEMP= TEMP+(-1)*(TEMP+C4+C3+C2+C1)

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).

2.9.2 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

THROUGH AN UNACCOUNTABLE MENTAL LAPSE ON MY PART, THE DATA BUFFERS CORRESPOND DIRECTLY TO THE OUTPUT DEVICES: DATA BUFFERSETC 1-8 CONTROL THE A/D'S, DATA BUFFER 9 CONTROLS THE RED 16:1 SELECT, ETC. THE FIRST PART OF THE OUTPUT ROUTINE CONTROLS THE A/D'S:

```
1)      OUT:    CMPB    TMRX,#11

2)              BPL     OUT1

3)              MOVB    TMRX,R2

4)              DEC     R2

5)              MOV     R2,@#LEWCHA

6)              MOV     6(R0),@#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     OUT2
3)              MOVB    TMRX,R2
4)              SUB     #11,R2
5)              ASL     R2
6)              ADD     #DONOUT,R2
7)              MOV     6(R0),(R2)
8)      INC     R5
9)              RTS     PC
```

IF THE BUFFER NUMBER IS GREATER THAN 12 BRANCH TO OUT2 (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,#20
2)              BPL     OUT3
3)              MOVB    TMRX,R2
4)              SUB     #15,R2
```

```
5)                ASL     R2

6)                ADD     #JEFOUT,R2

7)                MOV     6(R0),(R2)

8)      OUT3:     INC     R5

9)                RTS     PC
```

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):

OUTPUT ADDRESS= #JEFOUT+2*(TMRX-13)

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

2.9.3 EXIT ROUTINE

****

COMMAND WORD 47 - THIS SUBROUTINE IS INVOKED OVERTLY BY COMMAND WORD 47 AND COVERTLY BY 06, 07, 25, 26, 27, 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
****

LOCATION	LABEL	FUNCTION
170		INTERRUPT VECTOR
1000	BEGIN:	INITIALIZATION
1146	PBUF:	CONTROL WORDS FOR DATA BUFFERS
1346	TMR:	TIMING ROUTINE
1470	INT:	INTERPRETER
1536	JBUF:	COMMAND WORD TO SUBROUTINE CROSS-REFERENCE
1656	SUB00:	TIMING CONTROL SUBROUTINES
1664	SUB01:	
1672	SUB02:	
1700	SUB03:	
1706	SUB04:	
1716	SUB05:	
1726	SUB10:	DATA OUT SUBROUTINES
1734	SUB11:	
1742	SUB12:	
1750	SUB13:	
1756	SUB14:	
1764	SUB15:	
1772	SUB16:	
2002	SUB17:	
2012	SUB20:	
2026	SUB21:	
2042	SUB22:	
2050	SUB23:	
2056	SUB24:	
2066	OUT:	OUTPUT ROUTINE
2216	SUB30:	DATA IN SUBROUTINES
2230	SUB31:	
2242	SUB32:	
2254	SUB33:	
2266	SUB34:	
2300	SUB35:	
2312	IN:	INPUT ROUTINE
2512	LOOP:	LOOP ROUTINE
2540	EXIT:	EXIT ROUTINE

```
2541    TBL1:   TABLES
2741    EBUF:   ENABLE BUFFER
2751    DBUF1:  DATA BUFFERS

167770  LEWSTA: STATUS WORD
167772  LEWOUT: OUTPUT WORD
167774  LEWIN:  INPUT  WORD
167776  LEWCHA: CHANNEL ADDRESS

171040  DONOUT: RED    16:1 SELECT
171042          GREEN  16:1 SELECT
171044          BLUE   16:1 SELECT
171046          INVERSION REGISTER

171100  JEFOUT: RED    ALU
171102          GREEN  ALU
171104          BLUE   ALU

171510  LEDS:   LED DISPLAY

171620  DONIN:  REAL-TIME INPUT

171776  DONSTA: STATUS REGISTER
```

THE DATA BUFFERS, BEGINNING AT LOCATION 2541, BECOME A SEPERATE 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 TABL1, 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).

3.3 THE ENABLE BUFFER
****

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

