8-Bit Microprocessor Interfacing and Applications

EB-6820-40 WORKBOOK 595-4060-04

Core Technology

HEATHKIT™

EDUCATIONAL SYSTEMS

Prepare to succeed.™

EXPERIMENT 1

ADDRESS DECODING

PURPOSE

1. *To demonstrate the difference between full and partial address decoding.*
2. *To show how an address decoding chart is assembled.*
3. *To demonstrate how an address can be decoded using various types of logic circuits.*

Materials Required

1 ETW-3800 Microprocessor Trainer

2 74LS27 integrated circuit (#443-800)

1 74LS30 integrated circuit (#443-732)

1 74LS42 integrated circuit (#443-807)

Hookup wire

Introduction

Many different combinational logic circuits can be used to decode binary bit patterns. In this experiment, you will observe both the use of SSI and MSI logic devices to decode addresses generated by the MPU.

Procedure

1. Turn the Trainer power off and construct the circuit shown in Figure E-1. Notice that the output of one-third of the 2nd 74LS27 (pin 12) is connected to the logic probe input connector on the Trainer. NOTE: Make sure you hook-up power and ground to each IC—the pin numbers are shown in the schematic. Future experiments will assume that you remember to do this.

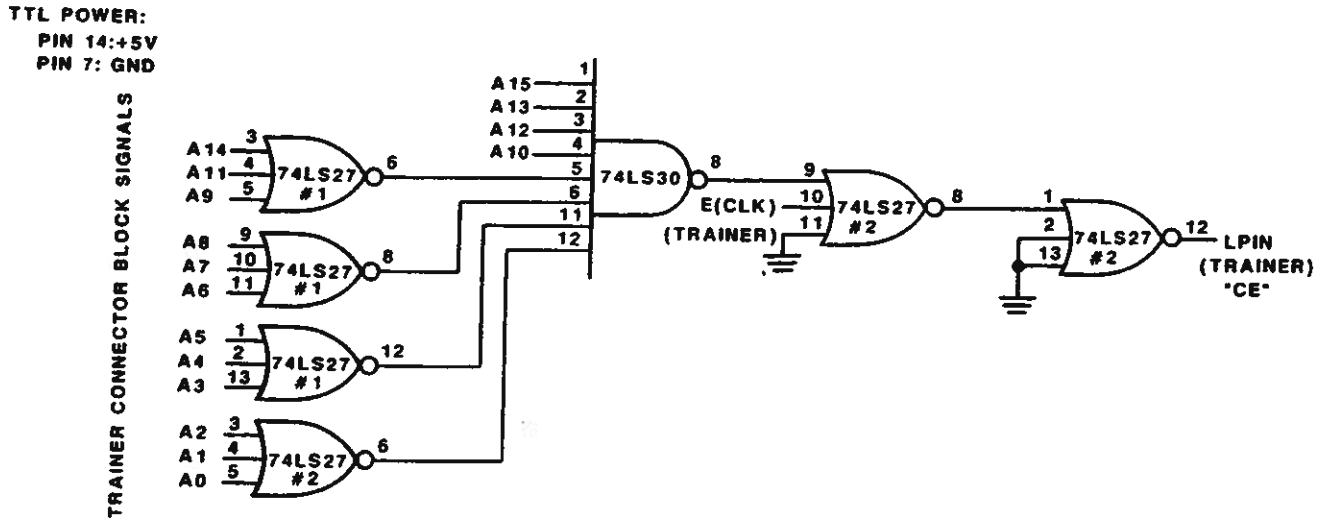

Figure E-1
An address decoder to completely decode address B400.

2. Turn the Trainer power on and verify that the logic probe is indicating a logic 1 output from the circuit. This is indicated by the red light in the lower left-hand corner of the Trainer and a high pitch tone emitting from the Trainer speaker. If you do not get a logic 1 indication, turn the Trainer power off and check your circuit.

3. Examine memory location $B400_{16}$. The logic probe should momentarily indicate a logic 0 via the green light and a lower pitch sound from the speaker. Do it again if you missed the action!

4. Press RESET and examine memory address $B3FF_{16}$. Was there any change in the logic output of the circuit?

5. Press RESET and examine memory address $B401_{16}$. Was there any change in the logic output of the circuit this time?

Discussion

The circuit in Figure E-1 completely decodes address $B400_{16}$. In other words, this circuit will only respond to this address. A decoding chart for the circuit is provided in Figure E-2. Comparing this chart to the schematic diagram in Figure E-1, you will find that the logic 0 address lines are being decoded by the NOR gates. Remember that the only time a NOR gate will generate a logic 1 is when all of its inputs are 0. Thus, when address $B400_{16}$ appears on the address bus each NOR gate generates a logic 1, since all of the NOR gate inputs are 0.

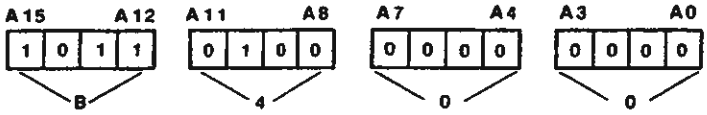

Figure E-2
An address decoding chart for the circuit in Figure E-1

Now, recall that the only time a NAND gate produces a logic 0 is when all of its inputs are 1. From Figure E-1, you see that the NAND gate decodes address lines *A15*, *A13*, *A12*, and *A10* as well as the outputs from the NOR gates. From the decoding chart you see that these four address lines are high (logic 1) when address $B400_{16}$ appears on the address bus. In addition, each NOR gate produces a logic 1 for this address. As a result, the output of the NAND gate goes low (logic 0) for address $B400_{16}$. Although at this point (the output of the NAND) the address bus is actually decoded, the generation of the final chip enable pulse requires the decoded address signal to be gated together with the ECLK (timer) signal of the 68HC11. This is done with the remaining two NOR gates. Notice that the last NOR gate is used simply as an inverter.* While many decoding circuits are possible, the circuit in Figure E-1 does the job of decoding the address B400 and providing a timed "chip-enable" or "CE" pulse, in this case to the logic probe, quite well.

