HEATHKIT
CONTINUING
EDUCATION

Individual Learning Program

MICROPROCESSORS

Unit 10

INTERFACING EXPERIMENTS

EE-3401

HEATH COMPANY
BENTON HARBOR, MICHIGAN 49022

CONTENTS

Unit 10

INTERFACING EXPERIMENTS

INTRODUCTION

This Unit contains nine interfacing experiments that are to be assembled and run on the Microprocessor Trainer. Most of the circuit parts for these experiments are supplied with this course. The remaining parts were part of the Trainer Kit.

You will be instructed to perform these experiments at the end of Units 7 and 8. Do not confuse them with the programming experiments in Unit 9. When you complete an experiment, you will be directed to the next experiment, or back to the Unit Activity Guide of the unit that directed you to the experiment.

If your Trainer is Model Number ET-3400, and has been modified for use with the Heathkit Memory I/O Accessory, Model ETA-3400, disconnect the 40-pin plug that connects the Trainer to the Memory I/O Accessory. Then reinstall the 2112 RAM IC's at IC-14 through IC-17 before starting the experiments in this unit.

If your Trainer is model number ET-3400A, and has been modified for use with the Heathkit Memory I/O Accessory, disconnect the 40-pin plug that connects the Trainer to the Memory I/O Accessory. Then reinstall the 2114 RAM IC's at IC14 and IC15 before starting the experiments in this unit.

Experiment 1
MEMORY CIRCUITS

OBJECTIVES:

Show how memory circuits can be connected to a microprocessor.

Demonstrate timing requirements when using memory circuits.

Show how data is stored and read from memory circuits.

Demonstrate an elementary memory test to ensure proper, reliable operation.

Introduction

In this experiment, you will construct a memory on the large connector block of the Microprocessor Trainer and interface the circuit with the Trainer circuits.

The initial sections of the experiment will examine memory and its characteristics. The remaining experiment section will interface the memory with the microprocessor and its support circuits. This program and all remaining hardware experiment programs will use a computer print-out listing. A detailed explanation of how to read the listing will be given later in the experiment.

TRAINER POWER REVIEW

With the Trainer plugged in and the Power switch off, the display LED's and the +5, +12, and −12 volt connector blocks are disconnected from Trainer power. The single LED next to the Power switch indicates this condition. Whenever you make connections between the Trainer and the large connector block, **always** switch the power off. This will not disturb any program stored in the Trainer. If you must remove or install a component in the Trainer circuits, such as an IC, remove the power plug from the wall receptacle.

Material Required

1 Microprocessor Trainer

1 1000 ohm, 1/4-watt, 10% resistor

1 Pushbutton switch (#1)

1 7400 integrated circuit (443-1)

1 74126 integrated circuit (443-717)

1 74LS30 integrated circuit (443-732)

1 74LS27 integrated circuit (443-800)

Hookup wire (22 gauge, solid)

1 IC puller tool } From Trainer Parts Package

Hookup wire (22 gauge, solid) }

ADDITIONAL MATERIAL REQUIRED (TRAINER ET-3400A)

2 2112-2 IC's (Heath Number 443-721)

Procedure

1. Turn the Trainer power off, then unplug your Trainer from its wall receptacle.

2. Insert the 74126 (443-717) and 7400 (443-1) integrated circuits (IC's) into the large connector block as shown in Figure 10-1. Always install an IC in the block with pin 1 toward the left.

NOTE: If you are using Trainer model number ET-3400, perform step 3. If your Trainer is an ET-3400A, perform step 3A. All other steps of this procedure are common to both Trainer types, except where indicated.

3. Using the IC puller tool, remove the 2112 (443-721) IC from its socket at location IC17. (Observe the precautions described in Unit 9 for MOS devices.) Then insert the IC into the large connector block as shown in Figure 10-1. This IC will be reinstalled in the ET-3400 Trainer in a later experiment.

3A. Locate a 2112-2 IC (Heath number 443-721) that was supplied with this course. Notice that this IC is packed in conductive foam.

NOTE: These IC's are rugged, reliable components. However, normal static electricity discharged from your body through an IC pin to an object can damage the IC. Install these IC's without interruption as follows:

- Remove the IC from its package with both hands.
- Hold the IC with one hand and straighten any bent pins with the other hand.
- Insert the IC into the large connector block as shown in Figure 10-1.

4. Install the pushbutton switch at the location shown in Figure 10-1. Be sure to press straight down when you insert the switch leads — they are fragile.

5. Using the 22 gauge, solid hookup wire, interconnect the IC's and switch as shown in Figure 10-1. Install the 1000 ohm, 1/4-watt, 10% resistor at this time also.

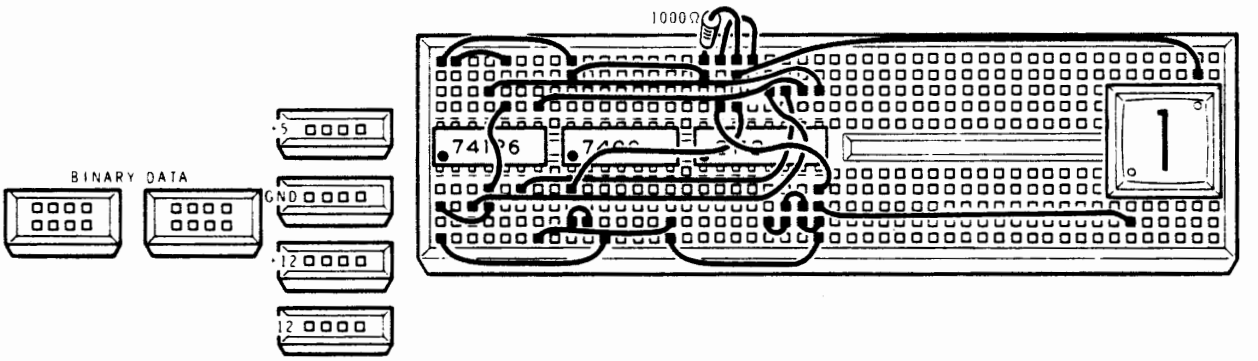


Figure 10-1
Part A of wire interconnect diagram.

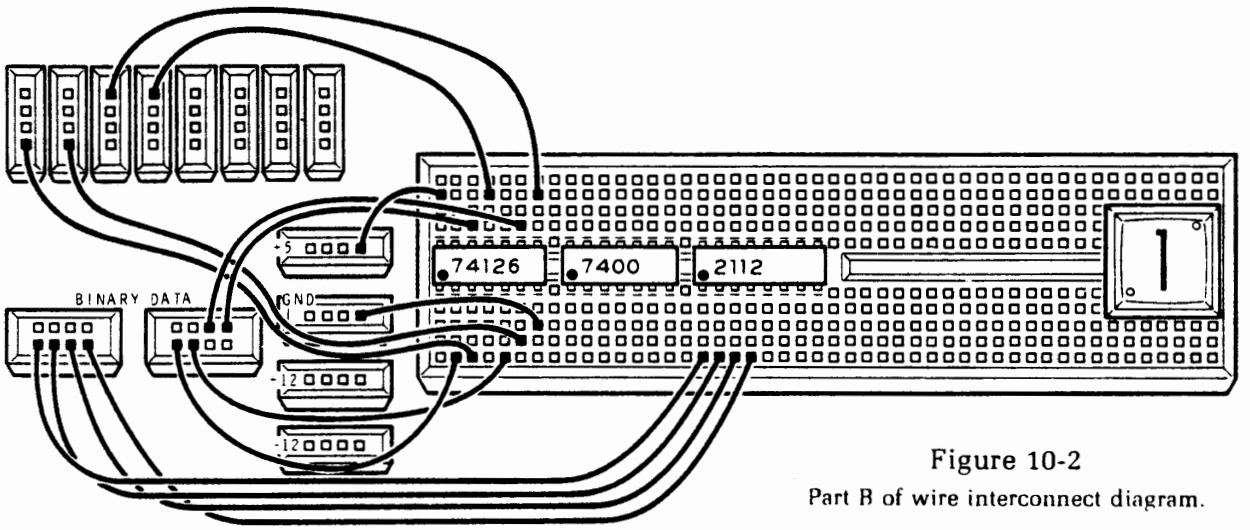


Figure 10-2
Part B of wire interconnect diagram.

6. Refer to Figure 10-2 and install hookup wire as shown. Now compare your circuit with the circuit shown in Figure 10-3. Figures 10-1 and 10-2 are supplied to familiarize you with the proper wiring technique. The remaining hardware experiments will show only the circuit diagram.

7. Connect your Trainer line cord plug to a wall receptacle, and switch the circuit power on. The four data LED's may or may not be randomly lit.

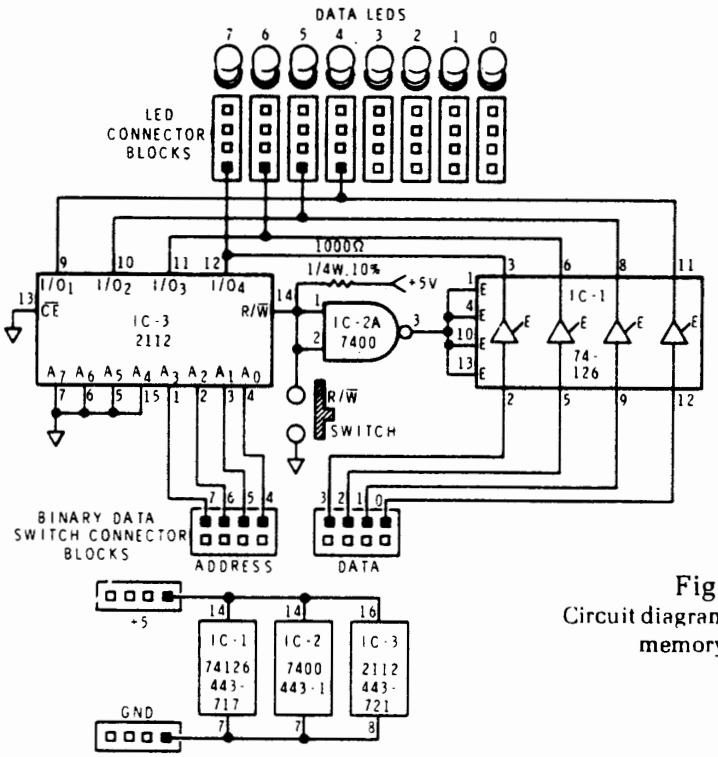


Figure 10-3
Circuit diagram of the first part of the memory experiment.

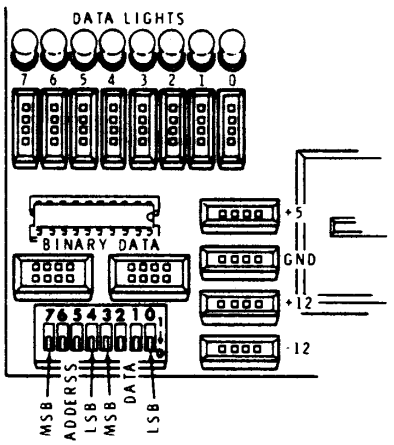


Figure 10-4
Binary data switch functions.

NOTE: The slide switch assembly in the lower left corner of the Trainer (Figure 10-4) is used to control address and data information in this experiment. The first four switches (0 thru 3) control data, while the next four (4 thru 7) control the address. Each group is arranged in a binary sequence with the least significant bits at 0 (data) and 4 (address). The physical position of each switch indicates a logic level; up for a logic 1 and down for a logic 0.

The pushbutton switch mounted on the large connector block functions as the memory Read/$\overline{\text{Write}}$ switch. In its out (off) position, the memory is in the read mode, and the four data LED's will display data stored in memory. When the R/$\overline{W}$ switch is pressed, the memory goes to the write mode, and the value stored in the four data switches is read into memory. The four data LED's immediately display the new memory data.

8. Set the four address switches to 0000_2, and the four data switches to 0000_2. Then press the R/$\overline{W}$ switch. If any of the data LED's were previously lit, they should now be out. This indicates that 0000_2 is now stored at address 0000_2.

9. Now select address 0001_2 and set the data switches to 0001_2. The display will show a random value produced at power-on. Press the R/$\overline{W}$ switch. The data lights will show 0001_2, which is now stored in memory at address 0001_2.

10. Refer to Figure 10-5 and use the slide switch assembly to load the remaining 14_{10} address locations (2 thru 15_{10}) with the data specified. NOTE: To write data in memory; select the address, select the data value, and load the data by pressing the R/$\overline{W}$ switch.

11. With the address switches, select memory location 9_{16}. The displayed data is $\_\_\_\__2$. Since the 16 memory locations contain a data value that matches the address, the displayed value should equal 9_{16}. Randomly select various memory locations. As each location is selected, the stored data will be displayed.

ADDRESS HEX	ADDRESS BINARY	DATA BINARY
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1
4	0 1 0 0	0 1 0 0
5	0 1 0 1	0 1 0 1
6	0 1 1 0	0 1 1 0
7	0 1 1 1	0 1 1 1
8	1 0 0 0	1 0 0 0
9	1 0 0 1	1 0 0 1
A	1 0 1 0	1 0 1 0
B	1 0 1 1	1 0 1 1
C	1 1 0 0	1 1 0 0
D	1 1 0 1	1 1 0 1
E	1 1 1 0	1 1 1 0
F	1 1 1 1	1 1 1 1

Figure 10-5
First data table for memory storage experiment.

12. Refer to Figure 10-6 and enter the specified data for each address location.

13. Randomly select a number of memory locations and note the stored data. You probably recognized the relationship between address and data while you entered the data. If not, do you now? The data corresponds to the 1's complement of the address. This circuit is being used as a look-up table. By selecting an address, you can retrieve data previously stored away. Similarly, this circuit can be used for code conversion. Using a binary sequence (as in Figures 10-5 and 10-6) for addressing, a value code can be stored to represent the address (for example, the Gray code as described in Unit 1). Thus, to find the Gray code value of 8_{16}, simply examine memory location 8_{16}. A second memory circuit can then be used to reconvert the code by using the Gray code values (in this example) for address locations, and the binary values for data.

ADDRESS HEX	ADDRESS BINARY	DATA BINARY
0	0 0 0 0	1 1 1 1
1	0 0 0 1	1 1 1 0
2	0 0 1 0	1 1 0 1
3	0 0 1 1	1 1 0 0
4	0 1 0 0	1 0 1 1
5	0 1 0 1	1 0 1 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 0 0
8	1 0 0 0	0 1 1 1
9	1 0 0 1	0 1 1 0
A	1 0 1 0	0 1 0 1
B	1 0 1 1	0 1 0 0
C	1 1 0 0	0 0 1 1
D	1 1 0 1	0 0 1 0
E	1 1 1 0	0 0 0 1
F	1 1 1 1	0 0 0 0

Figure 10-6
Second data table for memory storage experiment.

Discussion

In this section of the experiment, you used a 256 × 4-bit RAM integrated circuit. An IC of this type will have eight address pins and four I/O (input/output) pins through which a 4-bit data word may be stored (written) or read. The direction of I/O flow is determined by the R/$\overline{W}$ (read/$\overline{write}$) pin logic level. A logic 1 on this pin defines the four I/O pins as outputs, thus placing the IC in its **read** mode. A logic 0 on the R/$\overline{W}$ pin defines the four I/O pins as inputs, thus placing the IC in its **write** mode. A $\overline{chip\ enable}$ (CE) pin allows the IC to be enabled or disabled without disturbing its memory contents. This feature will be more meaningful in a later experiment.

You also used a 3-state buffer array (four buffers) with the memory circuit. This is necessary to isolate the data switches when the memory is in its read mode. Each buffer acts as an open circuit at its output pin unless a logic 1 is applied to each enable (E) pin. As shown in Figure 10-3, when the R/$\overline{W}$ switch is pressed, the memory R/$\overline{W}$ line goes low (write mode), and the enable lines to the buffers go high (through NAND gate IC-2A). Switch data is coupled through the buffers and is written into memory. The LED's display the data value. When the R/$\overline{W}$ switch is released, the memory returns to its read mode and the buffers return to their open circuit condition. The LED's now display the data value stored in memory.

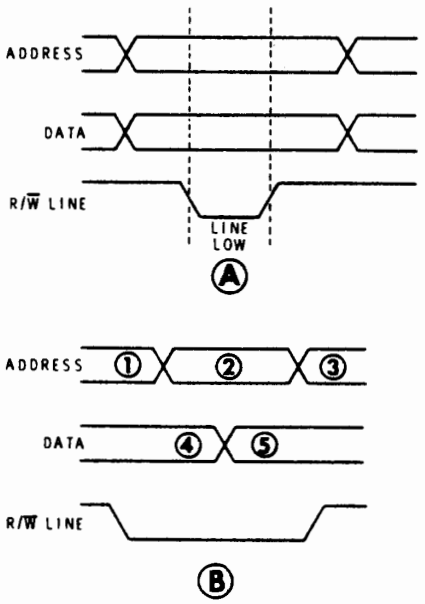


Figure 10-7
Memory write timing diagram.

Writing into memory requires careful timing. Normally the address and data lines must be stable while the R/$\overline{W}$ line is pulsed low, as shown in Figure 10-7, part A. Actually, the data is stored as soon as the R/$\overline{W}$ line reaches a logic 0 level. If the data changes value while the R/$\overline{W}$ line is low, the memory will store the new data. In like manner, if the address changes with the R/$\overline{W}$ line low, the same data will be stored at the new address.

Figure 10-7 part B shows an extreme example of **improper** timing. At address condition 1, data 4 will be stored. Then data 4 will be stored at address 2. However, while at address 2, data changes to condition 5. Therefore, data 5 is now stored at address 2. Finally, data 5 is stored at address condition 3.

Procedure (continued)

14. Turn the Trainer power off, then pull the power plug from the wall receptacle.

NOTE: If you are using Trainer model number ET-3400, perform step 15. For model number ET-3400A, perform step 15A. All other steps of this procedure are common to both trainers, except where indicated.

15. Using the IC puller tool, remove memory IC2112 (443-721) from its socket at location IC16. (Observe the necessary precautions for handling MOS devices.) Then insert the IC into the large connector block, next to the other memory IC.

15A. Locate the second IC 2112 (443-721) that was supplied with this course. (Observe the necessary precautions for handling MOS devices.) Then insert the IC into the large connector block, next to the other memory IC.

16A. Using hookup wire, rewire the connector block IC's and Trainer to form the circuit shown in Figure 10-8. Notice that most of the circuitry is identical to the first circuit you built. You may find it helpful to trace each on the Figure with a red pencil, as you install the wire.

16B. Connect your Trainer line cord plug to a wall receptacle, and switch the circuit power on.

17. The address, data, and R/$\overline{W}$ switches function as before. Refer to Figure 10-9 and enter the data shown, beginning at address 0000_2. NOTE: Data LED0 will be lit for the first eight address locations. Data LED1 will be lit for the last eight address locations.

18. Data LED's 0 and 1 indicate which memory IC is enabled. LED0 lights for IC4 and LED1 lights for IC3. Examine a number of addresses between 0000_2 and 0111_2. Notice which memory IC is enabled. The data stored should match the address. Now examine a number of addresses between 1000_2 and 1111_2. Notice which memory IC is enabled. The data stored should be the 1's complement of the address.

19. Switch the Trainer power off for a few seconds, then switch it back on. Examine a number of memory locations. Have their contents been altered? Do data LED's 0 and 1 still indicate which memory IC is enabled?

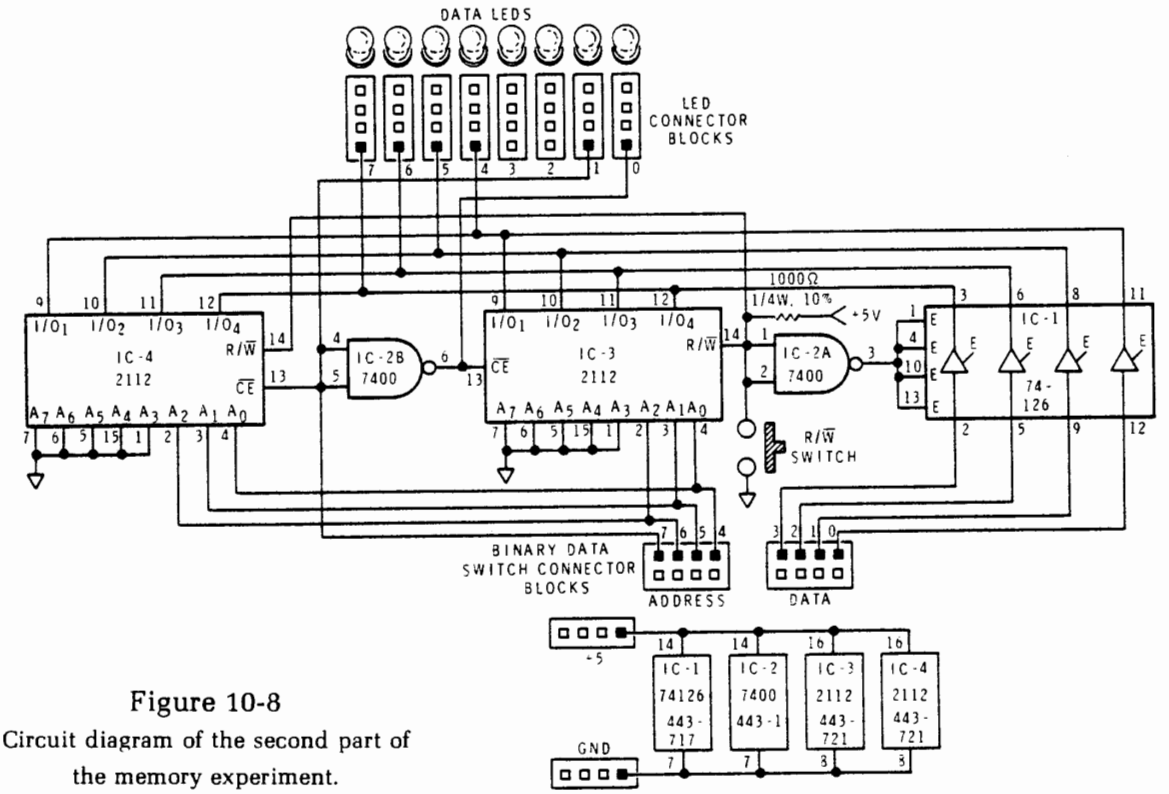


Figure 10-8
Circuit diagram of the second part of the memory experiment.

Discussion

Refer to Figure 10-8 and note how data LED's 0 and 1 are connected. They indicate which memory IC is enabled by the MSB address switch. Since IC3 and IC4 share address lines 4, 5, and 6, the chip enable ($\overline{CE}$) pins determine which IC is enabled for data transfer. The most significant address line is connected to pin 13 of IC4 and its complement is connected to pin 13 of IC3. Therefore, when bit 4 (switch 7) of the address is logic 0, IC4 is enabled; and when bit 4 is logic 1, IC3 is enabled.

When either memory IC is disabled, through pin $\overline{CE}$, stored data remains unaffected. The 3-state output pins go to an open circuit condition. This insures that only one memory IC on the common address lines will be active for data transfer.

RAM, unlike ROM, has a volatile memory. Therefore, when power is lost (even momentarily) data stored in memory is no longer valid. However, data can be reentered into memory and retained as long as power remains on.

ADDRESS HEX	ADDRESS BINARY	DATA BINARY
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1
4	0 1 0 0	0 1 0 0
5	0 1 0 1	0 1 0 1
6	0 1 1 0	0 1 1 0
7	0 1 1 1	0 1 1 1
8	1 0 0 0	0 1 1 1
9	1 0 0 1	0 1 1 0
A	1 0 1 0	0 1 0 1
B	1 0 1 1	0 1 0 0
C	1 1 0 0	0 0 1 1
D	1 1 0 1	0 0 1 0
E	1 1 1 0	0 0 0 1
F	1 1 1 1	0 0 0 0

Figure 10-9
Third data table for memory storage experiment.

Procedure (continued)

20. Switch the Trainer power off.

21. Remove the hookup wires (save them), pushbutton switch, resistor, and IC1 and IC2, used in the previous experiment, from the Trainer.

22. Refer to Figure 10-10, install the IC's, and wire the circuit to the Trainer. Caution: When you install the IC's, leave an extra set of holes between IC2 and IC3. You will be replacing the 14-pin IC2 with a 16-pin IC at a later time.

23. The memory circuit you have wired now interfaces with the microprocessor and will allow data transfer from address 0200_{16} through $02FF_{16}$. Use the Trainer Examine function and randomly select an address in this memory block. Change the data at this location to AA_{16}. Press the FWD key, then press the BACK key. Is the memory content still AA_{16}?

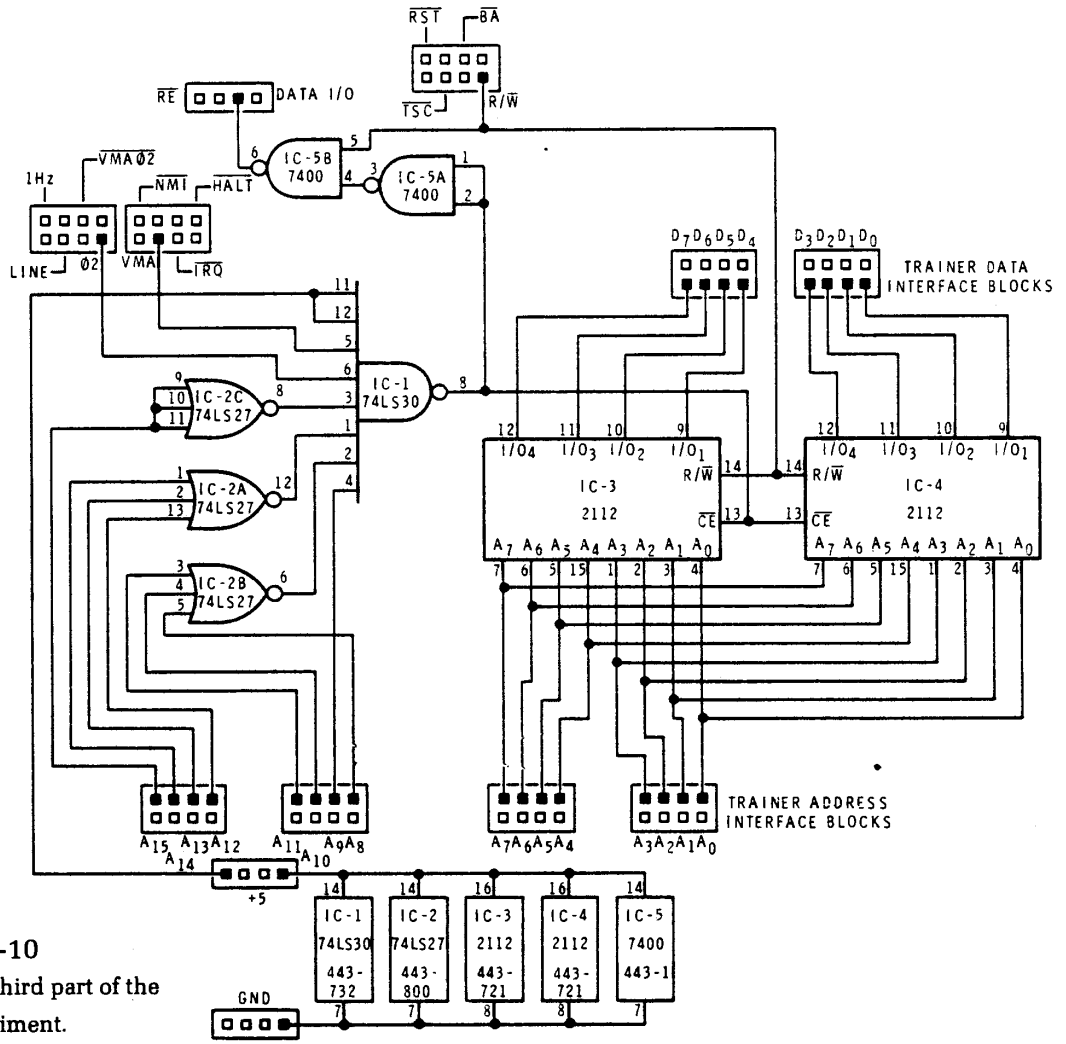


Figure 10-10
Circuit diagram of the third part of the memory experiment.

24. Examine address 0300_{16}. Now change the contents to AA_{16}. Press the FWD key, then press the BACK key. Is the memory content still AA_{16}.

    The data you entered in the previous step was retained because of the memory circuit you wired into the Trainer. The data at location 0300_{16} was not retained because no memory exists for that location.

25. Examine address 0200_{16} and change its contents to AA_{16}.

26. Without switching the Trainer power off, interchange the D_6 and D_7 wires at the Data Interface Block, as shown in Figure 10-11.

27. Press the FWD key, then press the BACK key. The indicated data is now $6A_{16}$. By interchanging the sixth and seventh data bits, $\underline{1}0\underline{1}0\ 1010_2$ became $\underline{0}1\underline{1}0\ 1010_2$. The memory IC still retains the orginal data you programmed. To see this relationship, return the two wires to their orginal position. Since the display has "latched in" the previous data, press the FWD key and then the BACK key. The correct memory contents are now displayed.

