HEATHKIT
CONTINUING
EDUCATION

# Individual Learning Program

# MICROPROCESSORS

## *Unit 7*
## INTERFACING — PART 1

EE-3401

HEATH COMPANY
BENTON HARBOR, MICHIGAN 49022

# CONTENTS

# INTRODUCTION

As mentioned earlier, there are two things you can do with a microprocessor. You can program it and you can interface it with other circuits. By now you should be able to write simple programs without too much trouble. You will continue to learn new programming techniques as you progress through the course. However, in the remaining portions of the course, the emphasis will shift to interfacing the MPU with other circuits.

Units 7 and 8 will give you enough general information to understand the interfacing experiments providing that you have a general knowledge of digital electronics. That is, these units are written on the assumption that you have training equivalent to that provided in the Heathkit Continuing Education, Individual Learning Program in Digital Electronics.

# UNIT OBJECTIVES

When you have completed this unit, you should be able to:

1. Define 3-state logic and explain the need for it.
2. Explain the purpose of each of the control lines on the 6800 MPU.
3. Explain the timing relationships between the clock signals and the information on the address, data, and R/$\overline{W}$ lines.
4. Identify several different arrangements used in static RAMS.
5. Draw the logic diagram of a simple address decoder.
6. Explain the operation of a static RAM storage cell.
7. Show four different methods by which an MPU can drive 7-segment displays.

# UNIT ACTIVITY GUIDE

- ☐ Read Section on Interfacing Fundamentals.
- ☐ Complete Self-Test Review Questions 1 through 14.
- ☐ Play Cassette Tape Section "Semiconductor Memories."
- ☐ Read Section on Interfacing with Random Access Memory.
- ☐ Complete Self-Test Review Questions 15 through 28.
- ☐ Read Section on Interfacing with Displays.
- ☐ Complete Self-Test Review Questions 29 through 40.
- ☐ Perform Interfacing Experiments 1 through 4.
- ☐ Complete Unit Examination.
- ☐ Check Examination Answers.

# INTERFACING FUNDAMENTALS

Before going into specific interfacing examples, we must first discuss some fundamental concepts that will be used. First, we will discuss the concept of a bus and the need for 3-state logic. Then we will examine the various control and bus lines of the 6800 MPU. Finally, we will consider the various timing relationships involved in the execution of instructions.

## Buses

In computer jargon, a bus is generally defined as a group of conductors over which information is transferred from one place to another. In many cases, the information can originate from any one of several sources and can be transferred to any one of several destinations. Moreover, on some buses, information can be transferred in either of two directions. These are called bi-directional buses. Of course for a given bus, only one transfer of information can occur at a time.

Figure 7-1 shows the data bus arrangement in a typical microcomputer application. Generally in this type of system, all data transfers involve the MPU. Thus, data can be transferred in either direction between RAM and the MPU. However, other data transfers are one way only. Data can be transferred from ROM or the input buffer to the MPU. Also, data can be transferred from the MPU to the output latches.

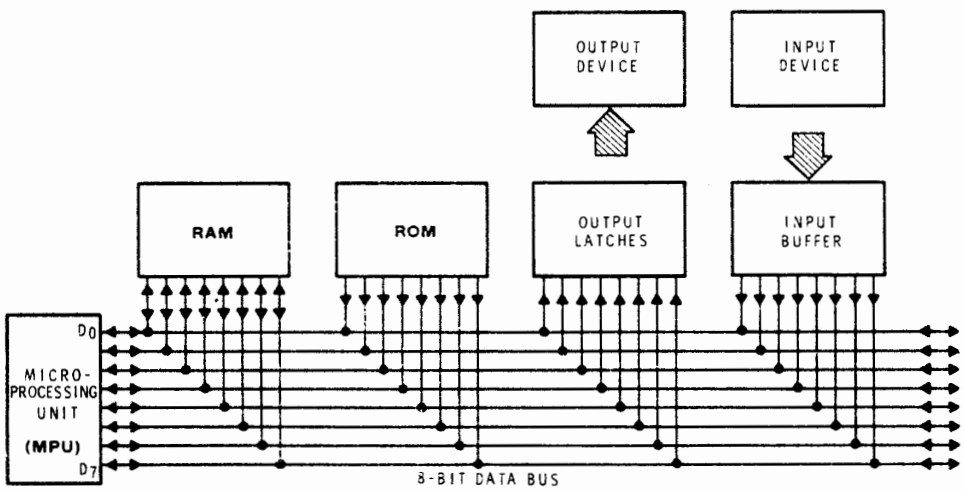



Figure 7-1
Typical Data Bus Arrangement.

In an arrangement like this, two problems arise. First, we must insure that only one data transfer is attempted at any given time. This is done by assigning each destination or source a different address. For example, the RAM, ROM, output latches, and input buffers all have one or more chip enable pins. The proper logic levels on these pins will select or activate the circuit. By assigning each circuit a different address, we insure that only one circuit at a time is enabled.

Figure 7-2 shows the addressing capability added to the block diagram. An address decoder is added for each circuit. The inputs to the address decoders come from the MPU via the address bus. The outputs go to the chip enable lines of the various circuits. Since only one address can appear on the address bus at any given instant, only one of the external circuits will be enabled at a time.

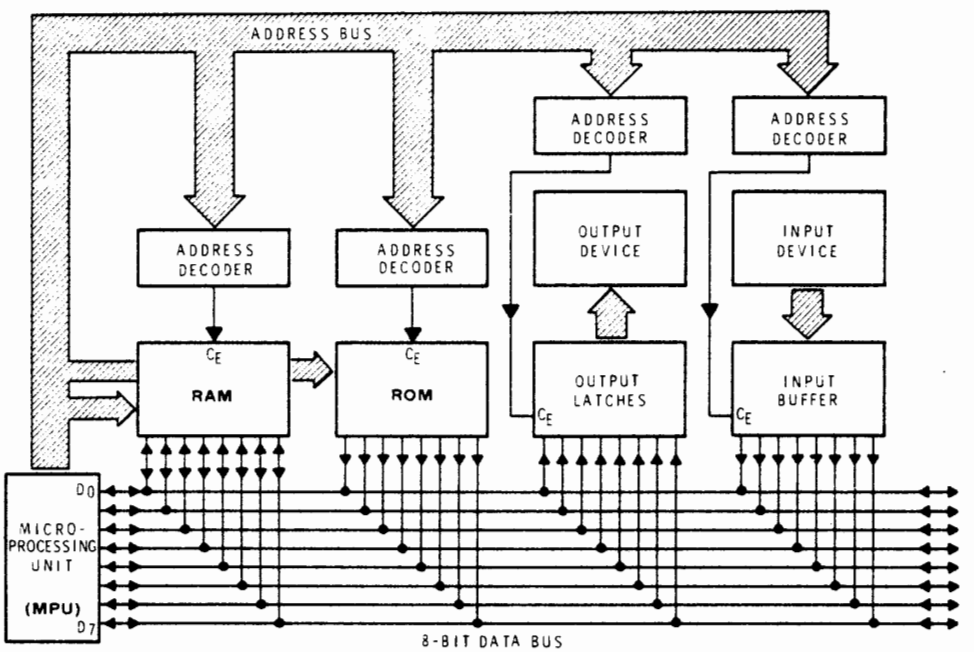



Figure 7-2
Adding the Address Capability.

The memories are assigned many addresses since each byte must have its own address. For example, if a $512_{10}$-byte RAM is used, it would probably be assigned addresses $0000_{16}$ through $01FF_{16}$. When any one of these addresses appear on the address bus, the RAM is selected via its chip enable line. Notice that a portion of the address bus connects directly to the RAM. This selects the individual byte within the RAM.

In the same way, the ROM is assigned a range of addresses. If a $1024_{10}$ byte ROM is used, it may be assigned addresses $FC00_{16}$ through $FFFF_{16}$. The ROM must be enabled whenever any of these addresses appear on the address bus. The output latch and input buffers are also assigned unique addresses. Thus, the MPU can communicate with any one of the external circuits simply by placing the proper address on the address bus.

The second problem is more fundamental. It arises because of the basic 2-state nature of digital logic circuits. Recall that the output of a standard logic gate will always be either logic 1 (high) or logic 0 (low). The problem is: Which state should the outputs of the circuits that are connected to the data bus assume when they are not selected? Regardless of which state they assume, they interfere with the output of the enabled circuit. For example, if the output of the disabled circuits assume a high state, they interfere with the low output of the enabled circuit. In other words, one circuit tries to pull the bus line high while the other is trying to force it low.

In the past, this problem has been overcome by using gates with open collector outputs. While open collector devices could be used to solve this problem in microprocessors, an entirely different approach is most often used. To understand how this problem is overcome, we must discuss *3-state logic*.

## 3-State Logic

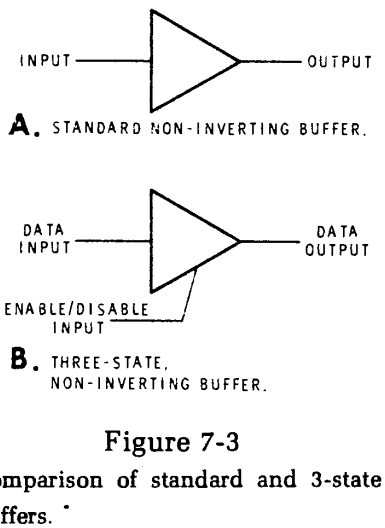



Figure 7-3
Comparison of standard and 3-state buffers.

As the name implies, 3-state logic devices have a unique third state in addition to the normal 1 and 0 output. Figure 7-3 compares a standard non-inverting buffer with a 3-state, non-inverting buffer.

Recall that a non-inverting buffer increases the current drive of the input signal without changing the logic levels in any way. Thus, the output may be able to drive ten times as many gates as the input. The standard buffer has one input and one output. The output always assumes the same logic level as the input. Because the input must be either 1 or 0, the output must be the same.

By contrast, the 3-state buffer has two inputs. In addition to the normal data input, the buffer has an enable/disable input. This input may be either logic 1 or logic 0 depending upon whether we wish to enable or disable the buffer. The buffer shown in Figure 7-3B is enabled by applying logic 1 to the enable/disable input.

When enabled, the 3-state buffer acts exactly like the standard buffer. The output will assume the same logic level as the data input.

The 3-state buffer is disabled by applying logic 0 to the enable/disable input. When disabled, the output assumes a very high impedance state that is neither logic 1 nor logic 0. While in this high impedance state, the output can be assumed to be disconnected from the rest of the circuit. That is, when the buffer is disabled, its output will not interfere with the circuits to which it is connected.

There are many different types of 3-state devices available. Figure 7-4 shows four different types of 3-state buffers. The buffer shown in Figure 7-4A is the same as that described above. It does not invert and is enabled by logic 1. Notice that the buffer shown in Figure 7-4B has a small circle at the enable/disable input. This means that the buffer is enabled by a logic 0 and disabled by a logic 1.

Figures 7-4C and D show inverting buffers. The first is enabled by a logic 1; the second is enabled by a logic 0.

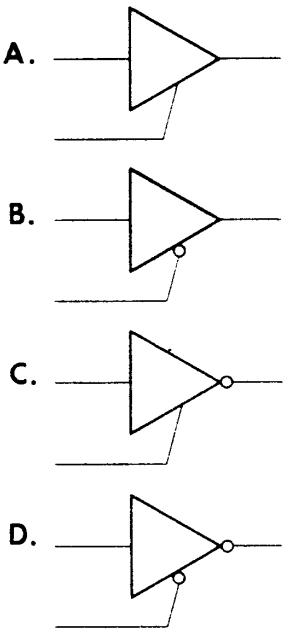



Figure 7-4
Four types of 3-state buffers.

Generally, four or more 3-state buffers are included in a single integrated circuit. Figure 7-5 shows the 74126 type TTL IC. It contains four 3-state buffers in a single 14-pin dual-in-line package.

Figure 7-6 shows eight 3-state buffers in a single 20-pin package. The four lower buffers are enabled by a logic 0 at pin 1 of the IC. The four upper buffers are enabled by a logic 1 at pin 19. The input buffer shown earlier in Figures 7-1 and 7-2 could use this type of IC. Buffers of this type are often called bus extenders, line drivers, line receivers, etc., depending on how they are used.

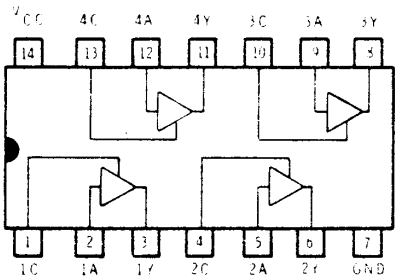



Figure 7-5
The 74126 IC contains four 3-state buffers in a single 14-pin package.

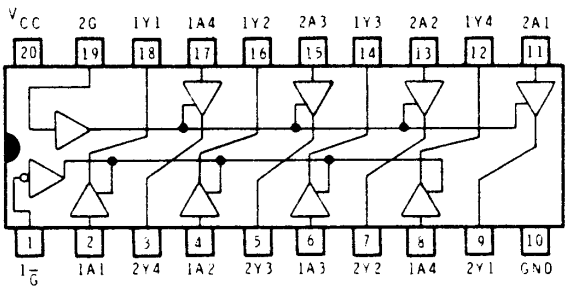



Figure 7-6
The 74LS241 contains eight 3-state non-inverting buffers.

While many different forms of 3-state buffers are available, many microprocessor support circuits do not require separate 3-state buffers. Most RAMS and ROMS have their own 3-state buffers built in. Thus, any time the RAM or ROM is not selected, it automatically goes to its third state. In this state, the outputs are said to be off, disconnected, disabled, floating, or in their high impedance state.

## The 6800 and 6808 MPU Interface Lines

Before you can interface any microprocessor to its support circuits or to the outside world, you must become familiar with its pin assignments, control lines, etc. Figure 7-7 shows the pin assignments of the 6800 and 6808 MPU. Notice that the MPU is in a single 40-pin dual-in-line package.