In step 3, you examined address $B400_{16}$ and observed a logic 0 output for this address. The examine operation causes the MPU to place address $B400_{16}$ on the address bus. The address decoder recognizes this address and generates a logic 0 output. In steps 4 and 5 you examined addresses $B3FF_{16}$ and $B401_{16}$, respectively. The decoder did not respond to these addresses, since it fully decodes address $B400_{16}$. In other words, the *only* address the decoder will respond to is address $B400_{16}$ Try examining any other address and you will not observe any response from the decoder.

*NOTE: While it is possible to construct a NOR-inverter by simply tieing all inputs together, good design practice is to, instead, tie all but the used input to ground. This is especially true with timer and clock signals, where race conditions can cause unstable operation.

Procedure (continued)

6. Turn the Trainer power off and remove the wire from pin 6 of the 74LS27(#2) to pin 12 of the 74LS30. Tie pin 12 of the 74LS30 to +5V. This eliminates the contribution of address lines A0, A1, and A2 to the decoder.

7. Turn the Trainer power on and examine memory address $B400_{16}$. Again the logic probe will indicate a logic 0 output from the decoder circuit when address $B400_{16}$ appears on the bus.

8. Examine addresses $B401_{16}$ through $B407_{16}$ and the logic probe should indicate that the decoder is responding to any addresses within this range. (Why?)

9. Press RESET and examine any address outside of the $B400_{16}$ - $B407_{16}$ range and the decoder will not respond. (Why?)

Discussion

By removing address lines A0, A1, and A2, you are only decoding part of the address bus as can be seen from the address decoding chart in Figure E-3. The chart shows that address lines *A0*, *A1*, and *A2* are not being decoded, and therefore are indicated as "don't cares" on the chart. Thus, the lowest address decoded is $B400_{16}$. when these three address lines are 000. The highest address decoded is $B407_{16}$ when these three address lines are 111. Of course, any address between these two extremes is also decoded.

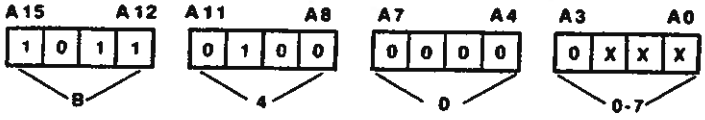

Figure E-3
An address decoding chart for the circuit in Figure E-1 with 74LS27 #2 removed.

Procedure (continued)

10. Turn the Trainer power off and construct the circuit shown in Figure E-4. Notice that one of the 3-input NOR gates on the 2nd 74LS27 is eliminated and address lines A0, A1, and A2 are left unconnected. The input to pin 12 of the 74LS30 is still tied to +5V.

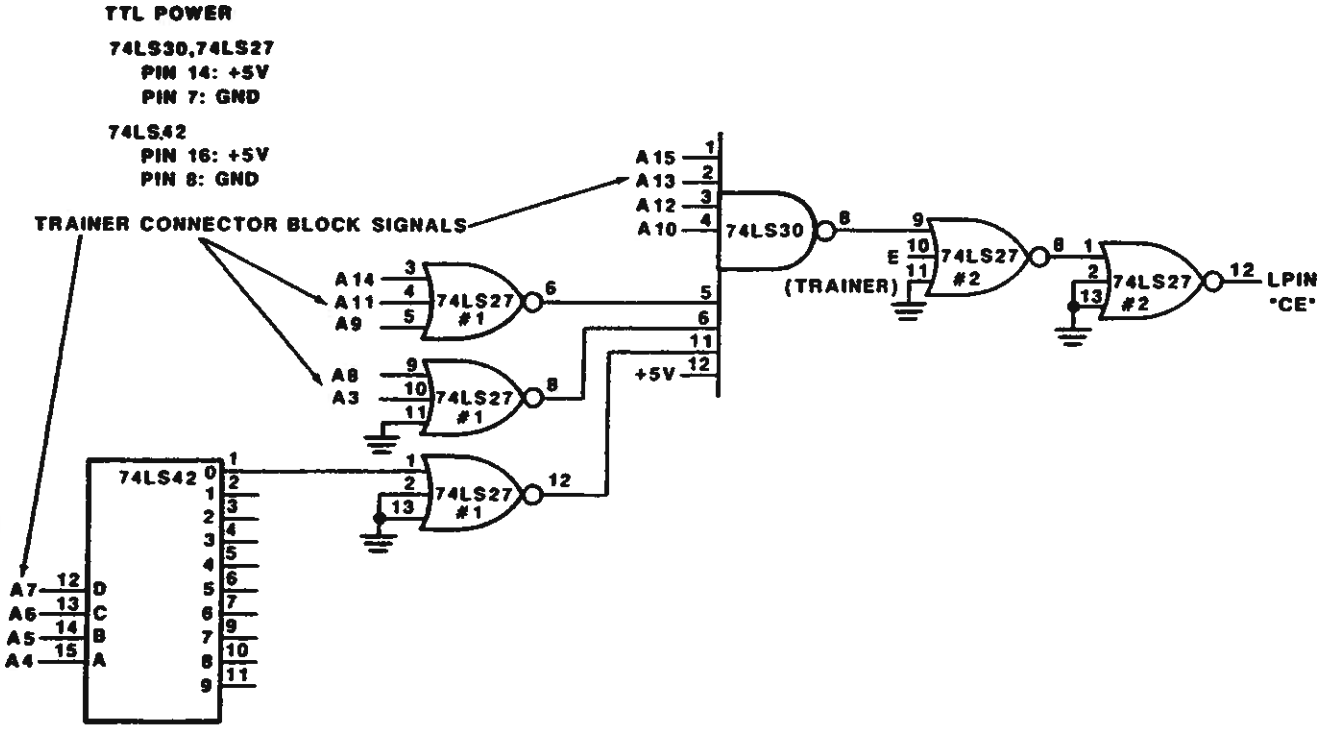

Figure E-4
An address decoder which employs a 74LS42 1-of-10 decoder IC.