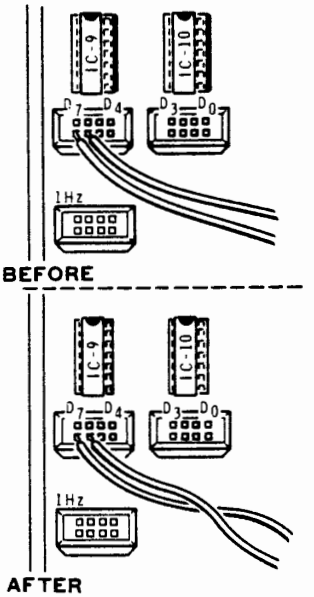


Figure 10-11
Physical manipulation of data by interchanging data lines.

Discussion

IC1 and IC2 shown in Figure 10-10, form the address decoder. The inputs of this decoder are connected to address lines A_8 thru A_{15}, ϕ_2, and VMA.

To enable the two memory IC's, the $\overline{CE}$ pins must be at logic 0. This will occur when all of the inputs to IC1 are at logic 1. Therefore, only address $0000\ 0010_2$ (02_{16}) will decode properly. This is the high order byte of a 16-bit address. ϕ_2 and VMA will go to a logic 1 sometime during a proper address cycle. When this occurs, the output of IC1 will go to a logic 0, and memory will be enabled.

The low order byte of the 16-bit address determines the memory location in IC3 and IC4 where data will be stored or retrieved. Since each memory IC can store only four bits of data, two IC's are connected in parallel. Thus, 256_{10} 8-bit data words can be stored from address 0200_{16} thru $02FF_{16}$. Therefore, in the circuit shown in Figure 10-10, address bits A_0 thru A_7 select the memory location and address bits A_8 thru A_{15} select the specific memory IC's to be enabled.

Data flow direction is determined by the R/$\overline{W}$ line. When this line is at logic 0, the MPU can write into memory. When this line is at logic 1, the MPU can read from memory.

In small microprocessor systems (less than ten devices), the MPU and its support IC's can be connected directly together. However, when more circuits are added to the address and data busses, the MPU can no longer supply the necessary current.

To solve this problem, bus extenders (buffers) are connected between the MPU and most of the surrounding IC's. These supply the necessary drive. Figure 10-12 illustrates a typical circuit.

The address bus extender is uni-directional. That is, it passes a signal in only one direction. It consists of 16 individual buffer drivers; one for each address line.

Unlike address signals, data signals can originate at the MPU or in peripheral circuits such as memory. Therefore, *bi-directional* bus extenders are required. Each data bus extender consists of two 3-state buffer drivers wired back-to-back as shown in Inset 2 of Figure 10-12. The 3-state feature in this case is complementary. That is, when the $\overline{\text{read enable}}$ ($\overline{\text{RE}}$) control line is low, the read buffer is enabled, and when the $\overline{\text{RE}}$ line is high, the write buffer is enabled. Remember, the terms read and write are always expressed in relation to the MPU.

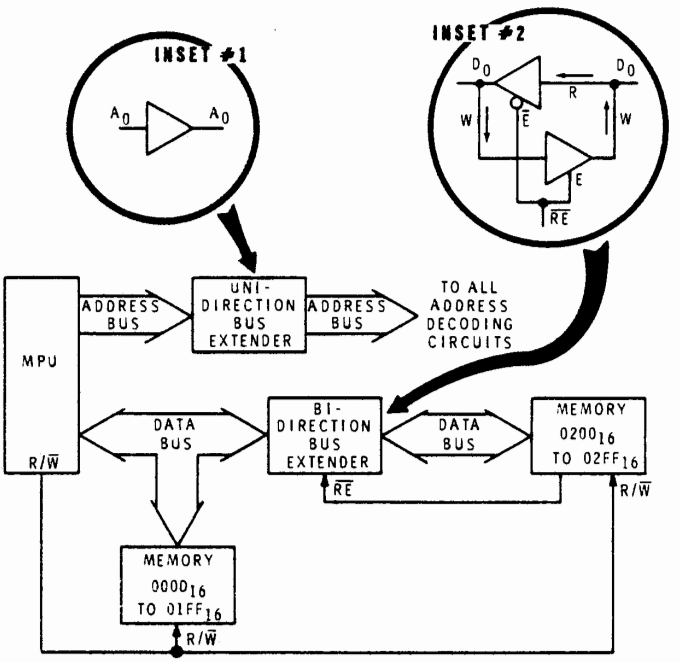


Figure 10-12
Block diagram of typical microprocessor system using bus extenders.

You may wonder why an $\overline{RE}$ line is required in addition to the R/$\overline{W}$ line. It wouldn't be, if all the memory and any other bus operated devices were electrically connected **outside** of the bus extenders. However, in most systems, there are some circuits connected directly to the MPU, with additional circuits connected to bus extenders. Thus, the $\overline{RE}$ control is necessary for valid logic transfer.

During a read cycle, data is transferred to the MPU from memory or other support devices. When a memory location "down line" from the bus extender is addressed, the bus extender will receive an $\overline{RE}$ logic 0 signal, which enables the buffer in the "read" direction. All devices not addressed, before or after the bus extender, will be 3-stated into their open circuit configuration. In this case, the R/$\overline{W}$ control could have been inverted and connected to the bus extender $\overline{RE}$ input. However, if memory "up line" from the bus extender (MPU side) is addressed, the bus extender must remain in its "write" configuration. It could not do this if the inverted R/$\overline{W}$ control line was used.

Since all of the devices "down line" are 3-stated to their open-circuit condition, the input to the "read" buffer of the bus extender would be undefined and its output would assume a logic level (usually logic 1 for TTL gates) that could interfere with data transfer. By using an $\overline{RE}$ control signal not totally defined by the R/$\overline{W}$ control, the bus extender can be forced into its "write" state and prevent any "down-line" interference.

The circuit you constructed from Figure 10-10 used the $\overline{CE}$ and R/$\overline{W}$ signals to produce the necessary $\overline{RE}$ signal. This is then used to control the bus extender where it interfaces the data connector blocks with the MPU data bus in the Trainer. As shown in Figure 10-10, the $\overline{CE}$ signal from pin 8 of IC1 is inverted by IC5A. Thus, when memory IC's 3 and 4 are enabled and the R/$\overline{W}$ line is logic 1, the output of IC5B is logic 0, which enables the bus extender "read" buffer. If IC's 3 and 4 are not enabled, the $\overline{RE}$ line remains high regardless of R/$\overline{W}$ level, and the bus extender remains in its "write" condition.

The final section of this experiment will examine a method for determining the validity of memory. First however, you will learn to read an assembled program listing. This format will be used from now on in these experiments. What you will see is a photocopy of each program as it is assembled and printed by a computer. This serves two purposes: First, it insures that no typographical errors have been introduced during manual reproduction. Second, it gives you an opportunity to become familiar with the format used by most periodicals and books for program listing.

All of the following programs were assembled with a Motorola EXORciser® and printed with a Digital Equipment Corporation decwriter II®. As shown in Figure 10-13, each listing can be divided into two main sections. The right half contains the assembly language program just as the programmer typed it into the computer. The left half of the listing was produced (assembled) from the data in the right half of the listing. This half contains the machine language code that must be entered into the Microprocessor Trainer.

The program listing contains eight columns of information. A brief explanation of each follows.

- Column 1 is a sequential list of numbers produced only as a reference to identify listing lines.

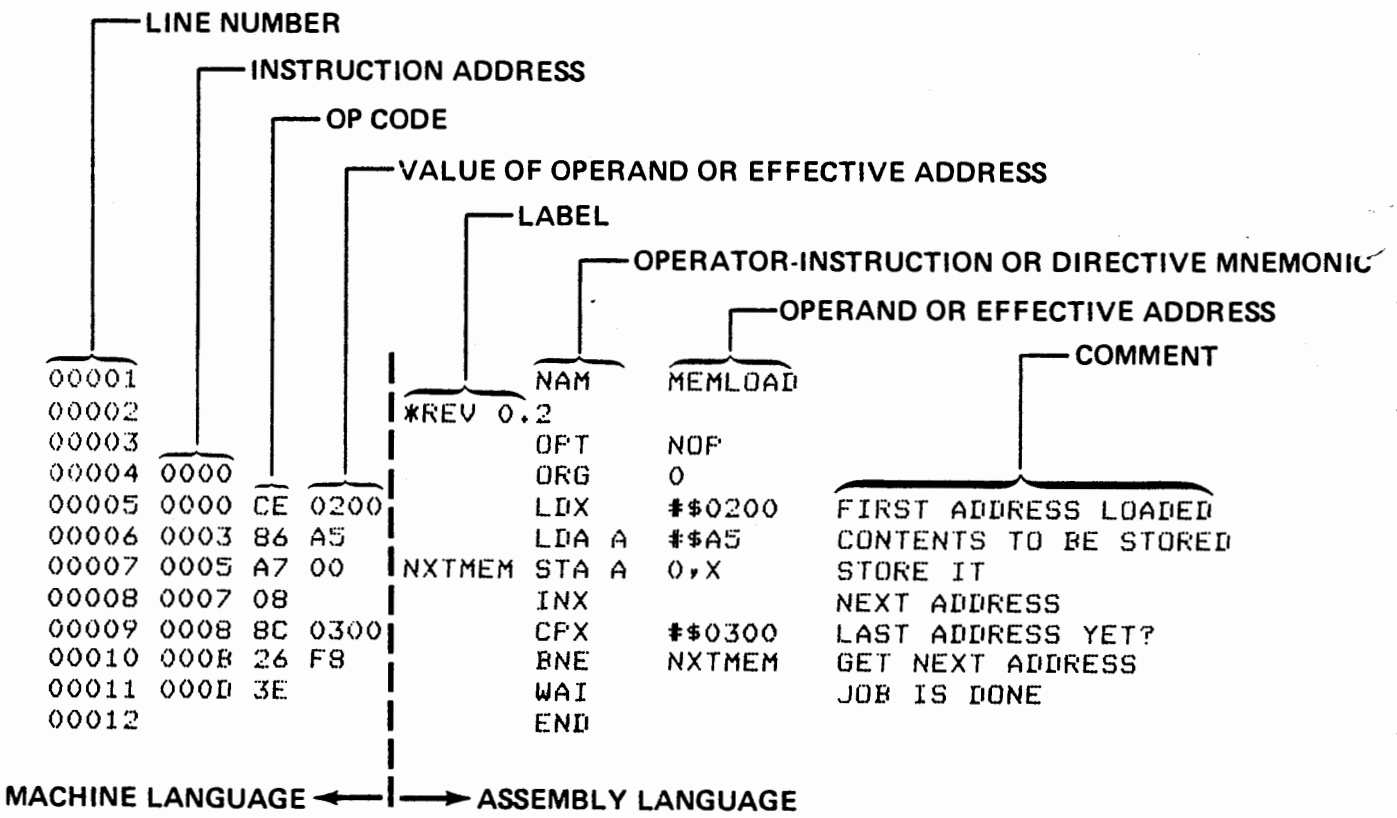


Figure 10-13
Assembled program for loading data into a block of memory.

- Column 2 lists each instruction address. Depending on the number of bytes in each instruction, these addresses can be sequential, or spaced 2 or 3 addresses apart. In Figure 10-13, line 00005 contains instruction address 0000, while line 00006 contains instruction address 0003. This occurred because op code CE required two address bytes 02 and 00 to complete the instruction.

- Column 3 lists instruction op codes only. This accounts for the variable address spacing in column 2.

- Column 4 contains the operand or effective address. Thus, this column may have none, one, or two bytes of data, depending on the op code.

- Column 5 begins the assembly language portion of the program listing. It contains any labels used to assemble the program. If the label is preceded by an asterisk (*), the following information is a comment only, and not used for program assembly. In Figure 10-13, line 00002 contains the label "*REV 0.2". This is a comment that states this is a program that has been revised two times. It serves only as a handy reference for the programmer to keep track of his program status.

  If the label is not preceded by an asterisk, the label becomes a way of finding an address in a branch or jump routine. Line 00010 contains the instruction BNE followed by NXTMEM. This says, branch if not equal to the address defined by the label NXTMEM. Thus, the program will jump to the address at line 00007, where the label NXTMEM is located.

- Column 6 lists the operator-instruction for the program, or the directive mnemonic used by the assembler.

- Column 7 lists the operand or the effective address of the instruction or directive.

- Column 8 contains any comments the programmer wishes to make. These are usually a description of the program steps to aid in understanding the program.

More information on assembly programming is available in Motorola's "M6800 Microprocessor Applications Manual" (Heath # EDP-244). For the purpose of this section, we will be primarily concerned with machine coding, columns 2, 3, and 4.

Procedure (continued)

28. Refer to Figure 10-13 and enter the program beginning at address 0000_{16}.

29. Examine your program beginning at address 0000_{16}. The addresses and their contents should agree with the list in Figure 10-14.

30. Press RESET, then press DO and enter address 0000_{16}. The display will go dark, indicating the microprocessor is working.

31. Press RESET and examine a number of memory locations between 0200_{16} and $02FF_{16}$. The contents will be $A5_{16}$. You should be able to single-step through each memory location and verify that it functions properly by observing that $A5_{16}$ is stored at each address. By changing the contents at address 0004_{16} to a different value, say $5A_{16}$, and executing the program you can verify that none of the memory defaulted to the original value, $A5_{16}$.

32. This completes the experiment. However, DO NOT disconnect the memory circuit from the Trainer. You will use this circuit in Experiment 3. Proceed to Experiment 2.

ADDRESS	DATA
0000	CE
0001	02
0002	00
0003	86
0004	A5
0005	A7
0006	00
0007	08
0008	8C
0009	03
000A	00
000B	26
000C	F8
000D	3E

Figure 10-14
Data listing for program in Figure 10-13.

Experiment 2

CLOCK

OBJECTIVES:

Show how the interrupt request can be implemented.

Show how an external timing signal can synchronize the MPU.

Introduction

In this experiment, you will improve the clock program you developed earlier. A line frequency (60 Hz) signal provides timing accuracy to the clock. The line frequency signal is connected to the interrupt request ($\overline{IRQ}$) input. This makes the clock extremely accurate.

Materials Required

1 ET-3400 Microprocessor Trainer (with hard-wired circuit).

1 6″ hookup wire.

Procedure

Programming Notes:

- Begin program (listed in Figure 10-15) at address 0003_{16} (line 00009) and enter data CE_{16}. The first three address locations are reserved for clock display time and will be entered after the main program has been entered.

- As you load the program, notice that no address is specified at lines 00008, 00015, 00023, 00028, 00041, 00047, and 00052. These lines are used for comments or assembler directives. Just follow the program address and ignore these lines.

- Line 00049 contains an ORG (originate) statement which is an assembler directive. Therefore, you can ignore the address specified. The next instruction you must enter goes to memory location $00F7_{16}$. Use the EXAM and CHAN keys to enter the opcode.

```
00001                          NAM     CLOCK-2  * REV  0.6
00002                  **LINE ACCURACY CLOCK PROGRAM
00003                          OPT     NOP
00004 0000                     ORG     0
00005 0000 0001        SECOND  RMB     1
00006 0001 0001        MINUTE  RMB     1
00007 0002 0001        HOURS   RMB     1
00008                  ** INTERRUPT HANDLING
00009 0003 CE 003D     TIMPAS  LDX     #$003D   61
00010 0006 09          ONE60T  DEX              TIME TICKING OFF
00011 0007 27 04               BEQ     TIMEUP   60 PULSES YET?
00012 0009 0E                  CLI
00013 000A 3E                  WAI              WAITING
00014 000B 20 F9               BRA     ONE60T   GO BACK & WAIT AGAIN!
00015                  ** INCR ONE SECOND AND UPDATE
00016 000D C6 60       TIMEUP  LDA B   #$60      SIXTY SECONDS,SIXTY MINUTES
00017 000F 0D                  SEC              , ALWAYS INCREMENT SECONDS
00018 0010 8D 11               BSR     INCR     INCREMENT SECONDS
00019 0012 8D 0F               BSR     INCR     INCREMENT MINUTES IF NEEDED
00020 0014 C6 13               LDA B   #$13     TWELVE HOUR CLOCK
00021 0016 8D 0B               BSR     INCR     INCREMENT HOURS
00022 0018 BD FCBC             JSR     REDIS    RESET DISPLAYS
00023      FCBC        REDIS   EQU     $FCBC
00024 001B 8D 17               BSR     PRINT
00025 001D 8D 15               BSR     PRINT
00026 001F 8D 13               BSR     PRINT    PRINT HOURS,MINUTES,SECONDS
00027 0021 20 E0               BRA     TIMPAS   DO IT ALL AGAIN
00028                  ** INCR - INCREMENT SUBROUTINE
00029 0023 A6 00       INCR    LDA A   0,X      DATA WORD INTO A
00030 0025 89 00               ADC A   #0       INCREMENT IF NECESSARY
00031 0027 19                  DAA              FIX TO DECIMAL
00032 0028 11                  CBA              TIME TO CLEAR?
00033 0029 25 01               BCS     INC1     NO
00034 002B 4F                  CLR A
00035 002C A7 00       INC1    STA A   0,X
00036 002E 08                  INX
00037 002F 07                  TPA
00038 0030 88 01               EOR A   #1       COMPLEMENT CARRY BIT
00039 0032 06                  TAP
00040 0033 39                  RTS
00041                  ** PRINT - PRINT HEX BYTES
00042 0034 09          PRINT   DEX              POINT X AT BYTE
00043 0035 96 02               LDA A   $02      WHAT'S IN HOURS?
00044 0037 27 05               BEQ     ADJUST   IF IT'S ZERO
00045 0039 A6 00       CONTIN  LDA A   0,X
00046 003B 7E FE20             JMP     OUTBYT
00047      FE20        OUTBYT  EQU     $FE20    MONITOR ROUTINE
00048 003E 7C 0002     ADJUST  INC     HOURS    MAKE IT ONE
00049 0041 20 F6               BRA     CONTIN   RESUME
00050 00F7                     ORG     $00F7
00051 00F7 3B                  RTI
00052                          END
```

Figure 10-15
Assembled program for real-time clock.

Procedure (continued)

1. If you have not done so, switch the Trainer on. Then enter the program listed in Figure 10-15.

2. Refer to Figure 10-16 and install the 6" hookup wire between the LINE socket and the $\overline{IRQ}$ socket. Do not disturb the other circuit you have wired into the Trainer.

3. Your clock is ready to run. Determine at what time you wish to start the clock; then enter the seconds at address 0000_{16}, the minutes at address 0001_{16}, and the hours at address 0002_{16}. For example, to set the clock for 9:25:30, enter the following:

Address	Data	
0000	30	(seconds)
0001	25	(minutes)
0002	09	(hours)

4. Press RESET, then press DO and then enter the first 3 digits of the starting address ($\underline{0\,0\,0}_{16}$). As the time you have set approaches, enter the fourth digit (3), but do not release the key. At precisely the correct time, release the "3" key. The display will momentarily go dark, and then show the correct time, with the seconds digit updating at a 1-second rate.

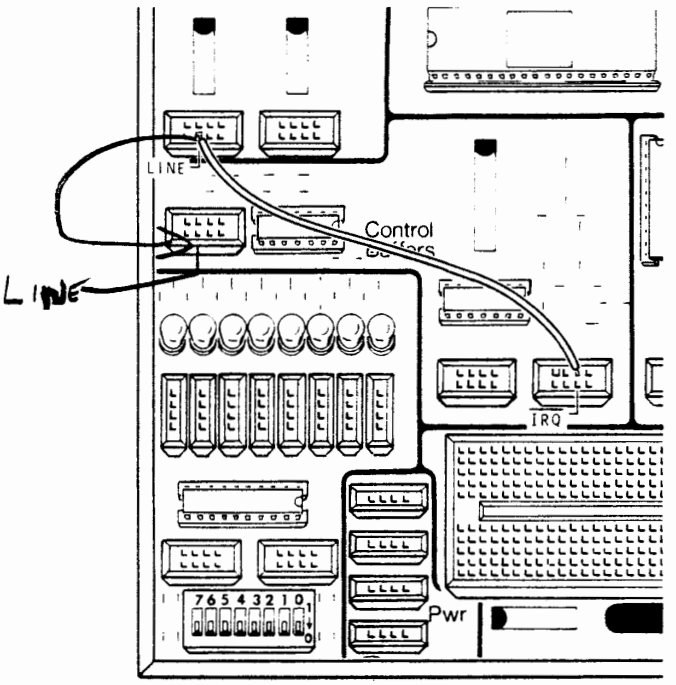


Figure 10-16
Clock interrupt request line connection.

Discussion

Although it appears to be a simple process to connect the LINE signal to the $\overline{IRQ}$ input of the microprocessor, AC line voltage had to be reduced in amplitude and processed into a waveform acceptable to the microprocessor. The circuit used by the Trainer is shown in Figure 10-17A. It uses a comparator with positive feedback to process the AC signal.

A sample of AC line signal is coupled through current limiting resistors R1 and R2 to the negative input of the comparator. Diode D1 limits the negative swing of the AC signal to approximately 0.7 volts. The comparator tracks the AC input and switches logic levels at its output, at the same rate. Positive feedback through resistor R6 speeds up the rise and fall times at the output.

A simpler circuit is shown in Figure 10-17B. Input current is limited by resistor R1, while diodes D1 and D2 limit the voltage swing within TTL levels. As before, the line frequency can be obtained from the secondary of a power transformer. The NAND gate is used to buffer the input signal and provide some speed-up of the rise and fall times.

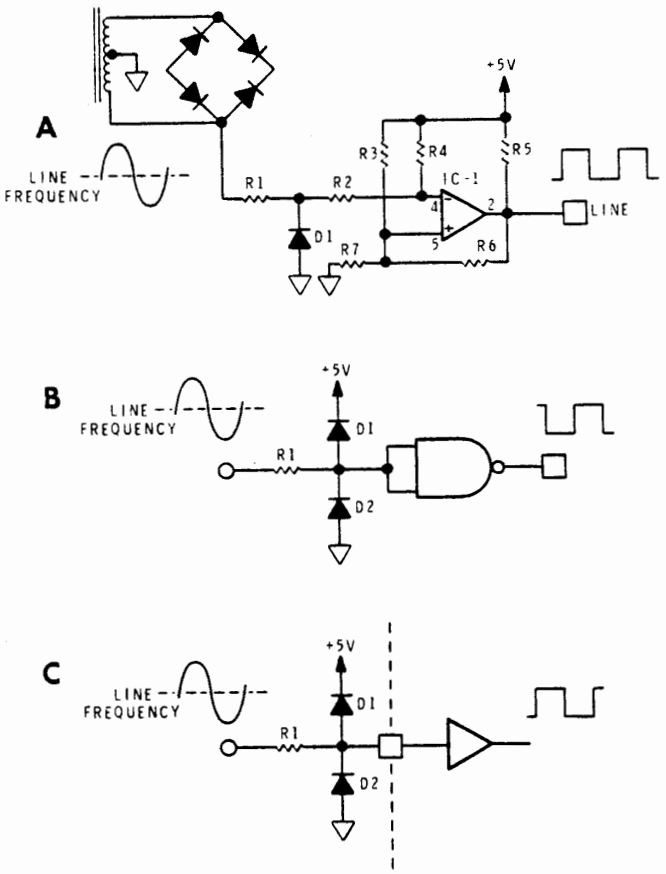


Figure 10-17
Line frequency signal processing.

If a buffer is already supplied in the system, as in Figure 10-17C, then it is only necessary to limit the signal current and voltage swing with a resistor and two diodes.

The clock program (Figure 10-15) is broken down into five main sections. These are:

Lines 00005-00007. Reserved addresses where the starting seconds, minutes, and hours are stored. As the program progresses, these addresses are updated to current time.

Lines 00009-00014, and 00051. They handle the interrupt routine, and count the seconds.

Lines 00016-00027. The main part of the program keeps track of seconds; increments seconds, minutes, and hours when appropriate.

Lines 00029-00040. A subroutine that handles the mathematical and updating part of the program.

Lines 00042-00049. Another subroutine that updates the display with new data. Also insures that hours never go to zero.

In addition, a number of monitor subroutines are used.

Since this experiment is concerned primarily with interrupt handling, this discussion will explain only that part of the program in detail.

Line 00009 — Load the index register with the line frequency plus one. (Line frequency is 60_{10}, plus 1 yields 61_{10} or $3D_{16}$.) Thus, when the index register is decremented in the next instruction, the count circuit will provide a precise division by 60.

Line 00010 — Decrement the index register by one. Clock timing has begun.

Line 00011 — The first time through, the index is not zero. Therefore, the branch instruction is not executed. On a later pass, when the index register is zero, the program will branch to TIMEUP.

Line 00013 — Wait for the interrupt request to arrive.

As discussed in Unit 6, when a non maskable interrupt ($\overline{NMI}$) occurs, the contents of the index register, program counter, accumulators, and condition code register are stored in the stack. The program counter is then loaded with a new address that is found at addresses $FFFFC_{16}$ and $FFFD_{16}$ (located in ROM). If you examine these addresses you will find 00_{16} and FD_{16} respectively. The microprocessor will then execute the instruction at address $00FD_{16}$.

Line 00050 — Return from interrupt. When this instruction is executed, the microprocessor retrieves the data previously stored in the stack. This includes the index register and program counter contents. It then executes the instruction pointed to by the program counter.

Line 00013 — Branch always is the instruction immediately following the wait for interrupt. This sends the microprocessor back to line 00010 (ONE6OT).

At this point, you should notice that the program is in a loop that repeats every sixtieth of a second. This will continue until the index register decrements to zero. When zero is attained, the branch-if-zero instruction will send the microprocessor off to the main part of the program, which increments the clock by one second. The interrupt routine repeats again after the clock advances.

The remainder of the program is very similar to the clock program presented earlier. A full explanation of the techniques used was discussed in a previous experiment and is not repeated here.

This completes this experiment. Switch the Trainer power off and remove the wire between LINE and $\overline{IRQ}$. Do not disturb the remaining wires. The circuit you previously constructed will be used in Experiment 3. Proceed with Experiment 3.

Experiment 3
ADDRESS DECODING

OBJECTIVES:

Demonstrate the difference between full and partial address decoding.

Show how an address decoding chart is assembled.

Demonstrate how an address can be decoded using various types of logic circuits.

Show how to construct a memory address map.

Introduction

Many different combinational logic circuits can be used to decode binary bit patterns. We will examine several decoding techniques in this experiment. The first example will use the circuit you wired in the first experiment.

Material Required

1 Microprocessor Trainer (with hard-wired circuit)

1 1000 ohm, 1/4-watt, 10% resistor

1 6" double-sided foam tape.

1 Large connector block

1 74LS42 integrated circuit (443-807)

1 74LS266 integrated circuit (443-719)

Hookup wire

Procedure

1. Carefully examine the circuit you wired to the Trainer in Experiment 1 to see if any wires have pulled out. Figure 10-18 is an electrical diagram of the circuit.

2. Load the program listed in Figure 10-19 beginning at address 0000_{16} with data CE_{16} and ending at address 0019_{16} with data $E6_{16}$.

3. Press RESET, then press DO and enter 0000_{16}. The display will go out. After a short period of time, all display segments and decimal points will light. This is an indication that the program has been executed.

