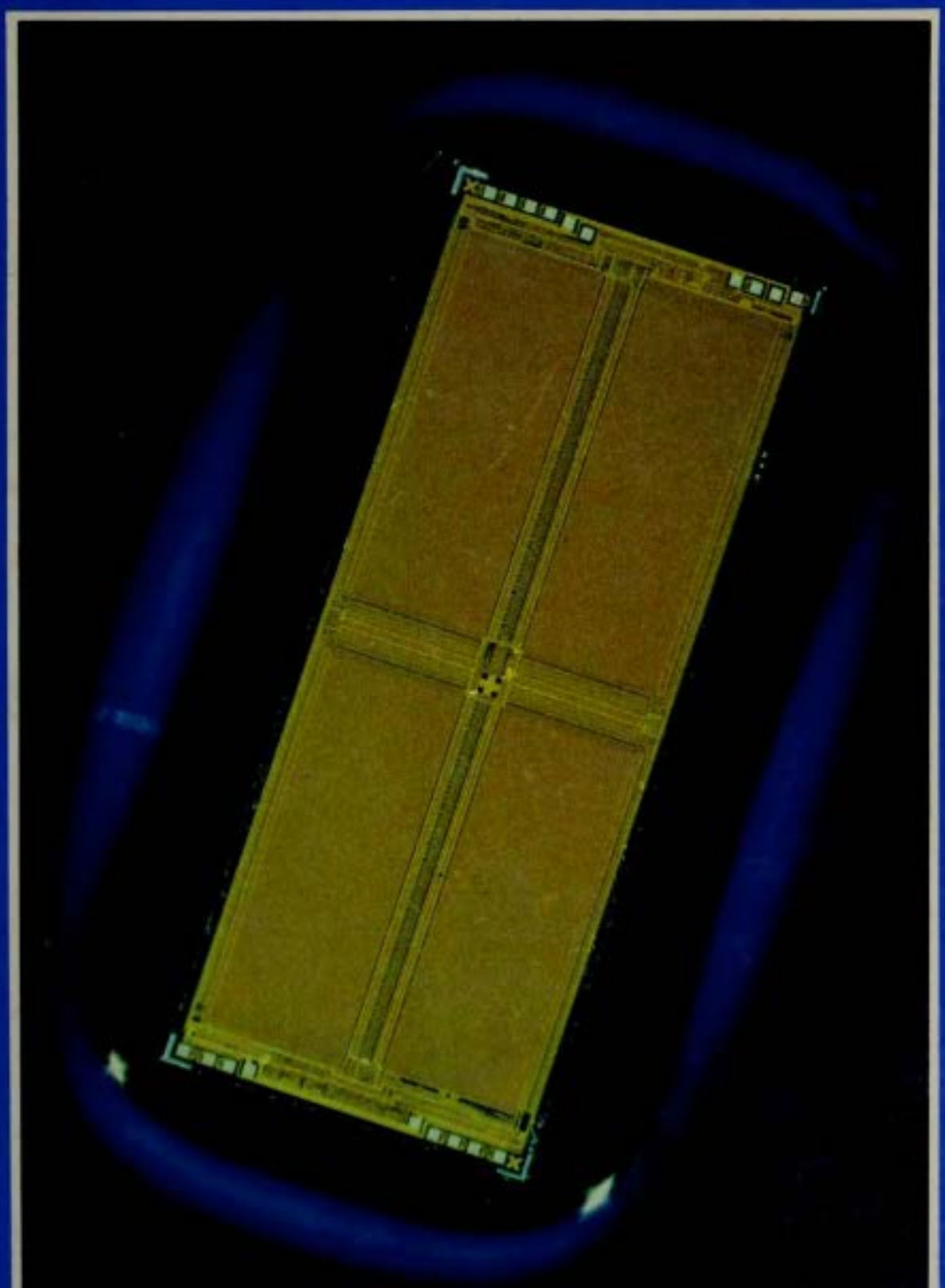
BREY

MICROPROCESSOR/ HARDWARE INTERFACING AND APPLICATIONS



4

The MC6800, MC6809, and MC68000 Microprocessors

This chapter introduces the current line of Motorola microprocessors. It is important to learn the operation and interfacing of either the MC6800 or MC6809 microprocessor first, since the remainder of this text emphasizes these microprocessors. Once they are learned, transition to the MC68000 or any other microprocessor manufactured by any of the IC houses is easy. For example, the 8085A is also covered in some detail.