Most of the pin assignments, as shown in Figure 7-7, are common to both the 6800 and 6808 microprocessors. There are six pin assignments that are not common, pin 3 and 35 through 39. The functions in parenthesis are for the 6808. The functions on these same pins that have no parenthesis are for the 6800 microprocessor. In the following description of the MPU interface lines, those common to both the 6800 and 6808 will be described first. Then those that apply to the individual MPU will be discussed under a separate heading.

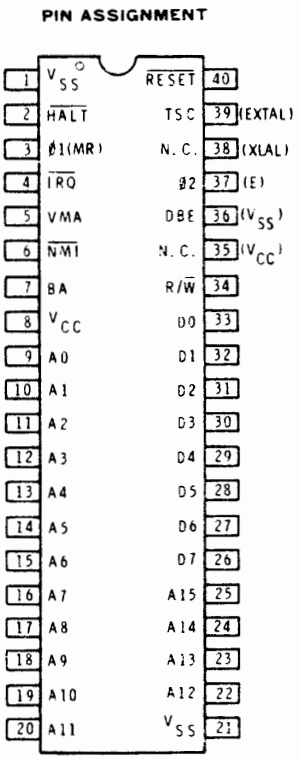



Figure 7-7
The 6800 and 6808 MPU pin assignments.

Notice that pins 9 through 20 and 22 through 25 make up the *16-bit address bus*. These pins connect to 16 three-state output drivers in the MPU. Each driver is capable of driving one standard TTL load. When disabled, the address lines act as open circuits. This capability is sometimes used to allow another device to gain control of the address bus. In this way, some external device can address memory. This is referred to as "direct memory access" (DMA).

Pins 26 through 33 are used for the *8-bit data bus*. This is a bi-directional bus that is used to transfer data to and from memory and the input-output circuitry.

You are already familiar with four of the control lines: the read/write line, the reset line, the non-maskable interrupt, and the interrupt request line. The *read/write (R/$\overline{W}$) line* tells the peripheral devices and memory whether the MPU is in the read or write mode. A read operation is indicated by a logic 1 on this line. In this mode, the MPU reads data from memory or from an input device. A write operation is indicated by a logic 0 on the R/$\overline{W}$ line. In this mode, the MPU sends data out to memory or an output device. Since it works hand-in-hand with the address bus, the R/$\overline{W}$ line has a 3-state capability.

The *reset line* (pin 40) was discussed in the previous unit. Recall that it is used to $\overline{\text{reset}}$ and start the MPU when power is initially applied or at anytime that we wish to initialize the system. When this line goes to logic level 1, the MPU starts the reset sequence. Recall that the reset vector is retrieved from addresses $FFFE_{16}$ and $FFFF_{16}$. This vector is loaded into the program counter so that the first instruction is fetched from that address. This capability is used to direct the MPU to the start of the monitor or control program.

The *non-maskable interrupt* ($\overline{\text{NMI}}$) was mentioned when interrupts were discussed. A high-to-low transition on pin 6 (the $\overline{\text{NMI}}$ line) will initiate the non-maskable interrupt sequence. Recall that this forces the MPU to pick up the $\overline{\text{NMI}}$ vector at addresses FFFC and FFFD. This vector is the address of the $\overline{\text{NMI}}$ service routine.

The *interrupt request* line ($\overline{\text{IRQ}}$) was also discussed earlier. While this is similar to the non-maskable interrupt, there are three fundamental differences. First, the $\overline{\text{IRQ}}$ signal is maskable; it will be ignored if the interrupt mask bit is set. Second, the $\overline{\text{IRQ}}$ sequence picks up the $\overline{\text{IRQ}}$ vector from addresses $FFF8_{16}$ and $FFF9_{16}$. Third, the $\overline{\text{IRQ}}$ line (pin 4) is level sensitive. That is, a logic 0 on this pin causes the interrupt sequence. Compare this to the $\overline{\text{NMI}}$ line which requires a logic 1-to-0 transition.

The *valid memory address (VMA)* line is an output. It indicates to peripheral devices that the address on the address bus is a valid one. This signal is necessary because occasionally a false address will appear on the address bus. The VMA signal is used to disable peripheral devices when the address is not valid. A logic 1 at pin 5 indicates that the address is valid and that the peripheral devices may respond accordingly. A logic 0 indicates that the address is not valid and that the peripheral devices should ignore the address. As you will see later, the VMA line is used in any decoding scheme.

The $\overline{\text{HALT}}$ line (pin 2) provides a hardware method of halting the MPU. If this input is forced low, the MPU will finish its present instruction, then it will halt. When halted, all 3-state lines go to their **off** state. This effectively disconnects the MPU from the address and data buses. This line is sometimes used to implement single instruction operation. By controlling the $\overline{\text{HALT}}$ line, the MPU can be forced to stop after each instruction is executed. This can be a valuable aid in troubleshooting hardware and debugging programs. In many applications, the halt capability is simply not required. In this case, the $\overline{\text{HALT}}$ line is permanently connected to +5 volts.

The final control line that is common to both the 6800 and 6808 microprocessors in identity and function is the **bus available** (BA) line. This output (pin 7) indicates whether or not the MPU is executing instructions. Recall that the MPU may stop executing instructions for either of two reasons. First, the WAI instruction will cause the MPU to stop until an interrupt is received. Or, the MPU can be stopped by forcing the $\overline{\text{HALT}}$ line low. A logic 0 on the bus available line indicates that the MPU is running. A logic 1 indicates that the MPU has stopped. When the MPU is stopped, all 3-state outputs go to their off state. Thus, the MPU is effectively disconnected from the buses. The BA signal indicates that this condition exists by going to the logic 1 state. During this period the buses are available for other purposes such as DMA operations.

## THE 6800 MPU INTERFACE LINES

While you are not yet familiar with the remaining 10 pins of the 6800 MPU, five of them are self-explanatory. For example, two of them (pins 35 and 38) are not connected. Also, three of the pins are used for power. Pin 8 is labeled $V_{cc}$. This is the +5-volt supply. Pins 1 and 21 are labeled $V_{ss}$. These are ground pins. Notice that the 6800 MPU requires a single +5-volt supply.

The remaining five lines require a brief explanation. First, there are two *clock signals* labeled $\phi 1$ (pronounced "phi one") and $\phi 2$. A 2-phase non-overlapping clock is required. This must be provided to the MPU from some external clock generator. The details of the clock signal will be discussed later. The normal clock frequency is 1 MHz, although higher speed versions of the 6800 MPU are also available.

The TSC line (pin 39) is used to enable or disable the 3-state address bus and the R/$\overline{\text{W}}$ line, which also has a 3-state capability. This input to the MPU is used in applications in which some external device must periodically take over the address bus. Normally, the MPU has complete control of the address bus and read/write line. However, an external device can effectively disconnect the MPU from the address bus by switching the TSC line to the high state. When TSC goes high, the address bus and read/write line of the MPU go to their off or high impedance state. This allows the external device direct access to memory without going through the MPU. This is called direct memory access or DMA. In many applications, DMA operations are not required and TSC is permanently connected to ground.

The *data bus enable (DBE) line* (pin 36) is the 3-state control line for the data bus. If this input to the MPU is forced low, the data bus will switch to its off or high impedance state. As you will see later, all data transfers between the MPU and memory take place when the $\phi 2$ clock is high. For this reason, the DBE line is often connected to the $\phi 2$ clock.

## THE 6808 MPU INTERFACE LINES

The 6808 microprocessor has a total of five pins for power. Pins 8 and 35 are labeled $V_{cc}$. This is the +5-volt supply. Pins 1, 21, and 36 are labeled $V_{ss}$. These are ground pins.

The 6808 has an internal oscillator that may be crystal controlled. The crystal is connected between pins 38 and 39. These are designated as **XLAL** and **EXTAL** on Figure 7-7. A divide-by-four circuit has been fabricated within the 6808 so that a 4 MHz crystal connected to pins 38 and 39 result in a 1 MHz output. This output (pin 37) is designated as "E". The 1 MHz output is similar to the $\phi 2$ signal on the 6800 microprocessor. The Enable (E output) supplies the clock for peripheral devices.

The final control line of the 6808 that we will consider is the **memory ready** (MR). This input (pin 3) is a control signal which allows stretching of the 1 MHz E output. When MR is high, E will be a normal pulse width. When MR is low, E may be stretched multiples of half periods. This feature allows for interfacing to slow memories.

# Instruction Timing

Before going further, the timing relationship between the various control and bus signals will be discussed. The discussion will start with the most basic timing relationship: the timing for a single instruction. The following discussion applies to both the 6800 and 6808 microprocessors. The only significant difference is where the $\phi 1$ and $\phi 2$ clocks are generated: internal to the 6808, external to the 6800. You should also keep in mind that $\phi 2$ in the following discussion is the E output of the 6808.

Figure 7-8 shows the 2-phase clock signals required by the microprocessor. These two clock signals control every single action that takes place in the MPU and its peripheral devices. To illustrate this, the significant events that occur during the fetch and execution of the LDAA immediate instruction will be discussed. Recall that this is a 2-byte instruction. The first byte is the opcode ($86_{16}$). The second byte is the number that is to be loaded into accumulator A. This instruction requires two MPU cycles. During the first cycle, the opcode is fetched and decoded. During the second cycle, the operand is retrieved from memory and is placed in accumulator A.

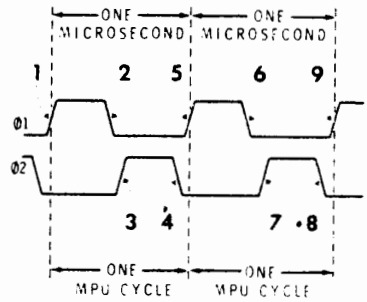



Figure 7-8.
Timing for the LDAA immediate instruction.

The significant events are illustrated. Notice that the events occur at the edges of the clock pulses. Assume that, prior to time 1, the program counter contains the address of the LDAA immediate instruction.

At time 1, the address is transferred from the program counter to the address bus via the memory address register. Notice that this occurs at the positive-going edge of the $\phi 1$ clock. If the VMA and R/$\overline{W}$ lines are not already at logic 1, they will be switched to logic 1. A logic 1 on VMA indicates to memory that this is a valid address. A logic 1 on R/$\overline{W}$ indicates to memory that the MPU wishes to read the byte at the indicated address.

Time 2 is the falling edge of the $\phi 1$ clock. At this time, the program counter will be incremented by one to the address of the next byte in memory. However, this will not change the address on the address bus. Remember that this address is latched into the memory address register. Thus, the output address is still that of the first byte of the LDAA instruction.

The events which occur during the $\phi 1$ clock are initiated from within the MPU itself. In fact, in most systems, the $\phi 1$ clock is applied *only* to the MPU. The $\phi 2$ clock, on the other hand, is applied to the peripheral circuits as well as the MPU. Thus, the RAMs, ROMs, etc., are controlled by the $\phi 2$ clock.

Time 3 is the rising edge of the $\phi 2$ clock. This positive-going edge forces memory to place the data from the indicated address onto the data bus. Recall that this is the opcode for the LDAA immediate instruction or $86_{16}$. Notice that the address has had from time 1 to time 3 to stabilize.

Time 4 is the falling edge of the $\phi 2$ clock. At this time, the MPU accepts the byte from the data bus. Notice that the data bus has from time 3 to time 4 to stabilize. The MPU transfers this byte ($86_{16}$) to the instruction register. There, it is decoded and is interpreted as an LDAA immediate opcode. This tells the MPU that the next byte in memory is the operand that is to be loaded into accumulator A.

This completes the first MPU cycle. During this cycle, the opcode was simply read from memory and decoded. This corresponds to the fetch phase discussed earlier for our hypothetical MPU. Notice that an MPU cycle corresponds to one cycle of the clock. Now let's see what happens during the second cycle or execute phase of the instruction.

At time 5, the address of the operand is transferred from the program counter to the address bus. At time 6, the program counter is incremented by one in anticipation of the next fetch phase.

At time 7, the rising edge of the $\phi 2$ clock causes the operand to be transferred from memory to the data bus. At time 8, the operand is latched into the MPU where it is transferred to accumulator A. This completes the second MPU cycle and the execution phase of the instruction. Time 9 represents the start of the fetch phase for the next instruction. The LDAA immediate instruction required two MPU cycles or two cycles of the clock. Assuming a 1 MHz clock, two microseconds are required for this instruction.

# Timing of Program Segment

Now that you are familiar with the timing of a single instruction, several instructions will be put together to form a sample program. You can then study the timing relationships between the bus signals, clock signals, the R/$\overline{W}$ line, etc.