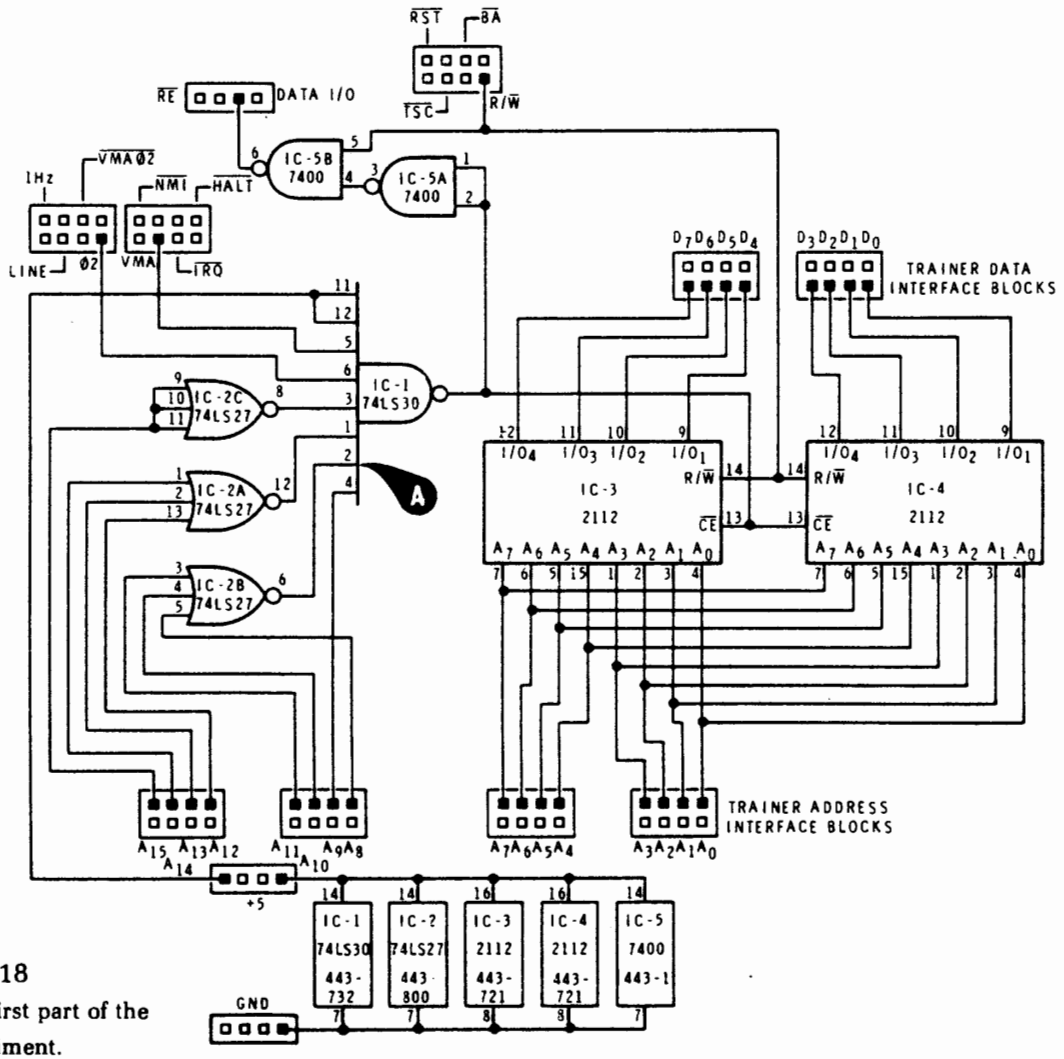


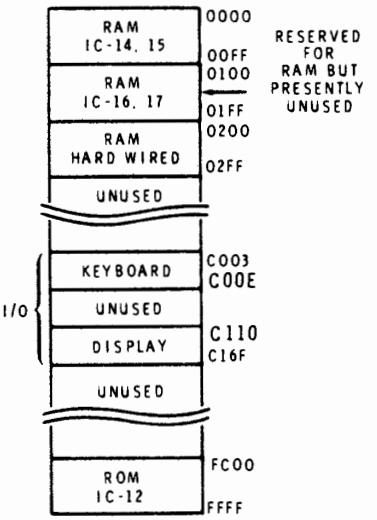
Figure 10-18
Circuit diagram of the first part of the decoding experiment.

```
00001                          NAM     DECODECK REV. 0.3
00002                          OPT     NOP
00003 0000 CE 001A REDO        LDX     #$001A    1ST BLOCK, 1ST ADR
00004 0003 86 BB               LDA A   #$BB      DATA TO BE STORED
00005 0005 A7 00   LOAD1       STA A   X         STORE IT
00006 0007 08                  INX               POINT TO NEXT ADR
00007 0008 8C 0200             CPX     #$0200    1ST BLOCK, LAST ADR(+1)
00008 000B 26 F8               BNE     LOAD1
00009 000D CE 0300             LDX     #$0300    2ND BLOCK, FIRST ADR
00010 0010 A7 00   LOAD2       STA A   X         STORE IT
00011 0012 08                  INX               POINT TO NEXT ADR
00012 0013 8C 0000             CPX     #0000     2ND BLOCK, LAST ADR(+1)
00013 0016 26 F8               BNE     LOAD2
00014 0018 20 E6               BRA     REDO      RECYCLE
00015                          END
```

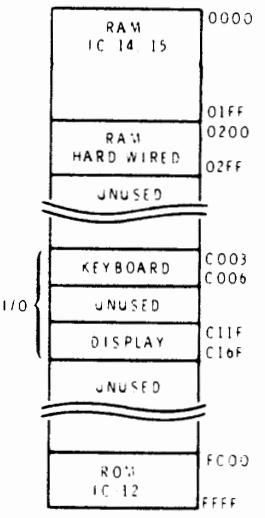
Figure 10-19
Program for memory decoding experiment.

4. You have written BB_{16} into every memory location except where the program resides, and between address 0200_{16} thru $02FF_{16}$. Verify this by examining a number of locations between $001A_{16}$ and $01FF_{16}$. Now examine a number of locations between 0200_{16} and $02FF_{16}$ (hard-wired RAM.)

   NOTE: Addresses $00D3_{16}$ thru $00F3_{16}$ will not contain BB_{16}. The Trainer monitor routine uses that portion of RAM.



A. ET-3400 MEMORY MAP



B. ET-3400A MEMORY MAP

Figure 10-20
Memory maps of the Microprocessor Trainers with additional off-board RAM at 0200_{16} through $02FF_{16}$.

Discussion

The memory you have hard-wired to the Trainer occupies memory space directly above that allocated for "on-board" memory. The assignments are shown in Figure 10-20. In Experiment 1, you manipulated data within this range. Then you wrote a program to load these addresses (0200_{16} thru $02FF_{16}$) with data. Thus, you found that each space in memory responds to a specific address.

In this experiment, you tried to load data into every possible memory location except the program location and memory block 0200_{16} thru $02FF_{16}$. You were unsuccessful with the addresses that do not presently contain RAM, if your Trainer is an ET-3400. Again, refer to Figure 10-20.

When addresses 0200_{16} thru $02FF_{16}$ were checked, the contents were not modified by the program. This proves that this section of memory is fully decoded. That is, each location can be accessed with only one specific address.

Although it was described in the Trainer assembly manual, now is a good time to briefly look at the Trainer display. As shown in Figure 10-20, the display occupies space in the Trainer memory network. In addition, each display segment and decimal point responds to a specific address. Thus, when you entered BB_{16} between 0300_{16} and $FFFF_{16}$ in memory, you also wrote into each display data latch. This is why all of the display segments lit while the program was running.

A decoding chart such as the one shown in Figure 10-21 can be used to indicate the address code for a memory location. The 1's and 0's in the high byte indicate the logic levels required to enable the memory block, while the X's in the low byte indicate that either a 1 or 0 may be present to select the actual address. Notice that all 16 bits help determine a specific address, which indicates this memory is fully decoded.

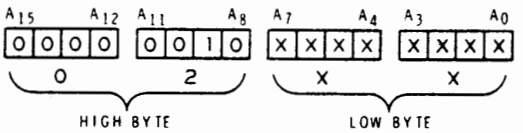


Figure 10-21
Decoding chart for a memory circuit that is fully decoding. Includes memory from 0200_{16} to $02FF_{16}$.

Procedure (continued)

5. Pull the wire end at location A (see Figure 10-18) from the connector block. Location A is pin 2 of IC1.

6. Now press DO and reexecute the program beginning at address 0000_{16}. Again the display will go out, and then light all segments and decimal points after a short period of time.

7. Press RESET and examine a number of addresses between 0200_{16} and $02FF_{16}$. Each address should now contain data BB_{16}.

8. Change the data at address 0210_{16} to AA_{16}.

9. Examine and record the data stored at the following addresses.

0010 _ _	0810 _ _
0110 _ _	0910 _ _
0210 _ _	0A10 _ _
0310 _ _	0B10 _ _
0410 _ _	0C10 _ _
0510 _ _	0D10 _ _
0610 _ _	0E10 _ _
0710 _ _	0F10 _ _

Discussion

When you disconnected IC2B from the circuit, address lines A_8, A_{10}, and A_{11} were no longer able to take part in address decoding. Since line A_9 was still connected, the circuit still decodes to $02XX_{16}$. However, other addresses will now decode the same circuit.

A new decoding chart (see Figure 10-22) can be assembled by examining the schematic in Figure 10-18. Bits A_0 thru A_7 are connected directly to memory and select specific addresses in that memory block. An X is placed in each of the corresponding boxes to indicate that the logic levels are unknown but do take part in address selection. Bits A_{12} thru A_{15} must be logic 0 so the correct logic level will be applied to the inputs of NAND gate IC1. Therefore, a 0 is placed in each box.

Disconnecting IC2B removed bits A_8, A_{10}, and A_{11} from the circuit. Since these bits have no effect in the decoding process, these are "don't care" bits. A dot (•) symbol is then placed in each of the corresponding boxes.

Address bit A_9 is still connected to IC1. Since it must be a logic 1 to enable memory, a 1 is placed in its box.

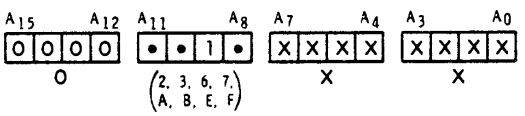


Figure 10-22
Decoding chart for a memory circuit that is partially decoded. Indicates that many addresses will decode this block of memory.

Once you have constructed a decoding chart, you can determine all of the addresses that will access the circuit. In this case, the most significant four bits are clearly defined as zeroes. The next four bits are more complicated. The only defined bit is A_9. Bits A_8, A_{10}, and A_{11} can be any logic level. Therefore, any hex number which contains the A_9 bit as a logic 1 will be valid. These hex numbers include 2, 3, 6, 7, A, B, E, and F. Bits A_0 through A_7 are variable and select the specific addresses within the circuit.

The chart you completed in step 9 will support this discussion. When you changed the contents of address 0210_{16} to AA_{16}, the addresses that had 3, 6, 7, A, B, E, and F as the second most significant hex digit also appeared to contain AA_{16}. This, of course is impossible, since no memory exists for those addresses.

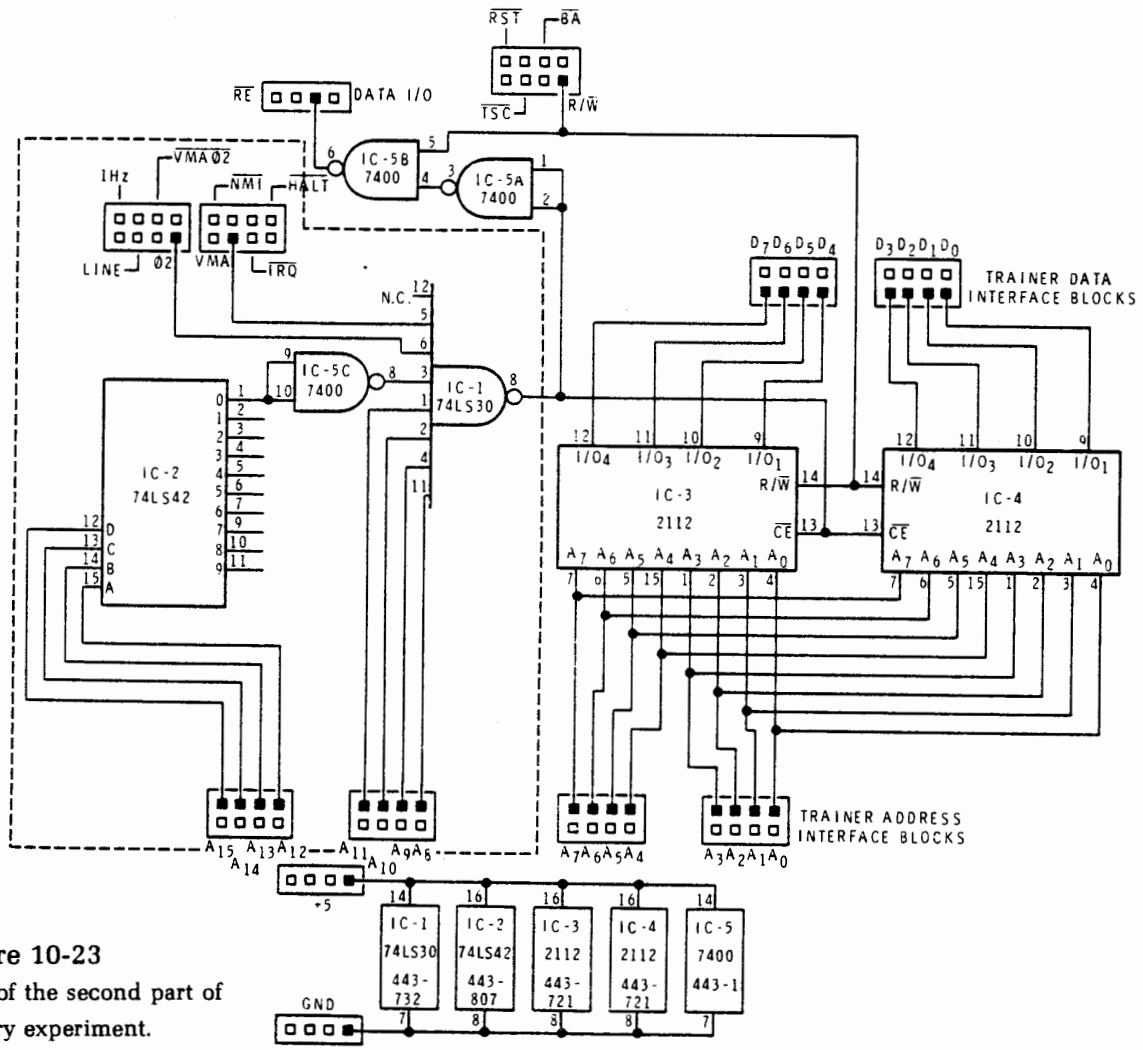


Figure 10-23
Circuit diagram of the second part of the memory experiment.

Procedure (continued)

10. Switch the Trainer power off. Then carefully remove all of the wires from IC1 and IC2, except for the +5 V and ground wires. The remaining circuit wires will remain unchanged in the circuit you are about to construct. To aid you, the new circuit is enclosed by a dashed line in Figure 10-23.

11. Using the IC puller tool, carefully remove IC2 (74LS27). Then install the 74LS42 16-pin IC. Since you are replacing a 14-pin IC with a 16-pin IC, you will have to reposition the +5V wires or ground wires (pin 8 or pin 16).

12. Refer to Figure 10-23 and wire IC1 and IC2 into the circuit. Note that gate C of IC5 is also wired into the circuit.

DEC	BCD INPUT				OUTPUT LINES									
NO.	D	C	B	A	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	1	1	1	1	1
2	0	0	1	0	1	1	0	1	1	1	1	1	1	1
3	0	0	1	1	1	1	1	0	1	1	1	1	1	1
4	0	1	0	0	1	1	1	1	0	1	1	1	1	1
5	0	1	0	1	1	1	1	1	1	0	1	1	1	1
6	0	1	1	0	1	1	1	1	1	1	0	1	1	1
7	0	1	1	1	1	1	1	1	1	1	1	0	1	1
8	1	0	0	0	1	1	1	1	1	1	1	1	0	1
9	1	0	0	1	1	1	1	1	1	1	1	1	1	0
>9	INVALID CODES				1	1	1	1	1	1	1	1	1	1

Figure 10-24
Logic truth table for 74LS42 4-to-10 line decoder.

13. Carefully examine the circuit to make sure all of the wires are properly routed. Also make sure none of the previously installed wires have pulled free.

14. Using the schematic in Figure 10-23, and the logic truth table for IC2, found in Figure 10-24, construct a memory decoding chart in Figure 10-25.

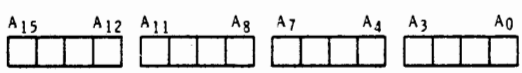


Figure 10-25
Blank decoding chart.

15. Is the circuit fully or partially decoded?
___ Fully ___ Partially
Why? _____

16. Now that you have determined the address block your memory resides at, load a few of the addresses with data.

Discussion

The circuit you just constructed contains a fully decoded memory. Its decoding chart is shown in Figure 10-26. Each bit position is used to define a specific memory address.

In an earlier experiment, you used a combinational logic decoding technique. With that technique, it is necessary to use an individual logic circuit for each memory block. In the circuit you just completed, a 4-to-10 line decoding IC was used to define a block of memory. Since it is possible to define ten different values with four input variables, this device could be used to address ten different memory blocks. Although there was no difference in the number of IC's used in the two experiments, expanding the amount of memory would require fewer device select IC's when the 4-to-10 decoder is used. This can be seen by reexamining the circuits in Figures 10-18 and 10-23.

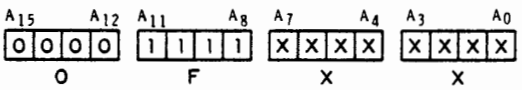


Figure 10-26
Decoding chart for circuit using 4-to-10 decoding IC.

Procedure (continued)

17. Switch the Trainer power off. Then disconnect the wire at pin 1 of IC2 (74LS42) and connect it to pin 6 of IC2.

18. The circuit is now fully decoded to address $5FXX_{16}$. This is because every address bit plays a part in determining a specific memory location. Switch the Trainer power on and examine a number of memory locations between $5F00_{16}$ and $5FFF_{16}$. Notice that you can store and read data at these locations. However, you can no longer store and read data at $0F00_{16}$ thru $0FFF_{16}$.

Discussion

The 4-to-10 line decoder can also be used to quickly change decoding addresses, as you have just done. Keep in mind that this experiment uses only one 4-to-10 line decoder to define the most significant hex number. It is not unusual to see them used at lower order addresses. Your Microprocessor Trainer uses this decoding technique for its "on-board" memory.

Procedure (continued)

19. At this time, you will need more breadboarding space. Locate the box containing the large connector block. Then refer to part A of Figure 10-27 and install the connector strips supplied with the block. You may have some strips left over.

20. Refer to Figure 10-27B. Then remove the paper backing from the vinyl strip supplied with the block, line up the long edges of the strip and block, and press the sticky side of the vinyl strip against the block.

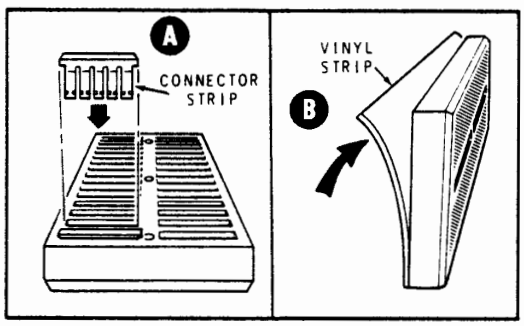


Figure 10-27
Large connector block assembly.

21. Read this whole step, then locate the foam tape and cut two 3/4" squares from it. Refer to Figure 10-28. Remove the paper backing from one side of each square and affix each square to the small edge of the Trainer cabinet as shown. The spacing between the squares should be a little less than the length of the connector block. Now remove the paper backing from the other side of the squares and affix the connector block to the squares. Try to center the squares on the back of the block. This block will provide additional bread-boarding space.

22. Switch the Trainer power off and install a 74LS266 (443-719) IC in the new large connector block, near the left end. This IC will become IC6 in the circuit you construct next.

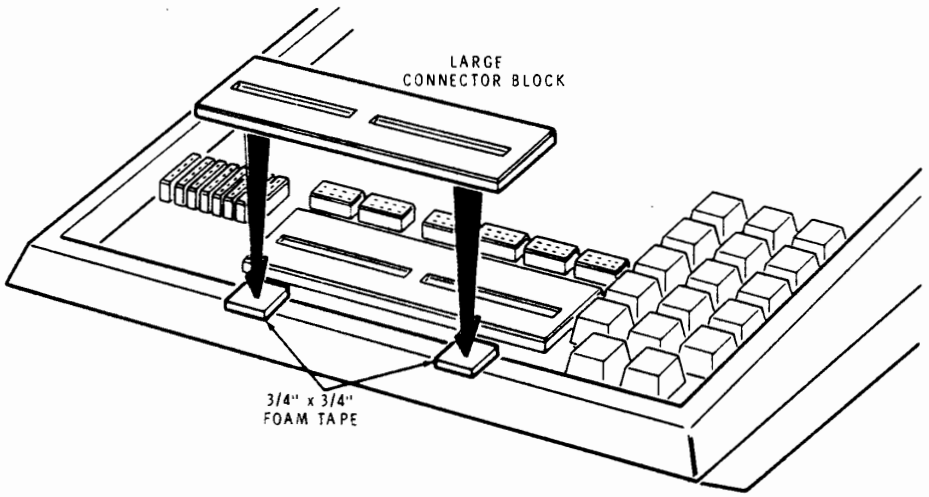


Figure 10-28
Outboard connector block installation.

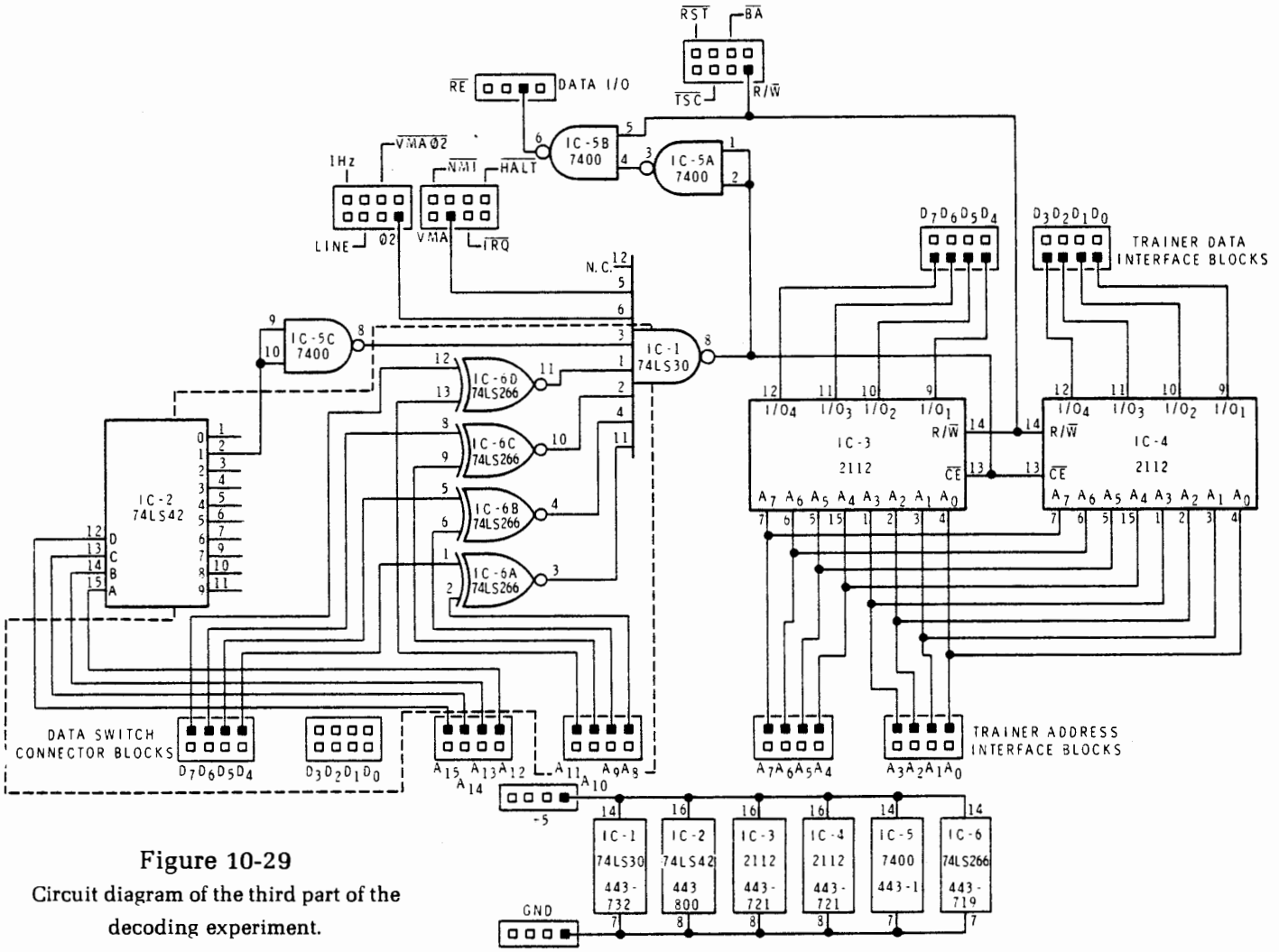


Figure 10-29
Circuit diagram of the third part of the decoding experiment.

23. Connect +5-volt power to pin 14 and ground to pin 7. Then refer to Figure 10-29 and construct the circuit shown. To aid you, the area enclosed by the dashed line is the only area where modifications are made to the previous circuit.

24. Recheck the wiring to insure it is correct. Be sure pins 9 and 10 of IC5C are connected to pin 2 of IC2.

25. Refer to Figure 10-30. This is the truth table for a gate in IC6. Notice that if the B input is held at logic 0, the relationship between input A and output Y is complementary (A is the inverse of Y). If the B input is held at a logic 1, the relationship between A and Y is direct (no inversion). Thus, input B can be used as a direct driver or an inverter driver. This feature will be used in the experiment.

A	B	Y
0	0	1
1	0	0
0	1	0
1	1	1

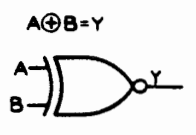


Figure 10-30
Truth table for exclusive NOR (ENOR) gate.

26. Position all of the data switches in the down (logic 0) position.

27. Using the schematic in Figure 10-29 and the table in Figure 10-30, determine the decoding chart for the circuit you constructed. Fill in the blank decoding chart in Figure 10-31.

28. Is the address fully or partially decoded?
Fully ___ Partially ___

29. What is the address block this circuit will decode?
$\_\_\_\__{16}$ thru $\_\_\_\__{16}$

30. Switch the Trainer on. Then enter data into a number of addresses in the address block you calculated.

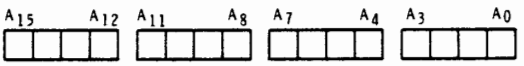


Figure 10-31
Blank decoding chart.

Discussion

The exclusive NOR gates used in the circuit all function as inverter drivers. Since each is connected to an individual input in IC1, and the 4-to-10 line decoder is still in the circuit, each address line plays a part in determining a specific address. Therefore, this circuit is fully decoded, and occupies addresses 1000_{16} thru $10FF_{16}$.

Procedure (continued)

31. Change data switches D_7 through D_4 to 0101_2. This equals hex 5.

32. What is the address block this circuit will now decode? $\_\_\_\__{16}$ thru $\_\_\_\__{16}$

33. Test the new address to see if the circuit will respond properly.

Discussion

Now you see the power of the exclusive NOR gate. Addresses are easily switched through them. Notice that whatever binary bit pattern you select with the data switch, the circuit responds to it. Now you will see one other feature of these particular exclusive NOR gates.

Procedure (continued)

34. Switch the Trainer power off. Refer to Figure 10-29. Remove the three wires interconnecting IC1 and IC6, from pin 11 to pin 3, pin 4 to pin 4, and pin 2 to pin 10. Refer to Figure 10-32. Then interconnect pins 3, 4, 10, and 11 of IC6. Finally, insert a 1000 ohm, 1/4-watt, 10% resistor between +5 volts and IC6, pin 11.

35. The circuit should still be located at address block 1500_{16} thru $15FF_{16}$. Check a few of the addresses where you previously entered data. They should contain the same data.

36. Change the data switches to a new bit pattern, determine the address code, then examine a few locations to assure yourself of address code accuracy. Repeat this procedure a number of times.

Discussion

The 74LS266 exclusive NOR gate has a special characteristic; it has an open-collector output. This means that a number of gates can be tied together, as you just did. The electrical term for this procedure is wire-OR'ing, where the outputs function as though they were inputs to an OR gate.

One more aspect of circuit decoding will be discussed before the next experiment. This is important, since an experimental error could possibly result in the destruction of a gate or memory package.

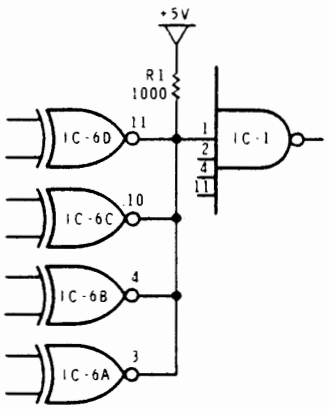


Figure 10-32
Modification to third circuit of experiment.

A problem arises when two circuits decode to the same address. If two memories containing different data occupy the same address, they will try to pull the data lines in two different directions. Since this is electrically impossible, one circuit will give in, usually resulting in permanent destruction to the circuit. Thus, it is important when designing circuits like those used in this experiment to always check for conficts in address decoding.

Procedure (continued)

37. Switch the Trainer power off and pull the line cord plug.

38. Pull the hookup wires from the circuit and save them for future use.

NOTE: If your Trainer is an ET-3400, perform step 39. If your trainer is an ET-3400A, perform step 39A.

39. Carefully remove IC's 3 and 4 (2112) from the large connector block and install them in IC sockets 16 and 17 in the Trainer. Observe the normal precautions for MOS devices, and make sure you align pin 1 of each IC to the proper position. Then remove all of the remaining components from the two large connector blocks.

39A. Carefully remove IC's 3 and 4 (2112) from the large connector block. Observing the precautions for MOS devices, place them on the anti-static pad prior to placing them in the small parts container furnished with this course. Then remove all of the remaining components from the two large connector blocks.

Discussion

Your Microprocessor Trainer now contains 512_{10} bytes of RAM. This is located at addresses 0000_{16} through $01FF_{16}$. Proceed to Experiment 4.

Experiment 4
DATA OUTPUT

OBJECTIVES:

Demonstrate microprocessor interfacing to an external data display.

Show how a 7-segment display is connected.

Demonstrate the trade-offs between hardware and software display decoding.

Provide an opportunity to write a number of output programs.

Introduction

Until now, you have been using programs that moved data within the Trainer, with any results displayed by the "on-board" LED's. This may be adequate for your purposes, but other methods are needed if external equipment uses the data. The data may take the form of a visual display for an operator to read, or a digital control signal to manipulate an electro-mechanical device. This experiment will present a number of interfacing methods and examine some of the advantages and disadvantages of each method.

Material Required

1 ET-3400 Microprocessor Trainer

8 470 ohm, 1/4-watt, 5% resistors

2 10 k ohm, 1/2-watt, 5% resistors

1 FND-500 7-segment LED (411-819)

1 TIL-312 7-segment LED (411-831)

1 7400 integrated circuit (443-1)

2 7475 integrated circuits (443-13)

1 9368 integrated circuit (443-694)

1 74LS30 integrated circuit (443-732)

1 74LS27 integrated circuit (443-800)

1 74LS259 integrated circuit (443-804)

Hookup wire

Procedure

1. In this part of the experiment, you will examine how the MPU can be interfaced to LED's. Make sure the Trainer power is switched off; then construct the circuit shown in Figure 10-33. Notice that +5 volts and ground are connected to pins 5 and 12 respectively for IC's 1 and 2 (7475). The other IC's use pin 14 for +5 volts and pin 7 for ground.