Whichever microprocessor you choose to study, you will find subsequent chapters interesting and useful.

4-1 PINOUTS

Figure 4–1 illustrates the MC6800, MC6809, and MC68000 microprocessors' pinouts. The MC6800 and MC6809 are both packaged in 40-pin dual in-line packages, while the MC68000 is integrated into a 64-pin dual in-line package. All three devices operate from a single 5V power supply with power dissipations of less than 1.5 W.

Output Loading

The MC6800 and MC6809 microprocessors are capable of providing 1.6 mA of sink current and 400 μA of source current at any of the output pins. This current will drive one 74XXX TTL unit load, four 74LSXXX TTL unit loads, or about ten NMOS or CMOS unit loads. Only ten NMOS or CMOS unit loads may be connected because each MOS input places a fairly large amount of capacitance on an output connection. Too much bus capacitance will degrade the timing signals issued by the microprocessor, causing performance problems. To prevent this, MOS loads are limited to ten or less. Refer to table 3–1 in chapter 3 for a detailed look at unit loading.

The MC68000 is capable of sinking 1.6 mA on the HALT pin, 3.2 mA on the address pins, and about 5.0 mA on the data bus connections. With this device, more TTL components can be driven directly from the output pins of the microprocessor. This means that a larger system may be connected directly to the MC68000 without the addition of external bus buffers. Since the source current at these outputs remains at 400 μA, the maximum number of MOS loads remains at ten or less.

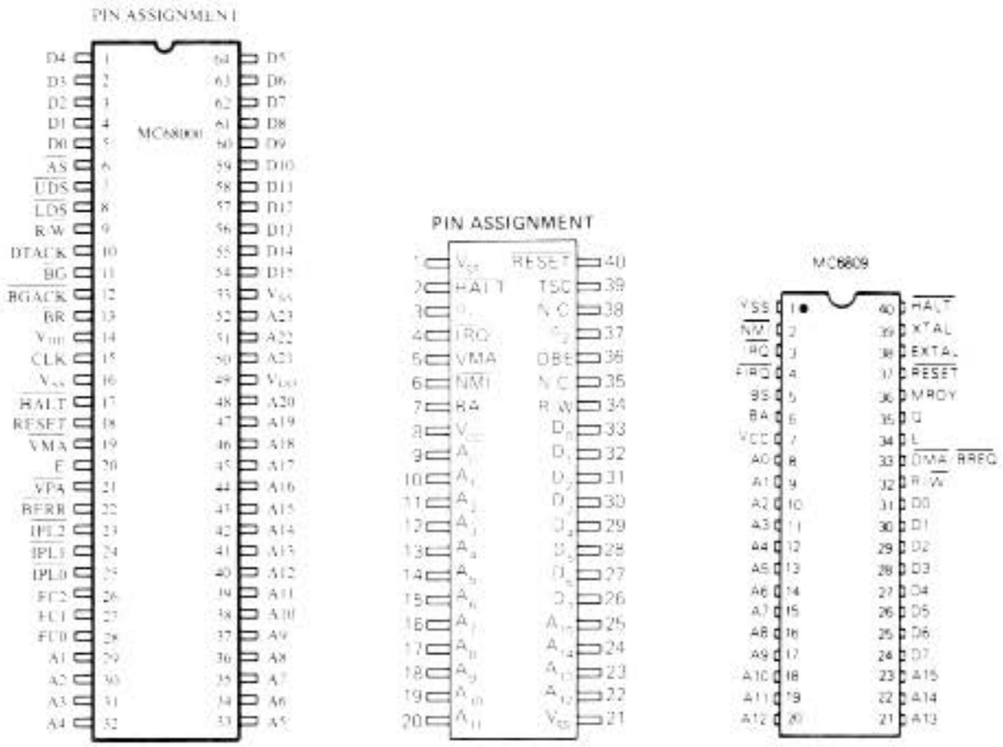


FIGURE 4–1 Pin diagrams or pinouts of the Motorola MC6800, MC6809, and MC68000 microprocessors.

