II PIEEE-77 COOKBOOK:
HOW TO USE THE BOARD

HOW TO OPERATE THE PIEEE-77 BOARD

Fig 21.1 shows the physical layout of your board. Let us start using it and learn about it as we go along.

(1) Touch a ground to discharge yourself. This is believed to save MOS chips in the board from being blown by high-voltage static discharge.

(2) Put a minigator clip lead to the TOUCH-POST.

(3) Plug in the power supply. One of the latch lights will go on; some other lights may or may not go on.

CONTROL PADS

(4) With the other end of your clip lead touch Reset, then Run pads. The Run and Examine lights should go on, and probably some of the ADDRESS and DATA LIGHTS.

(5) Touch the SS (Single Step) pad. SS light should go on, RUN light should go off.

(6) Touch the HALT pad. The EXAMINE light remains on. Touch the LOAD pad. All 3 control lights should now be off.

EXAMINING CONTENTS OF RAM CHIPS

Each pair of 6111 RAM chips contains 256 8-bit registers. Each register contains a BYTE (8 bits) of data. For convenience we refer to 256 8-bit registers as a PAGE of memory. It takes 8 address bits to address the 256 registers within a page. These 8 address bits we shall refer to as the LINE address. Since the address bus consists of 16 lines, we will call the 8 more significant bits the PAGE address. The page address picks out a particular pair of RAM chips. The first pair of RAM chips on this board are wired to be addressed as page 00. Let us examine the contents of some of the registers on page 00 to see whether they have any data in them.

ADDRESS PADS

(7) Touch EXAMINE pad. Run the minigator clip lightly along the OFF pads of the ADDRESS lights. ADDRESS should now read zero, i.e., all ADDRESS lights should now be out. DATA lights now show the contents of register 0000.

BEFORE NEXT STEP UNPLUG POWER SUPPLY MOMENTARILY.

(9) Record below on DATA 1 line the light pattern in DATA lights using the hex code.

ADDRESS	0000	0001	0002	0003	0004	0005	0006
DATA 1	....	....	....	....	....	....	....
DATA 2	....	....	....	....	....	....	....

(8) Touch the ON pad of ADDRESS bit 0. That light should go ON. The hex pattern of address lights is now 0001. Record the DATA pattern. Repeat until the line above is filled.

(10) Unplug the power supply and plug it in again. Repeat from (6) above, this time entering in DATA 2 line. DATA 2 line should be more or less identical to the DATA 1 line.

In any case, this "data" is background noise due to natural processes during manufacturing of the RAM chips. NOTE: Usually this noise does not interfere with the operation of the board: just like an audio tape recorder, WRITING data into a register automatically erases what was there before; however, ADDING or SUBTRACTING data from a register -- on the assumption that there was zero there to begin with -- will result in an error, of course.

DATA PADS

(11) Before you go on to the next step, touch some of the DATA pads. Note that nothing happens to the lights -- data in RAM chips can not be changed while the EXamine light is on.

(12) Now touch the LOAD pad. The EXamine light should go out. The ADDRESS and DATA registers now display what you wish to load into the RAM chips. Note that now you can change both the address and data at will. (However, the RAM chip is not affected until the DEPOSIT pad is touched.)

(13) Set the ADDRESS to any address on page 0; i.e., starting with 00. EXamine data. Load some other data. Touch the DEPOSIT pad. EXamine again to verify that data has indeed been written into the RAM chips. Try it on several data.