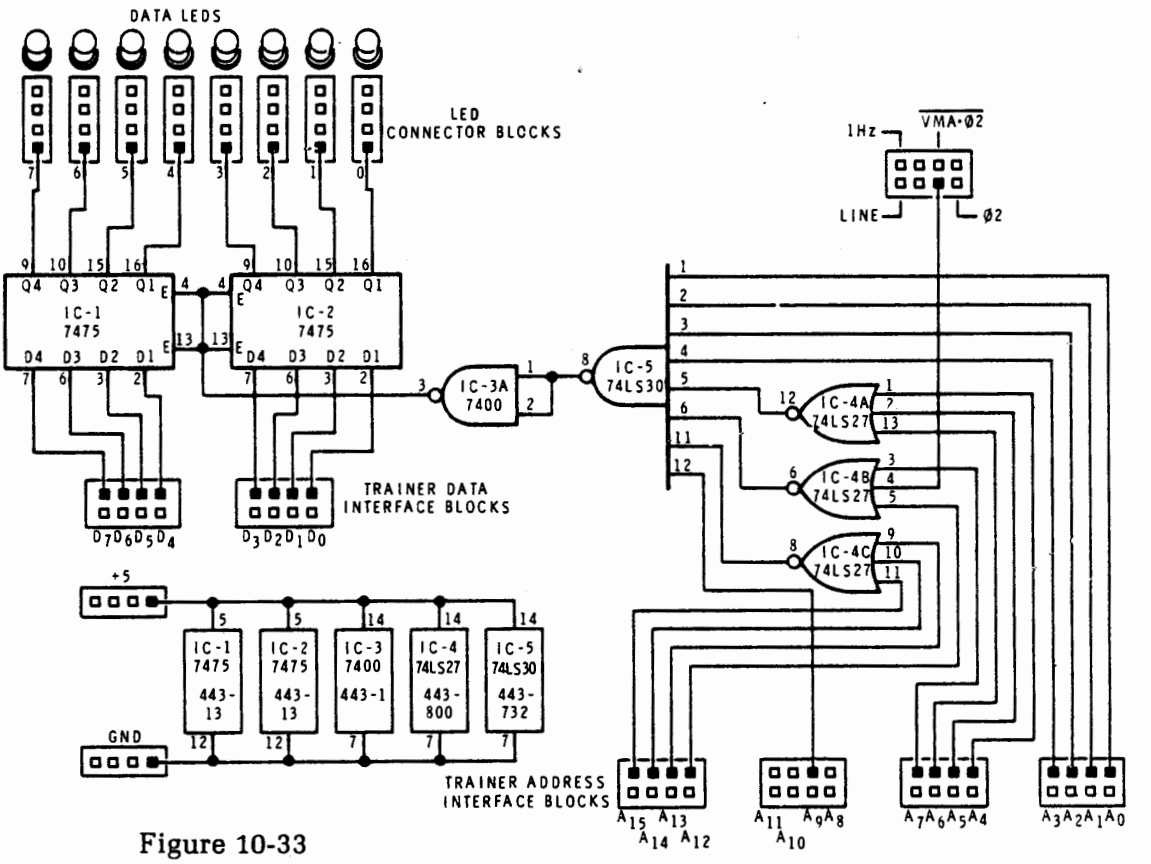


Figure 10-33
Circuit diagram of the first part of the output experiment.

2. Recheck your wiring; then switch the Trainer power on. The data LED's will show a random value.

3. Figure 10-34 is a decoding chart for the circuit you constructed. This shows that the circuit is partially decoded. A 2-digit hex number can be stored at any of these decoded addresses. Examine address $020F_{16}$. Then change the contents to 55_{16}.

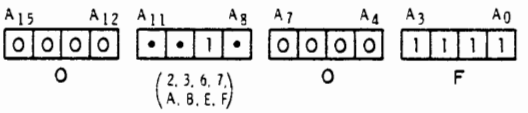


Figure 10-34
Decoding chart for outputting information to the data LED's.

4. The data LED's indicate _ _ _ _ _ _ _ _$_2$. This is the binary equivalent of the data you stored at address $020F_{16}$.

5. What hex value would be required to turn off all of the data LED's? _ _$_{16}$. Verify your answer.

6. What hex value would be required to turn on all of the data LED's? _ _$_{16}$. Verify your answer.

7. Change the data at address $020F_{16}$ a number of times and verify its value with the data LED's.

8. Write and execute a program that will alternately turn all of the data LED's on and off. Use a delay loop in the program so that the on and off cycles can be recognized. Remember that an MPU cycle takes approximately 2.5 microseconds in the unmodified Trainer. If your Trainer has been modified for use with the Memory I/O Accessory, an MPU cycle will take about one microsecond.

   If you have any difficulty, use the Trainer single-step function to examine the operation of your program.

Discussion

Refer to Figures 10-33 and 10-34. Notice that a partial decoding scheme is used. A fully decoded circuit could have been used by adding more combinational logic.

In previous decoding circuits, the VMA and $\phi 2$ signals were separate. A logic 1 indicated their true state. In this experiment, we took advantage of another Trainer output; the $\overline{VMA \cdot \phi 2}$ line. It is logic 0 when both the VMA and $\phi 2$ signals are at their true state. This reduces the number of logic gates needed for decoding.

The circuit you constructed appears as a **write only memory** to the microprocessor. That is, the MPU can write into the selected address, but it can not read the data stored. However, since eight data LED's monitor the stored information, you can **read** the data. Thus, the MPU is interfaced in a way that produces usable data.

Two bistable quad latch IC's are enabled when one of the eight preselected addresses is accessed. They act as an 8-bit memory storage device. Thus, any data appearing on the data lines is latched into the two devices. Since the output of each latch is active, the data LED connected to each will follow the data level. Storing 00_{16} will turn off all of the LED's, while storing FF_{16} will turn each LED on.

Right now, the data LED's should be switching on and off at a regular interval, because of the program you wrote and executed. If you had any difficulty with the program, refer to Figure 10-35. It lists a program to flash the data LED's. The contents of addresses 0005 and 0006 control the amount of delay. You may wish to change this number if your Trainer has been modified. While this program may not match your program, it is one of many ways to accomplish the same objective.

```
00001                          NAM     FLASHER1 REV. 0.1
00002                          OPT     NOP
00003 0000 4F                  CLR A                ACC NOW 0
00004 0001 B7 020F ALTER       STA A   $020F        STORE ACC TO LIGHTS
00005 0004 CE 5500             LDX     #$5500      *
00006 0007 09        WAIT      DEX                 *WAIT
00007 0008 26 FD               BNE     WAIT        *
00008 000A 43                  COM A               TOGGLE ACC
00009 000B 20 F4               BRA     ALTER       GO BACK TO RESTORE
00010                          END
```

Figure 10-35
Program to flash data LED's at regular interval.

Procedure (continued)

9. Write a program to alternately store 1's and 0's to the display LED's. But this time, adjust the timing so the LED "on" time is longer than the "off" time. Then execute the program.

Discussion

This program required two timing loops, to allow for the difference between on and off time. If your first program contained two timing loops of equal duration, it was a simple matter to modify the delay times. Figure 10-36 illustrates a second method for accomplishing the task. The delay times shown are for a **slow clock**. You may wish to change them if your Trainer has a **fast clock**.

In the next part of the experiment, you will add a decoder-driver and a common cathode, 7-segment display to the circuit.

```
00001                          NAM     FLASHER2 REV. 0.1
00002                          OPT     NOP
00003 0000                     ORG     0
00004 0000 4F                  CLR A               ACC NOW "0"
00005 0001 CE 5500 CYCLE       LDX     #$5500      LOGIC "0" TIME
00006 0004 B7 020F             STA A   $020F
00007 0007 43                  COM A               BITS NOW "1"
00008 0008 09        HOLD1     DEX
00009 0009 26 FD               BNE     HOLD1
00010 000B CE FF00             LDX     #$FF00      LOGIC "1" TIME
00011 000E B7 020F             STA A   $020F
00012 0011 43                  COM A               BITS NOW "0"
00013 0012 09        HOLD2     DEX
00014 0013 26 FD               BNE     HOLD2
00015 0015 20 EA               BRA     CYCLE       ONE CYCLE COMPLETE
00016                          END
```

Figure 10-36
Program to flash data LED's at a non-regular interval, with the on time longer than off time.

Procedure (continued)

10. Switch the Trainer power off. Then, without disturbing the circuit wired to the Trainer, add the circuit shown in Figure 10-37. Use the large connector block affixed to the Trainer cabinet to hold the new circuit. The FND-500 (#411-819) display is the larger of the two displays.

11. Recheck your wiring, then switch the Trainer power on, and press RESET.

12. The lower four bits of your data byte will determine the digit displayed. Enter $A5_{16}$ into address $020F_{16}$.

13. What is the bit pattern displayed by the lower four display LED's? ____$_2$.

14. What is the hex equivalent? $_{16}$.

15. What is displayed by the new 7-segment display? $_{16}$.

16. Write a program that will cause the 7-segment display to count from 0 to F_{16} and then continuously repeat. Include a delay loop so that each digit will remain on long enough to be identified. Execute the program.

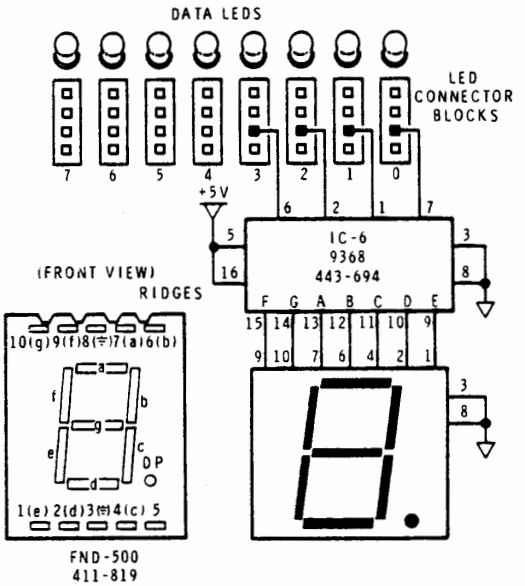


Figure 10-37
Additional data display for first output circuit.

Discussion

The circuit you just constructed contains a 4-line-to-7-segment decoder driver and a 7-segment, common cathode display. The decoder driver (9368) contains a large maze of combinational logic which allows it to decode four data bits and drive the proper segments in a 7-segment display to produce the corresponding hex digit.

The display circuit is a multiple LED array with common cathodes. The cathodes are grounded, and the decoder driver supplies the necessary power (approximately 30 mA at +5 volts) to light the selected LED segments.

If you had any questions concerning the program to increment the display, refer to Figure 10-38. It contains a simple program to increment the display from 0 to F_{16} at a slow rate. The simplicity of this program assignment removes the need to reset accumulator A after incrementing to $0F_{16}$. It continues beyond $0F_{16}$. But, since only the four lower bits of data are decoded, it appears to count to $0F_{16}$ and then reset to 00_{16}. Enter the program in Figure 10-38 and watch the eight data LED's. They show the actual value stored in accumulator A.

Next you will see how the MPU handles common-anode type displays. Also, you will see that a decoder driver is not necessary if you are willing to let the MPU do the decoding.

```
00001                          NAM     STEP-UP  REV.0.4
00002                          OPT     NOP
00003 0000 4F                  CLR A                START WITH 0
00004 0001 B7 020F UPDATE      STA A   $020F       STORE TO OUTPUT
00005 0004 4C                  INC A                ADD ONE
00006 0005 CE FFFF             LDX     #$FFFF     TIME TO WAIT
00007 0008 09        UPDAT2    DEX                  TIME RUNNING OUT
00008 0009 26 FD               BNE     UPDAT2     TIME UP YET?
00009 000B 20 F4               BRA     UPDATE
00010                          END
```

Figure 10-38
Program to increment the 7-segment display from 0 to F_{16} in an apparent cyclic manner.

Procedure (continued)

17. Switch Trainer power off and remove the wires, decoder driver, and display package from the large connector block affixed to the Trainer cabinet.

18. Refer to Figure 10-39 and construct the circuit shown. Since the resistor leads are too short to reach from the connector block to the data LED connectors, insert the free end of each resistor into an unused connector socket. Then run hookup wire to the appropriate LED connector block.

19. Reexamine the circuit to make sure it is properly wired, and the resistor leads do not touch adjacent resistor leads. Then switch Trainer power on and press RESET.

20. This circuit, like the previous circuit, uses the address decoder and latches initially wired to the Trainer. Data stored at address $020F_{16}$ will determine which display segment will light. Enter 00_{16} at address $020F_{16}$. What does the display indicate? $_{-16}$.

21. Change the data to FF_{16}. What does the display indicate? $_{-16}$.

22. To light a particular segment in the display, the corresponding data bit must be logic 0. The table below the circuit in Figure 10-39 indicates the segments connected to the data bits. What bit pattern will produce the number 1 in the display? $\_\_\_\_\_{}_{-2}$.

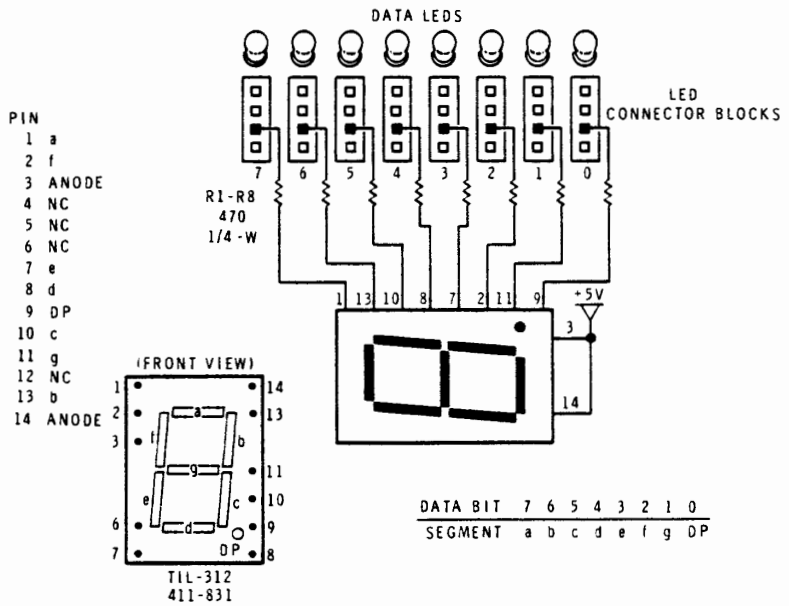


Figure 10-39
Additional data display.

23. Convert the bit pattern from step 22 to hex and enter it at address $020F_{16}$. Although it is possible to display two 1's, the correct 1 is produced when segments b and c are lit.

24. Load and execute the program shown in Figure 10-40. If your Trainer has a **fast clock**, you may want to change the contents of address 000B to provide a longer delay.

```
00001                          NAM     CHAROUT1 REV. 0.1
00002                          OPT     NOP
00003        020F     DISPLA   EQU     $020F
00004  0000                    ORG     0
00005  0000 CE 001A  RECYCL   LDX     #CODES    START OF TABLE
00006  0003 A6 00    NXTDIG   LDA A   X         LOAD BIT PATTERN
00007  0005 B7 020F           STA A   DISPLA    STORE TO DISPLA
00008  0008 86 FF             LDA A   #$FF      *
00009  000A C6 55    HOLD1    LDA B   #$55      *
00010  000C 5A       HOLD2    DEC B             * WAIT
00011  000D 26 FD              BNE     HOLD2     *
00012  000F 4A                 DEC A             *
00013  0010 26 F8              BNE     HOLD1     *
00014  0012 08                 INX               POINT TO NXT PATTERN
00015  0013 8C 002A            CPX     #FINAL+1  LAST ONE YET?
00016  0016 27 E8              BEQ     RECYCL    IF SO, START AGAIN
00017  0018 20 E9              BRA     NXTDIG    IF NOT, NXT PATTERN
00018  001A 03        CODES    FCB     $03,$9F,$25,$0D,$99,$49
       001B 9F
       001C 25
       001D 0D
       001E 99
       001F 49
00019  0020 41                 FCB     $41,$1F,$01,$19,$11,$C0
       0021 1F
       0022 01
       0023 19
       0024 11
       0025 C0
00020  0026 63                 FCB     $63,$85,$61
       0027 85
       0028 61
00021  0029 71        FINAL    FCB     $71
00022                          END
```

Figure 10-40
Program for incrementing the 7-segment display from 0 to F_{16} in a cyclic manner.

Discussion

In this experiment, you have successfully eliminated a decoder driver, but at the expense of increased software. The program sequentially stores bit patterns to the display to make it appear as number 0 thru F_{16} are being stored.

Addresses $001A_{16}$ thru 0029_{16} contain the sixteen display codes in numerical sequence. This "look-up" table is then accessed by the index register to obtain the required code.

You may have noticed that the B_{16} digit had a decimal point lit next to it. This is sometimes used to indicate it is a B rather than a 6. If you prefer not to have the decimal point, you can change address 0025_{16} to $C1_{16}$.

The display used in this circuit is of the common anode type, with the anodes connected to +5 volts. To turn on a segment, its cathode must be grounded. Therefore, a logic 0 turns on a segment while a logic 1 turns it off.

In some applications, it is convenient to assign each segment of the display its own address. In the next part of the experiment, you will see how this is accomplished.

Procedure (continued)

25. Switch the Trainer power off. Then remove all of the wires and components from both large connector blocks.

26. Refer to Figure 10-41 and construct this circuit on the Trainer's large connector block.

27. Switch the Trainer power on. Then enter 00_{16} at address $02F0_{16}$. Since only the D_0 bit is connected to the display circuit, a logic 0 will turn a display segment on, and a logic 1 will turn the segment off.

28. Advance the address and enter 00_{16}. Continue this process and observe the display. Stop after you enter 00_{16} at $02F7_{16}$. Notice that all of the display segments are lit, including the decimal point.

29. Examine address $02F0_{16}$ and watch the display. Now advance through the next seven address locations while you watch the display. What is finally displayed? _____. Why? _____

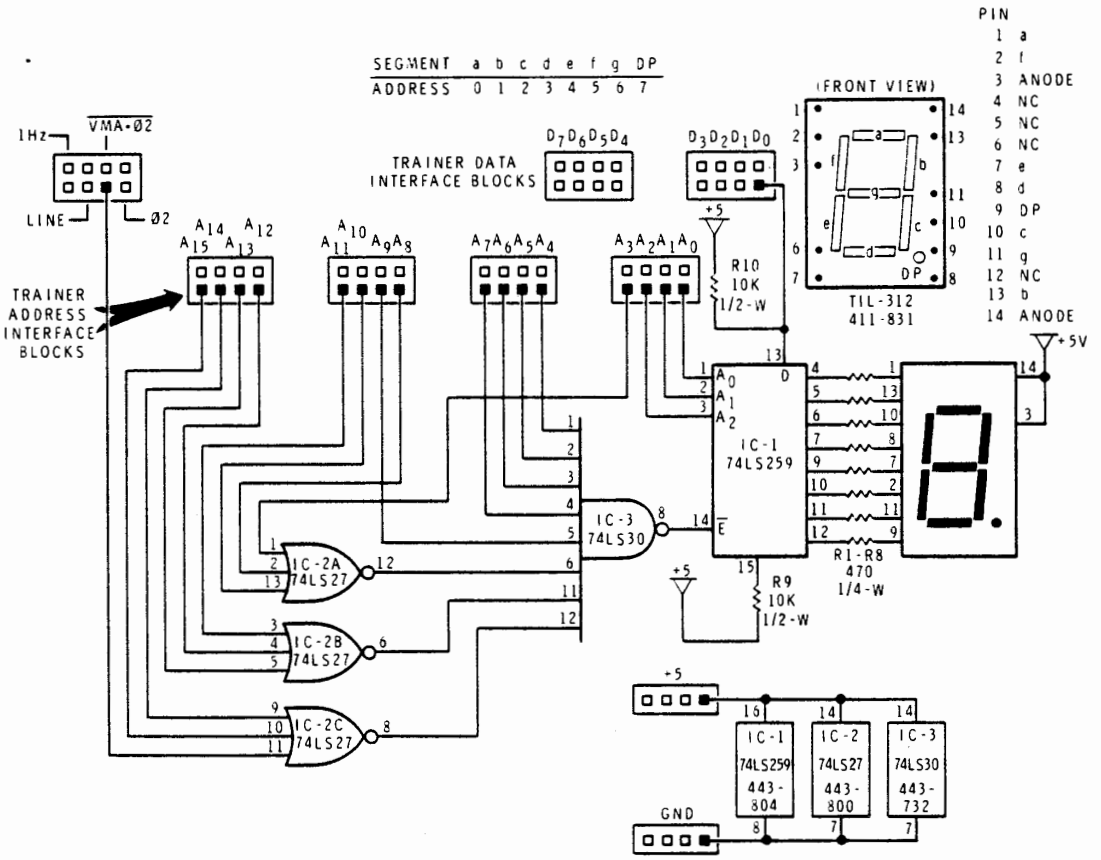


Figure 10-41

Circuit diagram of the fourth part of the output experiment.

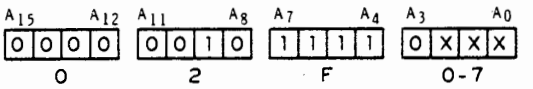


Figure 10-42

Decoding chart for the fourth output circuit.

Discussion

Each display segment now has its own address in memory. This is shown in the circuit decoding chart in Figure 10-42. Refer to Figure 10-41. Address bits A_0, A_1, and A_2 are decoded to select 1-of-8 bistable latches in IC1. Then during an MPU write operation, the logic information supplied by data bit D_0 is coupled into the selected latch. A logic 0 will turn on the appropriate display segment, while a logic 1 will turn off the segment. A table showing address/segment data is provide in Figure 10-41, above the circuit diagram. The remaining address bits, and $\overline{VMA \cdot \emptyset 2}$ are used to enable ($\overline{E}$) the latches.

Since this circuit is the write-only type, the D_0 line will "float" during an MPU read operation. Therefore, the D input of IC1 will go high (10 k ohm pull-up to +5 volts) and couple a logic 1 into the latch. This is why a segment went out when you examined its address without entering data.

Procedure (continued)

30. Load the program listed in Figure 10-43. Begin at address 0001_{16}. (Address 0000_{16} is reserved for data.) Notice that the comment column in step 0013_{10} indicates this program will be used as a subroutine in a future program.

31. Refer to the display data table in Figure 10-44 and select a hex digit. Then enter the corresponding data at address 0000_{16}.

32. Execute the program beginning at address 0001_{16}. The hex digit you selected will appear in the 7-segment display.

33. Load the program listed in Figure 10-45. Begin at address 0102_{16} (step 00006_{10}). (Addresses 0100 and 0101_{16} are reserved for data.) Then enter 39_{16} at address $000F_{16}$. The program located at addresses 0001 thru $000F_{16}$ is a subroutine for the program you just entered. The data stored at addresses 0124 thru 0133_{16} serve as a look-up table for the 16 hex digits you will display. If your Trainer has a **fast clock**, you may want to change the contents of address 0115 to provide a longer delay.

34. Execute the program beginning at address 0102_{16}. The 7-segment display will sequentially show the hex digits 0 thru F in a cyclic manner.

DIGIT	DATA
0	03
1	9F
2	25
3	0D
4	99
5	49
6	41
7	1F
8	01
9	19
A	11
B	C0
C	63
D	85
E	61
F	71

Figure 10-44
Display data table for the character output program.

```
00001                          NAM    CHAROUT2 REV. 0.2
00002                          OPT    NOP
00003 0000                     ORG    0
00004       02F0    SEGMEN     EQU    $02F0
00005 0000  0001    CHARAC     RMB    1
00006 0001  CE 02F7 OUTCHR     LDX    #SEGMEN+7  TOP OF SEG. LIST
00007 0004  D6 00              LDA B  CHARAC     GET PATTERN
00008 0006  E7 00   NXTSEG     STA B  0,X        STORE TO LATCH
00009 0008  56                 ROR B             SHFT FOR NXT BIT
00010 0009  09                 DEX
00011 000A  8C 02EF            CPX    #SEGMEN-1  LAST SEG. YET?
00012 000D  26 F7              BNE    NXTSEG
00013 000F  3E                 WAI               DONE(39 FOR SUBROUTINE)
00014                          END
```

Figure 10-43
Program for writing data into a 7-segment display.

```
00001                              NAM     OUTSTRIG REV. 0.3
00002 0100                         ORG     $0100
00003       0000     CHARAC EQU     00
00004 0100  0002     ISAVE  RMB     2
00005       0001     OUTCHR EQU     01
00006 0102  CE 0124  START  LDX     #CODES    POINT TO CODE TBL
00007 0105  A6 00    NXTDIG LDA A   0,X       GET PATTERN
00008 0107  97 00              STA A  CHARAC    STORE IT
00009 0109  FF 0100            STX    ISAVE     SAVE INDEX
00010 010C  BD 0001            JSR    OUTCHR    OUTPUT DIGIT
00011 010F  FE 0100            LDX    ISAVE     RESTORE INDEX
00012 0112  86 FF              LDA A  #$FF      *
00013 0114  C6 55    HOLD1  LDA B   #$55      *
00014 0116  5A       HOLD2  DEC B             *
00015 0117  26 FD              BNE    HOLD2     * WAIT
00016 0119  4A                 DEC A             *
00017 011A  26 F8              BNE    HOLD1     *
00018 011C  08                 INX               POINT TO NXT CODE
00019 011D  8C 0134            CPX    #FINAL+1
00020 0120  27 E0              BEQ    START     RECYCLE
00021 0122  20 E1              BRA    NXTDIG    GET NXT DIGIT
00022 0124  03       CODES  FCB     $03,$9F,$25,$0D,$99
      0125  9F
      0126  25
      0127  0D
      0128  99
00023 0129  49                 FCB    $49,$41,$1F,$01,$19
      012A  41
      012B  1F
      012C  01
      012D  19
00024 012E  11                 FCB    $11,$C0,$63,$85,$61
      012F  C0
      0130  63
      0131  85
      0132  61
00025 0133  71       FINAL  FCB     $71
00026                          END
```

Figure 10-45

Program for outputting hex digits in sequence and in a cyclic manner. Requires program from Figure 10-43 as a subroutine.

Discussion

With each reduction in hardware, there is generally an increase in support software. The program in Figure 10-43 is used only to output the necessary data bits to produce a single hex character. Since eight separate address locations are needed to light the 7-digit segments and the decimal point, the program must output eight bytes of data in order to produce the desired display. This is accomplished by using the index register to monitor each segment address and outputting the appropriate data from the B accumulator.

Remember that only the D_0 data bit is connected to the display latch. Thus, you can enter the appropriate 8-bit word (for the hex digit) into the B accumulator and then write the word to the display, which only accepts the D_0 bit. After the word is written, the B accumulator is rotated right, which places the next most significant data bit (for the hex digit) at the D_0 position. The index register decrements to the next segment address and the program branches back to the store B accumulator step. This process continues until all of the display latches are filled, then the branch step defaults and the MPU goes into a wait for interrupt condition. The second program you entered is similar to the previous cyclic character output programs. At step 00010_{10}, a jump to subroutine instruction sends the MPU back to the character output subroutine.

Procedure (continued)

35. Switch the Trainer power off. Then remove the hookup wire and the components from the large connector block.

36. This completes this experiment. Return to the Unit Activity Guide of Unit 7.

Experiment 5

DATA INPUT

OBJECTIVES:

Show how to construct a circuit for writing data to the microprocessor.

Demonstrate various methods for programming the microprocessor to accept externally applied data.

Demonstrate a software routine for debouncing a switch.

Show how to select a debounce routine to fit a specific system.

Introduction

Experiment 4 introduced you to various methods of outputting data from the microprocessor. In this experiment, you will learn how to input data. While many devices can be used to transfer data to a microprocessor (teletypewriter, tape reader, modem, transducer, etc.), they all accomplish their task in basically the same manner. You will use the Trainer binary data switches and four external pushbutton switches for data entry.