```
1)      EBUF:    .BYTE    Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø

2)               .BYTE    1,1,1,1,Ø,Ø,Ø,Ø

3)               .=EBUF+1Ø
```

IN THE EXAMPLE ONLY BUFFERS 9, 1Ø, 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:

```
DBUF:
DBUF1:
        .=DBUF+400
DBUF2:
        .=DBUF+1000
DBUF3:
        .=DBUF+1400
DBUF4:
        .=DBUF+2000
DBUF5:
        .=DBUF+2400
DBUF6:
        .=DBUF+3000
DBUF7:
        .=DBUF+3400
DBUF8:
        .=DBUF+4000
DBUF9:
        .=DBUF+4400
```

```
20)     DBUF10:
21)                 .=DBUF+5000
22)     DBUF11:
23)                 .=DBUF+5400
24)     DBUF12:
25)                 .=DBUF+6000
26)     DBUF13:
27)                 .=DBUF+6400
28)     DBUF14:   .
29)                 .=DBUF+7000
30)     DBUF15:
31)                 .=DBUF+7400
32)     DBUF16:
33)                 .=DBUF+10000
34)                 .END    TABLES
```

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 B40,0,1000.,L101

```
+10V    ----------*---------*---------*-----
                 **        **        **
                * *       * *       * *
               *  *      *  *      *  *
              *   *     *   *     *   *
  0V    -----*----*----*----*----*----*----*
            *     *   *     *   *     *   *
           *      *  *      *  *      *  *
          *       * *       * *       * *
         *        **        **        **
-10V    *---------*---------*---------*-----

        T = 0     T = 16 FIELDS
         0         1

        DELTA T= 1 FIELD
```

CHAPTER 4 - PROGRAMMING TECHNIQUES

4.1 CREATING TABLES

4.1.1 A SAMPLE TABLE

TABLES ARE FILLED IN AS ILLUSTRATED IN THIS EXAMPLE:

```
1)      TBL1:   .WORD   104210
2)              .WORD   177777
3)              .WORD   167356
4)              .WORD   156735
5)              .WORD   146314
6)              .WORD   135673
7)              .WORD   125252
8)              .WORD   114631
9)              .WORD   73567
10)             .WORD   63146
11)             .WORD   52525
12)             .WORD   42104
13)             .WORD   31463
14)             .WORD   21042
15)             .WORD   10421
16)             .WORD   0
```

THIS TABLE CONTAINS THE SIMPLEST BAR PATTERNS AVAILABLE ON DON MCARTHUR'S 16:1 SELECT MODULES.

LINE 1 -REPRESENTS A SOLID FIELD
LINE 2 -TWO HORIZONTAL BARS
LINE 3 -FOUR HORIZONTAL BARS
LINE 4 -EIGHT HORIZONTAL BARS
LINE 5 -SIXTEEN HORIZONTAL BARS
LINE 6 -THIRTY-TWO HORIZONTAL BARS
LINE 7 -SIXTY-FOUR HORIZONTAL BARS
LINE 8 -ONE HUNDRED AND TWENTY-EIGHT HORIZONTAL BARS
LINE 9 -TWO VERTICAL BARS
LINE 10 -FOUR VERTICAL BARS
LINE 11 -EIGHT VERTICAL BARS
LINE 12 -SIXTEEN VERTICAL BARS
LINE 13 -THIRTY-TWO VERTICAL BARS
LINE 14 -SIXTY-FOUR VERTICAL BARS
LINE 15 -ONE HUNDRED AND TWENTY-EIGHT VERTICAL BARS
LINE 16 -TWO HUNDRED AND FIFTY-SIX VERTICAL BARS

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   0,60.
2)              .WORD   10,31020
3)              .WORD   46
4)      L901:   .WORD   13,10421
5)              .WORD   46
6)              .WORD   40,0,777,L901
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 MCARTHUR RED 16:1 SELECT MODULE. FIRST THE TIMING INTERVAL IS SET TO 1 SEC (60 FIELDS, LINE 1). THE COMMAND WORD IS 0, 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 31020 (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 10421 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 40. 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:

```
1777**=  +1ØV
1ØØØ** =   ØV
   Ø** = -1ØV

   ** - LOW ORDER BITS Ø- 5 NOT USED
```

4.3.2 A SIMPLE RAMP
****

```
1)              Ø,6Ø.