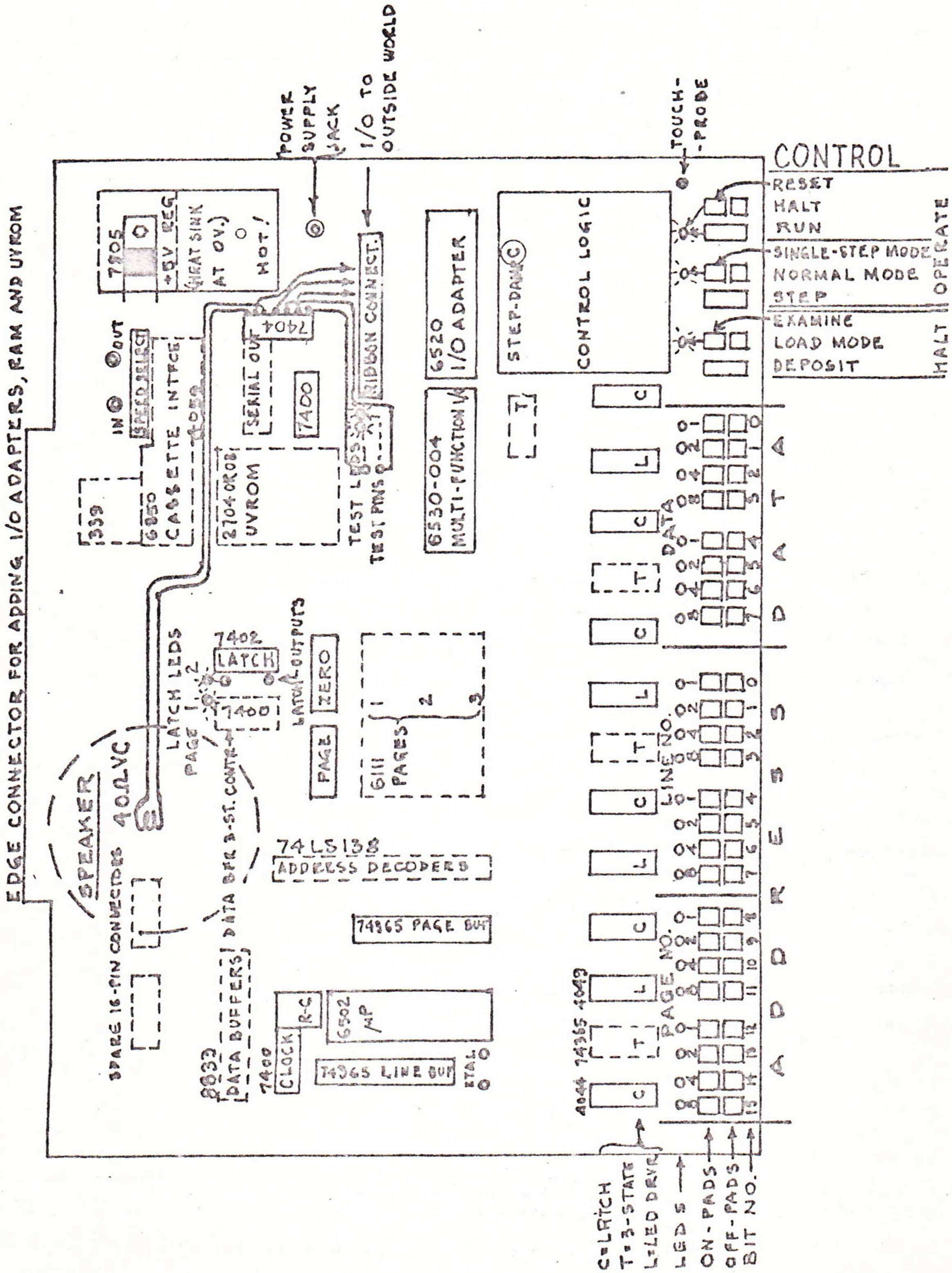


Fig. 21.1 Layout of PIEEE-77 board. Dotted chips and areas are not populated.

OUTPUT LATCH

(14) Finally we will see what makes the two output latch lights operate. Touch the EXamine pad. Now turn the page 1 bit ON and OFF. The latch-1 light should be on. Now turn the page 2 bit ON and OFF. That should turn the latch-2 light on. Latch-1 latches on whenever it passes a page 2 address. One or the other latch light is on at all times.

(15) Now we are familiar with the touch-pads and readout lights required for depositing our first program into the RAM chips.

(A)

PROGRAM 7-BIT ADD. Jump rope.

This is a simple-minded program for adding two 7-bit hex numbers.

(1) turn on the power supply and touch the LOAD PAD.

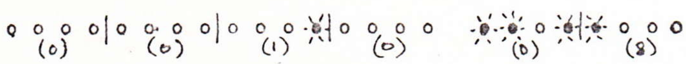
(2) LOAD the following instructions into the RAM chips:

ADDRESS OF REGISTER AT WHICH THIS INSTRUCTION IS TO BE STORED

INSTRUCTION

COMMENTS

0010	D8	This first address and instruction 0010 D8 -- should look like this on your board



(3) Now touch the DEPOSIT pad. This writes the instruction D8 in register 0010 of the RAM chips.

(4) Now write each of the following instructions. Touch the DEPOSIT pad after entering the address and the data on each line.*

		The D8 instruction above means this is a binary (not decimal) procedure.
0011	18	Clear Carry (C=0)
0012	A9	Put the next number into Accumulator A=11
0013	11	This is the first number to be added -- use any number
0014	69	Add the next number to the Accumulator A + 22
0015	22	This is the second number to be added -- use any number.
0016	85	Copy the accumulator into Line 00 of page 0 (0000) = A
0017	00	(Into line 00)
0018	4C	Jump to J rope
0019	18	Line 18
001A	00	of page zero
FFFC	10	When Restart pad is touched, go
FFFD	00	to line 10, page 00.

* In LOAD mode the DATA lights are often confusing. Touching the DEPOSIT pad should make them clear.

(5) Proofread your entries. Touch EXamine pad and one by one enter all the addresses and read the data at each address. If you have made any mistakes, correct them or do them over. Do not give up until you have gone through all the addresses once without errors.

(6) Touch the Reset-and-Run pad and then the HALT pad. The problem is now solved, and the answer is in register 0000. Touch the Examine pad register 0000 and find the answer to be 33 (for 11+22).

(7) Now try other pairs of 7-bit numbers. (What is the highest 7-bit number?) Start with step (2), but no need to repeat the whole program -- just touch the address 0013 and enter new number in the DATA register; then address 0015 and enter your second number.

(B)

DISCUSSION

Line 0010: The 6502 uP chip is capable of operating either with binary numbers or with binary-coded decimal numbers. The op-code (instruction) D8 orders binary operation (in computerese, it resets the "decimal flag" * to zero). Instruction F8 would have made the uP operate with decimal numbers (it sets the "decimal flag" to 1).

Line 0011: In the 6502 chip when an addition is performed, the current value of the Carry flag is added in. Since we are going to perform an addition we have to make sure that the Carry flag is zero.

Line 0012: The instruction A9 says to the 6502 chip: "put the immediately following number A9 into your Accumulator register. The immediately following number is 11 (on Line 0013), so 11 is entered into the Accumulator.

Line 0014: 69 says "add the number on the line immediately following 69 to the Accumulator". The line immediately following is 0015, and the number in it is 22, so 22 is added to the number already in the Accumulator.

Line 0016: op-code 85 says "copy the number in the Accumulator to page 0 of the RAM in the register shown on the line immediately following this line. The line immediately following is 0017, and the register is 00. So the result is copied from the Accumulator into RAM register 0000.

Line 0018: 4C is the op-code for "do not continue going down this program, but instead jump to the line and page in the two lines immediately following this line." The line immediately following is 0019, and the program will now jump to line 18; the page number is on line 001A, and that is page zero. So as soon as the uP has read

* A "flag" is a one-bit register inside the uP chip.

line 1A it knows that it must now jump to line 0018. Note the result: the uP will continuously jump from 001A to 0018, etc. We call it "jumping rope" because like a person jumping rope, the uP jumps and jumps, but stays in one place.

The purpose of jumping rope is to prevent the uP from reading the next line, 001B, finding some noise number there and interpreting it to mean some operation or other. This could be -- and usually is -- disastrous, since it can change the program just written and run itself into the ground. The 6502 does not have an op-code that will halt the uP. (It does, however, have a pin for halting it).