Materials Required

1 ET-3400 Microprocessor Trainer

1 #1 switch

1 #2 switch

1 #3 switch

1 #4 switch

1 7400 integrated circuit (443-1)

1 74126 integrated circuit (443-717)

2 74LS30 integrated circuits (443-732)

1 74LS27 integrated circuit (443-800)

Hookup wire

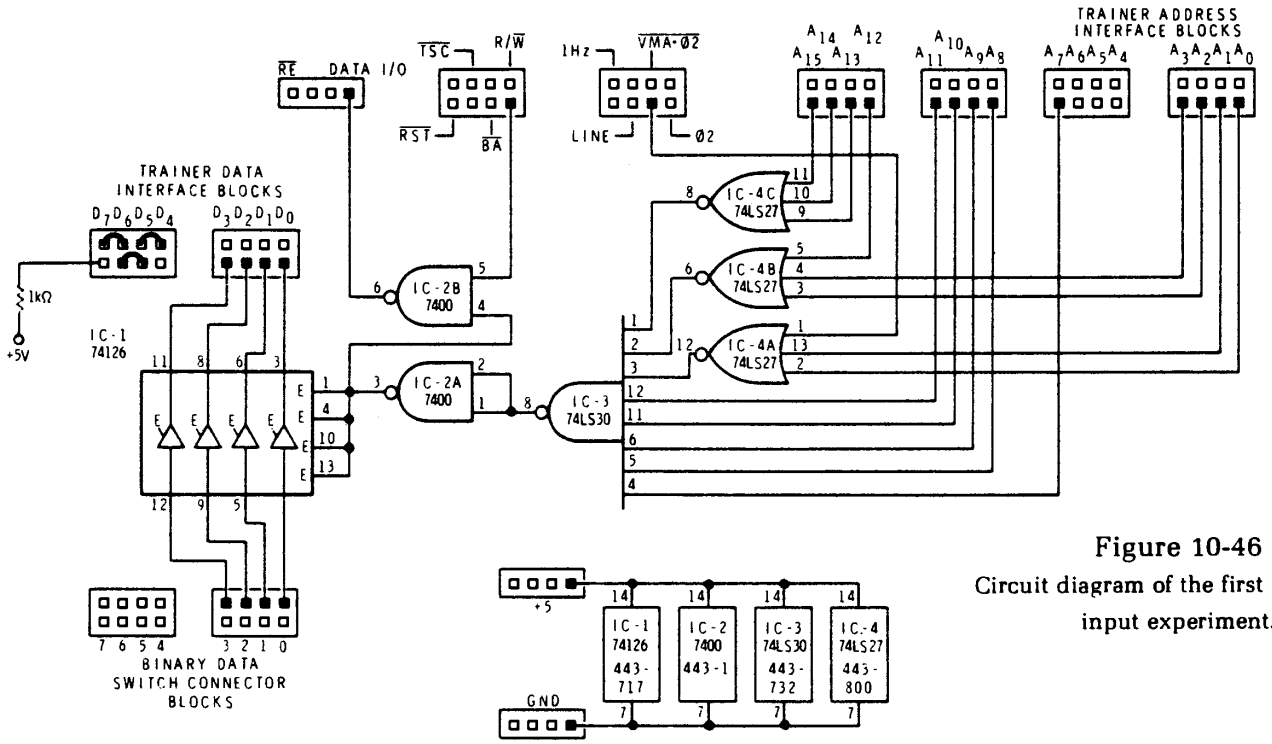


Figure 10-46
Circuit diagram of the first part of the input experiment.

Procedure

1. In the first part of this experiment, you will interface four slide switches to the data bus of the MPU. Make sure the Trainer power is switched off. Then construct the circuit shown in Figure 10-46.

2. Make sure all of the binary data switches are down (logic 0). Then position switch 0 up to logic 1.

NOTE: You may have noticed that the display is faintly illuminated with Trainer power off. This is caused by current from the data lines being coupled through IC1 to the +5-volt connector block, and from there to the displays. Disregard the display with power off.

3. Switch Trainer power on and enter the program listed in Figure 10-47. Then execute the program beginning at address 0000_{16}.

```
00001                          NAM     INPUT-01 REV. 0.2
00002                          OPT     NOP
00003 0000 B6 0F80             LDA A   $0F80     GET DATA
00004 0003 B7 0100             STA A   $0100     SAVE IT
00005 0006 3E                  WAI               DONE
00006                          END
```

Figure 10-47
Program for inputting data from the binary data switches.

4. Examine address 0100_{16}. What is the contents? $\_\_{}_{16}$.

5. Position data switch 0 down to logic 0. Then position data switch 1 up to logic 1.

6. Execute the program. Then examine address 0100_{16}. What is the contents? $\_\_{}_{16}$.

7. Position data switches 0 thru 3 up to logic 1.

8. Execute the program. Then examine address 0100_{16}. What is the contents? $\_\_{}_{16}$.

9. Enter the program listed in Figure 10-48.

10. Execute the program beginning at address 0000_{16}. Now flip data switch 0 between logic 1 and logic 0 a number of times and observe the decimal point of Trainer display H. Notice that the decimal point is lit for a logic 1 and off for logic 0.

Discussion

Refer again to the circuit in Figure 10-46. It is quite similar to the one used for outputting data. However, it operates like **read only memory**, with its data being influenced by external sources, (the "outside world").

```
00001                          NAM     INPUT-02 REV. 0.2
00002                          OPT     NOP
00003 0000 B6 0F80 REDO        LDA A   $0F80     GET DATA
00004 0003 B7 C167             STA A   $C167     STORE IT
00005 0006 20 F8               BRA     REDO      GO BACK AGAIN
00006                          END
```

Figure 10-48
Program to follow and display input from data switch 0.

The circuit is partially decoded as shown in Figure 10-49. When any of the specified addresses is selected, the buffer drivers of IC1 are enabled through inverter IC2A. This allows the data switch logic to be coupled to the Trainer data bus buffers. As soon as the R/$\overline{W}$ line goes high (MPU read), gate IC2B enables the input portion of the Trainer data bus buffers through the $\overline{RE}$ line.

You may have noticed one flaw in the circuit. The buffers in IC1 are always enabled when any of the circuit decoded addresses are selected. Therefore, it is important that you as the programmer do not try to write to these addresses. If you did so, the buffers would try to source or sink the data lines and result in possible circuit damage. One way to avoid this problem is to disconnect pin 4 of IC3 from A_7 and connect it to the R/W line. This will disable IC1 during an MPU write, but the circuit address coding is now changed to $00001111\bullet\bullet\bullet\bullet0000_{16}$.

Both programs in this experiment used address $0F80_{16}$ as an input port. The first retrieves data from $0F80_{16}$ and stores it at 0100_{16}.

The second program also retrieves data from $0F80_{16}$. But this time, it is stored at $C167_{16}$, the address of the decimal point for Trainer display H. Since only the D_0 data bit is connected to the Trainer display, data switch 0 is the only switch to affect the display. The program continuously branches back and retrieves switch data immediately after storing the previous data. Thus, when you changed the position of data switch 0, the decimal point appeared to follow the logic value of the changing switch position.

Next, some additional hardware and software features will be added to the circuit.

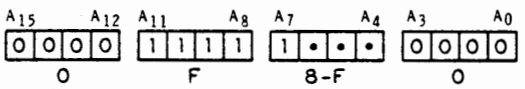


Figure 10-49
Decoding chart for the first input circuit.

Procedure (continued)

11. Refer to Figure 10-50 and construct the circuit shown. This circuit interconnects with the first circuit you constructed. Remember, the pushbutton pins are fragile. Press straight down when you install them in the large connector block, mounted on the Trainer cabinet.

12. Position all of the Trainer binary data switches up to logic 1.

13. Load the program listed in Figure 10-51, beginning at address 0000_{16} (program step 00007).

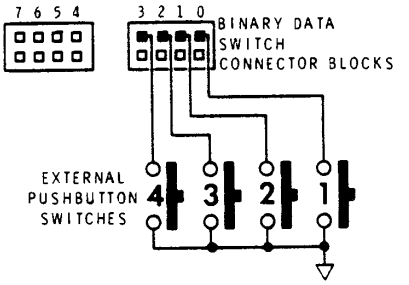


Figure 10-50
Added circuitry for the data input experiment.

```
00001                              NAM     KEYINDIC REV.0.2
00002                              OPT     NOP
00003 0000                         ORG     0
00004       FE3A     OUTCH   EQU     $FE3A
00005       FE52     OUTSTR  EQU     $FE52
00006       FCBC     REDIS   EQU     $FCBC
00007 0000 BD 0031   ONE     JSR     CLRDIS    CLEAR DISPLAY ROUTINE
00008 0003 BD FCBC           JSR     REDIS     RESET DIGIT POSITION (LEFT)
00009 0006 F6 0F80           LDA B   $0F80     LOOKING FOR KEY CLOSURE
00010 0009 86 30             LDA A   #$30      BIT PATTERN FOR #1
00011 000B 56                ROR B             MOVES "D0" BIT TO C REGISTER
00012 000C 25 02             BCS     TWO       NOT ONE? GO TO TWO
00013 000E 8D 17             BSR     XECUTE    OUTPUT A #1
00014 0010 86 6D     TWO     LDA A   #$6D      BIT PATTERN FOR #2
00015 0012 56                ROR B             MOVES "D1" BIT TO C REGISTER
00016 0013 25 02             BCS     THREE     NOT TWO? GO TO THREE
00017 0015 8D 10             BSR     XECUTE    OUTPUT A #2
00018 0017 86 79     THREE   LDA A   #$79      BIT PATTERN FOR #3
00019 0019 56                ROR B             MOVES "D2" BIT TO C REGISTER
00020 001A 25 02             BCS     FOUR      NOT THREE? GO TO FOUR
00021 001C 8D 09             BSR     XECUTE    OUTPUT A #3
00022 001E 86 33     FOUR    LDA A   #$33      BIT PATTERN FOR #4
00023 0020 56                ROR B             MOVES "D3" BIT TO C REGISTER
00024 0021 25 DD             BCS     ONE       NOT FOUR? GO BACK TO ONE
00025 0023 8D 02             BSR     XECUTE    OUTPUT A #4
00026 0025 20 D9             BRA     ONE       RETURN, RECHECK FOR CLOSURE
00027 0027 BD FE3A XECUTE    JSR     OUTCH     MONITOR ROUTINE OUTPUTS CHAR.
00028 002A CE 0100           LDX     #$0100    ENTER TIMING LOOP
00029 002D 09        HOLD    DEX               TIME RUNNING OUT
00030 002E 26 FD             BNE     HOLD      TIME OUT YET?
00031 0030 39                RTS               RETURN, RECHECK FOR CLOSURE
00032 0031 BD FE52 CLRDIS    JSR     OUTSTR    THE FOLLOWING CLEARS DISPLAY
00033 0034 00                FCB     00,00,00,00,00,$80
      0035 00
      0036 00
      0037 00
      0038 00
      0039 80
00034 003A 39                RTS
00035                        END
```

Figure 10-51
Program to display the pushbutton numbers in a sequential manner.

14. Execute the program beginning at address 0000_{16}. The decimal point in display C will light to show the program is working.

15. Press one of the four pushbuttons and note the displayed result.

16. Simultaneously press any two pushbuttons and note the result.

17. Simultaneously press any three pushbuttons and note the result.

18. Simultaneously press all four pushbuttons and note the result.

Discussion

The four pushbuttons in this experiment simply provide a convenient substitute for the four Trainer data switches. You could obtain the same result by manipulating the data switches. However, the pushbuttons will be needed in the next portion of the experiment.

The program shown in Figure 10-51 makes extensive use of the Trainer monitor routines located in ROM. These include OUTCH, OUTSTR, and REDIS. An earlier programming experiment showed how to use these routines.

Recall that OUTCH outputs a 7-segment code from accumulator A to the display indicated by a display pointer. OUTSTR outputs a string of characters to the displays. REDIS resets the display pointer so that the first character displayed by OUTCH or OUTSTR is in display H.

In addition, the program has two subroutines of its own. CLRDIS (for clear displays) is in addresses 0031_{16} through $003A_{16}$. It uses OUTSTR to clear the six displays. XECUTE is in addresses 0023_{16} through 0030_{16}. It uses OUTCH to display a character and then goes into a short delay.

The program starts at address 0000_{16}. The first instruction jumps the MPU to the CLRDIS subroutine. After the displays are cleared, the MPU returns to the instruction at address 0003_{16}. This instruction directs the MPU to the REDIS subroutine. This sets the display pointer to display H.

The MPU returns to the instruction in address 0006_{16}. Pushbutton data is now loaded into the B accumulator. Next, the 7-segment pattern for a "1" is loaded in the A accumulator. The D_0 bit in the B accumulator is examined. If it is a one (#1 pushbutton not pressed), the program branches foward to TWO. If the D_0 bit is a zero, the program branches to XECUTE. XECUTE displays the 1 in display H.

After XECUTE, an RTS sends the program back to address 0010_{16}. The A accumulator is loaded with the bit pattern for the digit "2". Then the B accumulator is again rolled right to test for a #2 pushbutton actuation. If #2 was pushed, it will be displayed; otherwise, the program will advance and test the remaining pushbuttons.

The pushbuttons that test true determine the numbers displayed. However, the display pointer determines the display that contains the number.

After all of the pushbuttons have been tested, the display is cleared and the display pointer again points to display H.

In many applications, the MPU constantly scans the input switches looking for input data. However, in some applications this would waste too much of the MPU's time. A better approach is to let the MPU ignore the keyboard until a key is depressed. This is possible through the use of interrupts. In the next part of the experiment you will see how a keyboard can control the MPU through the interrupt line. You will also see how a debounce subroutine works.

Procedure (continued)

19. Switch the Trainer power off. Then refer to Figure 10-52 and add the circuit shown to the circuit already wired to the Trainer. There should be enough room near the left end of the large connector block "on board" the Trainer to hold the additional 74LS30. Notice that the inputs to the 74LS30 are connected in parallel with the data lines leaving the four pushbutton switches. IC2C is one of the unused gates in IC2.

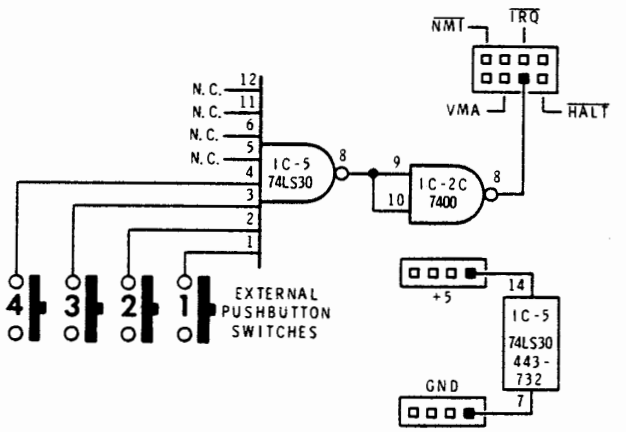


Figure 10-52
Interrupt circuitry for data input experiment circuit.

20. Switch Trainer power on. Then refer to Figure 10-53 and enter the program listed beginning at address 0000_{16}. Notice that after you enter the $3B_{16}$ at address $002B_{16}$, you must go to address $00F7_{16}$ to enter the remaining data. 002C and $002D_{16}$ are temporary registers.

21. Now enter 00_{16} into address 0100 thru 0110_{16}. These addresses are used as data storage registers.

22. Execute the program beginning at address 0000_{16}. The display will go blank.

23. Strike each pushbutton sequentially in a 1, 2, 3, 4 order. When you strike each button, use a moderate force, such as you would use when typing with a mechanical typewriter. The data you entered is stored in memory and will not be displayed.

```
00001                          NAM     DBOUNCE1 REV. 0.4
00002                          OPT     NOP
00003        0F80     INPUT    EQU     $0F80
00004 0000 0E                  CLI             READY FOR INTERRUPT
00005 0001 CE 0100 PROGRA      LDX     #$0100  POINT TO STORAGE
00006 0004 01                  NOP             *
00007 0005 01                  NOP             * LOCATION FOR PROGRAM
00008 0006 20 F9               BRA     PROGRA  *
00009 0008 B6 0F80 GETDAT      LDA A   INPUT   GET DATA
00010 000B B1 002C             CMP A   TEMP    IS IT LIKE BEFORE?
00011 000E 27 07               BEQ     SAME    IF SO, GO TO SAME
00012 0010 B7 002C             STA A   TEMP    IF NOT, STORE IN TEMP
00013 0013 7F 002D             CLR     COUNT   RESET COUNTER TO ZERO
00014 0016 3B                  RTI
00015 0017 C6 40    SAME       LDA B   #$40    NUMBER OF CHECKS
00016 0019 F1 002D             CMP B   COUNT   ENOUGH CHECKS YET?
00017 001C 27 04               BEQ     LEGAL   IF SO,GO TO LEGAL
00018 001E 7C 002D             INC     COUNT   IF NOT,INCREMENT COUNT
00019 0021 3B                  RTI
00020 0022 43       LEGAL      COM A           INVERT LOGIC
00021 0023 A7 00               STA A   X       PLACE IN STORAGE
00022 0025 7F 002D             CLR     COUNT   RESET COUNTER TO ZERO
00023 0028 08                  INX             POINT TO NEXT STORAGE PLACE
00024 0029 DF 02               STX     PROGRA+1 SAVE I DURING RTI
00025 002B 3B                  RTI
00026 002C 0001     TEMP       RMB     1
00027 002D 0001     COUNT      RMB     1
00028 00F7                     ORG     $00F7   INTERRUPT VECTOR
00029 00F7 7E 0008             JMP     GETDAT
00030                          END
```

Figure 10-53
Program to software debounce the input pushbuttons.

24. Examine address 0003_{16}. It should contain 04_{16}, which is the number of pushbutton contact closures made. Change the contents back to 00_{16}.

25. Examine addresses 0100 thru 0103_{16}. They should contain 01, 02, 04, and 08_{16} respectively. Change the data in these four locations back to 00_{16}. Even though the pushbuttons are labeled 1, 2, 3, and 4, they are connected to data lines D_0, D_1, D_2, and D_3. Therefore, the switches will enter the binary values 1, 2, 4, and 8.

26. Execute the program. Then press each pushbutton twice in succession (1, 1, 2, 2, 3, 3, 4, 4). Address 0003_{16} now contains 08_{16}, representing eight pushbutton contact closures. Enter 00_{16} at address 0003_{16}.

27. Examine addresses 0100 thru 0107_{16}. They will show the value of each pushbutton pressed and the sequence it was pressed. Change the data in these address back to 00_{16}.

28. Examine address 0018_{16}. It should contain data 40_{16}. Change the value to 00_{16}.

29. Execute the program. Then press each pushbutton once in sequence.

30. Examine address 0003_{16} and record the contents. $__{16}$. This number should equal 04_{16}. However, it may be higher.

31. Record the data in the following addresses. You need only examine the number of addresses that correspond to the value recorded in step 30.

0100 _ _	0109 _ _
0101 _ _	010A _ _
0102 _ _	010B _ _
0103 _ _	010C _ _
0104 _ _	010D _ _
0105 _ _	010E _ _
0106 _ _	010F _ _
0107 _ _	0110 _ _
0108 _ _	

Discussion

IC5 and gate IC2C provide an interface between the four external pushbuttons and the interrupt request line ($\overline{IRQ}$). The remaining circuitry functions as before. Thus, whenever you attempt to enter data with the pushbutton switches, a request for program interrupt signal is sent to the microprocessor.

The program listed in Figure 10-53 processes the interrupt and debounces the keys. The program is actually two programs in one. The first part (steps 00005 through 00008) serves as a "simulated" program that runs in an eternal loop until it is interrupted. The remaining program steps actually service the input data pushbuttons during the interrupt. This is the program we will deal with.

Figure 10-54 is a flow chart for the interrupt program. The numbers at each block represent the assembled program steps.

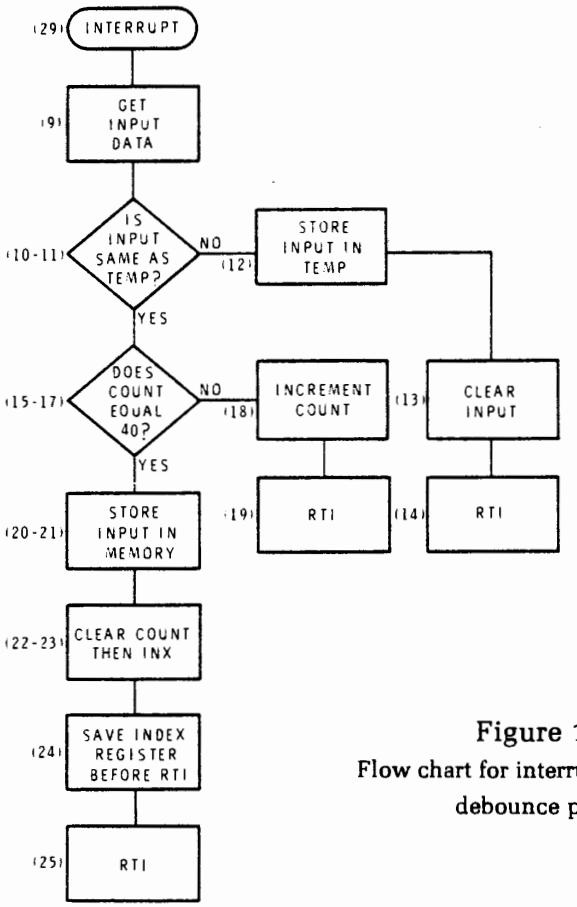


Figure 10-54
Flow chart for interrupt routine in the debounce program.

When the MPU receives an interrupt request; it completes the instruction it is presently performing, loads the internal registers and accumulators into the stack, sets the interrupt mask in the condition code register, then examines ROM to find out where the program counter is to be vectored. The vector address instruction sends the program counter to the beginning address of the interrupt program.

Pushbutton data is loaded and compared to the data in the temporary register (address $002C_{16}$). Since this is the first time data is examined, there can be no match. Therefore, the pushbutton data is stored in the temporary register, the counter register (address $002D_{16}$) is reset, and the MPU returns to the original program. This is the first time the MPU looks at the pushbuttons during the debounce routine. The data in the temporary register will serve as the reference for all future interrupts. If the input data changes, this new data will be entered, and the counter register will be reset. The counter is used later in the interrupt program to monitor the number of data examinations performed.

Upon return from the interrupt program, the MPU pulls the accumulator and register data from the stack. This clears the interrupt mask, and since you still have the pushbutton pressed, the MPU immediately acknowledges the interrupt request. Whereupon, it stores into the stack, sets the mask, and looks up the interrupt vector.

Pushbutton data is again compared with the temporary register. This time, it matches. Thus, allowing a branch to address 0017_{16}. Data 40_{16} is loaded into the B accumulator and then compared with the count register. Since the count is zero, there is no match. Therefore, the count is incremented and the MPU returns to the main program.

Assuming you are still holding the pushbutton down, the MPU goes through the interrupt routine 38 more times (39 total). During the 40^{th} cycle, if the data is still good, the MPU will be satisfied that the data supplied by the pushbutton is true, and the program is allowed to branch to address 0022_{16}.

The contents of accumulator A (pushbutton data) is complemented and stored at the address pointed to by the index register. This address was loaded into the index register in the main program. It is the first of 17_{10} addresses you reserved for data when you performed the experiment.

The counter register is cleared (in case the same pushbutton is again pressed). The index register is incremented and stored at address 0002_{16}. This points to the next data address, in preparation for the next pushbutton closure. Finally, the MPU returns to the main program.

You may have wondered why the pushbutton data was complemented before storage (address 0022_{16}). This was necessary since the pushbuttons were wired using inverse logic. That is, when the #1 pushbutton was pressed, data $1111\ 1110_2$ was transferred on the data bus, rather than $0000\ 0001_2$. Thus, it was necessary to invert the data for "logical" interpretation.

In the second part of this portion of the experiment, you changed the number of data examinations from 40_{16} to 00_{16} (address 0018_{16}). Then when you entered four pushbutton closures, you probably found more than four entries stored at address 0003_{16}. This occurred because the contacts of a switch tend to bounce open and closed a number of times before they stay closed. Since the bounce period can last many milliseconds, the MPU could treat each bounce as a separate entry, as you probably experienced.

Again look at the data you recorded in step 31. As you know, the program is designed to store one pushbutton closure in each address. A series of two or more identical entries indicates bounce. You may even have one or two zeroes recorded. This occurred because the contacts opened after an interrupt request, but before the data could be tested. Thus, a zero is stored.

Contact bounce can not be tolerated. But, what is a desirable number of switch samples? This will depend on the type of switch. If the sample is too low, bounce can occasionally get through. Large samples waste time and may require long switch hold-down periods. Normally five to eight samples are sufficient for a program of the type you used in this experiment. However, some switches will produce excessive bounce. As a precaution, 40 samples are used in the program.

Your Microprocessor Trainer uses a similar software routine for key debounce. This is stored in its ROM. Another method for debouncing a switch is to use cross-coupled NAND gates. They latch on the first closure and any additional bouncing is ignored. Regardless of the method used, you must debounce any mechanical switch used for data entry.

If you experiment with the sample rates in the program you entered, always be sure to change the data at addresses 0003_{16} and 0100 through 0110_{16} to 00_{16} before you execute the program.

Procedure (continued)

32. This completes this experiment. Switch the Trainer power off. Then remove all of the hookup wire and components from the two large connector blocks.

Experiment 6

INTRODUCTION TO THE PERIPHERAL INTERFACE ADAPTER (PIA)

OBJECTIVES:

Show how to interface the MPU with the outside world using a PIA (6820).

Demonstrate various ways the PIA can be initialized as an input, output, or input/output (I/O) device.

Introduction

As you have seen in the previous experiments, the need for latches and drivers to communicate with the MPU from the outside world can become quite burdensome. Then, once you have established a hardware interface circuit, you can not easily modify its function. However, the PIA can simplify your interface requirements in such a way that standardization is possible regardless of application. Therefore, you can easily develop interface systems compatible with your hardware and software needs. This is possible because most of the PIA performance characteristcs are software controlled. Thus, performance and design features can be modified with little difficulty. In this experiment, some of the PIA's characteristics will be examined.

Material Required

1 ET-3400 Microprocessor Trainer

2 1000 ohm, 1/4-watt, 10% resistors

1 7400 integrated circuit (443-1)

1 74LS30 integrated circuit (443-732)

1 74LS27 integrated circuit (443-800)

1 6820 PIA integrated circuit (443-843)

Hookup wire

Procedure

1. Make sure the Trainer power is off. Then construct the circuit shown in Figure 10-55. Caution: The PIA is a MOS device and should be handled properly.

2. Carefully reexamine the circuit you constructed. There should be a wire connected to each lead of the PIA.

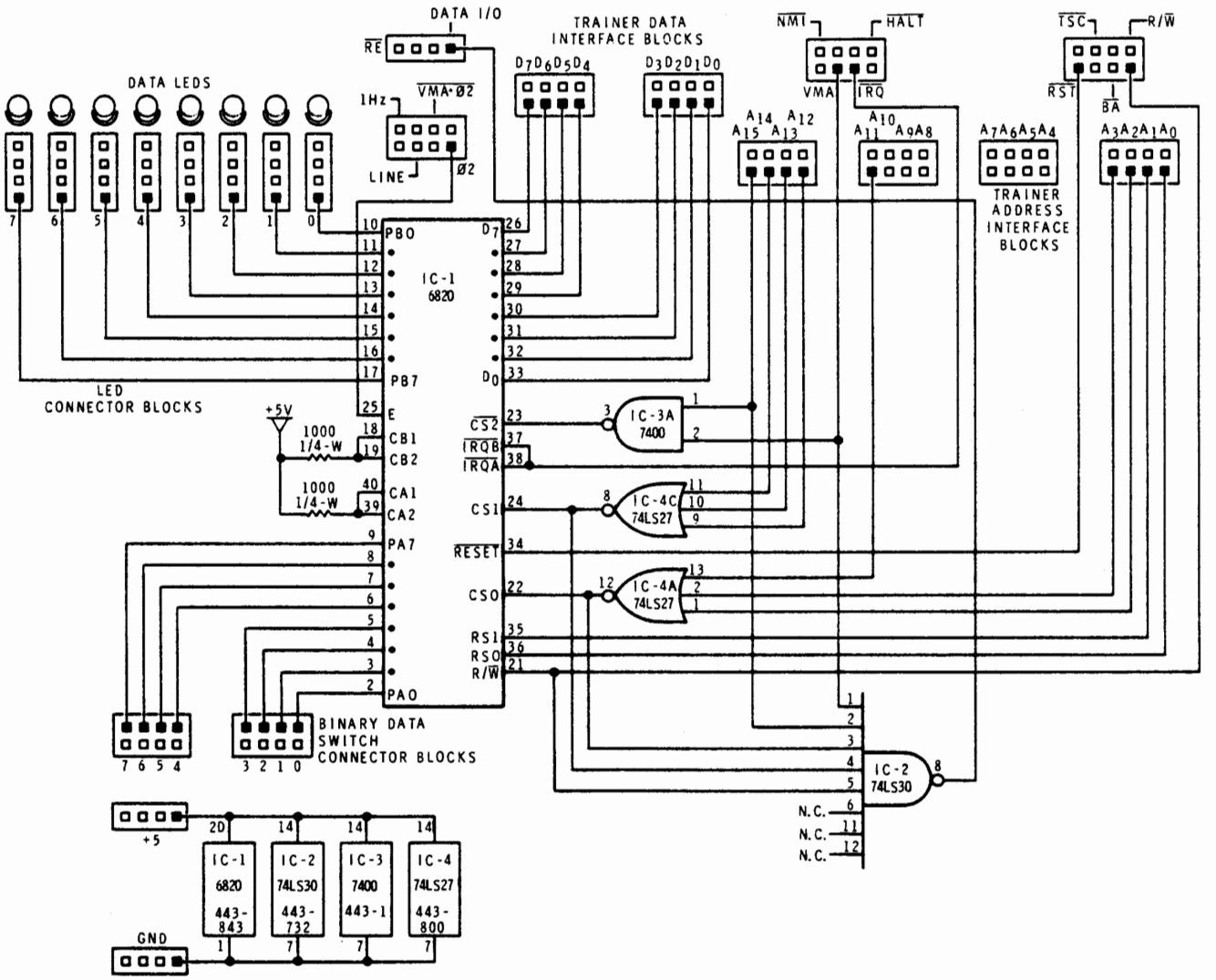


Figure 10-55
Circuit diagram for the first PIA experiment.

3. Switch the Trainer power on. Then enter the program listed in Figure 10-56.

4. Set all of the binary data switches to their down (logic 0) position. Then execute the program. The display will go blank.