DEC NO.	BCD INPUT D	C	B	A	OUTPUT LINES 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 E-5
Logic truth table for the 74LS42 1-of-10 decoder IC.

11. Using the schematic in Figure E-4 and the truth table for the 74LS42 shown in Figure E-5, fill in the address decoding chart in Figure E-6.

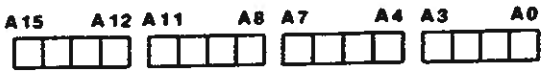

Figure E-6
A blank decoding chart.

12. Is the address bus fully or partially decoded? (Why?)

13. Now that you have determined the address range being decoded, examine all addresses within this range and verify that the decoder responds. Also examine addresses outside of the decoded range and verify that the decoder does not respond.

Discussion

The decoding chart for this circuit is shown in Figure E-7. Here, you see that the range of addresses being decoded is $B400_{16}$ through $B407_{16}$, since the lower three address lines are not being decoded.

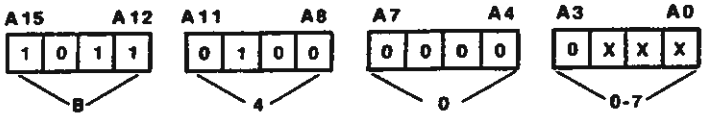

Figure E-7
An address decoding chart for the circuit in Figure E-4.

The 74LS42 IC decodes address lines *A7* through *A4*. Notice from the circuit that output line 0 is employed as the 74LS42 output line. From the truth table in Figure E-5 you see that this output line is a logic 0 when the decoder inputs (*A7* - *A4*) are 0000. This logic 0 output is inverted by the NOR gate and applied as a logic 1 to one of the input lines of the 75LS30 NAND gate. As a result, the circuit responds to address $B400_{16}$ - $B407_{16}$.

Now, suppose you were to reconnect the 74LS42 output from line 0 to line 1? Will this change the address range being decoded? Of course it will, since output line 1 of the 74LS42 only responds to a 0001 input. When this happens, the decoded address range becomes $B410_{16}$ through $B417_{16}$, right? What range would you get if you reconnected the 74LS42 output to line 2? You're right if you thought $B420_{16}$ through $B427_{16}$. Get the idea? In other words, the 74LS42 determines the second least significant hex digit within the decoded address range.

Procedure (continued)

14. Reconnect the 74LS42 output from output line 0 (pin 1) to output line 1 (pin 2).

15. Verify that the decoded address range is now $B410_{16}$ - $B417_{16}$ and *not* $B400_{16}$ - $B407_{16}$

16. Reconnect the 74LS42 output from output line 1 (pin 2) to output line 2 (pin 3).

17. Verify that the decoded range is now $B420_{16}$ - $B427_{16}$ and *not* any of the previous ranges.

How many different address ranges could be decoded with this circuit? What are they?

EXPERIMENT 2

DATA INPUT

PURPOSE

1. *To show how to construct a circuit for writing data to the microprocessor.*
2. *To demonstrate various methods for programming the microprocessor to accept externally applied data.*
3. *To demonstrate a software routine for debouncing a switch.*
4. *To show how to select a debounce routine to fit a specific system.*

Materials Required

1 ETW-3800 Microprocessor Trainer

4 SPST pushbutton switches (#64-910)

1 Black pushbutton switch cover-cap (#462-1144)

1 White pushbutton switch cover-cap (#462-1145)

1 Red pushbutton switch cover-cap (#462-1146)

1 Blue pushbutton switch cover-cap (#462-1147)

1 74LS125 integrated circuit (#443-811)

2 74LS30 integrated circuits (#443-732)

1 74LS27 integrated circuit (#443-800)

Hookup wire

Introduction

In this experiment, you will learn how to input data. While many devices can be used to transfer data to a microprocessor (keyboard, 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.

Procedure

1. In the first part of this experiment, you will interface four binary data switches to the data bus of the MPU. Make sure the Trainer power is switched off. Then construct the circuit shown in Figure E-8. Locate the three ICs next to each other at the extreme left-hand side of the connector block.