Line FFFC: When the Reset-and-Run pad is touched, the uP automatically looks at address FFFC to see with what line to start the program; the next address (FFFD) furnishes the page number. We have asked the uP to start at address 0010.

ENGINEERING LANGUAGE

Instead of commenting on the op-codes in long-hand, we can use algebra for most of them; for the others we can add some more-or-less obvious symbols which we will call engineering language. This, then, is what the program looks like with engineering language added:

Program 7*BIT*ADD

```
0010  D8   fD=0     clear Decimal flag
0011  18   fC=0     clear Carry flag
0012  A9   A=hh     load A with hh, where
0013  11   hh=11              hh=11
0014  69   A+fC+hh   to Accumulator add fC and
                     hh, where
0015  22     hh=22                 hh=22
0016  85   (Zll)=A   store A in RAM register
                     on page 0,
0017  00   ll=00                   line 00
0018  4C   J:ppll    Jump to program
0019  18     ll=18              line 18
001A  00   pp=00          page 00

FFFC  10   J: 10     Jump to 10, i.e. start
                     with line 10
FFFD  00   00                   on page 00
```

You will note that "f" stands for flag. A is Accumulator. hh is two hex digits. J: means jump to. pp means two .. digits for page number. ll means two hex digits for line number. () means the contents of. This is to avoid confusion between the look-alike number OF register (address) and number IN register (data). A complete list of engineering language symbols is in the frontispiece.

Line 0016: (Zll) means the contents of RAM register ll on page Zero. Notice that page Zero is inherent in the op-code 85; only the line number need be furnished.

SEGREGATING THE OP-CODES

Looking back at the last program listing, note that line 0012 contains an op-code followed by an uncoded hex number on the next line; line 0014 contains an op-code followed by an uncoded hex number on the next line; line 0018 has an op-code followed by uncoded numbers on next two lines. The program listing will become easier to read and to take in with a glance if the lines are rearranged so that every line either consists of an op-code or starts with one, thus:

NOTES

The * is a reminder that there is more than one address on this line. Note that now the first data column contains only op-codes; the second and third data columns contain constants.

Line 0018* illustrates a 6502 peculiarity; rather than the "natural" order of first the page, then the line. The 6502 requires first the line, then the page.

Note also the 2 comments on the right side: they give you a single-glance view of what procedure is followed (regardless of the particular microprocessor chip used). This is called a machine-independent program.

Program SEVEN-BIT-ADD

```
0010    D8 --  --   fD =0      set binary mode     ) . INITIALIZE
                                                   )
0011    18 --  --   fC=0       clear Carry  flag   )
0012*   A9 11  --   A=11       load first number into Accumulator      SUM=FIRST.NUMBER
0014*   69 22  --   fC+22      add second number to Accumulator        SUM+SECOND.NUMBER
0016*   85 00  --   (Z00)=A    copy sum into register 0000
0018*   4C 18  00   jump rope
FFFC*   -- 10  00   J:0010     start on line 0010
```

22.3 GENERATING A CONTINUOUS SQUARE WAVE ON PIA PIN

(A) Program SQ.W.ON.PIA.PIN

```
FFFC* -- 10 00          J:START

0010* A9 01 -- 2 START  A=01
0012* 8D 01 6E 4        (6E01)=A
0015* A9 01 -- 2 TOP    A=01
0017* 8D 00 6E 4        (6E00)=A
001A* A9 00 -- 2        A=00
001C* 8D 00 6E 4        (6E00)=A
001F* 4C 15 00 3        J:TOP
```

```
(6E01)=01     OUTPUT=PIA.PIN
              loop
(6E00)=01     PIA.PIN=ON

(6E00)=00     PIA.PIN=OFF
              pool
```

(B) INSTRUCTIONS

Connect scope to I/O pin A0 of 6530. This pin is available at ribbon connector pin 12.

(C) DISCUSSION

This is the same program as the previous one except that the output is now directed to pin A0 of the 6530 I/O adapter chip. Since there is only one LED -- not two as with the latch -- therefore it is difficult to observe whether there is indeed a square wave at the output. A scope should be connected to the A0 pin.

Line 0010* sets bit 0 for the next line.

Line 0012*: 6E01 is the DIRECTION register of 6530 side A. When we make (6E01)=01, its pin A0 becomes an output; the 7 other pins become inputs.

Line 0015* sets bit 0 for the next line.

Line 0017*: 6E00 is the DATA register for 6530 side A. When we make (6E00)=01, pin A0 goes ON.

Line 001A* resets bit 0 for the next line.

Line 001C*: When we make (6E00)=00, pin A0 goes OFF.

(D) QUESTIONS

(1) If we wrote (6E00)=FF, what would happen to the program?

(2) Why are bits A1 to A7 inputs?

(3) How would the program be changed to read out in side B of 6530?

(4) How would the program be changed to read out in bit 7 of side A?

GENERATING AN AUDIO-FREQUENCY WAVE

(A) Program AUDIO.WAVE

```
FFFC* -- 10 00          J:START

0010                    START
      A9 01 --          A=01        set binary mode ------------ start
      8D 01 6E 4        (6E01)=A    ---------------------------- set OUTPUT.PIN=PIA.B1
      A2 FF -- 2 TOP    X=FF        set counter                  loop
      A9 01 -- 2 ON     A=01                                     loop[FF]
      8D 00 6E 4        (6E00)=A    ---------------------------- OUTPUT=ON
      CA -- -- 2        X-1         count down
      D0 F8 -- 3        if ≠0,J:ON  is count-down done?          pool
      A2 FF -- 2        X=FF        set counter
      A9 00 -- 2 OFF    A=00                                     loop[FF]
      8D 00 6E 4        (6E00)=A    ---------------------------- OUTPUT=OFF
      CA -- -- 2        X-1         count down
      D0 F8 -- 3        if≠0,J:OFF  is count-down done?          pool
      F0 EA -- 3        if =0,J:TOP if yes,start from top        pool
```