2)              1Ø,Ø

3)              46

4)      L1Ø1:   13,1ØØ

5)              46

6)              4Ø,Ø,1776,L1Ø1
```

```
+10V    - - - - - - - - - - *
                          *
                        *
                      *
                    *
  0V    - - - - - * - - - - -
                *
              *
            *
          *
-10V    * - - - - - - - - - -

        T = 0                 T = 1024 SEC
         0                     1
```

DELTA T= 1 SEC

DELTA V= 20/1024 V

DURATION = 1024 SEC

AMPLITUDE= 20V PP

IN LINE 1 WE SET THE TIMING INTERVAL TO 60 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 L101 SETS THE TOP OF THE LOOP, THE COMMANDS TO BE REPEATED ARE ADD 100 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:

# REPEATS	OLD DATA	NEW DATA
1	0	0 +100= 100
2	100	100+100= 200
3	200	200+100= 300
4	300	300+100= 400
5	400	400+100= 500
6	500	500+100= 600
7	600	600+100= 700
8	700	700+100=1000

+--+

4.3.3 A REPEATING SAWTOOTH

****

```
1)                Ø,1

2)      L1Ø1:     1Ø,Ø

3)                46

4)      L1Ø2:     13,1ØØØØ

5)                46

6)                4Ø,Ø,17,L1Ø2

7)                4Ø,Ø,1ØØØ.,L1Ø1
```

```
+1ØV    ----------*---------*---------*-----
                 **        **        **
                * *       * *       * *
               *  *      *  *      *  *
              *   *     *   *     *   *
  ØV    -----*----*----*----*----*----*----*
            *     *   *     *   *     *   *
           *      *  *      *  *      *  *
          *       * *       * *       * *
         *        **        **        **
-1ØV    *---------*---------*---------*-----

        T = Ø     T = 16 FIELDS
         Ø         1
```

DELTA T= 1 FIELD

DELTA V= 1.25V

FREQUENCY= APPROX 4 HZ

AMPLITUDE= 2ØV PP

THIS COULD BE A NEGATIVE GOING SAWTOOTH:

```
1)                0,1
2)      L101:     10,177700
3)                46
4)      L102:     14,1000
5)                46
6)                40,0,17,L102
7)                14,7700
8)                46
9)                40,0,10000.,L101
```

IN BOTH EXAMPLES A PAIR OF NESTED LOOPS IS USED, A LOOP L101 REPEATS THE BASIC WAVE FORM 10,000 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)                0,1
2)                10,0
3)                46
4)      L101:     13,10000
5)                46
6)                40,0,20,L101
7)                40,0,10000.,L101
```

THIS PRODUCES EXACTLY THE SAME WAVEFORM AS THE FIRST EXAMPLE. ON THE SIXTEENTH REPETITION WE GET 170000+ 10000= 0, WHICH COMPLETES THE INSIDE LOOP. THE OUTSIDE LOOP REMAINS THE SAME.

EX

4.3.4 A REPEATING TRIANGLE

```
***************************

1)              0,1

2)              10,0

3)              46

4)      L101:   13,10000

5)              46

6)              40,0,17,L101

7)              13,7700

8)              46

9)              14,7700

10)             46

11)     L102:   14,10000

129             46

13)             40,0,17,L102

14)             40,0,1000.,L101
```

```
+10V    ----------*-------------------*---------------
                 * *                 * *
                *   *               *   *
               *     *             *     *
              *       *           *       *
  0V    -----*---------*---------*---------*---------*
            *           *       *           *       *
           *             *     *             *     *
          *               *   *               *   *
         *                 * *                 * *
-10V    *-------------------*-------------------*-----

        T = 0                    T = 32 FIELDS
         0                        1
```