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

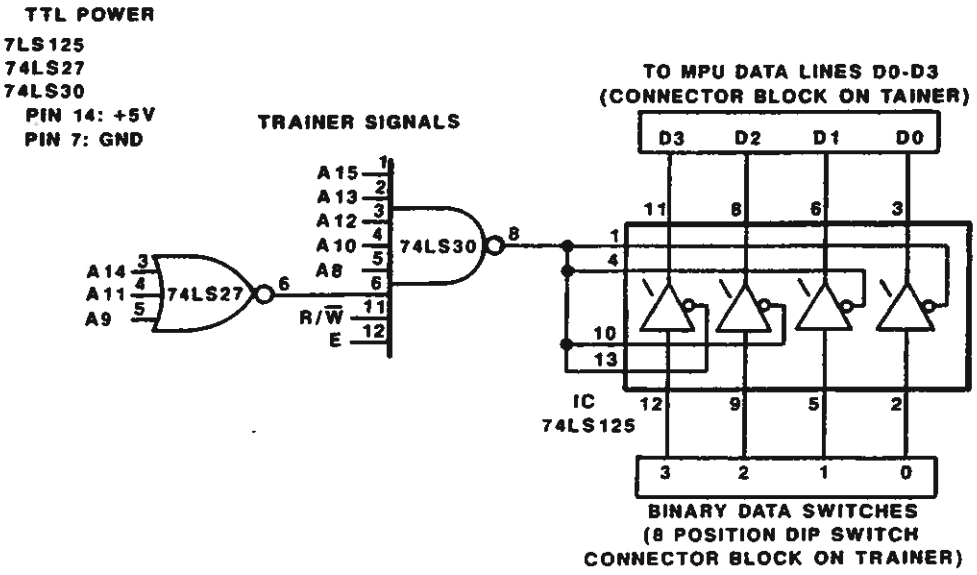

Figure E-8
Circuit diagram for the first part of the input experiment.

3. Switch Trainer power on and enter the program listed in Figure E-9. Then execute the program beginning at address 0000_{16}.

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS	COMMENTS
0000	F6	LDB$$	Read binary switch data at address $B500_{16}$
0001	B5	B5	
0002	00	00	
0003	F7	STB$$	Store switch data to address 0100_{16}
0004	01	01	
0005	00	00	
0006	CF	STOP	Stop

Figure E-9
Program for inputting data from binary data switches.

4. Examine address 0100_{16}. What is the contents? ___ ___16.

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

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

7. Position binary data switches 0 through 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 E-10.

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS	COMMENTS
0000	F6	LDB$$	Read switch data at address $B500_{16}$
0001	B5	B5	
0002	00	00	
0003	BD	JSR$$	Store value to the display
0004	C0	C0	
0005	15	15	
0006	C6	LDB#	Store a carriage return to the display
0007	0D	0D	
0008	BD	JSR$$	
0009	C0	C0	
000A	06	06	
000B	20	BRA	Do it again
000C	F3	F3	

Figure E-10
Program to read and display binary switch data.

10. Position the data switches to their logic 0 position. Execute the program beginning at address 0000_{16}. You should observe the hex value F0 being displayed in the upper left-hand corner of the Trainer display.

    Now, move binary data switch 0 from its logic 0 position up to its logic 1 position. The display should now reflect the change by displaying the hex value F1.

    Change the lower four binary data switches to any arbitrary logic pattern and the display should reflect the hex equivalent of the binary setting. Of course, only the least significant hex digit is affected since only the lower four MPU data lines are being employed.

Discussion

Refer again to the circuit in Figure E-8. It operates like read only memory, with its data being influenced by external sources, (the "outside world").

The circuit is partially decoded as shown in Figure E-11. When any of the specified addresses are selected, the 74LS125 three state buffer is enabled via the address decoder. This allows the data switch logic to be coupled to the Trainer data bus.

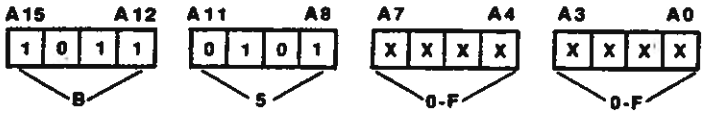

Figure E-11
Decoding chart for the first input circuit.

Notice that the $R/\overline{W}$ and ECLK signals are also used in the decoding. The $R/\overline{W}$ is being decoded so that the three state buffers are only enabled during a read operation. The ECLK is included to ensure consistent data transfer timing.

Both programs in this experiment have used address $B500_{16}$ as an input port. The first retrieves data from $B500_{16}$ and stores it at 0100_{16}.

The second program also retrieves data from $B500_{16}$. But this time, it jumps to a display subroutine at address $C015_{16}$. The display subroutine displays the hex contents of accumulator B, which contains the binary switch data. The program continuously branches back and retrieves switch data immediately after displaying the previous data. Thus, when you changed the position of data switches, the display followed the logic value of the changing switch positions.

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

Procedure (Continued)

NOTE: Before performing the next step, locate the four SPST pushbutton switches (64-910) and the four colored (black, white, red, blue) cover-caps. Prepare the switches for use by installing the caps on the switches. Do this by placing each cap on a switch and pressing it into place.

11. Refer to Figure E-12 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 breadboard. Locate the pushbuttons close together, just to the right of the ICs.