USING A TOGGLE

(A) Program TOGGLE.AN.AUDIO.WAVE

```
FFFC* -- 10 00          J:START

0010  D8 -- --    START fD=0 ----------------------------- start
      A9 01 --          A=01
      8D 01 6E 4        (6E01)=A  ------------------------ set OUTPUT.PIN=B1 A0
      EE 00 6E 6  TOGL  (6E00)+1  ------------ loop        OUTPUT=OUTPUT/
      A2 FF -- 2        X=FF
      CA -- -- 2  COUNT X-1                    loop [FF]
      D0 FD -- 2        if≠0,J:COUNT           pool
      F0 F6 -- 2        if =0,J:TOGL           pool
```

GENERATING A SUB-AUDIO SQUARE WAVE

(A) Program TWO.NESTED.LOOPS

```
FFFC* -- 10 00          J:START

0010  D8 -- --    START fD=0
      A9 01 --          A=01
      8D 01 6E 4        (6E01)=A  ------------------------ set OUTPUT.PIN=A0
      EE 00 6E 6  TOGL  (6E00)+1  -------------- loop      OUTPUT=OUTPUT/
      A0 FF -- 2        Y=FF
      A2 FF -- 2  T2    X=FF                     loop[FF]
      CA -- -- 2  COUNT X-1                      loop[FF]
      D0 FD -- 3        if ≠0,J:COUNT            pool
      88 -- -- 2        Y-1
      D0 F8 -- 3        if ≠0,J:T2               pool
      F0 F1 -- 3        if =0,J:TOGL             pool
```

GENERATING A TIME DELAY

(A) Program FOUR.NESTED.LOOPS

```
FFFC* -- 10 00          J:START

0010  D8 -- --    START fD=0
0011  A9 01 --          A=01
      8D 01 6E          (6E01)=A  ---------------------- set OUTPUT.PIN=PIA.A0
      EE 00 6E    TOGL  (6E00)+1  ---------- loop        OUTPUT=OUTPUT/
      A9 FF --          A=FF
      8D 00 00          (0000)=A
      A0 FF --    TIME3 Y=FF                  loop[FF]
      A2 FF --    TIME2 X=FF                  loop[FF]
      CA -- --    TIME1 X-1                   loop[FF]
      D0 FD --          if ≠0,J:TIME1         pool
      88 -- --          Y-1
      D0 F8 --          if ≠0,J:TIME2         pool
      CE 00 00          (0000)-1
      D0 F1 --          if ≠0,J:TIME3         pool
      F0 E7 --          if =0,J:TOGL          pool
```

STING
99 00
01
05
hh ≠ 0

```
0000* ENTER address of first note
0002  ENTER  PERIOD.MULTIPLIER (say, 01)
0003  reserve for DURATION
0004  reserve for PERIOD
0005  ENTER  DURATION.MULTIPLIER (say, 05)
0006  ENTER  REPEAT.ENABLE   (Zero keeps a tune from repeating).
0012  reserve for PERIOD.MULT.COUNTER
0014  reserve for PERIOD.COUNTER
0015  reserve for DURATION.MULT.COUNTER

ADRS OPC OPND T LABEL ENGINEERING INSTRUCTIONS           ALGORITHM
FFFC -- 1D 00          J:START
001D A2 FF --    START X=FF
001F 9A -- --          pS=X
0020 20 3C 00          J:sub INITIALIZE                   INITIALIZE
  23 A0 00 --    BEGIN Y=00                          loop set NOTE.POINTER=0
  25 20 46 00    NEXT  J:sub FETCH.DURATION             loopFETCH.DURATION
  28 C9 00 --          fl:A-0
  2A D0 04 --          if/0,J: PRIOD                     if DURATION=0
  2C A6 06 --          X=(Z06)
  2E D0 F3 --          if/0,J:BEGIN                        if REPEAT.ENABLE=ON
                                                              exit this loop
  30 20 4C 00    PRIOD J:sub FETCH.PERIOD                else FETCH.PERIOD
  33 20 5B 00          J:sub PLAY.NOTE                   PLAY.NOTE
  36 20 87 00          J:sub INCREMENT.NOTE.POINTER     poolINCREMENT.NOTE.POINTER&PAGE
  39 4C 25 00          J:NEXT                    &PAGE pool
                                   INITIALIZE
003C D8 -- --          fD=0    (clear decimal flag)
  3D 18 -- --          fC=0    (clear Carry)
  3E A9 01 --          A=01
  40 8D 01 6E          (6E01)=A    make PA0 an  output     set BIT.A0=OUTPUT
  43 A9 00 --          A=00
  45 60 -- --          J:ret S                           RETURN
                                 FETCH.DURATION
0046 B1 00 --          A=((Z00*)+Y)                      fetch DURATION
  48 85 03 --          (Z03)=A
  4A C8 -- --          Y+1                               NOTE.POINTER+1
  4B 60 -- --          J:ret S                           RETURN
                                  FETCH.PERIOD
004C B1 00 --          A=((Z00*)+Y)                      fetch PERIOD
  4E 20 54 00          J:sub ADJUST.PERIOD               ADJUST.PERIOD
  51 85 04 --          (Z04)=A                           store PERIOD
  53 60 -- --          J:ret S                           RETURN
                                  ADJUST.PERIOD
0054 C9 FF --          fl:A-FF
  56 F0 02 --          if=0,J:AEND                       if PERIOD/FF
  58 69 01 --          A+1                                  PERIOD+1
  5A 60 -- --    AEND  J:ret S                           RETURN
                                   PLAY.NOTE
005B A5 05 --    PLAY      A=(Z05)
  5D 85 15 --          (Z15)=A
  5F A5 03 --    TSET      A=(Z03)                  loop [DURATION.MULTIPLIER]
  61 8D 07 6E          (6E07)=A                            set CLOCK=DURATION
  64 EE 00 6E 6 TOGL   (6E00)+1                          loop set OUTPUT=TOGGLE
  67 20 76 00 6        J:sub TIME.THE.PERIOD                  TIME.THE.PERIOD
  6A AD 07 6E 4        A=(6E07)                               fetch CLOCK
  6D C9 80 -- 2        fl:A-80                                if CLOCK=DONE
  6F D0 F3 -- 3+1      if/0,J:TOGL                       pool   exit this loop
  71 C6 15 -- 5        (Z15)-1                          pool
  73 D0 EA -- 3+1      if/0,J:PLAY
  75 60 -- -- 6        J:ret S                           RETURN
                                 TIME.THE.PERIOD
0076 A5 02 -- 3            A=(Z02)
  78 85 12 -- 3        (Z12)=A
  7A A5 04 -- 3 PRSET      A=(Z04)                  loop [PERIOD.MULT]
  7C 85 14 -- 3        (Z14)=A
  7E C6 14 -- 5 PERD   (Z14)-1                          loop[ADJUSTED.PERIOD]
  80 D0 FC -- 3+1      if/0,J:PERD                       pool
  82 C6 12 -- 5        (Z12)-1
  84 D0 F4 -- 3+1      if/0,J:PRSET                     pool
  86 60 -- -- 6        J:ret S                           RETURN
                            INCREMENT.NOTE.POINTER&PAGE
0087 C8 -- --          Y+1                               if NOTE.POINTER=0
  88 D0 02 --          if/0,J:IEND                         then PAGE+1
  8A E6 01 --          (Z01)+1
  8C 60 -- --    IEND  J:ret S                           RETURN
```