SOURCE: Courtesy of Motorola, Inc.

It is interesting to note that the MC6800 and MC6809 microprocessors will not drive a 74SXXX series TTL load. This limitation can be overcome by using a 74ASXXX series gate or a FAST gate from Fairchild.

Input Loading

Input connections on all three microprocessors sink and source a maximum of 2.5 μA of current and present about 10 pF of capacitance. In addition to low loading, they are also compatible with the standard TTL voltage levels.

Noise Immunity

The system noise immunity in any of the Motorola processors is about 400 mV and is directly compatible with standard TTL noise immunities. In systems that contain heavy capacitive loads, long bus connections, or excessive current loads, bus buffers at the output connections are recommended. With additional buffering, it is possible to connect up to 100 MOS or 74LSXXX TTL unit loads to an output connection. Most buffers contain an enhanced pullup network that has been designed to drive capacitive loads.

CLOCK CIRCUITRY 4-2

The MC6809 contains an internal clock generator that, in most cases, generates the basic timing for the microprocessor. The MC6800 and the MC68000 both require the addition of an external clock generator to provide their basic timing.

MC6809 Clock Circuitry

Under normal operation, a crystal with a frequency of 8 MHz would be attached between the EXTAL and XTAL input pins of the MC6809 (as pictured in figure 4–2). The crystal is internally divided by a factor of four to produce the 2 MHz basic operating frequency. The range of allowable crystal frequencies is between 8.0 MHz and about 400 kHz for reliable operation. If

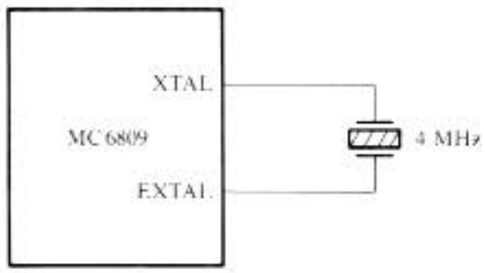


FIGURE 4–2 The MC6809 clock generation circuitry.

an operating frequency outside this range is chosen, Motorola will not guarantee the proper operation of the MC6809.

In addition to a crystal, the MC6809 may be driven from an external TTL source. This is accomplished by grounding the XTAL pin and connecting the external TTL clock signal to the EXTAL connection. This method of operation is used in multiple processor systems, where one timing source drives all of the processors.

MC6800 and MC68000 Clock Circuitry

The MC6800 and MC68000 require an external clock generator for proper operation. The MC6875 clock generator, as illustrated in figure 4–3, can generate the required multiphase clock inputs for the MC6800. The MC68000 requires a TTL compatible clock input of up to 8.0 MHz for proper operation. A circuit that can be used to generate this clock is illustrated in figure 4–4.

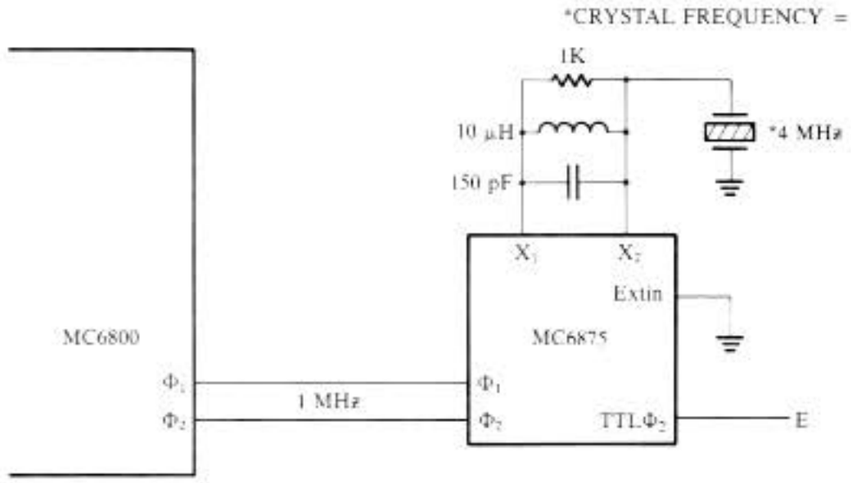
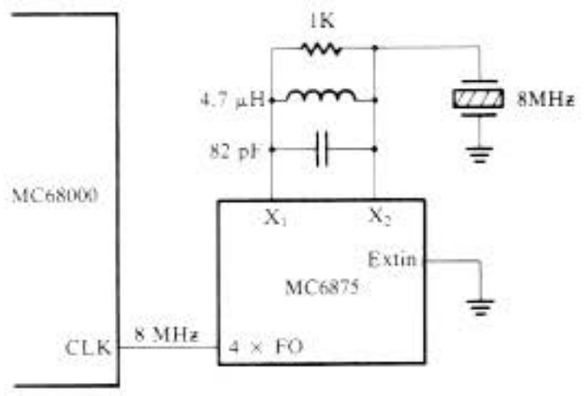