5. Randomly set the data switches and observe the data LED's. Notice that the LED corresponding to each switch follows the logic level of the switch.

6. Change the instruction at address 0017_{16} to 43_{16}.

7. Execute the program and again randomly set the data switches. Notice that the data LED's now show the complement logic level of the switches.

8. Refer to Figure 10-56 and briefly describe the "service routine" section of the program. _____

_____

_____

_____

_____

_____

_____

```
00001                        NAM     PIA-EXP1 REV. 0.2
00002                        OPT     NOP
00003                  *INITIALIZE PIA
00004 0000 86 00             LDA A   #00       0=INPUT
00005 0002 B7 8000           STA A   $8000     A SIDE NOW INPUT
00006 0005 86 04             LDA A   #04       SET TO COMMUN.
00007 0007 B7 8001           STA A   $8001
00008 000A 86 FF             LDA A   #$FF      1=OUTPUT
00009 000C B7 8002           STA A   $8002     B SIDE NOW OUTPUT
00010 000F 86 04             LDA A   #04       SET TO COMMUN.
00011 0011 B7 8003           STA A   $8003
00012                  *SERVICE ROUTINE
00013 0014 B6 8000 RESERV    LDA A   $8000     GET DATA
00014 0017 01                NOP
00015 0018 B7 8002           STA A   $8002     STORE TO OUTPUT
00016 001B 20 F7             BRA     RESERV    DO IT AGAIN
00017                        END
```

Figure 10-56
Program to initialize and use the PIA for data input and output.

Discussion

By now you are quite familiar with address decoding. Therefore, the discussion will deal with the PIA. Figure 10-57 is a decoding chart for the circuit you wired to the Trainer. If during this discussion you don't fully understand a specific function of the PIA, refer to the PIA section in Unit 8 and the PIA data sheet in Appendix B.

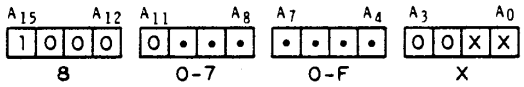


Figure 10-57

Decoding chart for the circuit in Figure 10-56.

Three chip-select pins on the PIA provide for easy decoding. They help eliminate address decoding gates. In small systems where partial address decoding can be tolerated, these three pins may be all that is needed to access the device. Notice that the PIA responds to addresses 8000_{16} through 8003_{16}.

The reset pin clears the PIA registers and is normally used at system turn-on. Therefore, it is connected to the system reset line.

The read/write pin controls data flow direction in the PIA in a manner similar to the RAM. Thus, it is connected to the R/W line from the MPU.

Interrupt request lines A and B can be wire OR'ed as in this experiment. Thus, each can transmit an MPU interrupt on the $\overline{\text{NMI}}$ or $\overline{\text{IRQ}}$ lines (in this experiment, the feature is not used).

Control pins A1 and B1 are inputs that are used to control the internal PIA interrupt flags. Control pins A2 and B2 can also serve as interrupt inputs or as peripheral control outputs. Since these features are not required for this experiment, each pin is pulled to a logic 1 to provide a termination and prevent undesired PIA interrupts.

The enable pin controls data transfer between the PIA and MPU. Since MPU data transfer occurs during time $\phi 2$, this pin is connected to Trainer $\phi 2$.

Data pins 0 through 7 are connected to the MPU data bus for device communication.

Peripheral pins A0 through A7 can be programmed as inputs or outputs. In this experiment, they are programmed as inputs and are connected to the binary data switches.

Peripheral pins B0 through B7 can also be programmed as inputs or outputs. In this experiment, they are programmed as outputs and are connected to the data LED's. Normally, the B side is used as an output because of its extra drive capabilities.

As you learned in Unit 8, the PIA must be initialized before it can function properly. This is accomplished through a software routine. Because initialization is accomplished by software, the PIA's operation can be modified at any time during the program.

When the PIA receives a reset pulse, its six memory accessible registers are cleared. Thus, whenever the Trainer RESET key is pressed, the PIA is reset. Because of this, the PIA must be initialized after each reset.

The program you entered (Figure 10-56) used the instructions in addresses 0000 through 0013_{16} to initialize the PIA. This programs the A side of the PIA as an input. Then, 04_{16} was loaded into control register A. This sets bit 2 of the control register high, which isolates the data direction register and accesses the output register.

In a like manner, the B side of the PIA is set up as an output by loading FF_{16} into the data direction register. Then the data direction register is isolated and the output register accessed by loading 04_{16} into the control register.

The remaining steps in the program comprise the service routine. The MPU reads data from the A side of the PIA and stores it to the B side. The "branch always" instruction holds the program in the service routine. Once the PIA is initialized, it will function as programmed until it is reset.

When you changed the instruction at address 0017_{16} to 43_{16}, you instructed the MPU to complement the data in the A accumulator before storing the data.

The program listed in Figure 10-58 is the same as the program you used, with one exception; the index register is used in place of the A accumulator for initializing the PIA. This reduced the number of program steps required.

Procedure (continued)

9. Do not disassemble the circuit you have wired to the Trainer. It will be used in the next experiment. Proceed to Experiment 7.

```
00001                          NAM     PIA-EXP2 REV. 0.2
00002                          OPT     NOP
00003                  *INITIALIZE PIA
00004 0000 CE 0004             LDX     #$0004
00005 0003 FF 8000             STX     $8000
00006 0006 CE FF04             LDX     #$FF04
00007 0009 FF 8002             STX     $8002
00008                  *SERVICE ROUTINE
00009 000C B6 8000 RESERV      LDA A   $8000     GET DATA
00010 000F 01                  NOP
00011 0010 B7 8002             STA A   $8002     STORE TO OUTPUT
00012 0013 20 F7               BRA     RESERV    DO IT AGAIN
00013                          END
```

Figure 10-58

Alternate program to initialize the PIA for data input/output.

Experiment 7

AUDIO OUTPUT

OBJECTIVES:

Show how a transducer can be interfaced with an MPU.

Provide an opportunity to write an output program that will supply the data to drive a speaker.

Demonstrate how different audible tones can be generated.

Introduction

With the proper interface, microprocessors are capable of producing meaningful audio sounds. These signals are often useful as indicators when the operator cannot monitor the display and would like to know when an event has occured.

Audible sounds can be produced in two ways. The first simply uses a buzzer that is activated by a change in output logic level, in the same manner as an LED. The second method actually drives an audio speaker. This experiment will use the second method to produce a variety of meaningful tones.

Materials Required

1 ET-3400 Microprocessor Trainer with PIA circuit wired to the large connector block

2 100 ohm, 1/2-watt, 10% resistors

1 100 μF electrolytic capacitor

1 Speaker

Foam tape (from previous experiment)

Solder (from Trainer kit)

Hookup wire

Procedure

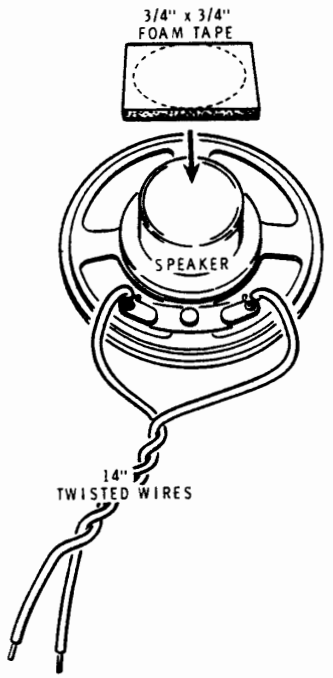


Figure 10-59
Speaker preparation.

1. Cut two 14" hookup wires and remove 1/4" of insulation from the ends of each wire. Then twist the wires together, leaving about 2" untwisted at each end.

2. Remove the speaker from its packing container. Then refer to Figure 10-59 and solder the two wires at one end of the 14" twisted wire pair to the two speaker terminals. Disregard any polarity marks on the speaker.

3. Cut a 3/4" × 3/4" piece of foam tape. Remove the paper backing from one side and press the tape onto the end of the magnet on the speaker. Then remove the paper backing from the other side of the tape and affix the speaker to the sloping back of the Trainer cabinet near the Power switch. Position the speaker lugs up away from the Trainer.

4. Switch the Trainer power off. Remove the eight wires interconnecting the data LED's and PIA. Then remove the eight wires interconnecting the binary data switches and PIA.

5. Refer to Figure 10-60 and construct the circuit shown. Connect the speaker wires and the capacitor to the unused large connector block. (Additional components will be added later in the experiment.) The gate is part of IC3 in the original circuit. You can use pin 7 of IC3 for speaker ground. Figure 10-61 shows the complete circuit wired to your Trainer.

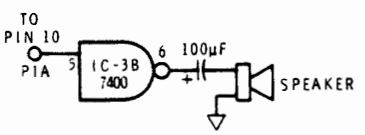


Figure 10-60
Speaker/PIA interface circuit.

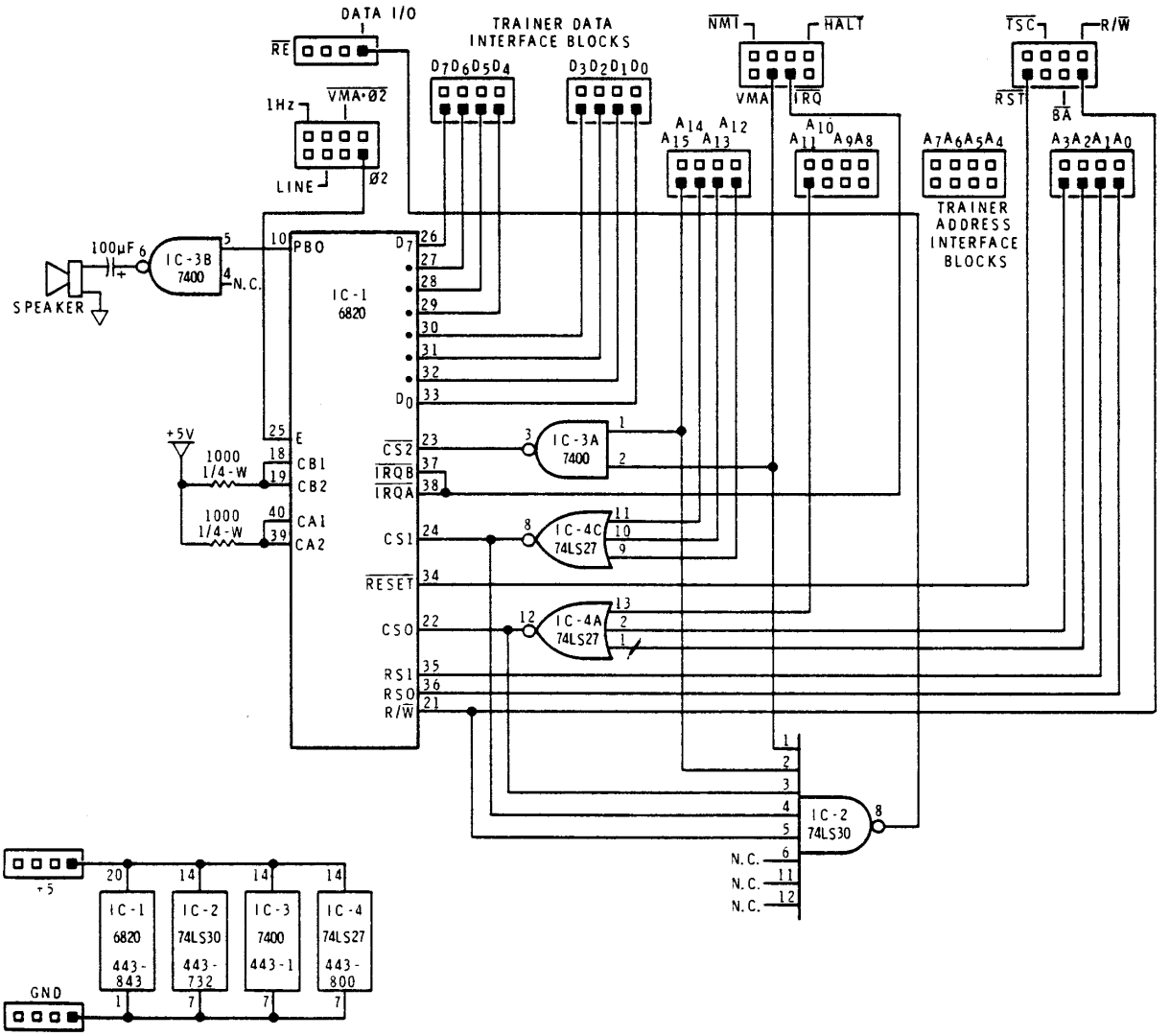


Figure 10-61
Circuit diagram for the audio output experiment.

6. Switch the Trainer power on. Load the program listed in Figure 10-62. Begin at address 0003_{16} (line 00008). Notice that a number of program steps have no data entry. Also, lines 00027_{16} and 00052_{16} contain assembler equate statements and should be ignored. After you enter 20_{16} at address $004B_{16}$, go to address $00F7_{16}$ to enter $3B_{16}$.

7. Press RESET, then install a hookup wire between LINE and $\overline{IRQ}$. This wire is not shown in the circuit diagram.

8. The program you entered is for a clock function. Addresses 0000, 0001, and 0002_{16} are reserved for the seconds, minutes, and hours of the clock respectively. Enter the desired time into these three addresses.

9. Execute the program beginning at address 0003_{16}. Each time the seconds count updates, you should hear a "tick" from the speaker.

Discussion

HARDWARE

Gate IC3B supplies the current needed to drive the speaker, while the 100 μF capacitor protects the gate. If the program stopped during a logic high output, the speaker would act as a direct short to ground. The capacitor coupling prevents this possibility. A side benefit of the capacitor is the RC time constant it forms with the internal gate circuitry. The pulse width for logic 1 and logic 0 transitions is different, producing a "tick-tock" sound.

SOFTWARE

The clock program you entered is very similar to the clock program from Experiment 2. Two instructions were added to initialize the PIA (lines 00008 and 00009). Also, a store instruction was added to the 60-second timer subroutine (line 00022).

The store instruction outputs the seconds digit information to the PIA (B side) every time the digit increments. Since the D_0 data bit is the only bit that changes during each time update, only peripheral output PB0 (pin 10) is needed to supply speaker data.

If you have any questions concerning the clock program, refer to Experiment 2.

```
00001                            NAM     CLOCK-3  * REV  0.2
00002                    **LINE ACCURACY CLOCK PROGRAM
00003                            OPT     NOP
00004 0000 0001          SECOND  RMB     1
00005 0001 0001          MINUTE  RMB     1
00006 0002 0001          HOURS   RMB     1
00007                    ** PIA INITIALIZATION
00008 0003 CE FF04               LDX     #$FF04
00009 0006 FF 8002               STX     $8002
00010                    ** INTERRUPT HANDLING
00011 0009 CE 003D       TIMPAS  LDX     #$003D   61
00012 000C 09            ONE60T  DEX              TIME TICKING OFF
00013 000D 27 04                 BEQ     TIMEUP   60 PULSES YET?
00014 000F 0E                    CLI
00015 0010 3E                    WAI              WAITING
00016 0011 20 F9                 BRA     ONE60T   GO BACK & WAIT AGAIN!
00017                    ** INCR ONE SECOND AND UPDATE
00018 0013 C6 60         TIMEUP  LDA B   #$60     SIXTY SECONDS,SIXTY MINUTES
00019 0015 0D                    SEC              ALWAYS INCREMENT SECONDS
00020 0016 8D 16                 BSR     INCR     INCREMENT SECONDS
00021 0018 96 00                 LDA A   SECOND
00022 001A B7 8002               STA A   $8002
00023 001D 8D 0F                 BSR     INCR     INCREMENT MINUTES IF NEEDED
00024 001F C6 13                 LDA B   #$13     TWELVE HOUR CLOCK
00025 0021 8D 0B                 BSR     INCR     INCREMENT HOURS
00026 0023 BD FCBC               JSR     REDIS    RESET DISPLAYS
00027      FCBC          REDIS   EQU     $FCBC
00028 0026 8D 17                 BSR     PRINT
00029 0028 8D 15                 BSR     PRINT
00030 002A 8D 13                 BSR     PRINT    PRINT HOURS,MINUTES,SECONDS
00031 002C 20 DB                 BRA     TIMPAS   DO IT ALL AGAIN
00032                    ** INCR - INCREMENT SUBROUTINE
00033 002E A6 00         INCR    LDA A   0,X      DATA WORD INTO A
00034 0030 89 00                 ADC A   #0       INCREMENT IF NECESSARY
00035 0032 19                    DAA              FIX TO DECIMAL
00036 0033 11                    CBA              TIME TO CLEAR?
00037 0034 25 01                 BCS     INC1     NO
00038 0036 4F                    CLR A
00039 0037 A7 00         INC1    STA A   0,X
00040 0039 08                    INX
00041 003A 07                    TPA
00042 003B 88 01                 EOR A   #1       COMPLEMENT CARRY BIT
00043 003D 06                    TAP
00044 003E 39                    RTS
00045                    ** PRINT - PRINT HEX BYTES
00046 003F 09            PRINT   DEX              POINT X AT BYTE
00047 0040 96 02                 LDA A   $02      WHAT'S IN HOURS?
00048 0042 26 03                 BNE     CONTIN   IF NOT ZERO
00049 0044 7C 0002               INC     HOURS    MAKE IT ONE
00050 0047 A6 00         CONTIN  LDA A   0,X
00051 0049 7E FE20               JMP     OUTBYT
00052      FE20          OUTBYT  EQU     $FE20    MONITOR ROUTINE
00053 00F7                       ORG     $00F7
00054 00F7 3B                    RTI
00055                            END
```

Figure 10-62
Clock program with audible tick-tock.

Procedure (continued)

10. Switch the Trainer power off. Then remove the wire interconnecting LINE and $\overline{IRQ}$. Switch the Trainer power on.

11. Write a program that will output the proper data to produce an audio tone from the speaker circuit. This program must: Initialize the PIA, alternately store 1's and 0's to the speaker in order to produce a tone, and provide a delay loop between each storage, to determine the frequency of the tone.

12. Execute the program.

Discussion

Figure 10-63 shows a program similar to the one you wrote. Notice that only two instructions were required to initialize the PIA. This is possible since only the B side will be used to output data.

Remember from the previous program, it is only necessary to change the D_0 bit of the output data, since that is the only bit connected to the speaker circuit. Therefore, you can start with a random number in the A accumulator (line 00007) and store the number to the PIA. After a short delay (lines 00008 thru 00010) the A accumulator is incremented (changing the D_0 bit logic level) and again stored to the PIA. This incrementing and storing of accumulator A can continue indefinitely since the only data of interest is the D_0 bit.

```
00001                          NAM     TONETEST REV. 0.2
00002                          OPT     NOP
00003                  *INITIALIZE PIA
00004 0000 CE FF04             LDX     #$FF04
00005 0003 FF 8002             STX     $8002
00006                  *PRODUCE TONE
00007 0006 B7 8002 ALTERN      STA A   $8002      OUTPUT BIT
00008 0009 C6 55               LDA B   #$55       DETERMINES FREQUENCY
00009 000B 5A          TONE    DEC B
00010 000C 26 FD               BNE     TONE
00011 000E 4C                  INC A              COMP. BIT
00012 000F 20 F5               BRA     ALTERN
00013                          END
```

Figure 10-63

Program to output a tone through the speaker.

Procedure (continued)

13. Enter the program listed in Figure 10-64. After you enter $F1_{16}$ at address 0024_{16}, go to address 0101_{16} and enter the remaining data bits. Notice that the program covers two pages. This listing second page has been condensed to show only the assembled program line numbers, addresses, and data. The data in addresses 000D and 000E controls the time of each note. The number in parentheses is for the **fast clock**. The number in addresses 0101 through 01BF determine the pitch, or frequency, of each note. Once more, the numbers in parentheses are for the **fast clock**.

14. Execute the program beginning at address 0000_{16}. Notice that after the program completes the song, there is a pause (of equal duration to the song) before the song repeats.

```
00001
00002 0000
00003 0000 7F 8003          CLR
00004 0003 7C 8002          INC
00005 0006 73 8003          COM
00006 0009 8E 0100          LDS
00007 000C CE 05FF (0CFF)   LDX
00008 000F 33               PUL B
00009 0010 5D               TST B
00010 0011 27 ED            BEQ
00011 0013 F7 0100          STA B
00012 0016 4C               INC A
00013 0017 F6 0100          LDA B
00014 001A 09               DEX
00015 001B 27 EF            BEQ
00016 001D 5A               DEC B
00017 001E 26 FA            BNE
00018 0020 B7 8002          STA A
00019 0023 20 F1            BRA
00020 0100                  ORG
00021 0100 0001
00022 0101 53 (9D)
      0102 42 (7C)
      0103 53 (9D)
      0104 42 (7C)
      0105 53 (9D)
      0106 42 (7C)
      0107 53 (9D)
      0108 42 (7C)
      0109 53 (9D)
00023 010A 42 (7C)
      010B 53 (9D)
      010C 42 (7C)
      010D 37 (69)
      010E 42 (7C)
      010F 37 (69)
      0110 42 (7C)
      0111 37 (69)
      0112 42 (7C)
00024 0113 37 (69)
      0114 42 (7C)
00025 0115 22 (41)
      0116 2B (52)
      0117 22 (41)
      0118 2B (52)
      0119 20 (3D)
      011A 29 (4D)
      011B 20 (3D)
      011C 29 (4D)
00026 011D 20 (3D)
      011E 29 (4D)
      011F 20 (3D)
      0120 29 (4D)
      0121 20 (3D)
```

Figure 10-64
Music program (Part 1 of 2).

```
                0122 29 (4D)        00031 014A 4A (8C)              0172 4A (8C)              019A 46 (84)
                0123 20 (3D)              014B 3E (75)              0173 37 (69)              019B 3A (6E)
                0124 29 (4D)              014C 4A (8C)              0174 4A (8C)              019C 46 (84)
                0125 20 (3D)              014D 3E (75)        00036 0175 31 (5C)              019D 3E (75)
00027           0126 29 (4D)              014E 4A (8C)              0176 3E (75)              019E 4A (8C)
                0127 20 (3D)              014F 3E (75)              0177 31 (5C)              019F 3E (75)
                0128 29 (4D)              0150 4A (8C)              0178 3E (75)              01A0 4A (8C)
                0129 20 (3D)              0151 3E (75)              0179 2B (52)              01A1 3E (75)
                012A 29 (4D)              0152 4A (8C)              017A 37 (69)        00041 01A2 4A (8C)
                012B 20 (3D)        00032 0153 3E (75)              017B 2B (52)              01A3 3E (75)
                012C 29 (4D)              0154 4A (8C)              017C 37 (69)              01A4 4A (8C)
                012D 20 (3D)              0155 3E (75)              017D 2B (52)              01A5 24 (45)
                012E 29 (4D)              0156 4A (8C)        00037 017E 37 (69)              01A6 3E (75)
00028           012F 20 (3D)              0157 3E (75)              017F 2B (52)              01A7 24 (45)
                0130 29 (4D)              0158 4A (8C)              0180 37 (69)              01A8 3E (75)
                0131 2B (52)              0159 3E (75)              0181 2B (52)              01A9 29 (4D)
                0132 37 (69)              015A 4A (8C)              0182 37 (69)              01AA 42 (7C)
                0133 2B (52)              015B 3E (75)              0183 2B (52)        00042 01AB 29 (4D)
                0134 37 (69)        00033 015C 4A (8C)              0184 37 (69)              01AC 42 (7C)
                0135 24 (45)              015D 3E (75)              0185 2B (52)              01AD 29 (4D)
                0136 2B (52)              015E 4A (8C)              0186 37 (69)              01AE 42 (7C)
                0137 24 (45)              015F 3E (75)        00038 0187 2B (52)              01AF 29 (4D)
00029           0138 2B (52)              0160 4A (8C)              0188 37 (69)              01B0 42 (7C)
                0139 29 (45)              0161 3E (75)              0189 2B (52)              01B1 29 (4D)
                013A 37 (69)              0162 5B (A7)              018A 37 (69)              01B2 42 (7C)
                013B 29 (45)              0163 3E (75)              018B 2B (52)              01B3 29 (4D)
                013C 37 (69)              0164 58 (A7)              018C 37 (69)        00043 01B4 42 (7C)
                013D 42 (7C)        00034 0165 3E (75)              018D 2B (52)              01B5 29 (4D)
                013E 37 (69)              0166 58 (A7)              018E 37 (69)              01B6 42 (7C)
                013F 42 (7C)              0167 3E (75)              018F 2B (52)              01B7 29 (4D)
                0140 37 (69)              0168 58 (A7)        00039 0190 37 (69)              01B8 42 (7C)
00030           0141 42 (7C)              0169 3E (75)              0191 31 (5C)              01B9 2B (52)
                0142 37 (69)              016A 58 (A7)              0192 3E (75)              01BA 31 (5C)
                0143 42 (7C)              016B 3E (75)              0193 31 (5C)              01BA 37 (69)
                0144 37 (69)        00035 016C 58 (A7)              0194 3E (75)              01BC 3E (75)
                0145 3A (6E)              016D 37 (69)              0195 37 (69)        00044 01BD 42 (7C)
                0146 46 (84)              016E 4A (8C)              0196 42 (7C)              01BE 4A (8C)
                0147 3A (6E)              016F 37 (69)              0197 37 (69)              01BF 00 (00)
                0148 46 (84)              0170 4A (8C)              0198 42 (7C)        00045     END
                0149 3E (75)              0171 37 (69)        00040 0199 3A (6E)
```

Figure 10-64

Music program (Part 2 of 2).

Discussion

The program you entered occupies memory locations 0000 through 0024_{16}. The remaining data represents the notes in the music. It was structured in this manner so that you could experiment with different songs. Figure 10-65 illustrates the various notes the program can produce, on the outline of an organ keyboard. Each note is listed with its actual fundamental frequency below the note letter. The number below the frequency is the hex number that will produce that approximate frequency. The number in parentheses is the one to use if your Trainer has been modified and has a **fast clock**. The notes your Trainer will produce depends on the MPU clock frequency. Even with the crystal-controlled clock in the modified Trainer, it is not possible to reproduce the exact frequency of the notes.

Although the music program is basically simple, there are a few unique features that should be examined. The first instruction clears control register B of the PIA. Naturally, this occurs prior to program execution. However, it will be necessary to modify the contents of data direction register B prior to each program cycle. Thus, bit two in the control register is cleared.

The second instruction turns bit PB0 on or off for each program cycle. Incrementing the data direction register will be of more value in the next section of this experiment.

Instruction four (LDS) tells the MPU that the data stored at 0101 thru $01BF_{16}$ now resides in the stack. However, the pointer contains address 0100_{16}. This is necessary, since each "pull" instruction adds "1" to the pointer prior to execution.

The last note in the stack is 00_{16}. This is used to indicate "end of music." Since a pull instruction does not affect any of the MPU condition codes, it is necessary to test for zero with instruction seven (TST B).

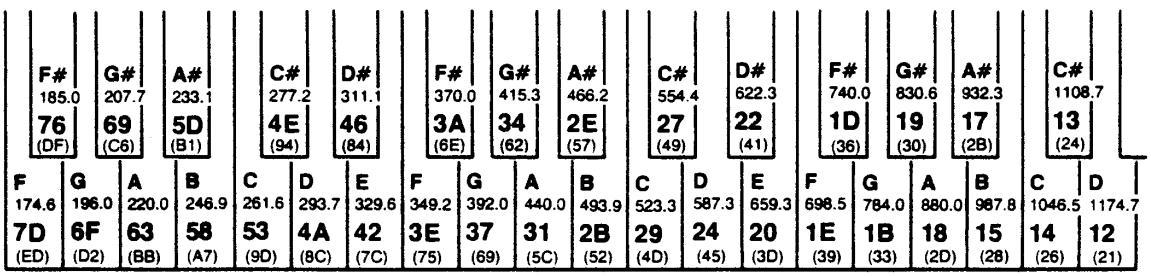


Figure 10-65
Music notes reproduced by the program in Figure 10-64.

The remaining program steps contain two timing loops. The first, starting at line 00007 sets the music tempo. The second, starting at line 00008 produces the notes.

NOTE: If you wish to listen to your ROM, enter FC_{16} at $000A_{16}$, and 01_{16} at 0010_{16}. It is necessary to remove the TST B instruction, since ROM contains a number of 00_{16} data bytes. Now, the program will continue until you press RESET.

Procedure (continued)

15. If you modified the music program (addresses 0000 through 0024_{16}), refer to Figure 10-64 and reenter the program. Its not necessary to reenter the same music notes if you have not modified them.

16. Refer to Figure 10-66 and modify your speaker circuit. Notice that a resistor is placed between gate IC3B and the speaker. Also, an additional gate (from IC3) and resistor interface pin 11 of the PIA with the speaker.

17. Execute the music program. This time, the music plays three times before there is a pause. The first repeat is an octave lower, and the second repeat simultaneously plays the original and octave lower music.

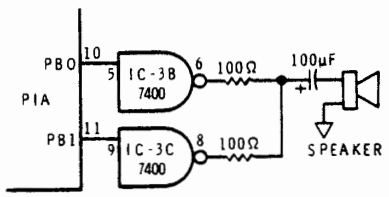


Figure 10-66
Modification to the speaker circuit.

Discussion

The gate connected to pin 11 of the PIA (data bit PB1) allows an additional output interface to the speaker. The two resistors reduce circuit loading so the outputs of the two gates can be combined at the coupling capacitor. However, the resistors also reduce the signal level and as a result, speaker volume.