Program MUSIC. An example of audio program is shown in Table 1.

	THE STING dur.	period	MORSE CODE				TICKING OF CLOCK		
0030	20	E3	40	40	dah	0004	C0	01	
	20	D6	10	01			01	10	tick 1
	20	CA	20	40	dit		C0	01	
	40	80	10	01			01	10	tick 2
	20	CA	40	40	dah		C0	01	
	40	80	10	01			01	10	tick 3
	20	CA	20	40	dit		C0	01	
	20	30	30	01			01	10	tick 4
	C0	80	40	40	dah		C0	01	
	20	72	10	01			01	10	tick 5
	20	6B	40	40	dah		C0	01	
	20	65	10	01			01	10	tick 6
	20	80	20	40	dit		C0	01	
	20	72	10	01			01	10	tick 7
	40	65	40	40	dah				
	20	88	40	01			00		end flag
	40	72							
	C0	80							
		00							

Table I. Audio programs.

$\sqrt[12]{2}=1.0594631$

MUSICAL SCALE

NOTE	HEX
C	48
B	4C
A#	51
A	55
G#	5B
G	60
F#	65
F	6B
E	72
D#	78
D	80
C#	88
C	90
B	98
A#	A1
A	AB
G#	B5
G	BF
F#	CA
F	D6
E	E3
D#	F1
D	FF

HALF-TONE=
1.0594631x

20 C
20 C
20 D
20 D
20 E
20 E
40 D

20 48
20 80
20 72
20 D6
20 BF
20 55
20 4C
F 00

PROGRAMMING: ARITHMETIC

ADDING TWO SEVEN-BIT NUMBERS

(A) Program SEVEN.BIT.ADD.1

```
FFFC* -- 10 00          J:0010 (START)

0010  D8 -- --   START  fD=0 ________________________________start
0011  18 -- --          fC=0
0012* A9 11 --          A=11 ________________________________SUM=FIRST.NUMBER
0014* 69 22 --          A+fC+22 _____________________________SUM+SECOND.NUMBER
0016* 85 00 --          (Z00)=A _____________________________SUM.DISPLAY=SUM
0018* 4C 18 00          J rope ______________________________end
```

(B) INSTRUCTIONS
Load the program. RESTART. HALT. Examine 0000 for sun. Enter two other numbers into 0013 and 0015. RESTART and HALT again. Examine 0000 for sum.

(C) QUESTIONS
(1) What is the difference between this program and the one that was used in section 21 to learn the board?
(2) In what registers are the two numbers to be added?
(3) What would happen if there was no fD=0 instructions?
(4) What would happen if there was no fC=0 instruction?
(5) On line 0012, what is the meaning of operand 11?
(6) In line 0014, what is the meaning of operand 22?
(7) In line 0014, why are we adding fC to the sum?
(8) In line 0016, why do we place the sum into register 0000?
(9) In line 0018, when does the uP actually perform the jump?
(10) Why is this a 7-bit program, not 8-bit?
(11) In the algorithm column, why make the sum equal to the first number?
(12) What is the meaning of the astterisk?
(13) What other way is there of clearing fC and fD?
(14) Would you clear the whole flag register if there was a subtraction operation ahead?
(15)

ADDING TWO UNSIGNED SEVEN-BIT NUMBERS FROM PARAMETER REGISTERS

(A) Program SEVEN-BIT.ADD.2

```
0001 ENTER FIRST number
0002 ENTER SECOND number
0000 reserve for SUM.DISPLAY

FFFC* -- 10 00          J:START ______________________ start

0010  D8 -- -- START    fD=0          set binary mode
0011  18 -- --          fC=0          clear Carry flag
0012* AD 01 00          A=(0001)______________________SUM=FIRST
0015* 6D 02 00          A+fC+(0002)___________________SUM+SECOND
0018* 8D 00 00          (0000)=A______________________SUM.DISPLAY=SUM
001B* 4C 1B 00          J rope _______________________end
```

(B) INSTRUCTIONS

Load the program. Enter two numbers into registers 0001 and 0002. RESTART. HALT. Examine register 0000 for sum.

(C) QUESTIONS

(1) What is the difference between this and the previous program, 23.10?
(2) 0012: what is the difference in this instruction?
(3) 0015: what is the difference in this instruction?
(4) 0015: why do we add in the fC-- why not just leave it out?