DELTA T= 1 FIELD

DELTA V= 1.25V

FREQUENCY= APPRX 2 HZ

AMPLITUDE= 20V PP

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:

1.	0	+100	15.	107700	"
2.	100	+200	16.	117700	"
3.	300	+400	17.	127700	"
4.	700	+1000	18.	137700	"
5.	1700	+2000	19.	147700	"
6.	3700	+4000	20.	157700	"
7.	7700	+10000	21.	167700	+4000
8.	17700	"	22.	173700	+2000
9.	27700	"	23.	175700	+1000
10.	37700	"	24.	176700	+400
11.	47700	"	25.	177300	+200
12.	57700	"	26.	177500	+100
13.	67700	"	27.	177600	+100
14.	77700	"	28.	177700	

NEXT THE TABLE IS CODED AS FOLLOWS:

```
1)              0,6.
2)              10,0
3)              46
4)              13,100
5)              46
6)              13,200
7)              46
8)              13,400
9)              46
10)             13,1000
11)             46
12)             13,2000
13)             46
14)             13,4000
15)             46
16)     L101:   13,10000
17)             46
18)             40,0,14,L101
19)             13,40000
20)             46
```

21) 13,2000

22) 46

23) 13,1000

24) 46

25) 13,400

26) 46

27) 13,200

28) 46

29) 13,100

30) 46

```
+10V     ----------------***
                       **
                     **
                    *
                   *
  0V     ---------*---------
                 *
                *
              **
            **
-10V     ***----------------

         T  = 0             T  = 156 FIELDS
          0                  1
```

DELTA T= 6 FIELDS

DELTA V VARIES

THIS IS TOO MUCH WORK FOR A SINE WAVE, IMPROVEMENTS WILL BE MADE. AT THIS POINT DEVELOPMENT STOPS.

CHAPTER 5 - SUMMARY
****

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.

```
DIV       Ø71RSS   DIVIDE                                  <R>  <R/S>
ASH       Ø72RSS   ARITHMETIC SHIFT
ASHC      Ø73RSS   SHIFT COMBINED

FADD      Ø75ØØR   FLOATING ADD
FSUB      Ø75Ø1R   FLOATING SUBTRACT
FMUL      Ø75Ø2R   FLOATING MULTIPLY
FDIV      Ø75Ø3R   FLOATING DIVIDE

BR        ØØØ4ØØ   BRANCH UNCONDITIONAL
BNE       ØØ1ØØØ   BRANCH IF   Ø,Z = Ø
BEQ       ØØ14ØØ   BRANCH IF = Ø, Z= 1
BPL       1ØØØØØ   BRANCH IF PLUS, N= Ø
BMI       1ØØ4ØØ   BRANCH IF MINUS, N= 1
BVC       1Ø2ØØØ   BRANCH IF OVERFLOW CLEAR, V= Ø
BVS       1Ø24ØØ   BRANCH IF OVERFLOW SET, V= 1
BCC       1Ø3ØØØ   BRANCH IF CARRY CLEAR, C= Ø
BCS       1Ø34ØØ   BRANCH IF CARRY SET, C= 1

BGE       ØØ2ØØØ   BRANCH IF   Ø, N V= Ø
BLT       ØØ24ØØ   BRANCH IF   Ø, N V= 1
BGT       ØØ3ØØØ   BRANCH IF   Ø, Z (N V)=Ø
BLE       ØØ34ØØ   BRANCH IF   Ø, Z (N V)=1

BHI       1Ø1ØØØ   BRANCH IF HIGHER, C Z=Ø
BLOS      1Ø14ØØ   BRANCH IF LOWER OR SAME, C Z=1
BHIS      1Ø3ØØØ   BRANCH IF HIGHER OR SAME, C=Ø
BLO       1Ø34ØØ   BRANCH IF LOWER, C=1

JMP       ØØØ1DD   JUMP                            PC  <D>
JSR       ØØ4RDD   JUMP SUBROUTINE
RTS       ØØØ2ØR   RETURN FROM SUBROUTINE
MARK      ØØ64NN   MARK
SOB       Ø77RNN   SUBTRACT 1 AND BRANCH IF   Ø

EMT       1Ø4***   EMULATOR TRAP
TRAP      1Ø4***   TRAP
BPT       ØØØØØ3   BREAKPOINT TRAP
IOT       ØØØØØ4   INPUT/OUTPUT TRAP
RTI       ØØØØØ2   RETURN FROM INTERRUPT
RTT       ØØØØØ6   RETURN FROM INTERRUPT, INHIBIT TRAP

HALT      ØØØØØØ   HALT
WAIT      ØØØØØ1   WAIT FOR INTERRUPT
RESET     ØØØØØ5   RESET BUS
NOP       ØØØ24Ø   NO OPERATION

CLC       ØØØ241   CLEAR C                         C  Ø
CLV       ØØØ242   CLEAR V                         V  Ø
CLZ       ØØØ244   CLEAR Z                         Z  Ø
CLN       ØØØ25Ø   CLEAR N                         N  Ø
CCC       ØØØ257   CLEAR ALL

SEC       ØØØ261   SET C                           C  1
SEV       ØØØ262   SET V                           V  1
SEZ       ØØØ264   SET Z                           Z  1
SEN       ØØØ27Ø   SET N                           N  1
SCC       ØØØ277   SET ALL
```