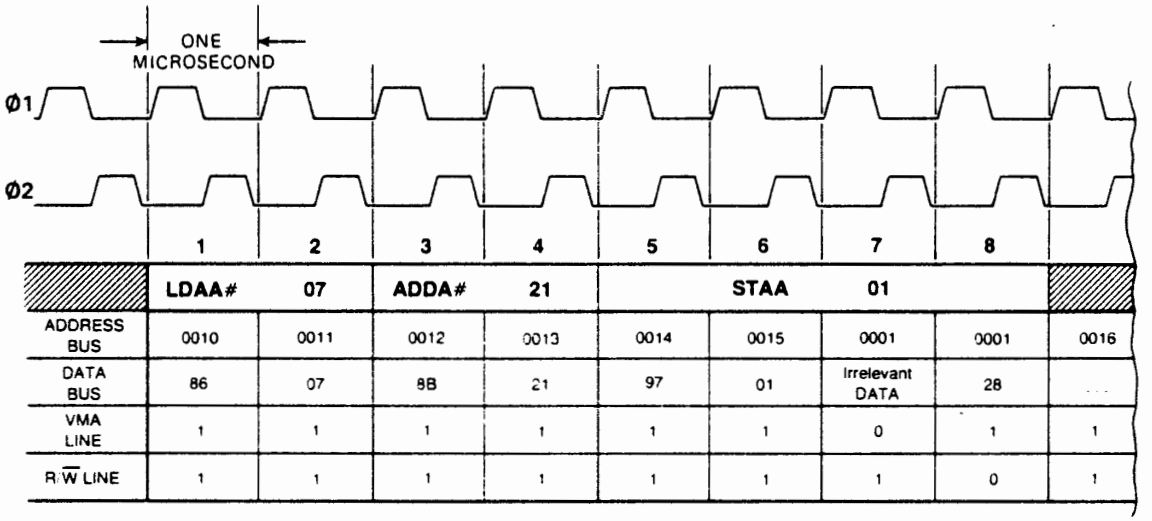
Our sample program segment is shown in Figure 7-9. Using the immediate addressing mode, it loads $07_{16}$ into accumulator A and adds $21_{16}$. The result is then stored in location $0001_{16}$. Notice that the first instruction resides at address $0010_{16}$.

| HEX ADDRESS | HEX CONTENTS | MNEMONIC/ HEX CONTENTS | COMMENTS |
|---|---|---|---|
| 0010 | 86 | LDAA# | Load Accumulator A immediate with |
| 0011 | 07 | 07 | 07. |
| 0012 | 8B | ADDA# | Add to Accumulator A immediate |
| 0013 | 21 | 21 | 21. |
| 0014 | 97 | STAA | Store the result |
| 0015 | 01 | 01 | at this address. |
| 0016 | ... | ... | Next Instruction |

Figure 7-9

Sample program segment.

Figure 7-10 illustrates the timing relationships. At the top the $\phi 1$ and $\phi 2$ clock signals are shown. The information that appears on the buses and control lines for each clock period is shown at the bottom. This program segment requires eight clock or MPU cycles. These are numbered one through eight. Next, you will see what happens during each of these cycles.




| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | LDAA# | 07 | ADDA# | 21 | STAA | | 01 | | |
| ADDRESS BUS | 0010 | 0011 | 0012 | 0013 | 0014 | 0015 | 0001 | 0001 | 0016 |
| DATA BUS | 86 | 07 | 8B | 21 | 97 | 01 | Irrelevant DATA | 28 | |
| VMA LINE | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| R/$\overline{W}$ LINE | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

Figure 7-10
Timing of sample program segment.

**Cycle 1.** During the first cycle, the address of the LDAA# instruction ($0010_{16}$) is placed on the address bus. As a result, the opcode $86_{16}$ is read from that address and is picked up by the MPU. Since this was a read operation from a valid address, the VMA and R/$\overline{W}$ lines are at logic 1. The MPU decodes the opcode and recognizes that this is an LDAA# instruction. Consequently, it knows that the next byte in memory is the operand that is to be loaded into the accumulator. During this cycle, the program counter was incremented to $0011_{16}$ so that it now points at the operand.

**Cycle 2.** This is the execution phase of the LDAA# instruction. The address of the operand ($0011_{16}$) is placed on the address bus. The operand ($07_{16}$) is read out on the data bus and is placed in accumulator A. In the process, the program counter is incremented to $0012_{16}$. This completes the execute phase of the first instruction.

**Cycle 3.** This is the fetch phase of the next instruction. The address $0012_{16}$ is placed on the address bus. The opcode at that address is read out and placed on the data bus. The MPU picks up the opcode, decodes it, and discovers that it is an ADDA# instruction. In the process, the program counter is incremented to $0013_{16}$.

**Cycle 4.** Here, the address $0013_{16}$ is transferred to the address bus and the selected memory location is read out. Therefore, the operand $21_{16}$ is placed on the data bus. The operand is picked up by the MPU and is added to the contents of the accumulator. The sum $28_{16}$ is retained in accumulator A. The program counter is incremented to $0014_{16}$.

**Cycle 5.** This is the fetch phase for the third instruction. The address $0014_{16}$ is placed on the address bus. The opcode for STAA is read out and decoded. The MPU recognizes that the direct address mode is used. Thus, it will interpret the next byte in memory as the address at which the sum is to be stored. The program counter is incremented to $0015_{16}$.

**Cycle 6.** Address $0015_{16}$ is placed on the address bus. Notice that the contents of this location is the address at which the sum is to be stored. Thus, 01 is read out on the data bus where it is picked up by the MPU. Because the MPU recognized that direct addressing is indicated, it assumes that the address at which the sum is to be stored is $0001_{16}$. This address is retained in the MPU. The program counter is incremented to $0016_{16}$.

**Cycle 7.** During this cycle, the MPU prepares to store the sum. To do this, it must transfer the address $0001_{16}$ to the address register. Also, it must transfer the sum from the accumulator to the data register. While this is happening, the MPU must refrain from all external data transfers. To prevent unwanted data transfers, the MPU switches the VMA line low. This tells all peripheral devices that the address is not a valid one and that no data transfers should be initiated. Thus, the peripheral devices simply ignore the data and address buses for this cycle.

**Cycle 8.** The MPU is now ready to store the sum in memory. The address at which the sum is to be stored ($0001_{16}$) is placed on the address bus. The data to be stored ($28_{16}$) is placed on the data bus. The VMA line is switched high, indicating that this is a valid address. The R/$\overline{W}$ line is switched low, indicating that this is a store operation. Thus, the sum ($28_{16}$) is stored away in memory location $0001_{16}$.

Of course, the computer does not stop here. During the next cycle, the next instruction is fetched and decoded. However, the eight machine cycles illustrated above should give you the idea. As you will see later, the timing relationships shown here become important when we interface the MPU with memory or I/O circuitry.

## The 6800 and 6808 Data Sheets

The preceding section has given you some information on the control lines and timing relationships of both microprocessors, but you may have some questions that have not been answered here. For this reason, a detailed data sheet on these microprocessors is included in Appendix B of this course. It explains in more technical language the capabilities of the microprocessors. At this point in your study, you should have little trouble understanding these data sheets. You may want to refer to these data sheets if you have a question that is not answered in the text.

## Self-Test Review

1. What is a 3-state logic gate?

2. How is the non-inverting buffer shown in Figure 7-4A switched to its high impedance state?

3. In the 6800 MPU, how can the address bus be forced to the off state?

4. How can the data bus be forced to its off state?

5. What does a logic 0 on the VMA line indicate?

6. What does a logic 1 on the BA line indicate?

7. How can the MPU be stopped by hardware?

8. How does the MPU indicate that it wishes to store data in memory?

9. List three ways in which an interrupt request differs from a non-maskable interrupt.

10. The following is a list of the MPU's buses and control lines. Characterize each as either input, output, bidirectional, or internally generated.

| | | | |
|---|---|---|---|
| $\overline{\text{Halt}}$ | _____ | R/$\overline{\text{W}}$ | _____ |
| $\phi 1$ | 6800 ____ 6808 ____ | DBE | _____ |
| $\overline{\text{IRQ}}$ | _____ | $\phi 2$ | _____ |
| VMA | _____ | TSC | _____ |
| BA | _____ | $\overline{\text{NMI}}$ | _____ |
| Address Bus | _____ | $\overline{\text{Reset}}$ | _____ |
| Data Bus | _____ | E | _____ _____ |

11. In reference to the clock signals, when is a new address placed on the address bus?

12. When is the program counter incremented?

13. When does memory place data on the address bus?

14. When does the MPU pick up or latch in data that is on the data bus?

# Answers

1. A gate which has an off state in addition to the normal logic 1 and logic 0 states.

2. By applying a logic 0 to the enable/disable input.

3. By applying a logic 1 to the 3-state control (TSC) line.

4. By applying a logic 0 to the data bus enable (DBE) line.

5. That the address is not valid and should be ignored.

6. That the MPU has halted.

7. By applying a logic level 0 to the $\overline{\text{HALT}}$ input.

8. By switching the R/$\overline{\text{W}}$ line to low.

9. First, the interrupt request is maskable; $\overline{\text{NMI}}$ is not.
Second, $\overline{\text{IRQ}}$ is level sensitive; $\overline{\text{NMI}}$ is edge sensitive.
Third, $\overline{\text{IRQ}}$ uses the interrupt vector at address $FFF8_{16}$ and $FFF9_{16}$, while $\overline{\text{NMI}}$ uses the vector at $FFFC_{16}$ and $FFFD_{16}$.

10.

| | | | |
|---|---|---|---|
| HALT | — input | R/W | — output |
| $\phi$1 | 6800 input 6808 internal | DBE | — input |
| IRQ | — input | $\phi$2 | — input |
| VMA | — output | TSC | — input |
| BA | — output | NMI | — input |
| Address Bus | — output | Reset | — input |
| Data Bus | — bi-directional | E | — output, internal |

11. On the rising edge of the $\phi$1 clock.

12. On the falling edge of the $\phi$1 clock.

13. On the rising edge of the $\phi$2 clock.

14. On the falling edge of the $\phi$2 clock.

# INTERFACING WITH RANDOM ACCESS MEMORY

The Cassette tape segment entitled "Semiconductor Memories" is an excellent overview of the types of memories encountered in microprocessor applications. If you have not already done so, you should complete this audio-visual activity at this time.

As shown in the audio-visual presentation, there are several different types of semiconductor memories. Today, the most popular type of memory used with microprocessors is the static RAM.

Recall that a RAM is a random access read/write memory. The static RAM uses bistable flip-flops to store data. Because the data is latched in these flip-flops, no refresh circuitry is required. That is, data can be maintained indefinitely without refreshing as long as power is applied.

Many different types of static RAMs are available. The smaller static RAMs use the bipolar TTL technique. Most of the larger RAMs use either MOS or CMOS technology. One of the most popular sizes of static RAMs is 1024 bits. This size RAM is packaged in a single IC having from 16 to 24 pins. Internally, the 1024 memory cells may be arranged as 128 8-bit words. Thus, a microprocessor system that requires 512 bytes of RAM would require four of these IC's. IC's of this type require eight pins as data lines. For this reason, 24-pin packages are often used. Since price is related to package size, a RAM of this type is often more expensive than a RAM that uses a smaller package.

A more popular scheme is to arrange the 1024 memory cells into 256 four-bit words. With this arrangement, only four data pins are required. Since the microprocessor works with 8-bit words, two of these RAMs must be used to form a 256 by 8-bit memory. As before, four packages are required to form the 512-byte memory. However, the four packages tend to be smaller and probably less expensive.

Another popular arrangement is the 1024 by 1-bit RAM. This scheme uses only one data line per package. Of course, eight of these packages must be used to form 8-bit bytes. Also, a problem arises if we wish to have a memory smaller than 1024 by 8.

# The Static Ram Storage Cell

The basic storage cell for an MOS type static RAM is shown in Figure 7-11. Keep in mind that this cell stores a single bit of data. There may be 1024 of these along with address decoders and bus drivers in a single IC.

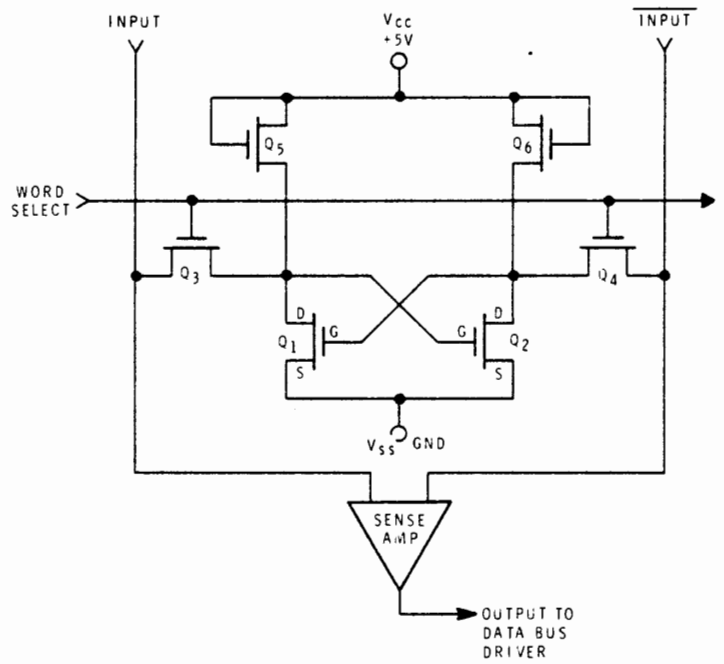



Figure 7-11
The static RAM storage cell.

The storage cell itself consists of the six MOS transistors. In keeping with current convention, the simplified symbol for the MOS transistor is used. Assume that these are N-channel enhancement type MOSFETs. Recall that an enhancement type MOSFET is normally off or non-conducting. However, since these are N-channel devices, they can be made to conduct by placing a positive voltage on the gate terminal. That is, when the gate is near ground potential, the transistor is off and represents a very high impedance. But, when the gate is high (near $+V_{cc}$), the transistor conducts and has a much lower impedance. If you keep these points in mind, the operation of the cell is easy to understand.

Transistors $Q_1$ and $Q_2$ are cross coupled so that they form a bistable latch or flip-flop. The bit of data is stored in this latch. $Q_5$ and $Q_6$ act as load resistors for $Q_1$ and $Q_2$ respectively. MOSFETs are used instead of physical resistors because they take up less chip space. $Q_3$ and $Q_4$ act as switches which connect input data to the latch during write operations. Also, they connect the data in the latch to the output sense amplifier during read operations. The word select line is connected to the gates of $Q_3$ and $Q_4$. A logic 1 on the word select line will turn $Q_3$ and $Q_4$ on. A logic 0 will keep them turned off.

Since this is a RAM storage cell, we must be able to write into it and to read from it. The following discussion will show how we write into it.