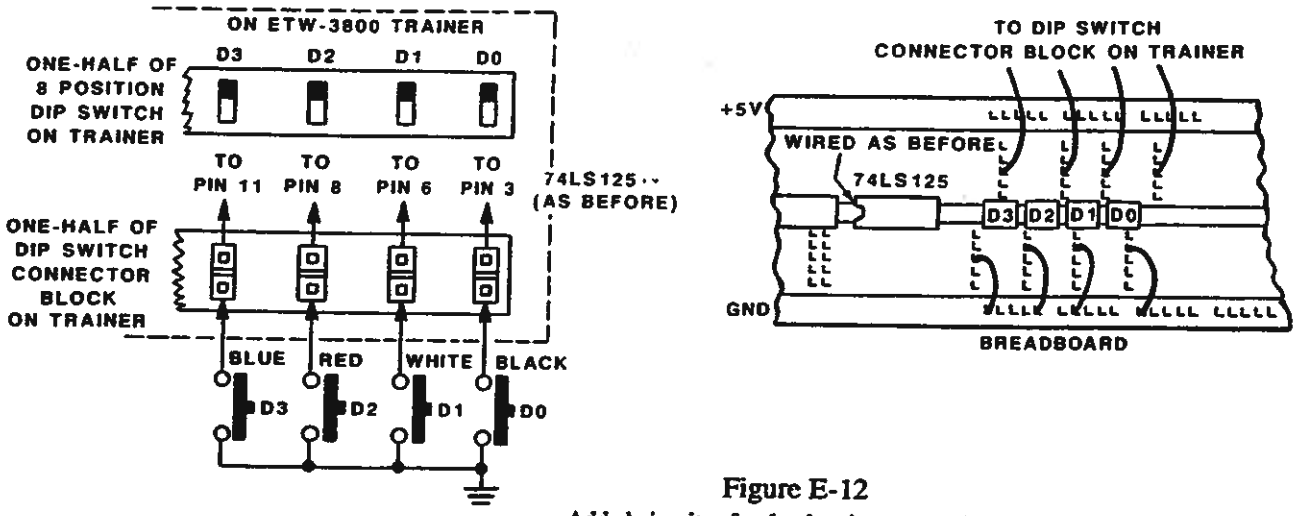

Figure E-12
Added circuitry for the data input experiment.

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

13. Execute the program beginning at address 0000_{16}. The display should display the hex value FF_{16}.

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

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

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

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

Discussion

The four pushbuttons that you added in Figure E-12 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 simply reads the pushbutton data at address $B500_{16}$ and repeatedly displays the hex equivalent of the data. The four pushbutton switches are connected in a pull up configuration. Thus, when a switch is depressed a ground potential (logic 0) is placed on the corresponding data bus line. Of course, the upper four data bus lines are not being affected by the circuit and are seen as logic 1's.

Try depressing any combination of the pushbutton switches while the program is running and observe the display. Write the binary equivalent of the hex display value and you should find logic 0's in the bit positions corresponding to the depressed pushbuttons.

Procedure (Continued)

18. Switch the Trainer power off. Then refer to Figure E-13 and add the circuit shown to the circuit already wired to the Trainer. There should be enough room near the right 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. The NOR gate is obtained from the 74LS27 IC already in the circuit. Make sure the NOR gate output (pin 12) is connected to the $\overline{\text{XIRQ}}$ interrupt input on the Trainer.

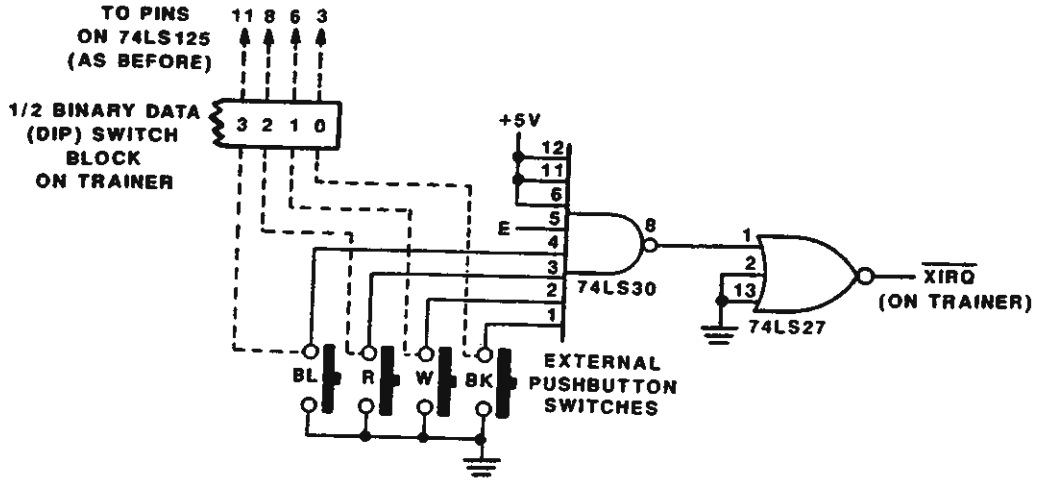

Figure E-13
Interrupt circuitry for data input experiment circuit.

19. Switch the Trainer power on. Then refer to Figure E-14 and enter the program listed beginning at address 0000_{16}.

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS	COMMENTS
0000	CE	LDX#	Store $\overline{\text{XIRQ}}$ interrupt vector to address $692E_{16}$
0001	00	00	
0002	0E	0E	
0003	FF	STX$$	
0004	69	69	
0005	2E	2E	
0006	0E	CLI	Clear I-flag
0007	CE	LDX#	
0008	01	01	Point to storage and loop until interrupt received.
0009	00	00	
000A	01	NOP	
000B	01	NOP	
000C	20	BRA	
000D	F9	F9	
000E	B6	LDA$$	Get data and compare to previous data.
000F	B5	B5	
0010	00	00	
0011	B1	CMPA$$	
0012	00	00	
0013	32	32	
0014	27	BEQ	If different, store in memory location 0032_{16}. If not, branch to debounce routine at address $001D_{16}$.
0015	07	07	
0016	B7	STA$$	
0017	00	00	
0018	32	32	
0019	7F	CLR$$	Reset counter and return from interrupt
001A	00	00	
001B	33	33	
001C	3B	RTI	
001D	C6	LDB#	Debounce routine causes switch to be read 64 times to eliminate contact bounce
001E	40	40	
001F	F1	CMPB$$	
0020	00	00	
0021	33	33	
0022	27	BEQ	
0023	04	04	
0024	7C	INC$$	
0025	00	00	
0026	33	33	
0027	3B	RTI	
0028	43	COMA	Complement switch logic and store.
0029	A7	STA X	
002A	00	00	
002B	7F	CLRA$$	
002C	00	00	
002D	33	33	
002E	08	INX	
002F	DF	STX$	
0030	08	08	
0031	3B	RTI	Return from interrupt

Figure E-14
Program to debounce the pushbutton switches.