ADDING TWO UNSIGNED NUMBERS I

(A) Program ADD.-

```
FFFC* -- 10 00          J:START

0001  ENTER number to be added. FIRST.NUMBER
0002  ENTER number to be added. SECOND.NUMBER
0003  reserve for Carry display
0004  reserve for SUM display

0010  D8 -- -- 2 START fD=0 _______________ start
0011  18 -- -- 2       fC=0
0012* AD 01 00 4       A=(0001)
0015* 6D 02 00 4       A+fC+(0002) ________ SUM=FIRST.NUMBER+SECOND.NUMBER
0018* 8D 04 00 4       (0004)=A ___________ SUM.DISPLAY=SUM
001B* A9 00 -- 2        A=0
001D* 69 00 -- 2       A+fC+00
001F* 8D 03 00 4       (0003)=A ___________ CARRY.DISPLAY=CARRY
0022* 4C 22 00 3       J rope _____________ end
```

(B) INSTRUCTIONS

Enter program. Enter two numbers to be added. RESTART. HALT. Examine register 0003 for Carry, 0004 for sum.

(C) QUESTIONS

(1) If there is no Carry, how will that show?
(2) If there is a Carry, how will that show?
(3) 0012: is there another instruction that could have been used to accomplish the same thing? What are the advantages and disadvantages?

ADDING TWO DECIMAL NUMBERS

(A) Program ADD.TWO.DECIMAL.NUMBERS

```
FFFC* -- 02 00          J:START
0002  F8 -- --   START  fD=1
0003  18 -- --          fC=0
0004* A9 d1 --          A=d1
0006* 69 d2 --          A+fC+d2 ____________ SUM=FIRST.DEC.NO+SECOND.DEC.NO
0008* 85 01 --          (Z01)=A ____________ SUM.DISPLAY=SUM
000A* B0 03 --          if fC=1,J:CARRY
000C* 4C 0C 00          J rope _____________ end
000F* E6 00 --   CARRY  (Z00)+1 ____________ CARRY.DISPLAY=CARRY
0011* 4C 11 00          J rope _____________ end
```

(B) INSTRUCTIONS

Load program. Enter the two decimal numbers into registers 0005 and 0007. RESTART. HALT. Examine 0000 for Carry, 0001 for sum.

(C) DISCUSSION

This is a demonstration of how the decimal mode operates in the 6502. One problem with the decimal mode is that there is no flag to alert you in case you have entered a hex number instead of a decimal by mistake. In present-day uP practice there does not appear to be much use for the decimal mode.

Note that in this program no room was left at the top of the page for parameters, and the program starts with line 0002.

(D) QUESTIONS

(1) What is the % register utilization of the decimal mode?
(2) What tells the uP to do this problem with decimal numbers?

ADDING TWO UNSIGNED NUMBERS II

(A) Program ADD.UNSIGNED.NUMBERS

```
0001   ENTER FIRST.NUMBER
0002   ENTER SECOND.NUMBER
0003 reserve for CARRY.DISPLAY
0004   reserve for SUM.DISPLAY

FFFC* -- 10 00          J:START

0010  D8 -- --   START  fD=0
0011  18 -- --          fC=0
0012* A5 01 --          A=(Z01)______________________ SUM=FIRST.NUMBER
0014* 65 02 --          A+fC+(Z02)___________________ SUM+SECOND.NUMBER
0016* 85 04 --          (Z04)=A______________________ SUM.DISPLAY=SUM
0018* 90 07 --          if fC=0,J:NOCAR______________ if CARRY=YES
001A* A9 01 --          A=01
001C* 85 03 --          (Z03)=A______________________   CARRY.DISPLAY=CARRY
001E* 4C 1E 00          J rope_______________________   end
0021* A9 00 --   NOCAR  A=00
0023* 85 03 --          (Z03)=A______________________ CARRY.DISPLAY=NO.CARRY
0025* 4C 25 00          J rope_______________________ end
```

ADDING TWO SIGNED OR UNSIGNED NUMBERS

(A) Program ADD.SIGNED.OR.NOT

```
0001   ENTER FIRST.NUMBER (such as 78)
0002   ENTER SECOND.NUMBER (such as FA)
0003   use for UNSIGNED.CARRY.DISPLAY
0004   use for SIGNED.OVERFLOW.DISPLAY
0005   use for SUM.DISPLAY

00FC* -- 10 00          J:START

0010  D8 -- -- 2 START  fD=0
0011  18 -- -- 2        fC=0
0012* AD 01 00 4        A=(0001)_____________________ SUM=FIRST.NUMBER
0015* 6D 02 00 4        A+fC+(0002)__________________ SUM+SECOND.NUMBER
0018* 8D 05 00 4        (0005)=A_____________________ SUM.DISPLAY=SUM
001B* A9 00 -- 2        A=00
001D  2A -- -- 2        AfC
001E* 8D 03 00 4        (0003)=A_____________________ UNSIGNED.CARRY.DISPLAY=CARRY
0021* A9 00 --           A=00
0023* 8D 04 00           (0004)=A                     SIGNED.OVERFLOW.DISPLAY=NONE
0026* 50 FE --           if fV=0,J rope               if OVERFLOW=NONE end
0028* A9 FE --           A=FE
002A* 8D 04 00           (0004)=A                     SIGNED.OVERFLOW.DISPLAY=OVERFLOW
002D* D0 FE --           if /0,J rope                 end
```

ADDING TWO 16-BIT NUMBERS (UNSIGNED)