FIGURE 4–3 The MC6875 clock generator connected to the MC6800 microprocessor.

FIGURE 4–4 The MC68000 clock generation circuitry.



ADDRESS AND DATA BUS CONNECTIONS 4-3

The Address Bus

Both the MC6800 and the MC6809 contain 16 pins that have been dedicated to addressing the memory and I/O. This feature allows either of these microprocessors to address 64K bytes of memory and I/O space directly.

The MC68000 contains 23 address connections, which allow it to access an astounding 16M bytes of memory and I/O directly. This is equal to eight million 16-bit words of memory information. In addition to the number of address connections present, the amount of drive current available is triple that of the MC6800 or MC6809.

The Data Bus

The data bus of the MC6800 and MC6809 microprocessors is 8 bits in width; whereas the MC68000 uses a 16-bit data bus. This bus, in all three cases, is a bidirectional, three-state bus that passes information out of, or into, the microprocessor.

As with the address bus, the data bus on the MC68000 possesses an enhanced drive capability. This capability allows the microprocessor to be structured into a larger system before bus buffering is required.

MC68000 Bus Buffering

Figure 4–5 illustrates the inclusion of a set of data and address bus buffers for the MC68000 microprocessor. The $\overline{AS}$, or address strobe, output is connected to the enable ($\overline{G}$) input on the address buffers. The $\overline{AS}$ signal becomes a logic zero whenever the address bus contains a valid memory address. In this circuit, $\overline{AS}$ switches the three-state buffers to their *enabled*, or on, condition.

The bidirectional bus transceivers, which are connected to the data bus, are controlled by the R/$\overline{W}$ control signal. Since the R/$\overline{W}$ signal selects the direction of data flow on the data bus, it is usable as a directional control input to the transceivers. During a memory or I/O read, R/$\overline{W}$ is high and causes the data to flow in from the data bus; during a memory or I/O write, this line is low and causes the data to flow out to the memory and I/O.

CONTROL BUS CONNECTIONS 4-4

The control bus structures of the MC6800, MC6809, and the MC68000 are almost identical when only the major control signals are examined. If the MC6800 and MC6809 are compared, they differ only in the way that direct memory access I/O is controlled. Comparing all three demonstrates many more differences. Table 4–1 contrasts these differences.

The Basic Memory and I/O Control Signals

All three microprocessors use the R/$\overline{W}$ signal to command the memory or I/O to read or write data. In the MC68000 this signal also works in conjunction