The music program is unchanged from the previous section. However, you are now using two additional features in the program that previously were not required or apparent. The first concerns the PIA. Each time the program repeats, the data direction register bits are incremented. This meant the PB0 bit cycled the music on and off. Now that two output pins are wired to the circuit, a new pattern develops. First, pin PB0 is active. Then pin PB1 is active. Next, both pins are active. Finally both pins are inactive. Thereafter, the cycle repeats.

That cyclic pattern accounts for two (apparent) channels of music. But, why does one channel sound like it is one octave lower in frequency?

At the end of each "note" timing loop, the A accumulator is stored, and then incremented. Thus, the speaker is driven by a cyclic logic level transition of the D_0 data bit. However, every two level transitions in the D_0 bit causes a single level transition in the D_1 bit. Figure 10-67 illustrates the process. Since bit D_0 is coupled to PIA bit PB0, and bit D_1 is coupled to PIA bit PB1, you effectively have identical music material generated at two different octave levels.

If you have a stereo sound system, you can connect the music output of each gate (through a 100 ohm resistor) to the AUX input of your amplifier. The sound reproduction will be better than that produced by your Trainer.

This completes this experiment. Leave the circuit intact for the next experiment.

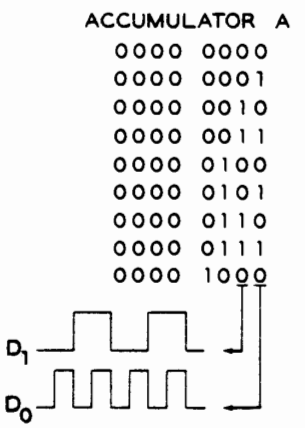


Figure 10-67
Music material coupled to the PIA.

Experiment 8

KEY MATRIX AND PARALLEL-TO-SERIAL CONVERSION

OBJECTIVES:

Demonstrate a method for using any combination of PIA I/O ports as inputs or outputs.

Show how a matrix-type keyboard decoder system works, and how it can be constructed.

Demonstrate parallel-to-serial conversion using the PIA.

Demonstrate a method for converting a hex digit to ASCII.

Show how a "one-shot" monostable works, using software.

Show how to add the parity bit to a serial word, using software.

Introduction

You are quite familiar with the PIA by now. In this experiment, you will demonstrate its versatility as an I/O device. In addition to using one peripheral port as both an input and output bus, you will see how a parallel data transfer device can be used to communicate in "serial." Since a great amount of serial data uses ASCII, this experiment will use the ASCII format.

Material Required

1 ET-3400 Microprocessor Trainer with PIA circuit wired to the large connector block

2 1000 ohm, 1/4-watt, 10% resistors

1 Pushbutton switch #1

1 Pushbutton switch #2

1 Pushbutton switch #3

1 Pushbutton switch #4

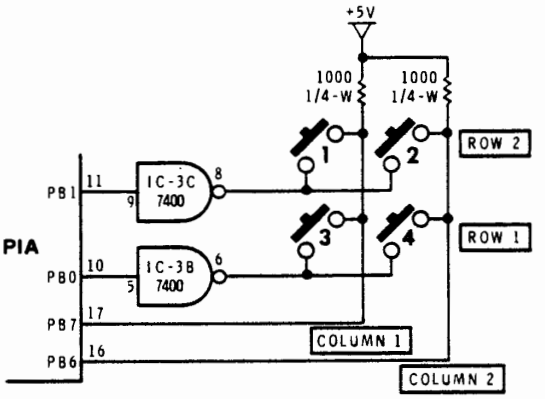


Figure 10-68
Matrix switch circuit.

Procedure

1. In this part of the experiment, you will see how the PIA interfaces with a switch matrix. Switch the Trainer power off. Then remove the 100 μF capacitor, two 100 ohm resistors, speaker and the wires associated with those parts.

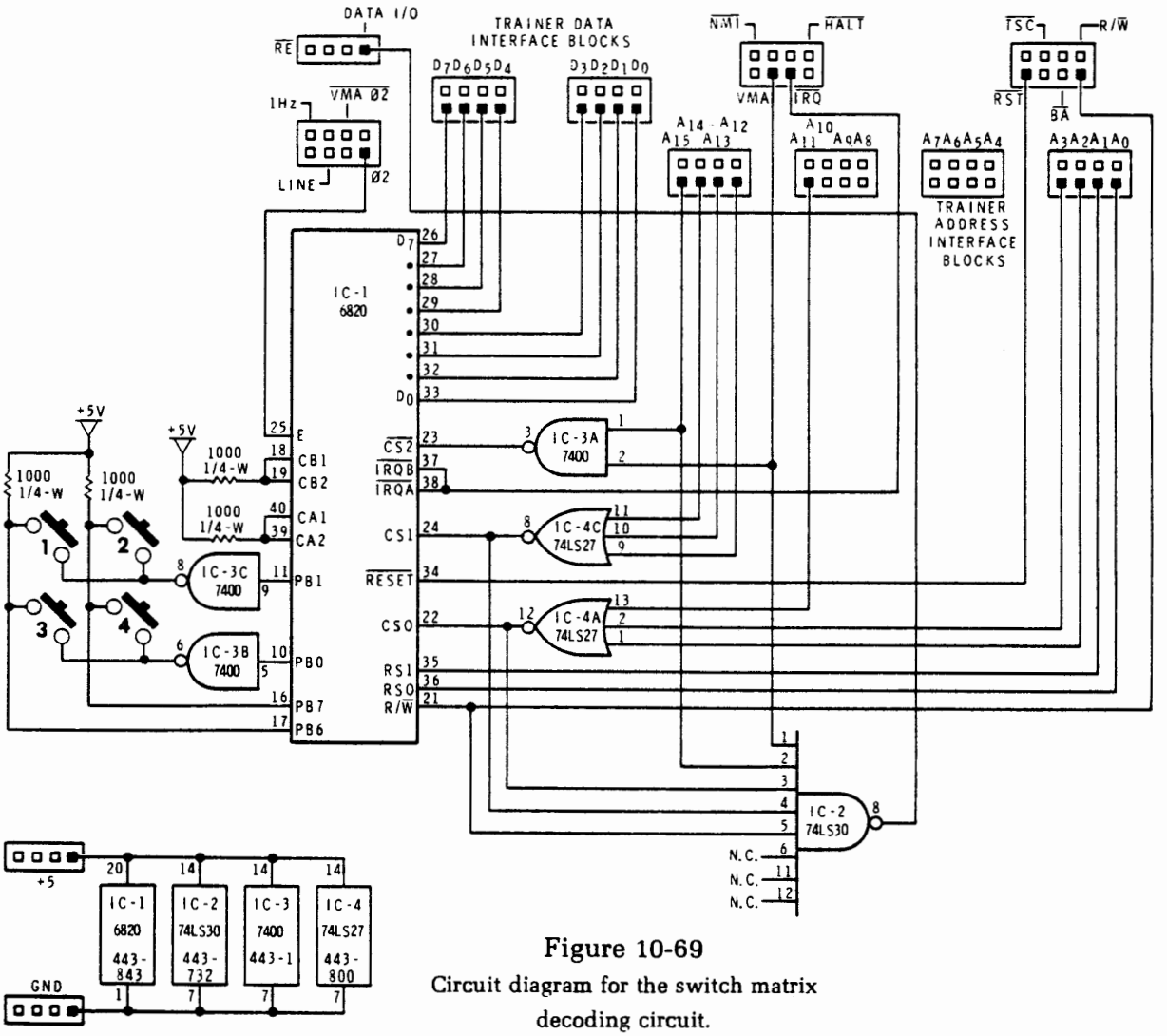


Figure 10-69
Circuit diagram for the switch matrix decoding circuit.

2. Refer to Figure 10-68 and add the circuit shown to the PIA circuit. Figure 10-69 shows the complete circuit wired to your Trainer.

3. Refer to Figure 10-70 and enter the program listed. Do not attempt to enter data at address $003E_{16}$. This will serve as a temporary storage register.

4. Execute the program. The display will go blank, except for the decimal point in digit H.

5. Randomly press the matrix circuit pushbuttons, individually and in combination, and observe display H.

```
00001                          NAM    KEYMATRI REV 0.4
00002                          OPT    NOP
00003       FE28    OUTHEX EQU    $FE28
00004       FCBC    REDIS  EQU    $FCBC
00005       FE52    OUTST1 EQU    $FE52
00006              *INITIALIZE PIA
00007 0000 CE 0F04             LDX    #$0F04
00008 0003 FF 8002             STX    $8002
00009              *SET DISPLAY LOCATION
00010 0006 BD FCBC NEWKEY JSR    REDIS     SET DISPLAY LOCATION
00011              *REFRESH WAIT
00012 0009 86 FF               LDA A  #$FF      *
00013 000B 4A      WAIT   DEC A               *WAIT
00014 000C 26 FD               BNE    WAIT      *
00015              *NUMBER OF KEYS
00016 000E 86 04               LDA A  #$04      TOTAL KEYS
00017 0010 B7 003E             STA A  KEYNUM    TO UPDATE
00018              *ROW SEARCH
00019 0013 86 01               LDA A  #$01      SELECT ROW ONE
00020 0015 B7 8002             STA A  $8002     OUTPUT TEST BIT
00021 0018 F6 8002 NXTROW LDA B  $8002     GET CLOSURE
00022 001B 2A 10               BPL    COL1      WAS IT COL1?
00023 001D 59                  ROL B
00024 001E 2A 10               BPL    COL2      WAS IT COL2?
00025 0020 7A 003E             DEC    KEYNUM
00026 0023 7A 003E             DEC    KEYNUM
00027 0026 27 10               BEQ    OUT       NO KEY INDICATION
00028 0028 78 8002             ASL    $8002     SHFT ROW BIT
00029 002B 20 EB               BRA    NXTROW    TEST NEXT ROW
00030              *COLUMN IDENTIFICATION
00031 002D 7A 003E COL1   DEC    KEYNUM
00032 0030 B6 003E COL2   LDA A  KEYNUM    GET KEY NUMBER
00033 0033 BD FE28             JSR    OUTHEX    MONITOR ROUTINE
00034 0036 20 CE               BRA    NEWKEY
00035              *BLANK DISPLAY
00036 0038 BD FE52 OUT    JSR    OUTST1    MONITOR ROUTINE
00037 003B 80                  FCB    $80       OUTPUT D.P. ONLY
00038 003C 20 C8               BRA    NEWKEY
00039 003E 0001    KEYNUM RMB    1         NUMBER TO BE DISPLAYED
00040                          END
```

Figure 10-70
Program for decoding key matrix.

Discussion

Refer to Figure 10-68. Notice that each pushbutton switch occupies a unique row and column electrically. Referring to the switch contacts, each "column" line is held at a logic 1 through a 1000 ohm resistor. Each "row" line is held at a logic 1 by the output of a NAND gate. In this circuit, the gates serve as inverter/drivers.

Assume that you close switch 2. In searching for the closed switch, the program first places a logic 0 on row 1 and then examines column 1 and column 2. Since both columns are still at a logic 1, row 1 is returned to logic 1, and row 2 is pulled to a logic 0. Again columns 1 and 2 are examined. This time, column 2 is found to be low, indicating that switch 2 is closed.

Refer to the program in Figure 10-70. Data direction register B is loaded with a bit pattern ($0F_{16}$) that sets pins PB0 and PB1 as outputs, and pins PB6 and PB7 as inputs. Although the other pins are set, they are not used in this experiment. Bit pattern 04_{16} then sets bit 2 of the control register high for access to the peripheral B interface.

A jump to monitor routine REDIS stores the address of display H in a temporary location in memory. This address will be used by other monitor routines to output data to display H.

The next three instructions provide a short time delay to help prevent character "ghosting." Some ghosting may be noticed in subdued light, due to the "rewriting" techniques used in this program.

Since four pushbutton switches (keys) are used in this experiment, decimal 4 is stored at temporary register 0041_{16}. This number or a decremented value of this number will be displayed when a switch closure is recognized.

The row search begins by storing 01_{16} to the PIA. This pulls the output of IC3B low (row 1). Then the PIA B side bus data is loaded into the B accumulator, and bit D_7 is tested for a 0. Assuming switch 2 was pressed, bit D_7 will test as a 1. The B accumulator is rotated left so that bit D_6 can be tested at the D_7 position. Since it also indicates 1, switches 3 and 4 have tested open.

Key number 4 in the temporary register (0041_{16}) is decremented twice prior to testing switches 1 and 2. Data stored at the PIA is shifted left. This pulls row 2 low.

The row search begins again at line 00021. When column 2 is tested, it is discovered that switch 2 is closed. The program branches to address 0033_{16}, and the A accumulator is loaded with the key number (2) from the temporary register. The monitor routine, OUTHEX, writes the number 2 into display H, and branches back to address 0006_{16} where the program begins again.

If all of the keys test open during row search, the program will branch to OUT, where a monitor routine lights the decimal point in display H and blanks all of the other displays. Then, the program branches back to address 0006_{16} and begins again.

You may have noticed that as you press more than one switch, the first switch tested will have priority. This occurs in a 3, 4, 1, 2 sequence.

Up to 16 switches can be tested with this program. By using both the A and B sides of the PIA, up to 64 switches can be tested. This can represent a big savings in peripheral interface logic.

Procedure (continued)

6. Switch the Trainer power off. Then remove the four switches, their two 1000 ohm resistors, and their associated wires. This includes the wires at pins 10 and 11 of the PIA.

7. Add the circuit shown in Figure 10-71 to the PIA circuit wired to your Trainer.

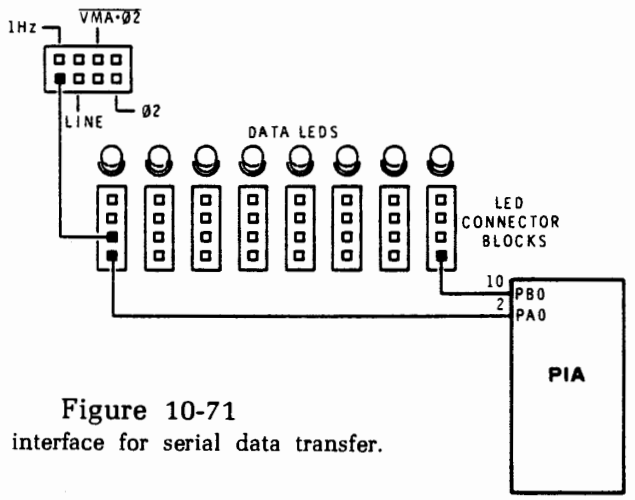


Figure 10-71
Display interface for serial data transfer.

8. Switch Trainer power on. Then enter the program listed in Figure 10-72. Do not attempt to enter data at address 0048_{16}. This address will serve as a temporary storage register.

When this program is executed, data LED7 will flash on and off at approximately a 1 Hz rate. The 1 Hz signal will be used as a program timing signal through PIA pin PA0. The flashing LED will serve as a visual timing reference.

```
00001                          NAM    SERIAL01 REV 0.6
00002                          OPT    NOP
00003        8000     PIAIN    EQU    $8000
00004        FDF4     INCH     EQU    $FDF4
00005        8002     PIAOUT   EQU    $8002
00006 0000 CE FE04            LDX    #$FE04
00007 0003 FF 8000            STX    PIAIN
00008 0006 CE FF04            LDX    #$FF04
00009 0009 FF 8002            STX    PIAOUT
00010 000C 7F 0048            CLR    TEMP
00011 000F 73 8002            COM    PIAOUT
00012                    *GET HEX CHARACTER
00013 0012 BD FDF4 NXTCHR JSR    INCH
00014 0015 01                 NOP                 *  8D   HOLD
00015 0016 01                 NOP                 *  39   FOR
00016 0017 01                 NOP                 *  8D   PROGRAM
00017 0018 01                 NOP                 *  47   MODIFICATION
00018                    *COMMENCE WITH START BIT
00019 0019 7F 8002 RESET  CLR    PIAOUT   RESETS ALL OUT BITS
00020 001C 8D 1B              BSR    DELAY
00021                    *OUTPUT THE CHARACTER
00022 001E B7 8002            STA A  PIAOUT   LSB IS STORED OUT
00023 0021 8D 16              BSR    DELAY
00024 0023 86 07              LDA A  #07      NO. OF TIMES SHIFTED
00025 0025 76 8002 WORD   ROR    PIAOUT   NEXT MSB IS STORED OUT
00026 0028 8D 0F              BSR    DELAY
00027 002A 4A                 DEC A           *CHECK FOR NUMBER OF
00028 002B 26 F8              BNE    WORD     *BITS SHIFTED
00029                    *OUTPUT 2 STOP BITS
00030 002D 86 01              LDA A  #$01     SET LSB
00031 002F B7 8002            STA A  PIAOUT   STORE IT TO OUTPUT
00032 0032 8D 05   WAIT   BSR    DELAY
00033 0034 4A                 DEC A           *WAIT FOR TWO
00034 0035 26 FB              BNE    WAIT     *BIT TIMES
00035 0037 20 D9              BRA    NXTCHR   DONE! GET NEXT CHAR.
00036                    *DELAY SUBROUTINE
00037 0039 F6 8000 DELAY  LDA B  PIAIN    SAMPLE INPUT LOGIC LVL.
00038 003C 50                 NEG B           INPUT NOW FF OR 00
00039 003D F1 0048            CMP B  TEMP     IS IT LIKE TEMP?
00040 0040 27 F7              BEQ    DELAY    IF SO, GET ANOTHER SAMPLE
00041 0042 73 0048            COM    TEMP     IF NOT, COMP TEMP
00042 0045 2B F2              BMI    DELAY    IF TEMP = FF, STAY IN LOOP
00043 0047 39                 RTS
00044 0048 0001   TEMP   RMB    1
00045                         END
```

Figure 10-72
Program to input serial data.

9. Execute the program. Data LED0 will light to indicate the program is running.

Discussion

Until now, you have observed data words being displayed in a parallel manner only. That is, all of the bits contained in the data word or byte were displayed simultaneously. You will now display a data word *serially*. In the serial mode, data bits are displayed one after the other. For this experiment, the bits-per-second or baud rate will be very slow so that you will be able to recognize each data bit. Baud is defined as one bit per second.

When you return to the experiment, you will use the Trainer keyboard to enter a hex number. The number will be converted to its binary form and transferred to data LED 0 one bit at a time. However, what you will see is not a 4-bit word, but rather an 11-bit word.

When serial data is transferred, it must fulfill certain format conventions. These include a start bit, to indicate the beginning of a word; the information being transferred; and a stop bit, to indicate the end of a word. Depending on the instrument sending data, and the baud rate, the data word can have one start bit, six, seven, or eight data bits, zero or one parity bit (often considered one of the data bits) and one or two stop bits.

The word format used in this experiment uses one start bit, eight data bits, and two stop bits. Figure 10-73 illustrates the serial word for hex 5. For added clarity, the timing signal for data LED7 is included. Notice that the actual data is transferred, beginning with the LSB. Also, only the first four data bits (plus the start bit) are of interest, since you are only transferring a single hex digit. Later in the experiment, the remaining four data bits will be used.

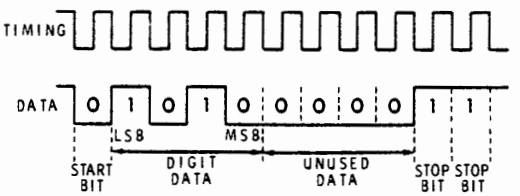


Figure 10-73
Serial data word format for the number 5_{16}.

Procedure (continued)

10. All timing is referenced to data LED7. Serial data is transferred through data LED0. When you are instructed to press a Trainer key, momentarily press the key while LED7 is **off**. It will take a little practice to be able to identify the data being transferred. Refer to Figure 10-73; it shows the data word you will observe when you press the 5 key. Press the 5 key and observe data LED's 0 and 7. Do it a number of times so that you can recognize each bit in the serial word.

11. Press a number of different Trainer keys and observe each serial word. You may find it helpful to illustrate each word, so you know what to expect.

Discussion

Data direction register A is set so all of the pins of peripheral bus A are outputs, except for pin PA0. Thus, it will not be necessary to electrically tie the unused pins to a logic 1. Data direction register B is set so all of the pins of peripheral bus B are outputs. Then, temporary register 0048_{16} is cleared.

Since output register B contains logic 0's from PIA reset, the contents of the register are complemented. This turns data LED0 on, its waiting condition. Everything is now prepared, and the main program can begin.

The program immediately jumps to monitor routine INCH and waits for an input from the Trainer keyboard. When a key is pressed, PIA pin PB0 is cleared, generating a start bit. The delay used to set the bit time will be described at the end of the program.

After the start bit delay, the key number, now residing in accumulator A, is stored in output register B. Data LED0 immediately displays the LSB of the number. Then accumulator A is loaded with the number representing the number of times output register B must be rotated right in order to display each data bit in data LED0. The data in output register B is rotated through pin PB0 with a time delay after each rotate.

Once all eight data bits have been stored, 01_{16} is stored to output register B from the A accumulator. This forces LED0 on, indicating the first stop bit. After two time delays, for the two stop bits, the program branches back to monitor routine INCH and waits for a new key closure.

The delay subroutine uses the 1 Hz clock for timing. This is why the serial data transfer coincides with the lighting of data LED7.

At the beginning of the delay routine, PIA peripheral interface A is examined. Pin PA0 should be logic 0 if you pressed the Trainer key while LED7 was off (logic 0). Therefore, 00_{16} is stored in the B accumulator. (01_{16} would be stored if LED7 was on.)

The data in the B accumulator is negated and then compared with the temporary register (0048_{16}). Since both registers are equal, the program loops back and examines pin PA0 again. This cycle continues until pin PA0 goes to a logic 1. Accumulator B is loaded with 01_{16}. The data is negated to produce FF_{16}.

Since the B accumulator and temporary register are no longer equal, the temporary register is complemented. This changes its contents to FF_{16}, which represents a negative number to the MPU. Because the temporary register contains a negative number, the program branches back to the beginning of the delay routine.

Pin PA0 is again repeatedly examined for a logic level change from 1 to 0. When this level transition occurs, the program returns the temporary register to its orginal 00_{16} condition, and then returns the program counter to the main program. Thus, you have effectively generated a software one-shot monostable with a time period determined by the 1 Hz signal.

Procedure (continued)

12. Enter the program listed in Figure 10-74. Notice that the program begins at address 0050_{16}.

```
00001                        NAM    ASCICONV REV 0.1
00002                        OPT    NOP
00003 0050                   ORG    $50
00004              *CONVERTS HEX TO ASCII
00005 0050 8A 30             ORA A  #$30      ASC NO. START WITH 3
00006 0052 81 39             CMP A  #$39      IS IT A NUMBER?
00007 0054 23 04             BLS    DONE      IF SO, DONE
00008 0056 80 09             SUB A  #$09      *LETTES START AT 1
00009 0058 8B 10             ADD A  #$10      *AND BEGIN WITH 4
00010 005A 39          DONE  RTS
00011                        END
```

Figure 10-74
Program to modify serial program for ASCII word format.

13. Now return to address 0015_{16} and enter $8D_{16}$. Then enter 39_{16} at address 0016_{16}.

14. Execute the program beginning at address 0000_{16}. This is the beginning of the serial output program.

15. A hex to ASCII conversion subroutine has been added to the serial output program. Refer to Figure 10-75. Notice that the ASCII representation for hex 5 is 35_{16}. Press the Trainer's 5 key and watch data LED0. Be sure to press the key while data LED7 is off. The serial data format remains unchanged, with one start bit, eight data bits, and two stop bits.

16. Press a number of Trainer keys, and observe the serial transfer for each key.

	COLUMN	0	1	2	3	4	5	6	7
ROW	BITS 765 ⇨ 4321 ⇩	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	\	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
10	1010	LF	SUB	*	:	J	Z	j	z
11	1011	VT	ESC	+	;	K	[	k	{
12	1100	FF	FS	,	<	L	\	l	|
13	1101	CR	GS	-	=	M	]	m	}
14	1110	SO	RS	.	>	N	^	n	~
15	1111	SI	US	/	?	O	_	o	DEL

Figure 10-75
Table of 7-bit American Standard Code for Information Interchange.

Discussion

Remember that the Trainer outputs the hex value in binary when a number is pressed. Therefore, when the 5 key is pressed, 05_{16} will be loaded into the A accumulator, in the serial output program. Then the program will branch to the ASCII conversion routine. This is caused by the branch instruction in address 0015_{16} and the **relative** address for the branch in address 0016_{16}.

The first instruction in the conversion routine OR's 05_{16} with 30_{16} to produce 35_{16}. The next instruction compares 35_{16} with 39_{16}. If the first value is smaller (which it is) or equal to 39_{16}, a valid ASCII number is present in the A accumulator, and the program counter is sent back to the main program.

If a valid number is not present, 09_{16} is subtracted from the value in the A accumulator. Assume that the C key was pressed. Then the number stored in the A accumulator is $0C_{16}$. When OR'ed with 30_{16}, it equals $3C_{16}$. Remember that a compare instruction does not alter the contents of the accumulator. Therefore, when 09_{16} is subtracted from the contents of the A accumulator, the result is $3C_{16}-09_{16}=33_{16}$. Finally, 10_{16} is added to the contents of the A accumulator, resulting in the value 43_{16}. Notice in Figure 10-75 that 43_{16} equals C in ASCII code. Again the program counter returns to the main program.

Procedure (continued)

17. Enter the program listed in Figure 10-76. Notice that this program begins at address 0060_{16}. Stop after you enter 39_{16} at address 0075_{16}. Address 0076_{16} is used as a temporary register.

```
00001                          NAM    ADDPARIT REV 0.1
00002                          OPT    NOP
00003 0060                     ORG    $60
00004                  *ADD PARITY BIT
00005 0060 7F 0076 PARITY CLR     PARIT1    START FRESH
00006 0063 C6 09           LDA B  #$09      TIMES TO SHIFT
00007 0065 49       BITCNT ROL A             SHIFT ONCE
00008 0066 24 03           BCC    NOINCR    SKIP INCR.
00009 0068 7C 0076         INC    PARIT1    COUNT LOGIC 1 BITS
00010 006B 5A       NOINCR DEC B             COUNT OFF TOTAL BITS
00011 006C 26 F7           BNE    BITCNT    ALL CHECKED YET?
00012 006E 76 0076         ROR    PARIT1    CHECK LSB OF PARIT1
00013 0071 24 02           BCC    FINIS     WAS IT ODD?
00014 0073 8A 80           ORA A  #$80      IF SO, SET PARITY BIT
00015 0075 39       FINIS  RTS
00016 0076 0001     PARIT1 RMB    1
00017                          END
```

Figure 10-76
Subroutine to add parity bit to serial output program.

18. Now return to address 0017_{16} and enter $8D_{16}$. Then enter 47_{16} at address 0018_{16}.

19. Execute the program beginning at address 0000_{16}. This is the beginning of the serial output program.

Discussion

Remember from Unit 1 that a parity bit is added to a serial word as a data transfer check. It indicates whether the sum of logic 1's in the data portion of the word are odd or even. Thus, if you desire an **even** parity check, the sum of the parity bit and all of the other data bits must equal an even number. Odd parity requires that all of the data bits plus the parity bit equal an odd number.

For example, the ASCII code for the number 5 is 35_{16}. Since 35_{16} equals $0011\ 0101_2$, the parity bit would be 0 for even parity and 1 for odd parity.

The parity bit occupies the eighth data bit position in the 8-bit data word. The first seven data bits contain the ASCII code.

Procedure (continued)

19. Determine the serial data word for hex 5. Then press the 5 key and observe LED0. Notice that the parity bit is 0 since the parity routine is configured for even parity.

20. Press a number of Trainer keys, and observe the serial transfer for each key.

21. Change the program data address 0071_{16} to 25_{16}. then press a number of Trainer keys, and observe the serial transfer for each key. The parity routine is now configured for odd parity.

Discussion

After the hex number is converted to ASCII code, the program branches from address 0017_{16} to the "even" parity routine. This routine determines if a 1 or 0 will be placed in the eighth data bit to provide an even parity indication.

To begin, the temporary register at address 0076_{16} is cleared. This register will store the count of the number of logic 1 bits found in the data word. Accumulator B is loaded with 09_{16}, which will be decremented to monitor the bit count routine.

Accumulator A is rotated left and the carry bit is checked. If the carry is set, the temporary register is incremented; then the B accumulator is decremented. If the carry is clear, the B accumulator is immediately decremented. Since the B accumulator is not zero yet, the program loops back, and the process continues.

Notice that the A accumulator is rotated nine times. This is necessary since there are actually nine data bits including the carry bit, and the contents of this accumulator will be used by the main program. The carry bit will not affect the logic 1 bit count, since it was cleared by instruction $7F_{16}$.

After all of the bits in the A accumulator have been examined, and all 1's stored in the temporary register, the temporary register is rotated right. This places bit D_0 in the carry bit position. the carry bit is then examined. If it was clear, the program counter returns to the main program; the ASCII code contains an even number of 1's, and does not require modification. If the carry bit was set, accumulator A is OR'ed with 80_{16}, which places a 1 in the eighth (MSB) bit of the ASCII code. This makes the total number of 1's even in count. The program counter then returns to the main program.

To convert the parity bit routine to odd parity recognition, the code at address 0071_{16} is changed to 25_{16}. This causes a branch if the carry is set.