(A) Program ADD.16-BIT

```
0000 ENTER FIRST.HIGH
0001 ENTER FIRST.LOW
0002 ENTER SECOND.HIGH
0003 ENTER SECOND.LOW
0004 reserve for CARRY.DISPLAY
0005 reserve for SUM.HIGH.DISPLAY
0006 reserve for SUM.LOW.DISPLAY

00FC* -- 10 00          J:START______________________ start

0010  D8 -- -- 2 START  fD=0
0011  18 -- -- 2        fC=0
0012* AD 01 00 4        A=(0001)_____________________ SUM.LOW=FIRST.LOW
0015* 6D 03 00 4        A+(0003)_____________________ SUM.LOW+SECOND.LOW
0018* 8D 06 00 4        (0006)=A_____________________ SUM.LOW.DISPLAY=SUM.LOW
001B* AD 00 00 4        A=(0000)_____________________ SUM.HIGH=FIRST.HIGH
001F* 6D 02 00 4        A+(0002)_____________________ SUM.HIGH+SECOND.HIGH
0022* 8D 05 00 4        (0005)=A_____________________ SUM.HIGH.DISPLAY=SUM.HIGH
0025* A9 00 -- 2        A=00
0027* 69 00 -- 2        A+fC+00
0029* 8D 04 00 4        (0004)=A_____________________ CARRY.DISPLAY=CARRY
002B* 4C 2B 00          J rope_______________________ end
```

ADDING TWO 16-BIT NUMBERS, SIGNED OR NOT

(A) Program ADD.16-BIT.SIGNED/UNSIGNED

```
0000 ENTER FIRST.HIGH
0001 ENTER FIRST.LOW
0002 ENTER SECOND.HIGH
0003 ENTER SECOND.LOW
0004 reserve for UNSIGNED.CARRY.DISPLAY
0005 reserve for SIGNED.OVERFLOW.DISPLAY
0006 reserve for SUM.HIGH.DISPLAY
0007 reserve for SUM.LOW.DISPLAY

FFFC* -- 10 00          J:START

0010  D8-- --   START  fD=0 ______________ start
0011  18 -- --         fC=0
0012* AD 01 00         A=(0001) __________ SUM.LOW=FIRST.LOW
0015* 6D 03 00         A+(0003) __________ SUM.LOW+SECOND.LOW
0018* 8D 07 00         (0007)=A __________ SUM.LOW.DISPLAY=SUM.LOW
001B* AD 00 00         A=(0000) __________ SUM.HIGH=FIRST.HIGH
001E* 6D 02 00         A+(0002) __________ SUM.HIGH+SECOND.HIGH
0021* 8D 06 00         (0006)=A __________ SUM.HIGH.DISPLAY=SUM.HIGH
0024* A9 00 --         A=00
0026  2A -- --         AfC↻
0027* 8D 04 00         (0004)=A __________ UNSIGNED.CARRY.DISPLAY=CARRY
002A* A9 00 --         A=00
002C* 8D 05 00         (0005)=A __________ SIGNED.OVERFLOW.DISPLAY=NONE
002F* 50 FE --         if fV=0,J rope ____ if OVERFLOW=NONE, end
0031* A9 FE --         A=FE
0033* 8D 05 00         (0005)=A __________ SIGNED.OVERFLOW.DISPLAY=OVERFLOW
0036* D0 FE --         if /0,J rope ______ end
```

ADDING TWO 24-BIT NUMBERS, SIGNED OR NOT

(A) Program ADD.24-BIT.SIGNED/UNSIGNED

```
0000 ENTER FIRST.HIGH
0001 ENTER FIRST.MIDL
0002 ENTER FIRST.LOW
0003 ENTER SECOND.HIGH
0004 ENTER SECOND.MIDL
0005 ENTER SECOND.LOW
0006 reserve for UNSIGNED.CARRY.DISPLAY
0007 reserve for SIGNED.OVERFLOW.DISPLAY
0008 reserve for SUM.HIGH.DISPLAY
0009 reserve for SUM.MIDL.DISPLAY
000A reserve for SUM.LOW.DISPLAY

FFFC* -- 10 00          J:START __________ start

0010  D8 -- --  START  fD=0
0011  18 -- --         fC=0
0012* AD 02 00         A=(0002) __________ SUM.LOW=FIRST.LOW
0015* 6D 05 00         A+(0005) __________ SUM.LOW+SECOND.LOW
0018* 8D 0A 00         (000A)=A __________ SUM.LOW.DISPLAY=SUM.LOW
001B* AD 01 00         A=(0001) __________ SUM.MIDL=FIRST.MIDL
001E* 6D 04 00         A+(0004) __________ SUM.MIDL+SECOND.MIDL
0021* 8D 09 00         (0009)=A __________ SUM.MIDL.DISPLAY=SUM.MIDL
0024* AD 00 00         A=(0000) __________ SUM.HIGH=FIRST.HIGH
0027* 6D 03 00         A+(0003) __________ SUM.HIGH+SECOND.HIGH
002A* 8D 08 00         (0008)=A __________ SUM.HIGH.DISPLAY=SUM.HIGH
002D* A9 00 --         A=00
002F  2A -- --         AfC↻
0030* 8D 06 00         (0006)=A __________ UNSIGNED.CARRY.DISPLAY=CARRY
0033* A9 00 --         A=00
0035* 8D 07 00         (0007)=A __________ SIGNED.OVERFLOW.DISPLAY=NONE
0038* 50 FE --         if fV=0,J rope ____ if OVERFLOW=NONE, end
003A* A9 FE --         A=FE
003C* 8D 07 00         (0007)=A __________ SIGNED.OVERFLOW.DISPLAY=OVERFLOW
003F* D0 FE --         if /0,J rope ______ end
```

#1 ADDING TWO UNSIGNED 4-DIGIT DECIMAL NUMBERS

(A) Program ADD.4-DIGIT.DECIMAL.NUMBERS

```
0000 ENTER FIRST.HIGH
0001 ENTER FIRST.LOW
0002 ENTER SECOND.HIGH
0003 ENTER SECOND.LOW
0005 reserve for CARRY.DISPLAY
0006 reserve for SUM.HIGH.DESPLAY
0007 reserve for SUM.LOW.DISPLAY

FFFC* -- 10 00          J:START________________________________start

0010  F8 -- --   START  fD=1    set decimal mode
0011  18 -- --          fC=0
0012* AD 01 00          A=(0001)_____________________________SUM.LOW=FIRST.LOW
0015* 6D 03 00          A+(0003)_____________________________SUM.LOW+SECOND.LOW
0018* 8D 07 00          (0007)=A ____________________________SUM.LOW.DISPLAY=SUM.LO

001B* AD 00 00          A=(0000)_____________________________SUM.HIGH=FIRST.HIGH
001E* 6D 02 00          A+(0002)_____________________________SUM.HIGH+SECOND.HIGH
0021* 8D 06 00          (0006)=A ____________________________SUM.HIGH.DISPLAY=SUM.HIGH

0024* A9 00 --          A=00
0026* 69 00 --          A+fC+00
0028* 8D 05 00          (0005)=A ____________________________CARRY.DISPLAY=CARRY
002B* A9 00 --          A=00
002D* F0 FE --          if =0,J rope ________________________end
```