**Write Operation.** Before writing a bit of data into the cell, the discussion will first define what constitutes a 1 and 0. Assume that the latch contains a binary 1 when $Q_2$ conducts and $Q_1$ is cut off. And, of course, it contains a binary 0 when $Q_1$ conducts and $Q_2$ is cut off.

When power is initially applied, the flip-flop will latch in one condition or the other but we can never be sure of which. We write data into the cell by controlling the input, $\overline{\text{input}}$, and word select lines. To store a binary 1, we place a logic 1 (a positive voltage) at the input line. The $\overline{\text{input}}$ line is always the opposite state of the input line, so it will go to logic 0 ($\approx$0 volts). The binary 1 is now stored by momentarily applying a logic 1 (positive pulse) to the word select line.

A positive pulse on the word select line will turn switches $Q_3$ and $Q_4$ on. Thus, the positive voltage on the input line is applied through $Q_3$ to the gate of $Q_2$. This forces $Q_2$ to conduct. When $Q_2$ conducts, its drain voltage falls to a low value. This reduced voltage is felt on the gate of $Q_1$, cutting $Q_1$ off. When $Q_1$ cuts off, its drain voltage goes high. This increased voltage is felt on the gate of $Q_2$, holding $Q_2$ in the on state.

When the word select line returns to logic 0, $Q_3$ and $Q_4$ cut off. However, the conduction of $Q_2$ keeps $Q_1$ cut off and the high drain voltage of $Q_1$ keeps $Q_2$ conducting. Thus, the binary 1 is latched in the flip-flop. It will remain there until power is removed or until a binary 0 is deliberately written in.

We can later write in a 0 if we like by applying a logic 0 to the input, a logic 1 to $\overline{\text{input}}$ and then pulsing the word select line. When the word select line goes high, $Q_4$ conducts, applying the logic 1 from $\overline{\text{input}}$ to the gate of $Q_1$. This tends to bring $Q_1$ out of cutoff. At the same time, $Q_3$ conducts, applying the logic 0 from the input to the gate of $Q_2$. This tends to cut off $Q_2$. As you can see, the flip-flop latches in the opposite state. This represents a binary 0.

The sense amplifier and output line (shown at the bottom) are disabled during this period by the read/write line (not shown).

**Read Operation.** The input and $\overline{\text{input}}$ lines are 3-state lines that are enabled or disabled by the read/write line. When the read/write line (not shown) is high, the input and $\overline{\text{input}}$ lines are effectively disconnected. During this period, we can read data from the storage cell by pulsing the word select line. Assume that the cell is presently storing a logic 1. This means that $Q_1$ is cut off and that $Q_2$ is conducting. Consequently $Q_1$'s drain voltage is high (logic 1) while $Q_2$'s drain voltage is low. When the word select line swings high, $Q_3$ conducts, connecting the high voltage at the drain of $Q_1$ to the left input of the sense amplifier. At the same time, $Q_4$ conducts, connecting the low voltage at the drain of $Q_2$ to the right input of the sense amplifier. The sense amplifier interprets this as a logic 1 condition and sets the output line accordingly.

If the flip-flop contains a logic 0 when the word select pulse occurs, the right input of the sense amplifier receives the higher voltage while the left input receives the lower. The sense amplifier interprets this as a logic 0.

## A 128-Word by 8-Bit Ram

Because a storage cell can store only one bit of information, large numbers of these cells are required to form useful memory sizes. Figure 7-12 shows how 1024 cells can be arranged to form a 128-word by 8-bit RAM. Each of the squares represents one of the 6-transistor cells just discussed.

The word select lines are shown entering on the left. While only four are shown, there would actually be 128 of these lines — one for each word. Notice that word select line 00 connects to each of eight storage cells across the top of the figure. In an actual system, these eight storage cells might make up the 8-bit byte we call memory location $0000_{16}$.

The input lines are shown at the top of the diagram. For simplicity, only four of the eight lines are shown. Notice that each input line is inverted so that complement inputs can be applied to each cell. Although the details are not shown, both the input and $\overline{\text{input}}$ lines are 3-state lines so that they are effectively disconnected except during a write operation.

The output lines are shown at the bottom of the diagram. These lines are disabled during a write operation but they are enabled during a read operation. One sense amplifier is required for each output line.

Keep in mind that Figure 7-12 does not show the complete RAM. It merely shows the memory storage matrix and the sense amplifiers. Some additional circuits are required to turn this into a working RAM. One of these is an address decoder.

The memory array is arranged as 128 bytes. An address decoder is required that can select any of these 128, 8-bit storage locations. Thus, a 1 of 128 decoder is required. The input to the decoder is the seven address lines from the MPU. Recall that seven bits can specify 128 different addresses.

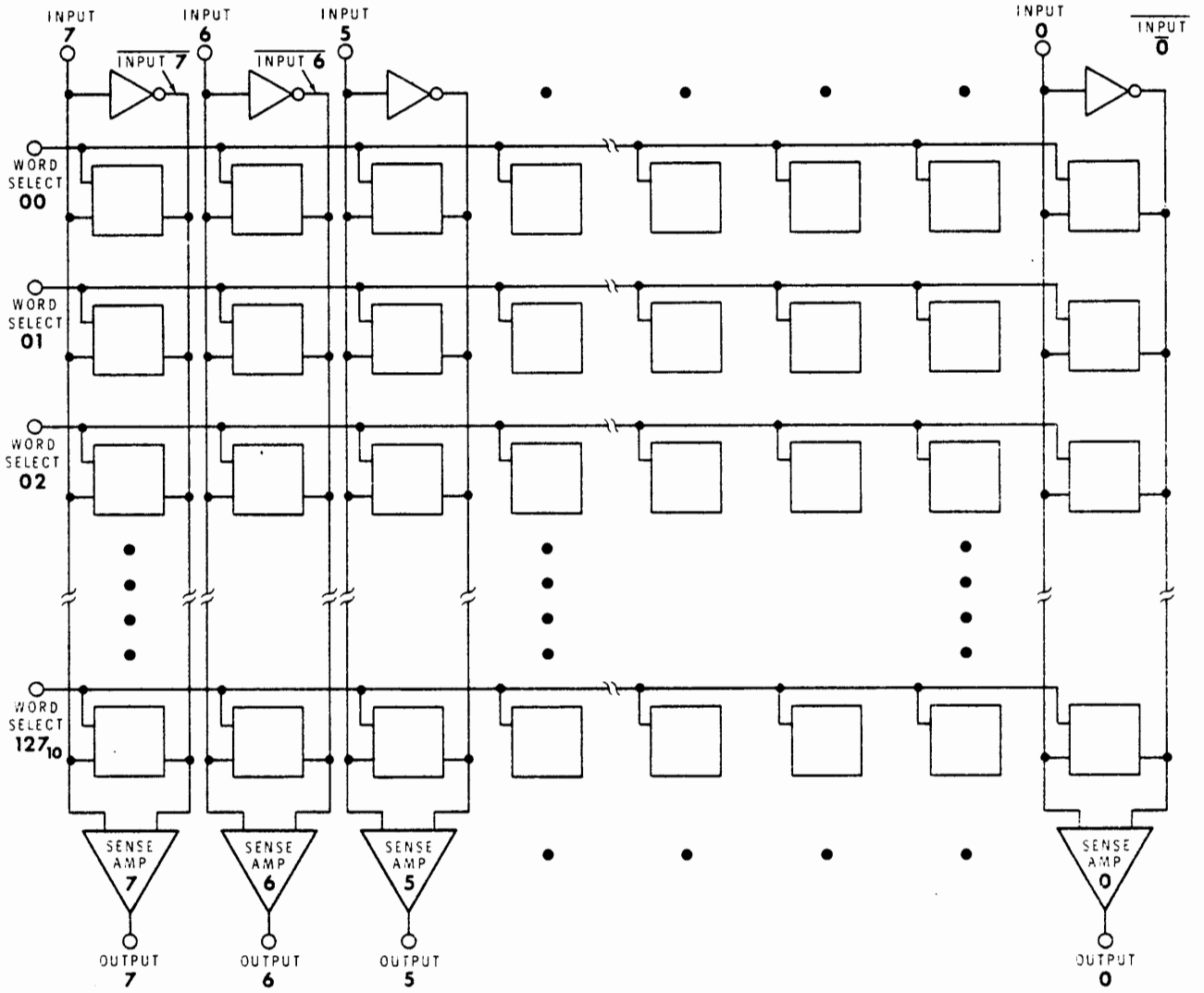



Figure 7-12

Here 1024 storage cells are arranged to form a 128-byte by 8-bit RAM.

The address decoder is made up of 128 7-input AND gates. A simplified diagram is shown in Figure 7-13. Here, only three gates are shown — the first two and the last. Word select line 00 should go high when address lines $A_0$ through $A_6$ are all low. Notice that seven inverters are used to form $\overline{A_0}$ through $\overline{A_6}$. Notice also, that these complements are the inputs to the top AND gate. If $A_0$ through $A_6$ are all low, then $\overline{A_0}$ through $\overline{A_6}$ must be all high. Consequently, the output of the AND gate goes high. Thus, word select line 00 is selected when the low order address is $0000000_2$.

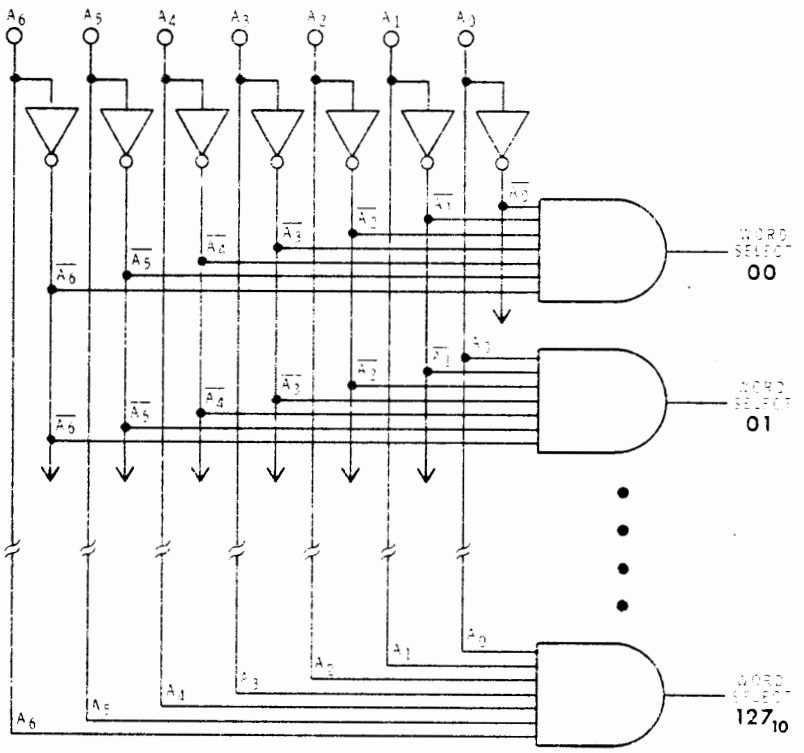



Figure 7-13
The 1 of 128 address decoder.

The 01 select line is activated when the address is $0000001_2$. The next $125_{10}$ gates are not shown. Word select line $127_{10}$ is selected when $A_0$ through $A_6$ are all high. Thus, it is activated when the address is $1111111_2$.

Most RAMs do not have separate input and output lines. Instead they have data lines which can serve either as inputs or outputs. This is possible because the MPU cannot read and write data simultaneously.

Figure 7-14 shows a simplified arrangement. The data lines are shown on the left. The 3-state input buffers are enabled by a high signal on the WRITE line. This line is controlled by the R/$\overline{W}$ signal and the chip enable (CE) signal. As you will see, the WRITE line is high when the MPU is writing data into memory. This enables the input buffers and allows data to be written into the selected address. The output buffers are disabled during this period by the low signal on the READ line.

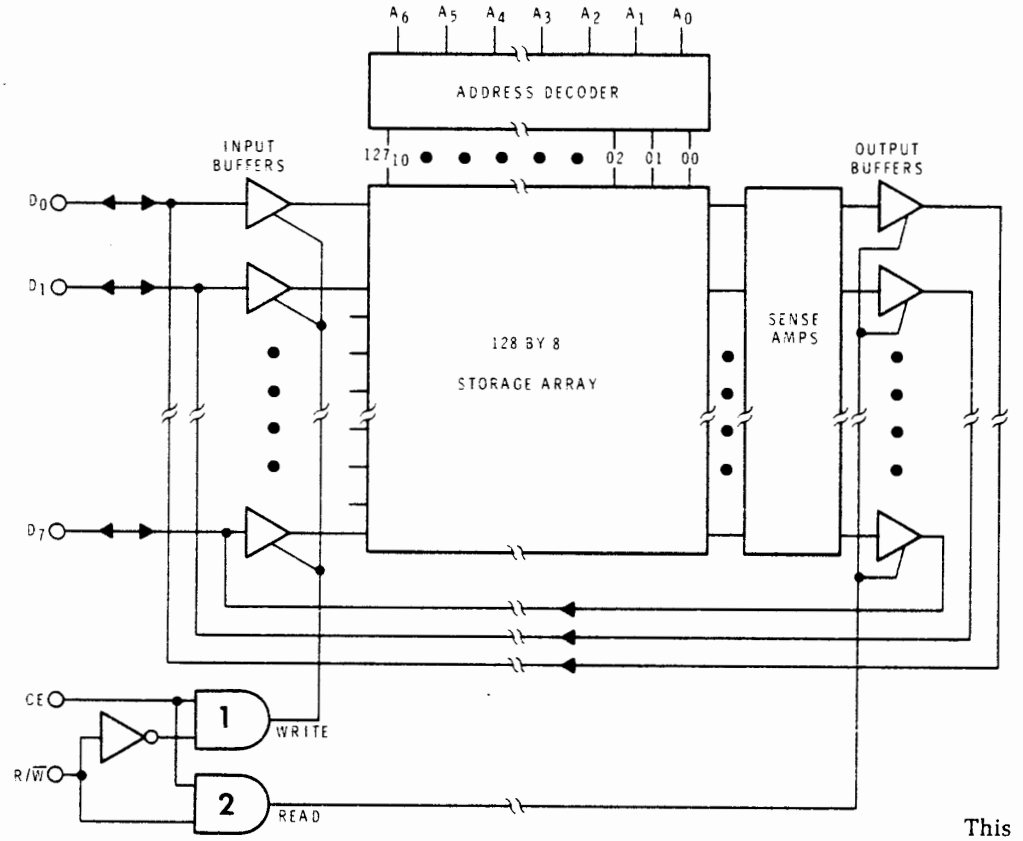



Figure 7-14
This RAM has bi-directional data lines.