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

21. Execute the program beginning at address 0000_{16}.

22. Strike each pushbutton sequentially in a 4, 3, 2, 1 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.

23. Examine address 0009_{16}. It should contain 04_{16}, which is the number of pushbutton contact closures made. Change the contents back to 00_{16}.

24. Examine addresses 0100 through 0103_{16}. They should contain 08, 04, 02, and 01_{16} respectively. Change the data in these four locations back to 00_{16}. Note (Figure E-12) that the pushbuttons are connected to data lines D_3, D_2, D_1 and D_0. Therefore, the switches will enter the binary values 8, 4, 2, and 1.

25. Execute the program. Then press each pushbutton twice in succession (4, 4, 3, 3, 2, 2, 1, 1). Address 0009_{16} now contains 08_{16}, representing eight pushbutton contact closures. Enter 00_{16} at address 0009_{16}.

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

27. Examine address $001E_{16}$. It should contain data value 40_{16}. Change the value to 00_{16}.

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

29. Examine address 0009_{16} and record the contents. $\_\_\_\_{16}$. This number should equal 04_{16}. However, it may be higher.

30. Record the data in the following addresses. You need only examine the number of addresses that correspond to the value recorded in step 29.

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

Discussion

The additional NAND and NOR gates provide an interface between the four external pushbuttons and the interrupt request line $\overline{\text{XIRQ}}$. 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 E-14 processes the interrupt and debounces the keys. The program is actually two programs in one. The first part serves as a "simulated" program that runs in an endless 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 E-15 is a flowchart for the interrupt program.

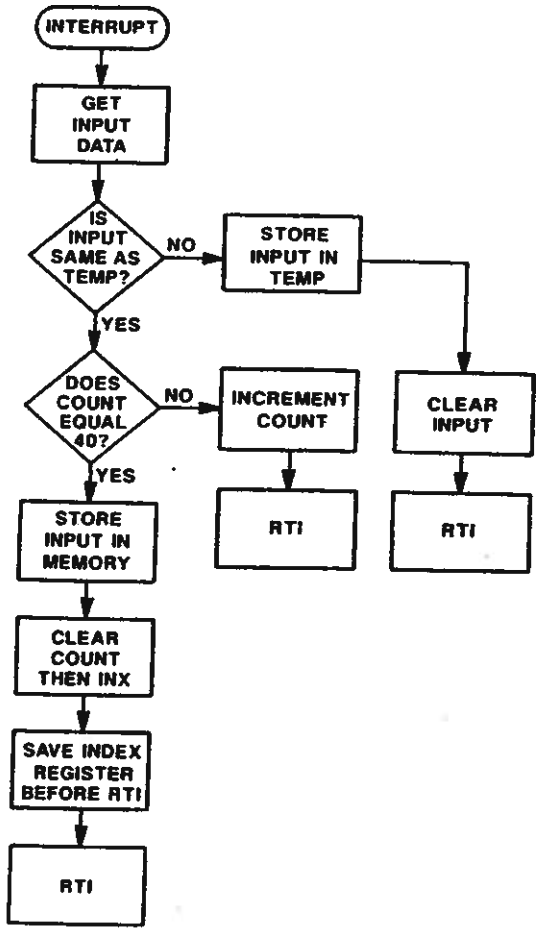

Figure E-15
Flowchart for interrupt routine in the debounce program.

When the MPU receives an interrupt request; it completes the instruction it is presently performing, stores 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 0032_{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 0033_{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 again. Whereupon, it stores it 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 $001E_{16}$. Data value 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 62 more times (63 total). During the 64th 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 $002B_{16}$.

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 0008_{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 0028_{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 $001E_{16}$). Then when you entered four pushbutton closures, you probably found more than four entries stored at address 0009_{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 as separate entry, as you probably experienced.

Again look at the data you recorded in step 30. 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, 64 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 0009_{16} and 0100_{16} through 0110_{16} before you execute the program.

Procedure (Continued)

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

EXPERIMENT 3

DATA OUTPUT

PURPOSE

1. *To demonstrate microprocessor interfacing to an external data display.*
2. *To show how a 7-segment display is connected.*
3. *To demonstrate the trade-offs between hardware and software display decoding.*
4. *To provide an opportunity to write a number of output programs.*

Materials Required

1 ETW-3800 Microprocessor Trainer

8 470 Ω, 1/4-watt, 5% resistors (#6-471-12)

1 TIL-312 7-segment LED (#411-831) or 5082-7731 7-segment LED (#411-875)

2 7475 integrated circuits (#443-13)

1 7447 integrated circuit (#443-36)

1 74LS30 integrated circuit (#443-732)

1 74LS27 integrated circuit (#443-800)

Hookup wire

Introduction

Until now, you have been using programs that moved data within the Trainer, with any results displayed by the "on-board" display or binary LEDs. 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.

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 E-16. Notice that +5 volts and ground are connected to pins 5 and 12 respectively for the 7475 ICs. The other ICs use pin 14 for +5 volts and pin 7 for ground.