NEA REPORT

APPENDIX A - COMMAND WORDS

```
Ø,N      ;SET THE TIMING INTERVAL
```

INTERVAL= N, WHERE Ø<N<2ØØØØØ OCTAL

THE INTERVAL IS THE NUMBER OF FIELDS THE MAIN PROGRAM WAITS BEFORE RETRUNING TO THE DATA BUFFER FOR THE NEXT COMMAND WORD.

1,N ;ADD TO THE TIMING INTERVAL

INTERVAL= INTERVAL+N

2,N ;SUBTRACT FROM THE TIMING INTERVAL

INTERVAL= INTERVAL-N

3 ;COMPLEMENT THE TIMING INTERVAL

INTERVAL= INTERVAL 177777

4,N ;SHIFT THE TIMING INTERVAL RIGHT

INTERVAL= INTERVAL/2

AN INTERVAL OF 1 SEC BECOMES 2 SEC.

5 ;SHIFT THE TIMING INTERVAL LEFT

INTERVAL= INTERVAL*2

AN INTERVAL OF 1 SEC BECOMES 2 SEC.

1Ø,N ;SET THE DATA WORD

DATA= N, WHERE -1<N<2ØØØØØ OCTAL

11 ;INCREMENT THE DATA WORD

DATA= DATA+1, 177777+1= Ø

12 ;DECREMENT THE DATA WORD

DATA= DATA-1, Ø-1= 177777

13,N ;ADD TO THE DATA WORD

DATA= DATA+N

14,N ;SUBTRACT FROM THE DATA WORD

DATA= DATA-N

15 ;COMPLEMENT THE DATA WORD

DATA= DATA 177777

16 ;SHIFT THE DATA WORD RIGHT

DATA= DATA/2

17 ;SHIFT THE DATA WORD LEFT

DATA= 2*DATA

2Ø ;ROTATE THE DATA WORD RIGHT

15 Ø

BIT N BECOMES BIT N-1

BIT 0 BECOMES BIT 15

21 ;ROTATE THE DATA WORD LEFT

```
15                0
                            BIT N BECOMES BIT N+1

BIT 15 BECOMES BIT 0
```

22,N ;BIT CLEAR DATA WORD WITH N

DATA= DATA (N)

```
DATA     0 110 001 011 010 111    065327
N        0 100 001 101 100 010    041542

DATA     0 010 100 010 010 101    024225
```

23,N ;BIT SET DATA WORD WITH N

DATA= DATA N

```
DATA     0 110 101 011 010 111
N        0 100 001 101 100 010

DATA     0 110 101 111 110 111    065767
```