When data is to be read from the RAM, the READ line is switched high and the WRITE line is switched low. This disables the input buffers and enables the output buffers. Thus, the data at the selected address is read out and placed on the data bus.

As Figure 7-14 illustrates, the READ and WRITE signals are controlled by the chip enable (CE) signal and the R/$\overline{W}$ line. The CE input line is switched high when this particular memory chip is selected. If this line is low, gates 1 and 2 are disabled. This causes both the READ and WRITE signals to go low, disabling both the input and output buffers. In effect, it disconnects this chip from the data bus.

However, when CE is high, the R/$\overline{W}$ line controls the READ and WRITE signals. This R/$\overline{W}$ line is connected to the R/$\overline{W}$ line of the MPU. Recall that the R/$\overline{W}$ line is low when the MPU is writing data into RAM, and high when the MPU is reading from RAM. When R/$\overline{W}$ is high, the output of gate 2 goes high, enabling the output buffers. The output of gate 1 is held low, disabling the input buffers. This places the RAM in the read mode.

When R/$\overline{W}$ goes low, the output of gate 1 goes high and the output of gate 2 goes low. This places the RAM in the write mode.

Some RAMs have a single chip enable line. In many RAMs, the chip enable line is labeled $\overline{CE}$, meaning that the chip is selected when the enable line is low. Some RAMs have several chip enable (CE) or chip select (CS) lines.

Figure 7-15 shows a 128 by 8 RAM that is designed to be used with the 6800 MPU. As shown in the simplified block diagram, this RAM has six chip select lines. The large number of chip selects allows this RAM to be used with little or no external address decoding. As shown in the pin assignment diagram, a 24-pin package is required. This RAM is called the 6810. Data sheets on this device are included in Appendix B of this course.

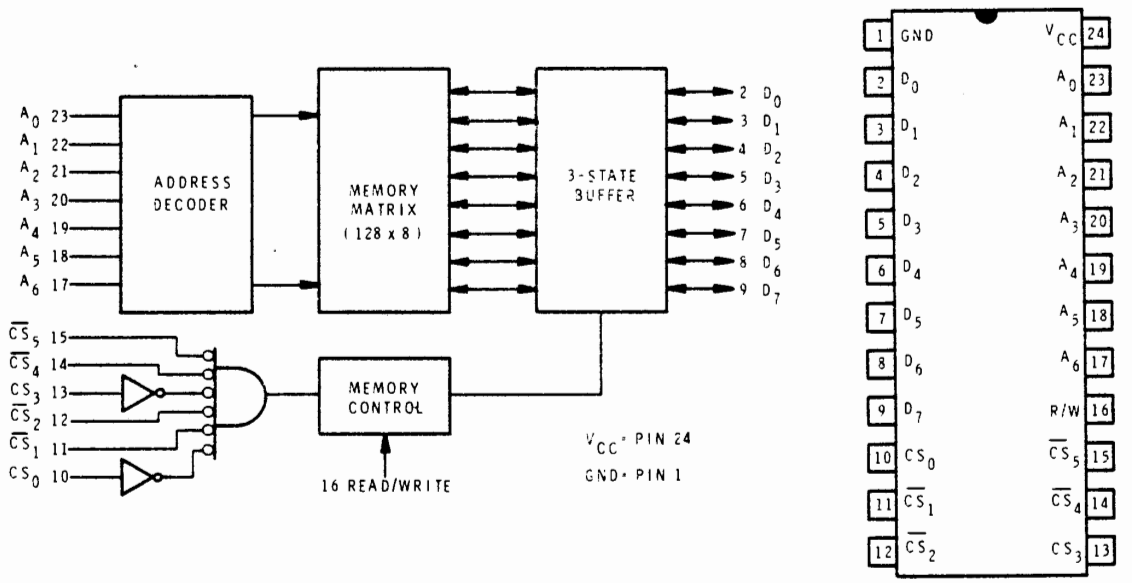



Figure 7-15
The 6810 is a 128-byte by 8-bit RAM.

# A 256 by 4-Bit Ram

Figure 7-16 shows the block diagram of a popular 256 by 4-bit static RAM. In 8-bit systems, two of these would be required to form a $256_{10}$ byte memory. Like the $128_{10}$ by 8 RAM, this circuit has $1024_{10}$ storage cells. To simplify the address decoders, the cells are arranged in 32 rows of 32 cells each. The 32 cells in each row are further divided into 8 columns of 4 bits each. Thus, the array consists of 32 rows by 8 columns by 4 bits (or 256 by 4 bits).

Two address decoders are used. The row select decoder is a 1-of-32 decoder which chooses the row. Five address lines ($A_0$ through $A_4$) are used to specify the proper row.

A 1-of-8 column decoder is used to select the proper column. Three address lines ($A_5$ through $A_7$) are used to specify the proper column. The selected 4-bit word is at the point where the row and column lines intersect.

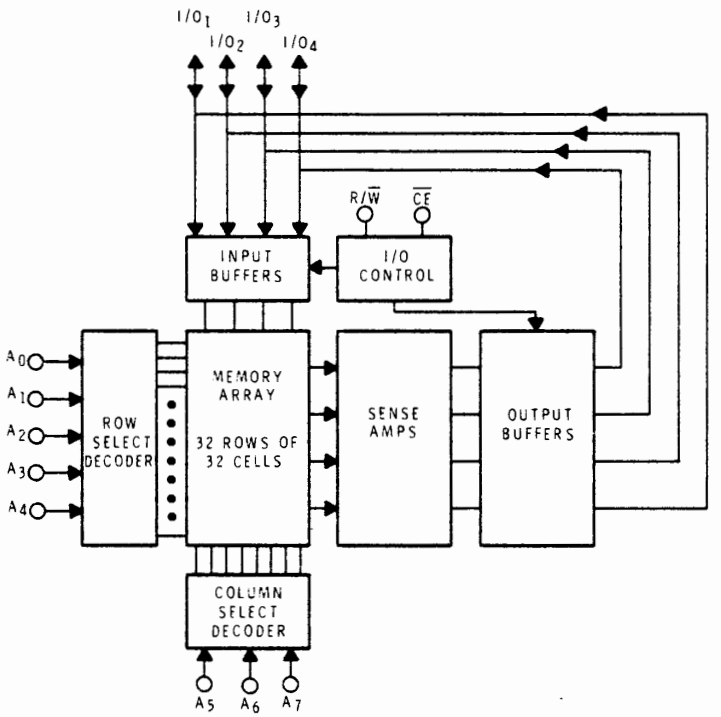



Figure 7-16
Block Diagram of a 256 by 4 RAM.

In this particular IC, the data lines are called I/O lines. The R/$\overline{W}$ line determines whether the IC is in the read or write mode. The $\overline{CE}$ line determines if this particular IC has been selected. If you count the pins shown and add one for $V_{cc}$ and another for ground, you will see that a 16-pin package is required. Figure 7-17 shows the pin assignments and logic diagram of the popular 2112 static RAM. It has the same general arrangement as that shown in the block diagram.

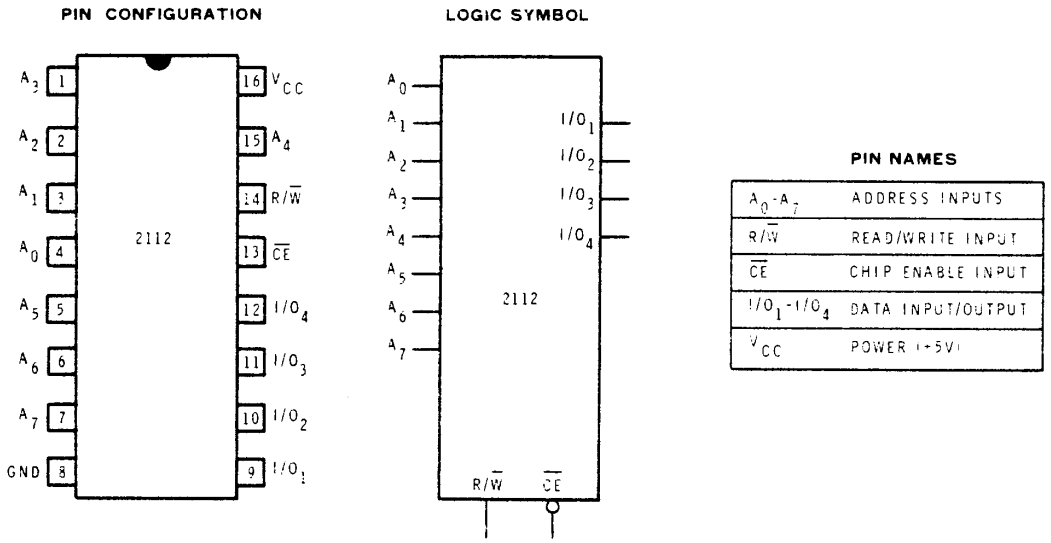



Figure 7-17
Pin assignments and logic symbol for the 2112 static RAM.

## Connecting Ram to the MPU

Figure 7-18 is a partial schematic of a 6800 based microprocessor system. Two 2112 IC's are used as a $256_{10}$-by-8 RAM. Although not shown, the address and data buses from the MPU are also connected to a ROM, input and output circuits, and possibly additional RAMs. The MPU can communicate with only one of these devices at any one time. To communicate with the two 2112 ICs, the MPU must first select them by switching their $\overline{CE}$ lines low. Notice that the $\overline{CE}$ lines are connected to the output of the RAM address decoder. This decoder monitors the eight high-order address lines. When the address of the RAM appears on these lines, the two 2112 chips are enabled.

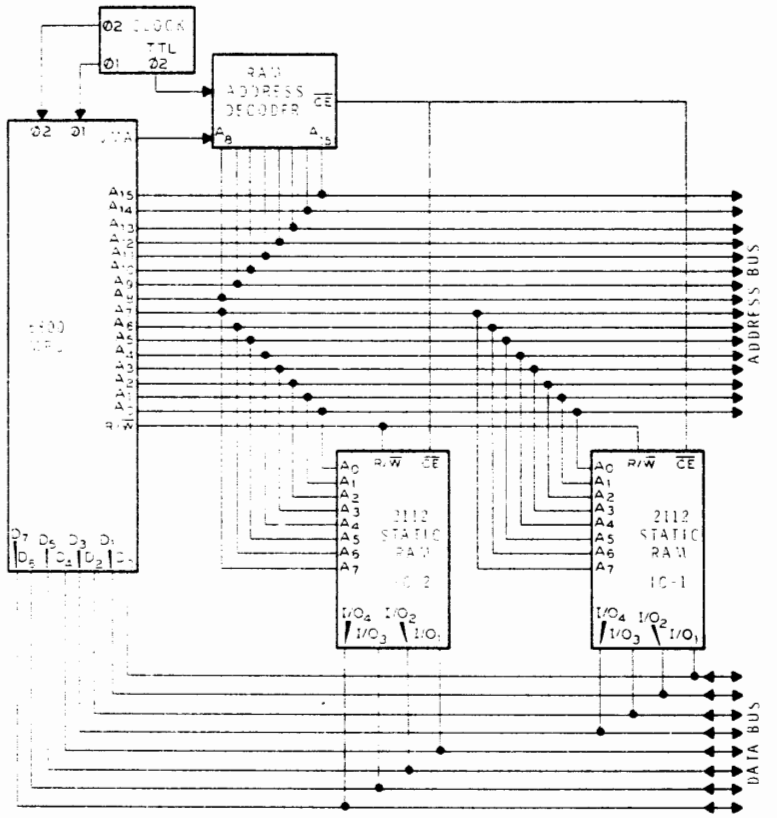



Figure 7-18
Partial schematic of a 6800 based microprocessor using two 2112 static RAMs.

The two 2112 IC's make up a 256 by 8-bit RAM. IC1 connects to the four least significant bits of the data bus ($D_0$ through $D_3$), while IC2 connects to the four most significant bits. Thus, when a byte of data is stored in the RAM, the four LSB's go in IC1, while the four MSB's go in IC2. This is possible because the two IC's are enabled at the same time. Notice that the address, $\overline{CE}$, and R/$\overline{W}$ lines of the two ICs are tied together.

The address lines of the IC's monitor the $A_0$ through $A_7$ lines from the MPU. Once the chips are enabled, any one of the 256 memory locations can be selected by placing the proper address on the lower eight address lines.

# Address Decoding

The $256_{10}$-byte RAM must be assigned some starting address. In 6800-based systems, a common practice is to assign RAM the lowest addresses. Thus, a $256_{10}$-byte RAM would be given address $0000_{16}$ through $00FF_{16}$. In binary, these are addresses 0000 0000 0000 $0000_2$ through 0000 0000 1111 $1111_2$. Notice that the eight LSB's can specify any one of the $256_{10}$ memory locations. However, it is the upper eight bits that specify that the starting address is at the low end of memory. That is, the RAM must be enabled when the upper eight bits of the address bus are 0000 0000.