#2 MULTIPLYING TWO HEX DIGITS BY SUCCESSIVE ADDITION

(A) Program MULT.2.DIGITS.BY.ADDITION

```
0001 ENTER FIRST digit thus: 0X
0002 ENTER SECOND digit, thus: 0Y
0003 reserve for PRODUCT.DISPLAY

FFFC* -- 10 00          J:START ____________________________ start

0010  18 -- --   START  fC=0
0011  D8 -- --          fD=0
0012* A6 02 --          X=(0002)
0014* A9 00 --          A=00________________________________ PRODUCT=0
0016* E0 00 --          fl:X-00
0018* F0 07 --          if =0,J:STORE
001A* 65 01 --   CUMUL  A+fC+(Z01)_______________ loop[SECOND]
001C  CA -- --          X-1                            PRODUCT+FIRST
001D* D0 FC --          if /0,J:CUMUL_____________ pool
001F* EA EA
0021* 85 03 --   STORE  (0003)=A __________________________ PRODUCT.DISPLAY=PRODUCT
0023* 4C 23 00          J rope ____________________________ end
```

#3 DETERMINING WHICH OF TWO UNSIGNED NUMBERS IS LARGER

(A) Program COMPARE

```
0000  reserve for DISPLAY.RESULT (if FIRST is larger, 01; if second, 02; if equal, 03)
0001  ENTER FIRST number
0002  ENTER SECOND number

FFFC* -- 10 00          J:START

0010  D8 -- --   START  fD=0
0011  38 -- --          fC=1
0012* A9 00 --              A=00
0014* 85 00 --          (Z00)=A
0016* A5 01 --          A=(Z01)
0018* C5 02 --          fl:A-(Z02)____________try FIRST-SECOND
001A* F0 10 --          if =0,J:EQUAL ________if /0
001C* B0 07 --          if fC=1,J:FIRST_______   if BORROW=YES
001E* A9 02 --          A=02 _________________      DISPLAY=SECOND.IS.LARGER; end
0020* 85 00 --          (Z00)=A ______________else DISPLAY=FIRST.IS.LARGER ; end
0022* 4C 22 00          J rope _______________else DISPLAY=EQUAL; end
0025* A9 01 --   FIRST  A=01
0027* 85 00 --          (Z00)=A
0029* 4C 29 00          J rope _______________
002C* A9 03 --   EQUAL  A=03
002E* 85 00 --          (Z00)=A
0030* 4C 30 00          J rope _______________
```

#1 GENERATING A RANDOM NUMBER

(A) Program ULTRA.SIMPLE.RANDOM.NOISE.GENERATOR

```
0000 store random number

FFFC* -- 10 00          J:START

0010* C6 00 --     START (Z00)-1 <-------------------- loop RANDOM.NUMBER-1
0012* 4C 10 00           J:START ---------------------- pool
```

(B) PROCEDURE
Enter program, set switch register to 0020. RESET. RUN. EXAMINE. RUN and EXAMINE again.

(C) DISCUSSION
This program decrements a register until you halt it to examine. It will then display a random number between 00 and FF. The number is random because the uP loops many thousands of times before you can possibly stop it.

Advantage is taken of the fact that the switch register at the PIEEE-77 board is latched to the register it examined last. By repeatedly touching RUN and EXAMINE, you can generate a sequence of random numbers.

#2 GENERATING A DECIMAL RANDOM NUMBER WITHIN STATED LIMITS

(A) Program DECIMAL.RAND.NUM.GEN

```
0000  reserve for DISPLAY
  01  ENTER upper limit UPLIM, such as 77    ENTER: UPPER.LIMIT as decimal number
  02  ENTER lower limit LWLIM, such as 43    ENTER: LOWER.LIMIT as decimal number

FFFC* -- 10 00         J:START
0010  F8 -- --         fD=1
0011  38 -- --         fC=1
  12  A5 01 -- RESET   A=(Z01)      <-- loop        COUNTER=UPPER.LIMIT
      E9 01 -- DECRM   A-fC/-1      <--    loop     COUNTER-1
      85 00 --         (Z00)=A                      DISPLAY=COUNTER
      C5 02 --         fl:A-(Z02)
      D0 F8 --         if/0,J:DECRM -->    pool if COUNTER=LOWER.LIMIT, exit this loop
      4C 12 00         J:RESET      --> pool
```

(B) PROCEDURE: Enter UPPER.LIMIT. LOWER.LIMIT: set address to 0000. RESET. RUN. HALT. EXAMINE: Data readout now displays a random decimal number between UPPER.LIMIT AND LOWER.LIMIT To generate another random number, RESTART and HALT.

(C) DISCUSSION: This program counts down a register over and over again. The number remaining in the register when the computer is stopped is random because a random number of clock cycles has passed. The switch register remembers the register it looked at last, and the program takes advantage of this to display the result without the need to readdress it each time the program is run.

(D) QUESTIONS

1 What happens if a non-decimal number is entered for UPPER.LIMIT? for LOWER.LIMIT?

2 What happens if LOWER.LIMIT is larger than UPPER.LIMIT?

3 This generator is not truly random. It shows a bias in favor of LOWER.LIMIT. Why? What% bias?

4 What will make the numbers more truly random?