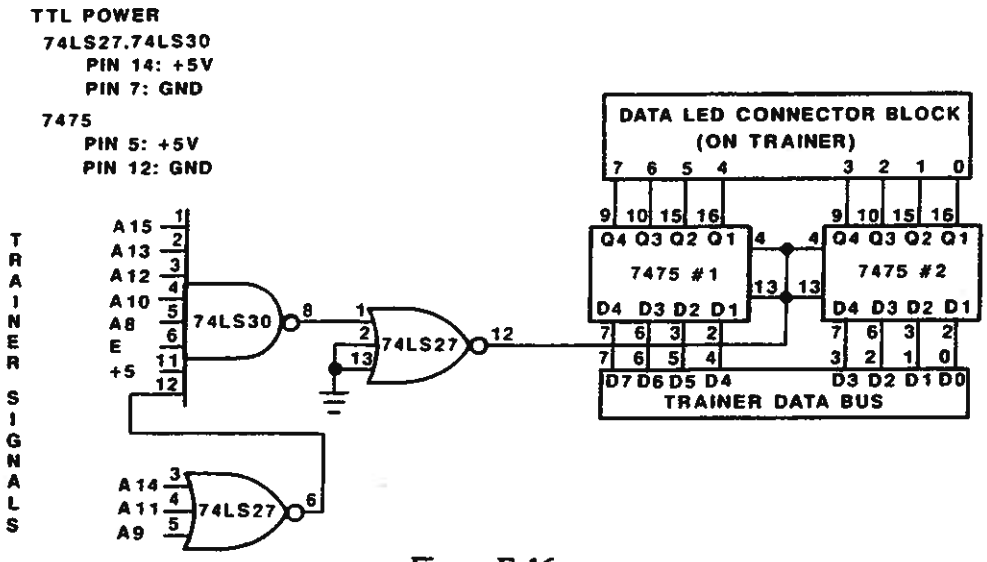

Figure E-16
Latching binary data for output.

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

3. Figure E-17 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.

Figure E-17
Decoding chart for the circuit in Figure E-16.

4. Enter the program listed in Figure E-18. Execute the program beginning at address 0000_{16}. The data LEDs indicate _ _ _ _ _ _ _ _$_2$. This is the binary equivalent of the data the program stored to address $B500_{16}$.

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS	COMMENTS
0000	86	LDA#	Load A with the value 55_{16}
0001	55	55	
0002	B7	STA$$	Store A to data LEDs
0003	B5	B5	
0004	00	00	
0005	CF	STOP	Stop

Figure E-18
Program to store data to the LEDs.

5. What hex value would be required to turn off all of the data LEDs? _ _$_{16}$. Verify your answer by placing this value in the program at address 0001_{16} and executing the program.

6. What hex value would be required to turn on all of the data LEDs _ _$_{16}$. Verify your answer by placing this value at address 0001_{16} and executing the program.

7. Change the data at address 0001_{16} a number of times and verify its value with the data LEDs. Each time you must re-execute the program.

8. Write and execute a program that will alternately turn all of the data LEDs 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 1 microsecond in the Trainer.

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

Discussion

Refer to Figures E-16 and E-17. Notice that a partial decoding scheme is used. A fully decoded circuit could have been used by adding more combinational logic.

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 LEDs monitor the stored information, you can see the data. Thus, the MPU is interfaced in a way that produces usable data.

Two bistable quad latch ICs 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 the LEDs, while storing FF_{16} will turn each LED on.

Right now, the data LEDs 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 E-19. It lists a program to flash the data LEDs. While this program may not match your program, it is one of many ways to accomplish the same objective.

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS	COMMENTS
0000	4F	CLRA	Clear A
0001	B7	STA$$	Store A to LEDs
0002	B5	B5	
0003	00	00	
0004	CE	LDX#	Delay
0005	55	55	
0006	00	00	
0007	09	DEX	
0008	26	BNE	
0009	FD	FD	
000A	43	COMA	Toggle A
000B	20	BRA	Do it Again
000C	F4	F4	

Figure E-19
Program to flash the data LEDs.

Alternatively, you may display a binary upcount on the eight LED indicators with the following program:

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS	COMMENTS
0000	4F	CLRA	Clear ACCA
0001	B7	STAA	Store ACCA
0002	B5	B5	to LEDs
0003	00	00	
0004	CE	LDX	Delay
0005	40	40	
0006	00	00	
0007	09	DEX	
0008	2E	BGT	
0009	FD	FD	
000A	4C	INCA	Upcount
000B	81	CMPA	
000C	FF	FF	
000D	24	BHS	
000E	F1	F1	
000F	20	BRA	Do it again
0010	F0	F0	

Procedure (Continued)

9. **Write a program to alternately store 1's and 0's to the display LEDs. 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 E-20 illustrates a method for accomplishing the task.

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS	COMMENTS
0000	4F	CLRA	Store FF_{16} to LEDs and delay
0001	CE	LDX#	
0002	55	55	
0003	00	00	
0004	B7	STA$$	
0005	B5	B5	
0006	00	00	
0007	43	COMA	
0008	09	DEX	
0009	26	BNE	
000A	FD	FD	
000B	CE	LDX#	Store 00_{16} to LEDs and delay
000C	FF	FF	
000D	00	00	
000E	B7	STA$$	
000F	B5	B5	
0010	00	00	
0011	43	COMA	
0012	09	DEX	
0013	26	BNE	
0014	FD	FD	
0015	20	BRA	Do it again
0016	EA	EA	

Figure E-20
Program to flash LEDs at nonregular intervals.

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

Procedure (Continued)

10. Switch the Trainer power off. Then, without disturbing the circuit wired to the Trainer, add the circuit shown in Figure E-21.