In Figure 7-18, the RAM address decoder monitors the upper eight bits of the address bus. When this decoder finds that the upper eight bits are all zeros, it switches the $\overline{CE}$ line low, enabling the RAM. However, notice that VMA and the $\phi 2$ clock signals are also applied to the decoder. Recall that VMA is 1 when the address is valid. Thus, the decoder must also monitor the VMA line, since the RAM should not be enabled unless the address is valid. Recall also that the MPU must receive and transmit data only while the $\phi 2$ clock is at logic 1. Thus, the decoder also monitors the $\phi 2$ clock. If the high-order address lines are all zeros, the VMA line is high, and the $\phi 2$ clock is high, the RAM will be enabled.

The address decoder can be any type of logic circuit that meets the above requirements. A typical circuit is shown in Figure 7-19. If you trace through the various logic levels, you will see that $\overline{CE}$ is low only when $A_8$ through $A_{15}$ are low and VMA and $\phi 2$ are high. NAND gate 1 produces a low at its output when VMA and $\phi 2$ are high. The other inputs to NOR gate 2 and the inputs to NOR gates 3 and 4 are low when the high-order address is 0000 $0000_2$. The three NOR gates produce high outputs. Thus, the inputs to NAND gate 5 are all high. This forces the output of gate 5 low. As you can see, this circuit fulfills the requirements of the address decoder.

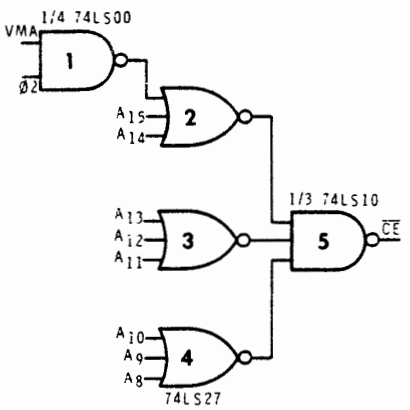



Figure 7-19

Address decoder using discrete logic gates.

A memory location in the RAM shown in Figure 7-18 is selected by all 16 address lines. The low-order address lines connect directly to the RAM IC's, while the high-order lines connect to the RAM address decoder. Thus, each memory location in RAM has only one address. The addresses are said to be *fully decoded*.

We can save some decoding logic by only *partially decoding* the address. Figure 7-20 shows an address decoder that monitors only two of the address lines ($A_{14}$ and $A_{15}$). If you trace the logic levels through, you will see that $\overline{CE}$ is low any time that $A_{14}$ and $A_{15}$ are low and VMA and $\phi 2$ are high. $A_{14}$ and $A_{15}$ will be low for any address at or below $3FFF_{16}$. If the $\overline{CE}$ line is used to enable a RAM that has fewer than $3FFF_{16}$ bytes, some of the addresses will be duplicated. To illustrate this point, assume that we replace the address decoder shown in Figure 7-18 with the circuit shown in Figure 7-20. Memory location $0000_{16}$ can be selected by placing address $0000_{16}$ on the address bus. However, location $0000_{16}$ is also selected when $0900_{16}$ is placed on the address bus. The reason for this is that the RAM is enabled because $A_{14}$ and $A_{15}$ are low. And, the lowest address in the RAM is read out because $A_6$ through $A_7$ are low. Actually, there are dozens of addresses that will select any given location. For example, addresses $3F00_{16}$, $2700_{16}$, $1C00_{16}$, and many others will all select memory location $0000_{16}$. This is the price that must be paid for saving a few gates in the address decoder.

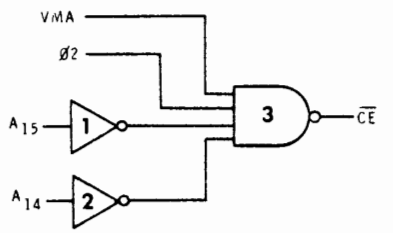



Figure 7-20
We can save gates by only partially decoding the address.

In practice, this does not actually cause many problems. All we have done is sacrifice the lower $16{,}384_{10}$ addresses to $256_{10}$ bytes of RAM. However, this still leaves three-fourths of the 65 K of addresses untouched. In most cases, this leaves more than enough addresses for I/O devices, ROMs, etc.

The fact that a given byte of data appears at several addresses is also no problem as long as we remember this limitation in our programming. Partial decoding schemes are frequently used because they save decoding logic.

## Self-Test Review

15. List three common arrangements for a 1024-bit static RAM.

16. Using 256 by 4-bit static RAMs, what is the smallest 8-bit RAM possible?

17. What is the advantage of using two 256 by 4-bit RAMs to form 256 by 8-bit memory rather than using two 128 by 8-bit RAMs.

18. How many bits of data can be stored in the circuit shown in Figure 7-11?

19. Explain how data is written into this circuit.

20. Explain how data is read from this circuit.

21. Refer to Figure 7-13. Which address lines would connect to the input of the AND gate that drives word select line 02?

22. Refer to Figure 7-14. What is the state of the input and output buffers if CE is low?

23. When CE is high and R/$\overline{W}$ is low, the circuit shown in Figure 7-14 is in the _____ state.

24. In addition to some address lines, what other signals are connected to an address decoder in a 6800-based microcomputer system?

25. How is the 2112 static RAM enabled?

26. Refer to Figure 7-19. What conditions must be met before $\overline{CE}$ will go low?

27. What is the advantage and disadvantage of only partially decoding an address?

28. When each byte of memory has one and only one address, the memory is said to be _____ decoded.

# Answers

15. The most common arrangements for a 1024-bit static RAM are:

    1024 by 1-bit
    256 by 4-bits
    128 by 8-bits

16. 256 by 8-bits.

17. Because the 128 by 8-bit RAM requires 8 data lines, it uses a larger and usually more expensive package.

18. Only one at a time.

19. The input data bit sets the INPUT and $\overline{\text{INPUT}}$ lines to the proper complementary states, then the word select line is pulsed high. This turns $Q_3$ and $Q_4$ on connecting the inputs to the flip-flop. This sets or resets the flip-flop to the proper state.

20. The two input lines are tri-stated. The word select line is pulsed high turning on $Q_3$ and $Q_4$. This connects the outputs of the flip-flop to the sense amplifier. In turn, the sense amplifier sets the data line to the proper state.

21. $\overline{A}_0$, $A_1$, $\overline{A}_2$, $\overline{A}_3$, $\overline{A}_4$, $\overline{A}_5$, and $\overline{A}_6$.

22. Both the input and output buffers are disabled.

23. Write.

24. The VMA line and the $\phi 2$ clock signal.

25. The 2112 static RAM is enabled by applying a logic 0 to the $\overline{\text{CE}}$ line.

26. VMA and $\phi 2$ must be high. At the same time, address lines $A_8$ through $A_{15}$ must be low.

27. The advantage is that generally fewer logic gates are required. The disadvantages are that it wastes addresses and that a given byte of data may be accessed at several different addresses.

28. Fully.

# INTERFACING WITH DISPLAYS

One of the most popular output devices used with the microprocessor is the 7-segment LED display. It can be used to display the decimal digits 0 through 9 or the hexadecimal digits 0 through F. It can also display many special characters. Its low cost and flexibility make the 7-segment LED display ideal for low cost microprocessor based systems.

## The 7-Segment Display

The 7-segment display consists of seven LED's arranged in the pattern shown in Figure 7-21A. An eighth LED is generally included to act as a decimal point. By lighting the LED's in different combinations, $256_{10}$ patterns are possible. These range from a blank display (all segments off) to the digit 8 with a decimal point (all segments on). Figure 7-21B shows some of the patterns that can be formed.

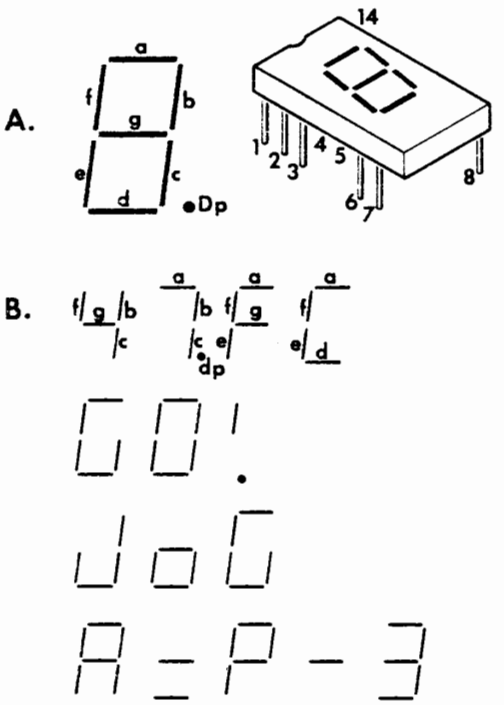



Figure 7-21
The 7-segment display can be used to form many numerals, letters, and symbols.

Two types of 7-segment displays are popular. One is called the common-anode type. As shown in Figure 7-22A, the anodes of the eight light-emitting diodes are tied together and connected to $+V_{cc}$. A diode is forward biased (turned on) by applying a low logic level to its cathode. An external series resistor is required to reduce the current to an acceptable level.

The other type is called the common-cathode type. As Figure 7-22B illustrates, the cathodes are tied together and connected to ground. In this case, a diode is forward biased (turned on) by applying a high logic level to its anode.

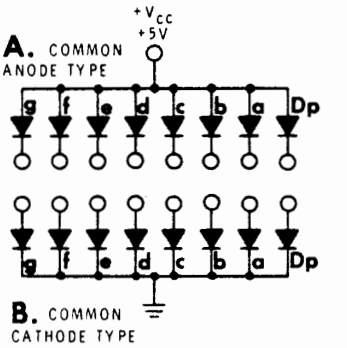



Figure 7-22
Two types of 7-segment displays.

## Driving the 7-Segment Display

A number of IC's are especially designed to drive the 7-segment display. A typical example is the 7447 shown in Figure 7-23A. This is a BCD-to-7-segment decoder-driver. It receives a 4-bit binary number at inputs A, B, C, and D. It provides the proper patterns at outputs a through g to form the numerals 0 through 9 and six special characters as shown in Figure 7-23B.

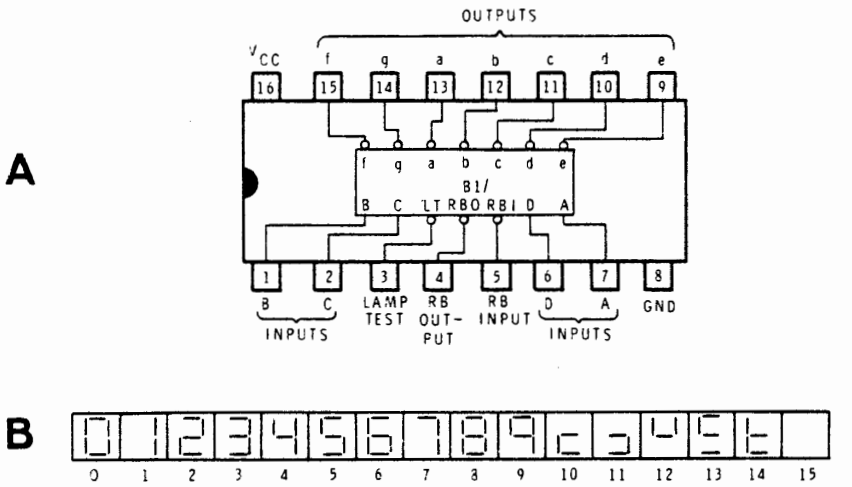



Figure 7-23
The 7447 seven segment decoder-driver and its resultant displays.

When used with a microprocessor, the MPU does not drive the decoder-driver directly. Recall that the information on the data bus is there for a microsecond or less. Since the 7447 has no latch capability, a separate latch must be used to store the data at the right instant. Figure 7-24 shows a representative circuit.

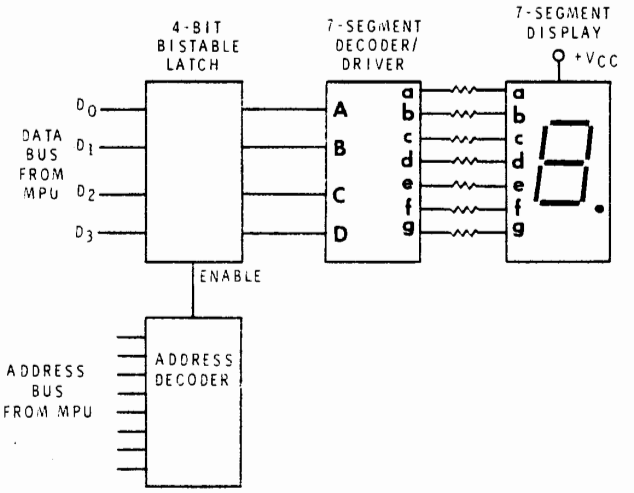



Figure 7-24
Using the decoder/driver.

Here a 4-bit bistable latch is used between the data bus and the decoder-driver. The latch is enabled only at the instant that its address appears on the address bus. For example, assume that the latch has been given the address $8000_{16}$. An instruction such as STAA $8000_{16}$ will activate the display. When this instruction is executed, the address 8000 is placed on the address bus. The address decoder recognizes this address and enables the latch. Thus, the data on lines $D_0$ through $D_3$ are latched into the 4-bit latch. An instant later, the MPU places new information on the address and data buses. However, the display responds only to what has been preserved in the 4-bit latch.

The advantage of this circuit is that it requires only one instruction from the MPU to display a given number indefinitely. Its disadvantage is its lack of flexibility. Of the $256_{10}$ displays possible, only the $16_{10}$ provided by the decoder-driver can be used. Thus, this arrangement can display only those symbols shown in Figure 7-23B.

A more versatile way of driving a 7-segment display is shown in Figure 7-25. Here, a low current display is used so that the latch can drive the display directly. Notice that the BCD-to-7-segment decoder is eliminated. The seven segments of the display (and the decimal point) are now controlled directly by the eight bits of data on the data bus.