24,N ;XOR DATA WORD WITH N

DATA= DATA N

```
DATA     0 110 101 011 010 111
N        0 100 001 101 100 010

DATA     0 010 100 110 110 101    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 GENERATER.

IF N1= 9 TO 14 THEN N2 IS NOT USED AND THE COMMAND TAKES THE FORM- 30,N1.

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 3Ø AND 23.

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

COMBINES COMMANDS 3Ø AND 24.

4Ø,N1,N2,LABEL ;LOOP COMMAND

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

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

EXAMPLE OF A SINGLE LOOP:

```
1)    LABEL1: COMMAND
2)            COMMAND
3)            4Ø,Ø,1ØØ.,LABEL1
```

EXAMPLE OF A NESTED LOOPS:

```
1)    LABEL1: COMMAND
2)    LABEL2: COMMAND
3)            COMMAND
4)            4Ø,Ø,1ØØ.,LABEL2
5)            4Ø,Ø,1ØØ.,LABEL1
```

EXAMPLE OF MULTIPLE LOOPS:

```
1)    LABEL1: COMMAND
2)            COMMAND
3)            4Ø,Ø,1ØØ.,LABEL1
4)    LABEL2: COMMAND
5)            COMMAND
6)            4Ø,Ø,1ØØ.,LABEL2
7)            4Ø,Ø,1ØØ.,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, FINIS

APPENDIX A - LSI-11 OPERATION CODES

B	- Ø FOR WORD, 1 FOR BYTE
SS	- SOURCE FIELD 6 BITS
DD	- DESTINATION FIELD 6 BITS
R	- GENERAL REGISTER 3 BITS, Ø- 7
	- AND
	- INCLUSIVE OR
	- EXCLUSIVE OR, XOR
	- NOT
<S>	- CONTENTS OF SOURCE
<D>	- CONTENTS OF DESTINATION
<R>	-CONTENTS OF REGISTER
<-	- BECOMES
N	- SIGN CONDITION CODE
Z	- ZERO CONDITION CODE 1 BIT
V	- OVERFLOW CONDITIN CODE 1 BIT
C	- CARRY CONDITION CODE 1 BIT

MNEMONIC	OPCODE	INSTRUCTION	NOTES
CLR(B)	BØ5ØDD	CLEAR	<D> Ø
COM(B)	BØ51DD	COMPLEMENT	<D> <D>
INC(B)	BØ52DD	INCREMENT	<D> <D>+1
DEC(B)	BØ53DD	DECREMENT	<D> <D>-1
NEG(B)	BØ54DD	NEGATE	<D> -<D>
TST(B)	BØ57DD	TEST, SETS STATUS BITS	
ROR(B)	BØ6ØDD	ROTATE RIGHT	<C,D>
ROL(B)	BØ61DD	ROTATE LEFT	<C,D>
ASR(B)	BØ62DD	SHIFT RIGHT	<D> 2
ASL(B)	BØ63DD	SHIFT LEFT	2*<D>
SWAB	ØØØ3DD	SWAP BYTES	
ADC(B)	BØ55DD	ADD CARRY	<D> <D>+<C>
SBC(B)	BØ56DD	SUBTRACT CARRY	<D> <D>-<C>
SXT	ØØ67DD	SIGN EXTEND	Ø OR -1
MFPS	1Ø67DD	MOVE BYTE FROM PS	<D> PS
MTPS	1Ø64SS	MOVE BYTE TO PS	PS <D>
MOV(B)	B1SSDD	MOVE	<D> <S>
CMP(B)	B2SSDD	COMPARE	<S-D>,SETS STATUS BITS
ADD	Ø6SSDD	ADD	<D> <S+D>
SUB	16SSDD	SUBTRACT	<D> <D-S>
BIT(B)	B3SSDD	BIT TEST	<S D>,SETS STATUS BITS
BIC(B)	B4SSDD	BIT SET	<D> <(S) D>
BIS(B)	B5SSDD	BIT SET	<D> <S D>
XOR	Ø74RDD	XOR	<D> <R D>
MUL	Ø7ØRSS	MULTIPLY	<R> <R*S>