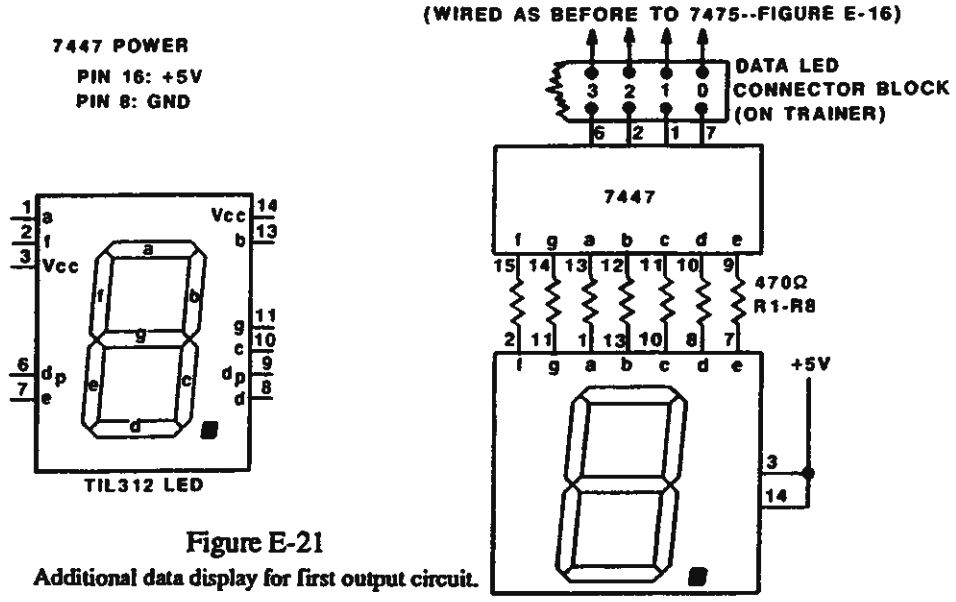

Figure E-21
Additional data display for first output circuit.

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 the program back in Figure E-18.

13. Execute the program. What is the bit pattern displayed by 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 9 and then continuously repeat. Include a delay loop so that each digit will remain on long enough to be identified. Execute the program.

Discussion

The circuit you just constructed contains a 4-line-to-7-segment decoder driver and a 7-segment, common anode display. The 7447 decoder driver 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 decimal digit.

The display circuit is a multiple LED array with common anodes. The anodes are tied to +5V and the decoder driver supplies the necessary grounds to light the selected LED segments.

If you had any questions concerning the program to increment the display. refer to Figure E-22. It contains a simple program to increment the display from 0 to 9 at a slow rate. Enter the program in Figure E-22 and watch the eight data LED's. They show the actual value stored in accumulator A.

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS	COMMENTS
0000	4F	CLRA	Start with 0
0001	81	CMPA#	
0002	0A	0A	
0003	27	BEQ	
0004	FB	FB	
0005	B7	STA$$	Store to display
0006	B5	B5	
0007	00	00	
0008	4C	INCA	Add one
0009	CE	LDX#	Delay
000A	FF	FF	
000B	FF	FF	
000C	09	DEX	
000D	26	BNE	
000E	FD	FD	
000F	20	BRA	Repeat
0010	F0	F0	

Figure E-22
Program to increment the 7-segment display from 0 to 9.

Next, you will see that a decoder driver is not necessary if you are willing to let the MPU do the decoding.

Procedure (Continued)

17. Switch Trainer power off and remove the decoder driver, and display package. Leave the 74LS27, 74LS30, and 7475 ICs wired to the Trainer per Figure E-16.

18. Refer to Figure E-23 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.

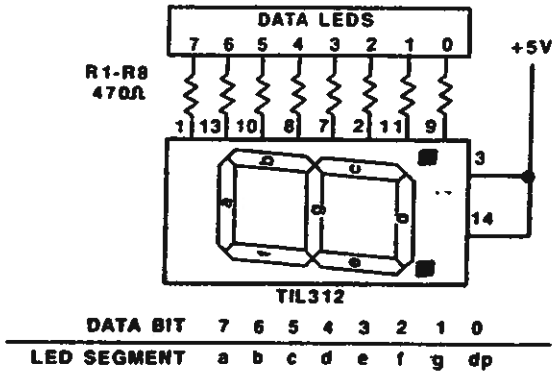

Figure E-23
Additional data display.

19. Re-examine 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 $B500_{16}$ will determine which display segment will light. Enter the program back in Figure E-18.

21. Change the data at address 0001_{16} to 4. Execute the program. 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 E-23 indicates the segments connected to the data bits. What bit pattern will produce the number 1 in the display? _ _ _ _ _ _ _ _$_2$.

23. Convert the bit pattern from step 22 to hex and enter it at address 0001_{16}. Although it is possible to display two 1s, the correct 1 is produced when segments b and c are lit.

24. Load and execute the program listed in Figure E-24.

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS	COMMENTS
0000	CE	LDX#	Load beginning table address
0001	00	00	
0002	1A	1A	
0003	A6	LDA X	Load display code
0004	00	00	
0005	B7	STA$$	Store display code to display
0006	B5	B5	
0007	00	00	
0008	86	LDA#	Delay
0009	FF	FF	
000A	C6	LDB#	
000B	FF	FF	
000C	5A	DECB	
000D	26	BNE	
000E	FD	FD	
000F	4A	DECA	
0010	26	BNE	
0011	F8	F8	
0012	08	INX	Get next display code
0013	8C	CPX#	
0014	00	00	
0015	2A	2A	
0016	27	BEQ	
0017	E8	E8	
0018	20	BRA	
0019	E9	E9	
001A	03	03	Display codes
001B	9F	9F	
001C	25	25	
001D	0D	0D	
001E	99	99	
001F	49	49	
0020	41	41	
0021	1F	1F	
0022	01	01	
0023	19	19	
0024	11	11	
0025	C0	C0	
0026	63	63	
0027	85	85	
0028	61	61	
0029	71	71	

Figure E-24
Program for incrementing 7-segment display from 0_{16} to F_{16}.

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 the numbers 0 through F_{16} are being stored.

Addresses $001A_{16}$ through 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 $C1_{16}$.