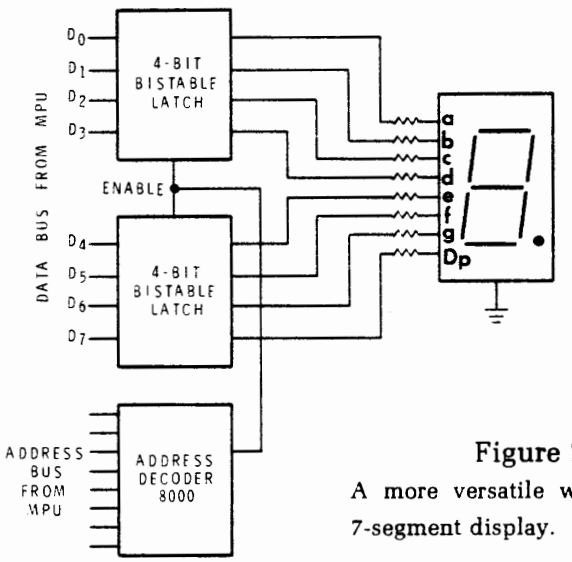



Figure 7-25
A more versatile way of driving a 7-segment display.

The display shown is a common-cathode type. Thus, a segment is turned on by applying a logic high to its input. Figure 7-26 shows the 8-bit pattern that is required to display the digit 1. Since this 8-bit pattern comes from the MPU, the microprocessor must do the decoding. This requires more MPU time and instructions but it greatly increases the versatility of the display. We can now use any of the $256_{10}$ possible displays by providing the proper 8-bit pattern.

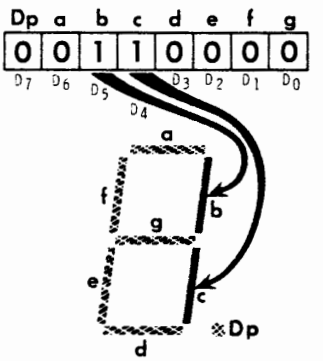



Figure 7-26
Each bit controls one segment of the display.

A display of this type is often used to display the hexadecimal digits 0 through F. The MPU must convert the binary numbers 0000 through 1111 into the proper 8-bit patterns to display the equivalent hexadecimal digit. The easiest way to do this is to use a "look-up" table. The 8-bit patterns are placed in $16_{10}$ consecutive memory locations. Assume that the starting address of the table is $FF96_{16}$ as shown in Figure 7-27.

| HEX ADDRESS | HEX CONTENTS | MNEMONIC/ CONTENTS | COMMENTS |
|---|---|---|---|
| 0010 | 97 | STAA | Store the binary number at the |
| 0011 | 16 | 16 | variable offset. |
| 0012 | CE | LDX# | Load the index register |
| 0013 | FF | FF | with the starting address |
| 0014 | 96 | 96 | of the table. |
| 0015 | A6 | LDAA,X | Load A indexed using the |
| 0016 | — | variable offset | variable offset |
| 0017 | B7 | STAA | Store the 7-segment |
| 0018 | 80 | 80 | code at $8000_{16}$. |
| 0019 | 00 | 00 | |
| . | . | | |
| . | . | | |
| . | . | $D_p$ a b c d e f g | NEXT INSTRUCTION |
| . | . | | |
| FF96 | 7E | 0 1 1 1 1 1 1 0 | 7-Segment pattern for 0 |
| FF97 | 30 | 0 0 1 1 0 0 0 0 | 7-Segment pattern for 1 |
| FF98 | 6D | 0 1 1 0 1 1 0 1 | 7-Segment pattern for 2 |
| FF99 | 79 | 0 1 1 1 1 0 0 1 | 7-Segment pattern for 3 |
| FF9A | 33 | 0 0 1 1 0 0 1 1 | 7-Segment pattern for 4 |
| FF9B | 5B | 0 1 0 1 1 0 1 1 | 7-Segment pattern for 5 |
| FF9C | 5F | 0 1 0 1 1 1 1 1 | 7-Segment pattern for 6 |
| FF9D | 70 | 0 1 1 1 0 0 0 0 | 7-Segment pattern for 7 |
| FF9E | 7F | 0 1 1 1 1 1 1 1 | 7-Segment pattern for 8 |
| FF9F | 7B | 0 1 1 1 1 0 1 1 | 7-Segment pattern for 9 |
| FFA0 | 77 | 0 1 1 1 0 1 1 1 | 7-Segment pattern for A |
| FFA1 | 1F | 0 0 0 1 1 1 1 1 | 7-Segment pattern for B |
| FFA2 | 4E | 0 1 0 0 1 1 1 0 | 7-Segment pattern for C |
| FFA3 | 3D | 0 0 1 1 1 1 0 1 | 7-Segment pattern for D |
| FFA4 | 4F | 0 1 0 0 1 1 1 1 | 7-Segment pattern for E |
| FFA5 | 47 | 0 1 0 0 0 1 1 1 | 7-Segment pattern for F |

Figure 7-27
Program segment and table for converting binary to 7-segment display format.

A program segment is required that will store the proper 7-segment pattern in the latches shown in Figure 7-25. Assume that the address of the latches is $8000_{16}$. The program segment might look like that shown in Figure 7-27.

The program assumes that the binary number we wish to convert to its 7-segment hexadecimal equivalent is in accumulator A. The first instruction stores the binary number at address $0016_{16}$. Assume that the number is $0000\ 0110_2$ or $06_{16}$. Thus, this number is stored at address $0016_{16}$. Notice that this number becomes the offset for the LDAA, X instruction at address $0015_{16}$.

The second instruction loads the starting address of the table ($FF96_{16}$) into the index register. Then, accumulator A is loaded using indexed addressing. The offset, which is now $06_{16}$, (by virtue of the first instruction) is added to the contents of the index register ($FF96_{16}$) to form the address of the operand ($FF9C_{16}$). Thus, the number at address $FF9C_{16}$ is loaded into accumulator A. This number is $5F_{16}$, which is the proper 7-segment code to form the numeral 6.

Finally, this number is stored at address $8000_{16}$. This is the address of the latches that drive the display. Therefore, if $06_{16}$ is in accumulator A when this program segment is executed, the digit 6 will be displayed. Also, if $02_{16}$ is initially in accumulator A, the program segment will cause a 2 to be displayed.

Using this arrangement, two 4-bit latches are required for each display. There are eight bit latches available in a single IC. However, most come in 20 to 24-pin packages, which are relatively expensive. There is one type of 8-bit latch that comes in a 16-pin package. This device will be discussed next.

# Using an Addressable Latch

Figure 7-28 shows the pin outs and schematic for the 8-bit addressable latch. It can store an 8-bit byte and drive a low current display. Its most striking characteristic is that data is entered into the device in serial form. That is, it has a single data input (pin 13). Thus, the eight bits of data must be entered into the device one bit at a time. This explains how the IC can get by with only 16 pins.

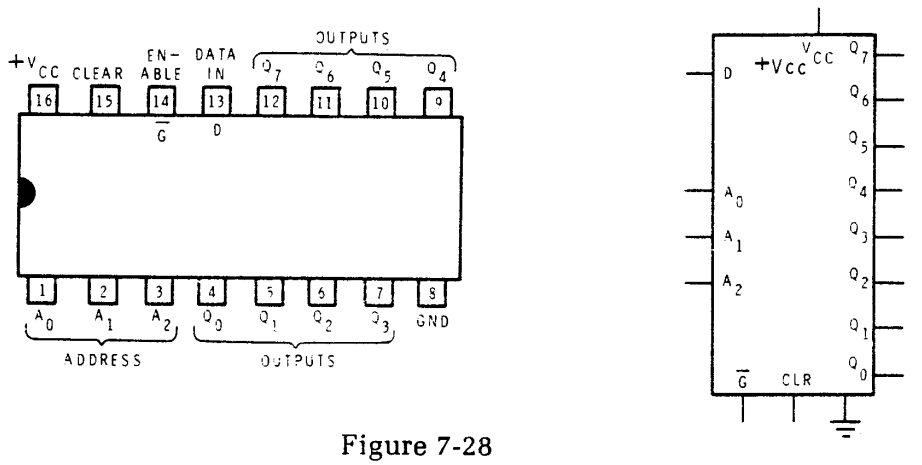



Figure 7-28
Pin outs and schematic diagram of the 74LS259 latch.

The addressable latch has an enable pin that allows it to be selected by an address decoder. A low at pin 14 will enable the latch and allow it to receive data. Actually, there are eight latches on the IC. They are numbered 0 through 7. The particular latch to which the input data bit is routed is determined by the 3-bit address at pins $A_0$, $A_1$, and $A_2$. Figure 7-29 shows which latch is selected for each address.

| ADDRESS INPUTS | | | LATCH SELECTED |
|---|---|---|---|
| $A_2$ | $A_1$ | $A_0$ | |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

Figure 7-29.
Latch selection table.

How can this latch be used by the microprocessor to drive a display? Figure 7-30 shows a representative circuit. The three address lines on the addressable latch are connected to their corresponding lines on the address bus. The remaining lines of the address bus are connected to the address decoder. Assume that the address decoder is arranged so that the addressable latch is enabled for addresses $C160_{16}$ through $C167_{16}$. That is, the addressable latch is enabled for eight different addresses. Address $C160_{16}$ selects latch 0 of the addressable latch; $C161_{16}$ selects latch 1; and so forth.

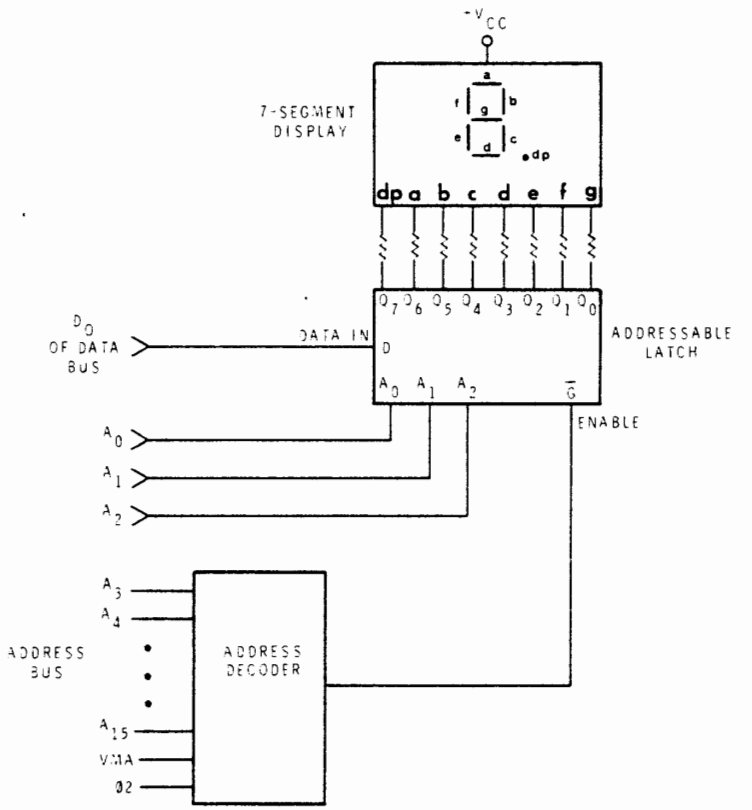



Figure 7-30.
Using the addressable latch to drive a 7-segment display.

The only data line connected to the addressable latch is $D_0$ from the microprocessor's data bus. The outputs of the eight latches ($Q_0$ through $Q_7$) drive the seven segments and the decimal point of the display.

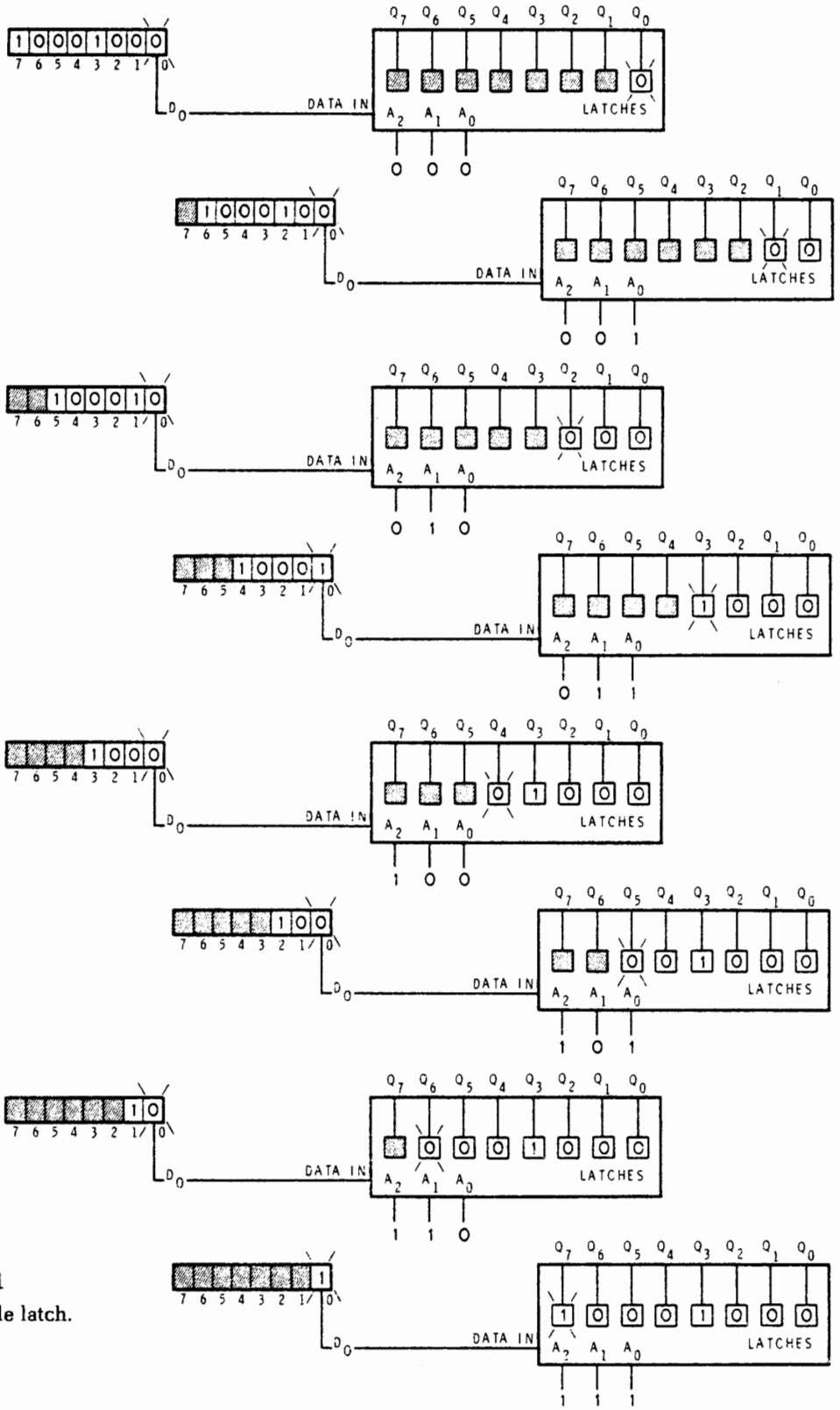



Figure 7-31
Loading the addressable latch.