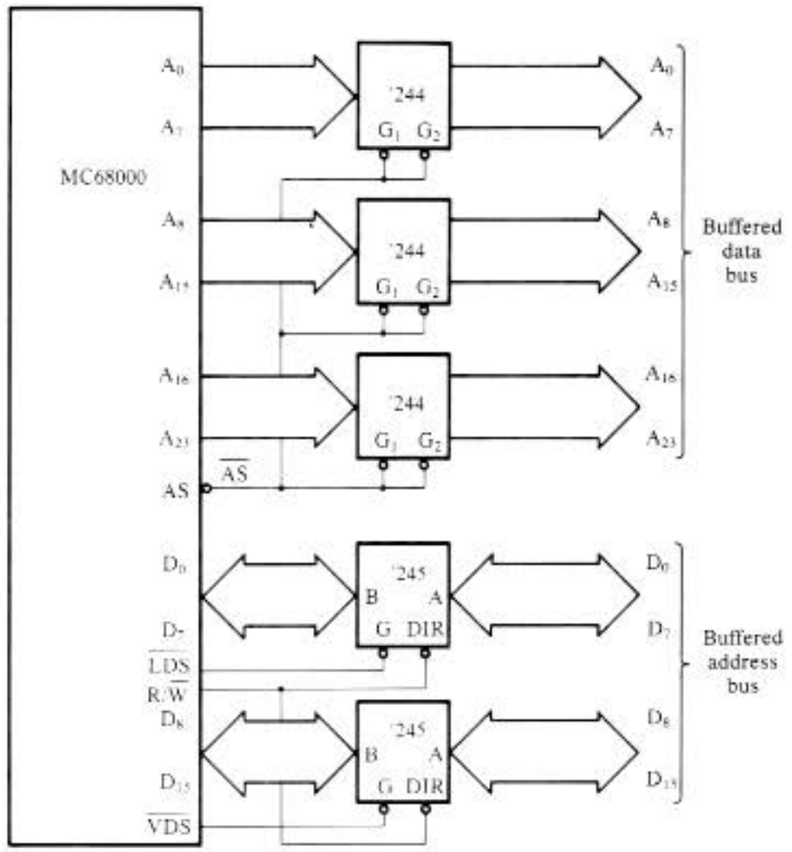


FIGURE 4–5 A fully buffered MC68000 microprocessor.

with the $\overline{LDS}$ and $\overline{UDS}$ data strobes, which indicate how the microprocessor will react with the data bus during the current bus cycle. Table 4–2 illustrates how the MC68000 interprets the data bus for each combination of the R/$\overline{W}$, $\overline{LDS}$, and $\overline{UDS}$ signals.

In addition to these three control signals, the $\overline{VMA}$ and $\overline{E}$ signals are also present on all three microprocessors. The $\overline{VMA}$ signal indicates that the address bus contains a "valid memory address"; the $\overline{E}$ signal, or *enable*, is used to enable the memory or I/O device. The E signal is actually not present on the MC6800, but it is the phase two TTL output of the clock generator circuitry.

Additional MC6809 Control Signals

The MC6809 has a few other control signals that are not present on the MC6800. These include BS, $\overline{FIRQ}$, MRDY, and DMA/$\overline{BREQ}$. The BA and BS signals indicate the present state of the MC6809, as illustrated in table 4–3.

The MRDY signal extends the access time provided for the memory by extending the current read or write cycle. The extension may be anything

TABLE 4–1 Comparative control signals.

MC6800/MC6809	MC68000	Function
R/$\overline{W}$	R/$\overline{W}$	Controls reading or writing to memory or I/O
VMA	$\overline{AS}$, $\overline{VMA}$	Valid memory address present on the address bus
62,E	$\overline{UDS}$, $\overline{LDS}$, $\overline{E}$	Enables data bus so that no contention can occur
$\overline{NMI}$	LEVEL 7	Nonmaskable interrupt
$\overline{IRQ}$, $\overline{FIRQ}$	LEVEL 0–6 TRAPS	Maskable interrupts
$\overline{MRDY}$	$\overline{DTACK}$	Slow memory control
DMA/$\overline{BREQ}$, BA BS, DBE, TSC	$\overline{BR}$, $\overline{BG}$ BGACK	DMA control and bus arbitration

TABLE 4–2 MC68000 bus control strobes.

$\overline{UDS}$	$\overline{LDS}$	R/$\overline{W}$	Function
0	0	0	Valid data bus
0	0	1	Valid data bus
0	1	0	Bits 8–15 appear on both halves of bus
0	1	1	Bits 8–15 appear on upper half of bus. Lower half contains unknown information
1	0	0	Bits 0–7 appear on both halves of bus
1	0	1	Bits 0–7 appear on lower half of bus. Upper half contains unknown information
1	1	X	Data bus contains unknown information

from one clocking period up to 10 μs in duration. This is most useful if a slower external device is to be interfaced to the MC6809. An example is an analog-to-digital converter, as illustrated for the 8085A in chapter 6.