Procedure (continued)

22. Pull the circuit timing wire from the 1 Hz socket and connect it to the LINE socket.

23. Press a number of Trainer keys and observe the serial transfer at data LED0.

Discussion

When you press each Trainer key, you can see data LED0 flash on and off at a rapid rate. This is because the baud rate is now equal to the line frequency. Although the transfer rate appears to be quite fast, it is considered very slow for computer work. Typical speeds for a teletypewriter are almost twice as fast as the line frequency. Speeds as high as 96.2 kilobaud are quite common.

Procedure (continued)

24. Do not disturb the circuit wired to your Trainer. It will be used in the next experiment. Proceed to Experiment 9.

Experiment 9

DIGITAL-TO-ANALOG AND ANALOG-TO-DIGITAL CONVERSION

OBJECTIVES:

Show how to connect a digital-to-analog converter (DAC) to a microprocessor system.

Demonstrate how a DAC converts digital information into an analog equivalent.

Demonstrate a number of programs that will produce variable analog signal levels from a DAC.

Show how to convert a DAC into an analog-to-digital converter with a voltage comparator.

Demonstrate a program that implements the analog-to-digital conversion.

Introduction

When analog input and output capabilities are added to a microprocessor, its power can be greatly expanded. Basically, the microprocessor is a digital device that is ideal for control of discrete input/output levels. However, many analog signals can also be processed with a minimum of additional hardware. With this addition, such devices as temperature sensors and photo cells can be monitored, and analog signals can be coupled to various peripherals such as oscilloscopes and audio amplifiers.

In this experiment, you will learn how to use the digital-to-analog converter (DAC) for outputting an analog signal. You will also learn how to translate an analog signal into its equivalent digital value using the same DAC.

Material Required

1 ET-3400 Microprocessor Trainer with the PIA circuit wired to its large connector block

3 1000 ohm, 1/4-watt, 10% resistors

1 2000 ohm, 1/2-watt, 5% resistor

2 2700 ohm, 1/2-watt, 5% resistors

2 10k ohm, 1/2-watt, 5% resistors

1 1M ohm, 1/2-watt, 5% resistor

2 1000 ohm controls

1 47 pF ceramic capacitor

3 100 pF ceramic capacitors

1 1N4149 diode (56-56)

1 5.6-volt zener diode (56-616)

1 741 op amp integrated circuit (442-22)

1 301 op amp integrated circuit (442-39)

1 MC1406 DAC integrated circuit (443-842)

1 VOM, VTVM, or DVM with input impedance greater than 100 k ohm (11 M ohm desirable)

1 Oscilloscope (optional)

Procedure

1. Switch the Trainer power off. Then remove the wires interconnecting LINE and data LED7, data LED7 and PIA pin 2, and data LED0 and PIA pin 10. The remaining PIA circuitry will be used in this experiment.

2. Refer to Figure 10-77 and construct the circuit shown, on the large connector block affixed to the Trainer cabinet. Be careful when you insert 1/2-watt resistor leads. It is easy to push a connector strip out of the bottom of the block. The 1000 ohm control leads will fit into the connector block if you straighten them out and then insert the control at a slight angle. You can also solder a short wire to each control lead. Figure 10-78 shows the complete PIA circuit with the DAC circuit added.

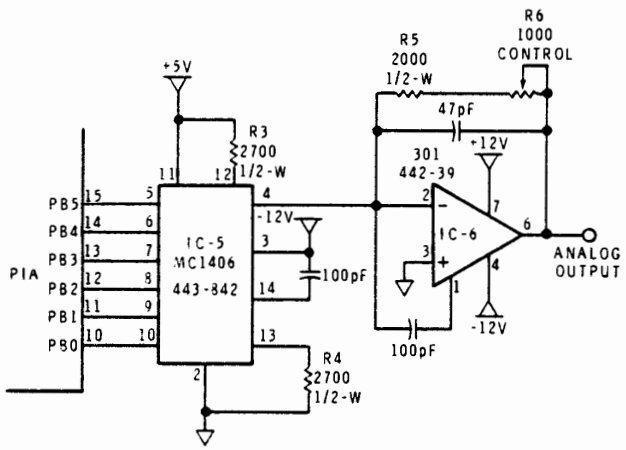


Figure 10-77
DAC circuit added to the PIA interface circuit.

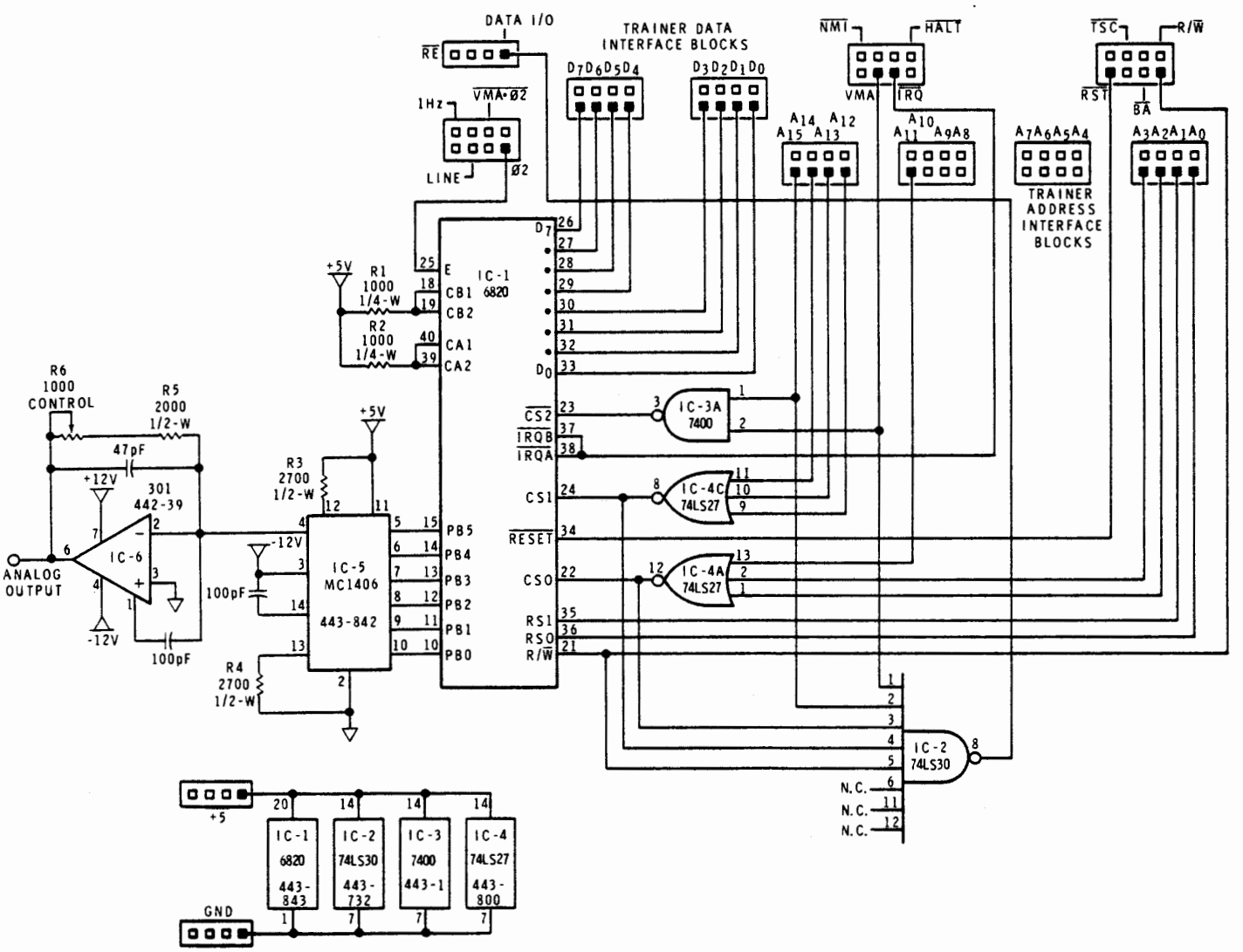


Figure 10-78

Circuit diagram for the digital-to-analog conversion circuit.

3. Switch the Trainer power on. Then enter the program listed in figure 10-79. Notice that the program begins at address 0100_{16}.

4. Execute the program. Then, using the Trainer keyboard, press the 3 key and then the F key.

5. Set your voltmeter to measure 5 volts DC. Then connect the common lead to circuit ground, and the input lead to pin 6 of IC6 (analog output). You should measure approximately 0 volts DC.

6. Press 00 with the keyboard. Then adjust control R6 for a 5-volt output level. If you can not obtain a 5-volt level: switch the Trainer power off, change the value of resistor R5 from 2000 ohms to 1000 ohms, and then switch the Trainer power on again.

7. Press 20. What is the voltage level?_____

8. Press 30. What is the voltage level? _____

9. Press 10. What is the voltage level? _____

```
00001                          NAM    LINOUT   REV. 0.1
00002                          OPT    NOP
00003       FCBC     REDIS     EQU    $FCBC
00004       FE09     IHB       EQU    $FE09
00005 0100                     ORG    $0100
00006                *CALIBRATION PROGRAM
00007 0100 CE FF04             LDX    #$FF04
00008 0103 FF 8002             STX    $8002    B SIDE IS OUT
00009 0106 BD FCBC NEWIN       JSR    REDIS
00010 0109 BD FE09             JSR    IHB
00011 010C B7 8002             STA A  $8002
00012 010F 20 F5               BRA    NEWIN
00013                          END
```

Figure 10-79
Program to convert digital values to their analog equivalent.

Discussion

The DAC converts a binary word into a proportional output current. This particular DAC is a 6-bit device. That means it can accomodate a 6-bit binary word. Therefore, the two most significant data bits from the PIA are not used.

This DAC requires a complemented input, that is, maximum current out occurs when the PIA output is 00_{16}. Minimum current occurs when all of the data lines are at a logic 1 level. This equals $3F_{16}$ (data bits PB6, PB7 not used).

Op amp IC6 (301) functions as a current-to-voltage converter. Feedback through resistor R5 and control R6 determines the gain of the op amp.

Since the DAC accomodates a 6-bit binary word, it is possible to have a conversion resolution of 2^6 or 64 discrete current levels. With IC6 calibrated for a voltage swing between 0 and 5 volts, each binary unit equals approximately 0.08 volts DC. Therefore, the approximate voltage levels you should have observed in steps 7, 8, and 9 are:

20_{16} = 2.44 VDC

30_{16} = 1.16 VDC

10_{16} = 3.72 VDC

Refer to the program shown in Figure 10-79. The first two instructions set up the PIA so the B side operates as an output. REDIS then points to display H for data display. A second monitor routine, IHB, couples key closure data to the display, and also loads the data into the A accumulator. Two key entries are required to complete the monitor routine. The data is then stored to the PIA. Since only the first six data bits are used by the DAC, 00, 40, and $C0_{16}$ will convert to the same analog levels.

Procedure (continued)

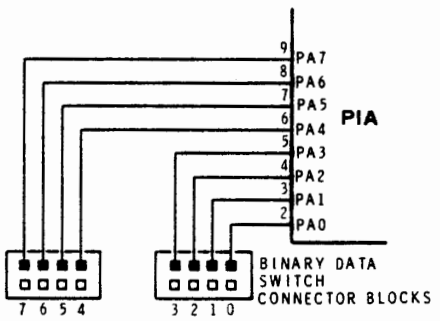


Figure 10-80
Peripherial data entry circuit for use with the digital-to-analog circuit.

10. Switch the Trainer power off. Then refer to Figure 10-80 and interconnect the PIA and the binary data switches.

11. Switch the Trainer power on. Then enter the program listed in Figure 10-81. Notice that the program begins at address 0120_{16}.

12. Execute the program located at address 0120_{16}. Then set the binary data switches for $1F_{16}$. The voltmeter should indicate a slowly decreasing voltage that cyclically ramps from 5 volts to 0 volts.

13. The rate of voltage change is determined by the binary data switches. If you have an oscilloscope, connect its input to pin 6 of IC6 and circuit ground. Now change the binary data switches to 00_{16}. You should observe a descending voltage ramp.

Discussion

The voltage ramp in this program is a composite of 64 discrete voltage steps. Each step represents a binary value. Therefore, an 8-bit DAC would have produced a ramp with 256 discrete steps, while a 4-bit DAC would contain only 16 steps.

```
00001                          NAM     OUTRAMP1 REV. 0.1
00002                          OPT     NOP
00003 0120                     ORG     $0120
00004 0120 CE 0004             LDX     #$0004
00005 0123 FF 8000             STX     $8000     A SIDE IN
00006 0126 CE FF04             LDX     #$FF04
00007 0129 FF 8002             STX     $8002     B SIDE OUT
00008 012C 4F                  CLR A
00009 012D B7 8002 NXTRMP      STA A   $8002
00010 0130 FE 8000             LDX     $8000     TIME TO WAIT
00011 0133 4C                  INC A             INCR RAMP STEP
00012 0134 09      STHOLD      DEX
00013 0135 26 FD               BNE     STHOLD
00014 0137 20 F4               BRA     NXTRMP
00015                          END
```

Figure 10-81
Program to generate a voltage ramp.

Procedure (continued)

14. Change the data at address 0133_{16} to $4A_{16}$. Then execute the program. You should observe an ascending voltage ramp. If you only have a voltmeter, change the binary data switches to $1F_{16}$. This will slow the ramp rate so you can observe the voltage change.

Discussion

The ramp program sets the A side of the PIA as an input, and the B side as an output. The A accumulator is cleared so the ramp will begin at +5 volts. Accumulator A is stored to the DAC through the PIA. Then the index register is loaded with the binary switch data. This will determine the waiting period between voltage ramp steps.

Remember that the index register always contains two bytes of data. The first byte (high byte) is obtained from the specified address, and the second byte (low byte) is obtained from the specified address plus one. Therefore, when you load the index register from the PIA, the high byte contains the binary switch data, and the low byte contains 04_{16}. 04_{16} is the data stored in the A side control register of the PIA during initialization.

Accumulator A is incremented for the next voltage level, and the index register is decremented until it reaches 0000_{16}. Then the program branches back to the store the A accumulator, and the cycle continues. It is not necessary to clear accumulator A once the voltage ramp reaches zero, since the next increment sets the six least significant binary data bits to zero.

Procedure (continued)

15. Enter the program listed in Figure 10-82. Notice that this program begins at address 0140_{16}.

16. Execute the program beginning at address 0140_{16}. Set the binary data switches to $1F_{16}$. The voltage should step from 0 to 5 volts and then back to 0 volts in a cyclic manner. You can observe this dual ramp (triangle) waveform on the oscilloscope, if you increase the ramp rate with the binary data switches.

Discussion

This program uses the A side of the PIA to enter step delay data and the B side to output the discrete voltage level data. However, this time the A accumulator is loaded with $3F_{16}$, which is the binary equivalent of 0 volts.

```
00001                          NAM     TRIANGLE REV. 0.1
00002                          OPT     NOP
00003 0140                     ORG     $0140
00004 0140 CE 0004             LDX     #$0004
00005 0143 FF 8000             STX     $8000      A SIDE IN
00006 0146 CE FF04             LDX     #$FF04
00007 0149 FF 8002             STX     $8002      B SIDE OUT
00008 014C 86 3F               LDA A   #$3F       DAC OUT = 0
00009 014E FE 8000 UP          LDX     $8000      DELAY TIME
00010 0151 B7 8002             STA A   $8002      OUT TO DAC
00011 0154 27 10               BEQ     DOWN1
00012 0156 4A        UP1       DEC A
00013 0157 09        LOOP1     DEX
00014 0158 26 FD               BNE     LOOP1
00015 015A 20 F2               BRA     UP
00016 015C FE 8000 DOWN        LDX     $8000      DELAY TIME
00017 015F B7 8002             STA A   $8002      OUT TO DAC
00018 0162 81 3F               CMP A   #$3F
00019 0164 27 F0               BEQ     UP1
00020 0166 4C        DOWN1     INC A
00021 0167 09        LOOP2     DEX
00022 0168 26 FD               BNE     LOOP2
00023 016A 20 F0               BRA     DOWN
00024                          END
```

Figure 10-82
Program to generate a triangle waveform.

Beginning at address $014E_{16}$, the index register is loaded with the level step delay time. Then the A accumulator is stored to the PIA. If accumulator A is zero, the program branches to address 0166_{16}. Since it equals $3F_{16}$, the A accumulator is decremented. Then, the index register is decremented until it equals 0000_{16}.

At the end of delay, the index register is again loaded, and the contents of the A accumulator are stored to the PIA. This cycle continues until the A accumulator equals 00_{16} (5-volt level). Then the program branches to address 0166_{16}.

Addresses 015C through $016B_{16}$ are similar to the first part of the program. They differ in that the A accumulator is incremented and compared to $0F_{16}$. Thus, the voltage level is cycled from 0 to 5 volts in the UP program section, and from 5 to 0 volts in the DOWN program section.

Procedure (continued)

17. Enter the program listed in Figure 10-83. Notice that this program begins at address 0000_{16}.

18. Execute the program. This program produces a sine waveform at a fixed frequency. If you only have a voltmeter, it should indicate approximately 1.7 volts AC. An oscilloscope will show a slightly distorted waveform. This is because of the resolution provided by the 6-bit DAC. An 8-bit DAC would produce a more symmetrical waveform.

Discussion

The program uses a "look-up" table (addresses 003B through $004C_{16}$) of constant values to generate a sine wave signal. The table was produced by deriving the sine of the angles between 0° and 90° in 5° increments, and then multiplying each value by a constant.

Because the sine wave must reside within a 0 to 5-volt "window," each 90° segment of the waveform can only be generated by 32 of the possible data bits. The first 90° of the sine wave starts at approximately 2.5 volts and steps to 0 volts. The second 90° steps from 0 volts up to 2.5 volts. The third 90° steps from 2.5 volts up to 5 volts. Finally, the fourth 90° steps from 5 volts down to 2.5 volts.

```
00001                                NAM     SINEWAVE REV.0.2
00002                                OPT     NOP
00003 0000 CE FF04                   LDX     #$FF04
00004 0003 FF 8002                   STX     $8002
00005 0006 CE 003B                   LDX     #BTABLE
00006 0009 C6 11     SIN1    LDA B   #$11
00007 000B A6 00     SIN1A   LDA A   X
00008 000D 01                        NOP
00009 000E B7 8002                   STA A   $8002
00010 0011 08                        INX
00011 0012 5A                        DEC B
00012 0013 26 F6                     BNE     SIN1A
00013 0015 C6 11     SIN2    LDA B   #$11
00014 0017 A6 00     SIN2A   LDA A   X
00015 0019 01                        NOP
00016 001A B7 8002                   STA A   $8002
00017 001D 09                        DEX
00018 001E 5A                        DEC B
00019 001F 26 F6                     BNE     SIN2A
00020 0021 C6 11     SIN3    LDA B   #$11
00021 0023 A6 00     SIN3A   LDA A   X
00022 0025 43                        COM A
00023 0026 B7 8002                   STA A   $8002
00024 0029 08                        INX
00025 002A 5A                        DEC B
00026 002B 26 F6                     BNE     SIN3A
00027 002D C6 11     SIN4    LDA B   #$11
00028 002F A6 00     SIN4A   LDA A   X
00029 0031 43                        COM A
00030 0032 B7 8002                   STA A   $8002
00031 0035 09                        DEX
00032 0036 5A                        DEC B
00033 0037 26 F6                     BNE     SIN4A
00034 0039 20 CE                     BRA     SIN1
00035 003B 20        BTABLE  FCB     $20,$22,$25,$27,$2A,$2D,$2F
      003C 22
      003D 25
      003E 27
      003F 2A
      0040 2D
      0041 2F
00036 0042 31                        FCB     $31,$33,$35,$37,$39,$3A,$3B
      0043 33
      0044 35
      0045 37
      0046 39
      0047 3A
      0048 3B
00037 0049 3C                        FCB     $3C,$3D,$3E,$3E
      004A 3D
      004B 3E
      004C 3E
00038                                END
```

Figure 10-83
Program to generate a sine waveform.

With the program using only 32 data levels for each waveform segment, the first five data bits in each byte determine the digital voltage level, while the sixth bit (D_5) determines whether the lower two waveform segments or the upper two waveform segments are being generated. Refer to the program in Figure 10-83. Notice that the table of values begins at 20_{16}, which approximately equals 2.5 volts. The values increase (voltage decrease) from there. When the upper half of the sine waveform is produced, the table values are complemented to generate the voltages between 2.5 and 5 volts.

In the program, the first two steps initialize the PIA. Then, the index register is loaded with the address of the first value in the look-up table. Accumulator B is loaded with 11_{16}, which will be decremented to show when all of the table values have been used.

The next six instructions form a "fetch and display" loop that generates the first waveform segment. Accumulator A is loaded with the first value (20_{16}). The NOP occupies time to make the loop time identical to the time used when the third and fourth segments are generated. Accumulator A is stored to the PIA. The index register is incremented, to point to the next value in the table, and the B accumulator is decremented. Since the register is not zero, the program branches back to address $000B_{16}$. This continues until the B accumulator equals 00_{16}.

Then, the B accumulator is loaded with 11_{16}. Remember that the index register contains the address of the last value in the table ($004C_{16}$). The next six instructions form another fetch and display loop to generate the second waveform segment. The only instruction that differs from the first loop is "decrement the index register" rather than increment.

After the loop is completed, the B accumulator is loaded with 11_{16}. The index register now contains the address ($003B_{16}$) of the value at the top of the table. The next six instructions form the third fetch and display loop. This is identical to the first program loop; except, the NOP instruction is replaced with a "complement the A accumulator" instruction. Thus, the table values can now be used to generate the upper half of the sine waveform. Program loop four also operates in this manner.

Procedure (continued)

19. Switch the Trainer power off. Then remove the eight wires interconnecting the binary data switches and the PIA.

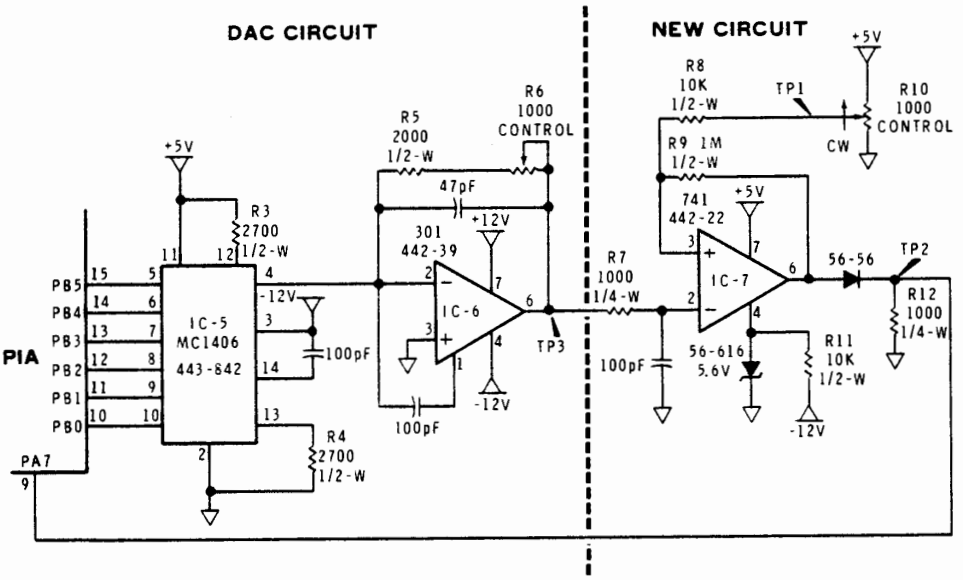


Figure 10-84
New circuit adds a voltage comparator to produce an analog-to-digital converter.

20. Refer to Figure 10-84 and add the new circuit shown to the DAC circuit. The output of the new circuit is connected to pin 9 (PA7) of the PIA. If you changed the value of R5 (in the DAC circuit) to 1000 ohms, in a previous section of this experiment, remove the 1000 ohm resistor and reinstall the 2000 ohm, 1/2-watt resistor in its place.

21. Switch the Trainer power on. Then, enter the program listed in Figure 10-85. Notice that the program begins at address 0170_{16}.

22. Turn control R10 fully clockwise, then execute the program beginning at address 0170_{16}.

23. Connect your voltmeter to TP1 (wiper of control R10) and circuit ground. Then, adjust control R6 for a Trainer display equal in value to the voltage at TP1. If you cannot adjust the display value down to the indicated voltage level, place a 1000 ohm, 1/4-watt resistor in series with resistor R5. Make sure you switch the Trainer power off before you add the resistor.

24. The circuit you have constructed acts like a digital voltmeter and will measure the voltage at the wiper of control R10 (TP1). Turn control R10 and compare the voltage indicated by your voltmeter with the voltage indicated by the Trainer display.

```
00001                              NAM     A-TO-D   REV.0.1
00002                              OPT     NOP
00003 0170                         ORG     $0170
00004       FCBC     REDIS  EQU     $FCBC
00005       FE20     OUTBYT EQU     $FE20
00006                *INITIALIZE PIA
00007 0170 CE 0004                 LDX     #$0004
00008 0173 FF 8000                 STX     $8000    A SIDE IN
00009 0176 CE FF04                 LDX     #$FF04
00010 0179 FF 8002                 STX     $8002    B SIDE OUT
00011                *FIND EQUAL POINT
00012 017C C6 FF   NEWIN  LDA B    #$FF     FF=LOW VOLTAGE
00013 017E 4F                      CLR A
00014 017F F7 8002 NXTSTP STA B    $8002
00015 0182 CE 0055                 LDX     #$0055   *
00016 0185 09      WAIT   DEX                      *HARDWARE SETTLE WAIT
00017 0186 26 FD                   BNE     WAIT     *
00018 0188 7D 8000                 TST     $8000    CHECK COMPARITOR
00019 018B 2A 06                   BPL     FOUND
00020 018D 5A                      DEC B
00021 018E 8B 01                   ADD A   #01
00022 0190 19                      DAA
00023 0191 20 EC                   BRA     NXTSTP
00024                *OUTPUT RESULTS
00025 0193 BD FCBC FOUND  JSR      REDIS    SET DISPLAY LOCATION
00026 0196 BD FE20                 JSR     OUTBYT   OUTPUT BYTE
00027 0199 86 01                   LDA A   #01      *  TURN ON
00028 019B B7 C16F                 STA A   $C16F
00029 019E 20 DC                   BRA     NEWIN    *  DECIMAL POINT
00030                              END
```

Figure 10-85
Program to convert an analog signal to a digital value.

Discussion

In this analog-to-digital conversion circuit, the MPU supplies a known voltage to a voltage comparator, and looks for a match with the unknown voltage at the comparator. Because the program operates in digit increments, each voltage level step will equal 0.1 volts after the circuit has been calibrated. Thus, the unknown voltage can be resolved to one-tenth of a volt.

At the beginning of the program, the PIA is initialized (A side in, B side out); the B accumulator is loaded with FF (to start comparison at 0 volts); and the A accumulator is cleared (used to store the voltage count, for the display). The B accumulator is stored to the PIA for conversion to a

voltage level. The next three instructions use the index register to supply a short time delay. This is needed since the analog circuit cannot operate as quickly as the MPU.

After the short delay, the comparator output to the A side of the PIA is checked for a voltage match. If there is no match, the B accumulator is decremented, increasing the voltage level, and the A accumulator is incremented, by adding 01_{16} to the contents, to indicate the voltage increase. Since this circuit is emulating a digital voltmeter, the displayed voltage must be in decimal (base 10). Therefore, the next instruction performs a decimal adjust on the A accumulator. This converts the binary addition to a BCD format.

The program then branches back to address $017F_{16}$ and repeats until the comparator test indicates that the voltage generated by the MPU equals the unknown voltage. You can observe the conversion routine by connecting the Y1 input of a dual-trace oscilloscope to TP2 and circuit ground, and the Y2 input to TP3 and circuit ground. Figure 10-83 shows the location of the test points, while Figure 10-86 shows the display of the 0.5-volt conversion.

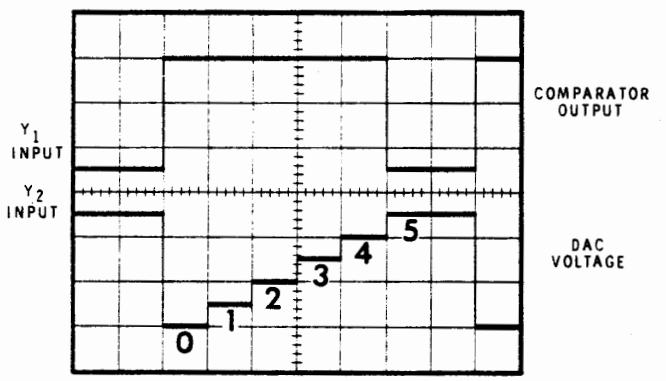


Figure 10-86
Voltage comparator timing in relation to the DAC voltage signal.

As soon as the comparator senses that the known and unknown voltages are equal, it switches low to tell the MPU a match has occurred. This causes the program to branch to the display output routine. REDIS stores the location of display H. OUTBYT writes the contents of the A accumulator to displays H and I. Next, the contents of the A accumulator is changed to 01_{16}. This is stored to address $C16F_{16}$, which lights the decimal point in display H.

Procedure (continued)

25. Switch the Trainer power off. Then remove the wires and components from the two large connector blocks. Caution: The PIA is an MOS device. When you remove it from the Trainer, press it onto its conductive foam pad. This will reduce the possibility of damage from static electricity.

26. Return to the Unit Activity Guide of Unit 8.

INDEX