While this arrangement results in an inexpensive circuit, it places an extra burden on the MPU. As in the previous example, the microprocessor must make the binary to 7-segment conversion. But in this case, it must also convert the parallel 7-segment code into a serial bit stream that is acceptable to the addressable latch. You are already familiar with the solution to the first problem. Now, the discussion will show how the MPU can convert parallel data to serial data.

The procedure is illustrated in Figure 7-31. The register on the left represents one of the accumulators. It contains the proper 7-segment code for displaying the letter A. This code must be transferred a bit at a time into the addressable latch shown on the right. Remember that the addressable latch actually contains eight latches and that the particular latch selected is determined by the 3-bit address at $A_0$ through $A_2$.

The first step is to store the contents of the accumulator at the basic latch address which is $C160_{16}$. If you convert this address to binary, you will find that address bits $A_0$, $A_1$, $A_2$ are all 0's. Since these lines connect to pins $A_0$, $A_1$, and $A_2$ on the addressable latch, latch 0 is selected as shown. Notice that the only data line used is $D_0$. Thus, the 0 in bit 0 of the accumulator is stored in latch 0.

Next, the contents of the accumulator are shifted to the right so that the next bit is available at $D_0$. The address of the latch is incremented by 1 and a store instruction is executed. Thus, the second bit is stored in latch 1.

This procedure continues as shown until all eight bits have been shifted from the accumulator to the addressable latch. Of course, a short program segment is required to control this operation. A typical program is shown in Figure 7-32. Step through the program and verify that it works. Remember that the 7-segment code must be in accumulator A before running the program.

The ET-3400 Microprocessor Trainer uses a method similar to this for driving the displays. The program shown in Figure 7-32 will light the left-most display. However, in the ET-3400 Microprocessor Trainer, the $D_0$ data line is inverted. Therefore, a 1 must be used in the accumulator for each segment that you wish to light. For example, the letter A is formed by placing $01110111_2$ in accumulator A before you run the program.

| HEX ADDRESS | HEX CONTENTS | MNEMONICS/ CONTENTS | COMMENTS |
|---|---|---|---|
| 0010 | CE | LDX # | Load the index register immediate |
| 0011 | C1 | C1 | with the address of the latch. |
| 0012 | 60 | 60 | |
| 0013 | A7 | STAA, X | Store accumulator A in the latch. |
| 0014 | 00 | 00 | |
| 0015 | 46 | RORA | Get ready to send next bit. |
| 0016 | 08 | INX | Set up next address. |
| 0017 | 8C | CPX# | Compare with final address. |
| 0018 | C1 | C1 | |
| 0019 | 67 | 67 | |
| 001A | 26 | BNE | If a match does not occur, branch back |
| 001B | F7 | F7 | to here. |
| 001C | 3E | Wait | Otherwise, wait. |

Figure 7-32
A simple program for loading the addressable latch.

# Multiplexing Displays

The previous approaches required one or more latches for each display. There is a method by which we can drive up to eight displays using only two 8-bit latches. However, it requires some additional components and a lot of microprocessor time.

A typical circuit is shown in Figure 7-33. Eight common cathode displays are used in this example. The display cannot light unless its associated transistor is turned on. The transistors are controlled by the contents of the digit select latch. In turn, this latch is loaded by the MPU. $Q_1$ is turned on by placing 1 in bit 7 of the digit latch; $Q_2$ is turned on by placing 1 in bit 6; etc. Generally, only one transistor at a time is turned on. A common procedure is to store 10000000 in the digit select latch and rotate the contents so that the 1 appears at each bit in turn.

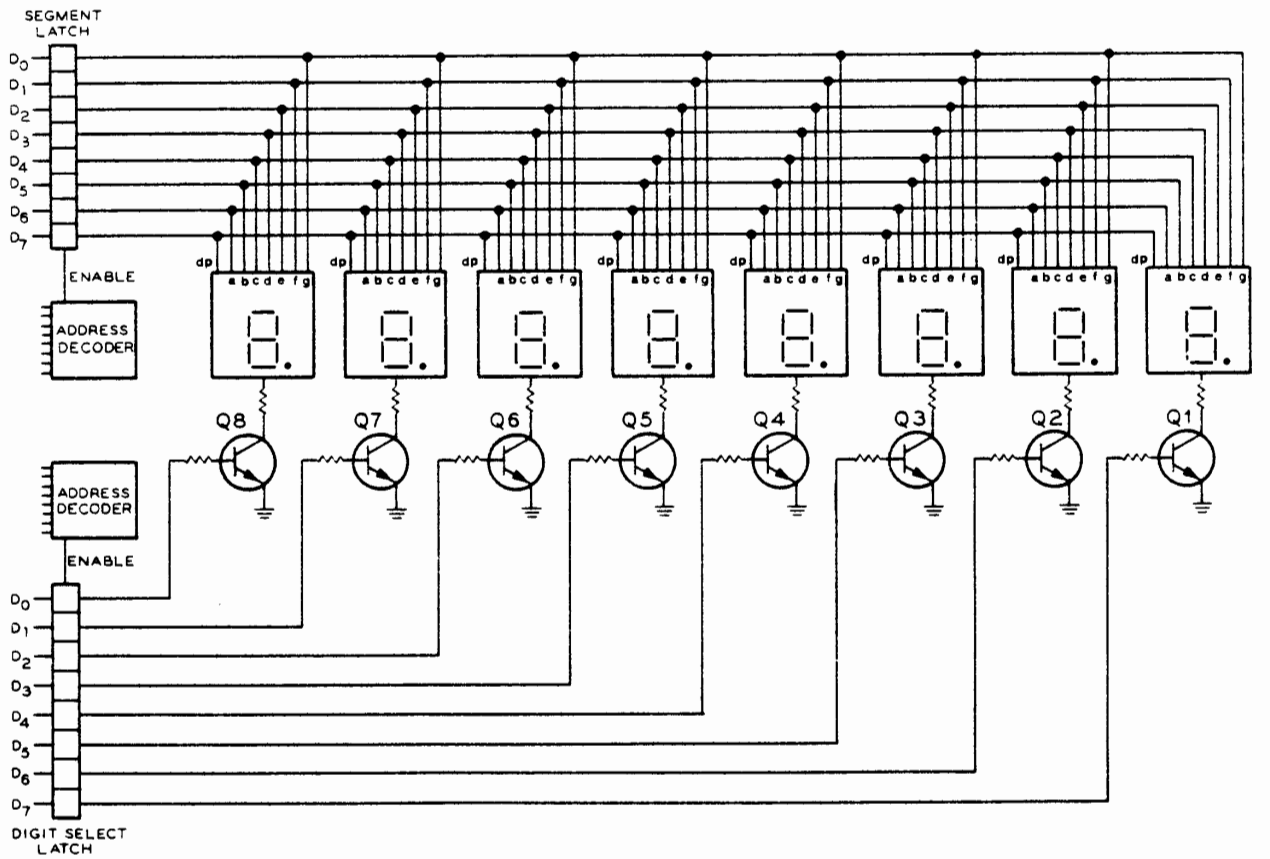



Figure 7-33
Multiplexing Displays.

The individual segments of each display are turned on by the segment latch. For example, to turn on the decimal point in the right-most display, the segment latch must contain a 1 at bit 7. At the same time, the digit select latch must turn on $Q_1$.

To display an 8-digit message, the procedure looks like this: Load the digit select latch with $10000000_2$. Load the segment latch with the 7-segment information for the right-most display. This causes the right-most display to show the first digit. After a brief delay, the contents of the digit display latch are changed to 01000000. This turns on the second display. Next the contents of the segment latch must be changed to the 7-segment code for the second display. This procedure continues until all eight displays are lit. Because no two displays are lit simultaneously, the displays must be refreshed many times each second to give the illusion of a constant steady display.

The advantage of this technique is that a minimum amount of hardware is required. However, it requires more MPU time than the previous techniques since the displays must be constantly refreshed.

## Self-Test Review

29. How do we turn on a common-anode type 7-segment display?

30. What is a disadvantage of using a decoder-driver, such as the 7447, to drive a display?

31. Why is a latch required between the MPU and the display?

32. When the latch drives the display directly, what performs the decoding function?

33. List four instructions that could be used to output data to a display.

34. Using the arrangement shown in Figure 7-26, what 8-bit code is required to form the letter P?

35. Refer to the program shown in Figure 7-27. If the number in accumulator A is $08_{16}$ when this program segment is run, what binary number will be outputted to the display?

36. What is the purpose of the program shown in Figure 7-27?

37. Refer to Figure 7-30. What determines which of the latches the input data bit goes into?

38. The program shown in Figure 7-32 converts _____ data to _____ data.

39. Refer to Figure 7-33. What determines which character is displayed?

40. Refer to Figure 7-33. What determines the display on which the character appears?

## Answers

29. A common-anode type display is turned on by applying logic 0 to the proper cathode.

30. The pattern is limited to those provided by the decoder.

31. Because the output data is stable for only an instant, a latch must be used to capture the data and hold it for the display.

32. The microprocessor.

33. Any of the store instructions could be used to output data to a display. These include: STAA, STAB, STX, and STS.

34. $0110\ 0111_2$.

35. $0111\ 1111_2$. This is the 7-segment pattern for the numeral 8.

36. The program converts binary numbers between 0000 and 1111 to a 7-segment code to form the appropriate hexadecimal numeral.

37. The 3-bit address at $A_0$ through $A_2$.

38. parallel, serial.

39. The number in the segment latch.

40. The number in the digit select latch.

## INTERFACING EXPERIMENTS

The hardware experiments are included in Unit 10 of this course. Go to Unit 10 and perform experiments 1 through 4. Some of these experiments are quite involved and you should not attempt more than one experiment per sitting.

# UNIT EXAMINATION

1. A 3-state logic gate:
   A. Has an enable/disable input as well as the normal data inputs.
   B. Is effectively disconnected from the circuit when it is disabled.
   C. Is generally used when two or more gates drive the same bus line.
   D. All the above.

2. Which of the following lines or buses is bi-directional?
   A. The read/write line.
   B. The address bus.
   C. The data bus.
   D. All the above.

3. Which of the following lines or buses is an input to the 6800 MPU?
   A. The address bus.
   B. The Ø2 clock.
   C. Valid Memory Address (VMA).
   D. Bus available (BA).

4. The MPU can be stopped by external hardware by:
   A. Forcing the $\overline{\text{halt}}$ line low.
   B. Forcing the $\overline{\text{halt}}$ line high.
   C. Forcing the BA line low.
   D. Forcing the BA line high.

5. RAM places data on the data bus when:
   A. The positive-going edge of the Ø2 clock occurs.
   B. The negative-going edge of the Ø2 clock occurs.
   C. The positive-going edge of the Ø1 clock occurs.
   D. The negative-going edge of the Ø1 clock occurs.

6. In 6800-based systems, two control lines are generally connected to the address decoder along with the address line. These are:
   A. BA and Ø1.
   B. VMA and Ø2.
   C. TSC and Ø2.
   D. NMI and Ø1.

7. What is the minimum number of 256 by 4 RAM IC's required to develop a memory of 8-bit words?
   A. One.
   B. Two.
   C. Four.
   D. Eight.

8. The output of the address decoder normally connects to:
   A. The R/$\overline{W}$ line on the RAM IC's.
   B. The $\overline{CE}$ line on the RAM IC's.
   C. The $\overline{HALT}$ line of the MPU.
   D. The $\overline{NMI}$ line of the MPU

9. What type of circuit is required between the MPU data lines and the 7-segment display?
   A. A latch.
   B. A decoder/driver.
   C. An address decoder.
   D. All of the above.

10. Refer to the program shown in Figure 7-27. The number in accumulator A when the program starts is used as:
   A. The address of the corresponding 7-segment code.
   B. The 7-segment code.
   C. The offset address for the LDAA, X instruction.
   D. The address of the 7-segment display.

11. The program shown in Figure 7-32:
   A. Converts the 7-segment code in accumulator A into a serial bit stream acceptable to the latch.
   B. Converts the 7-segment code in accumulator A into a parallel data byte acceptable to the addressable latch.
   C. Converts the binary number in accumulator A into the corresponding 7-segment code.
   D. Loads eight addressable latches with the proper 7-segment codes to display an 8-character message.

12. An advantage of the circuit shown in Figure 7-33 is:
   A. It requires fewer MPU instructions than the other methods of driving the display.
   B. It requires less MPU time than the other methods of driving the displays.
   C. It requires less external circuitry than the other methods.
   D. All of the above.