The DMA/$\overline{BREQ}$ input is used for a direct memory access or bus arbitration; it is covered, along with the $\overline{FIRQ}$ input, later in this chapter.

Additional MC68000 Control Signals

The advanced architecture of the MC68000 microprocessor includes some additional control pins, such as $\overline{BGACK}$, $\overline{VPA}$, $\overline{DTACK}$, $\overline{BERR}$, $\overline{BR}$, $\overline{BG}$.

TABLE 4–3 BA and BS processor state signals.

BA	BS	Processor State
0	0	Normal
0	1	Interrupt or reset acknowledge
1	0	Sync acknowledge
1	1	Halt/bus grant acknowledge

$\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$, FC0, FC1, and FC2. These pins control such features as interrupts and direct memory access or bus arbitration.

The FC0, FC1, and FC2 signals indicate the status of the MC68000 as depicted in table 4–4.

TABLE 4–4 MC68000 bus function control signals.

FC2	FC1	FC0	Cycle Type
0	0	0	Undefined
0	0	1	User data
0	1	0	User program
0	1	1	Undefined
1	0	0	Undefined
1	0	1	Supervisor data
1	1	0	Supervisor program
1	1	1	Interrupt acknowledge

FC0, FC1, and FC2 are basically used to indicate the mode of operation, either supervisor or user. In the supervisor state, the MC68000 can control an external memory management device and system software. This capacity provides security, since the memory management unit and system software cannot be accessed by the normal user. The access requires a shift to the supervisor state, which is a privileged state.

The $\overline{BERR}$ input signal informs the processor of a bus error and is provided by the external hardware. The type of hardware most likely to generate this signal is a memory parity checking circuit. If a parity error is detected, this signal becomes active, and the processor executes an exception sequence or an interrupt. This sequence reads the address of the user supplied bus error handling subroutine from memory address $00008. Control is then transferred to this error-handling subroutine for a possible repeat of the bus cycle.

The $\overline{BR}$, $\overline{BG}$, and $\overline{BGACK}$ signals are used when more than one MC68000 or a DMA controller is connected in a system. $\overline{BR}$ is an input that requests the use of the bus. $\overline{BG}$ is an output that indicates that the MC68000 will release bus control at the end of the current cycle. The $\overline{BGACK}$ input indicates that some other device has become the bus master. These signals are discussed in greater detail in the section on bus arbitration.

The $\overline{VPA}$ input is activated whenever an external MC6800 peripheral device is addressed. This is provided so that the wealth of MC6800 8-bit peripheral devices can function with the MC68000. It also signals the processor to use automatic vectoring for an interrupt, as described in the interrupt section of this chapter.

4-5 RESET OR RESTART

If the MC6800 or MC6809 microprocessors are reset, they look at memory location $FFFE for the restart vector. The restart vector holds the starting address of the system program.

Resetting or restarting the MC68000 is completely different because two vectors apply to this function. When the MC68000 is first powered up, locations zero through three must contain the supervisor stack pointer (SSP). Locations four through seven must contain the location of the first instruction to be executed after a reset. These vectors are used only during a power up sequence.

The RESET instruction in the MC68000 will not cause the reset vectors to be called. This instruction will only cause the $\overline{RESET}$ output pin to become active for 124 clocking periods after it has been executed. This instruction and the resulting signal on the $\overline{RESET}$ pin are only used for reinitializing the external peripheral components in the system. It has absolutely no effect on the internal registers of the MC68000.

If repowering the processor is desirable, it can be accomplished by using the reset vectors stored in the vector table.

4-6 BUS TIMING

The standard operating frequency for the MC6800 and the MC6809 is 1 MHz. At this rate they are capable of transferring 1 byte of information per clocking period or 1 byte every microsecond. The MC68000 works with an internal clock frequency of 8 MHz and can transfer 1 byte of data every 500 μs since the internal timing is set up so that four external clock pulses are required for a bus transfer.

MC6800 Read and Write Timing

Figure 4–6 illustrates the basic read and write timing diagrams of the MC6800 microprocessor and its AC characteristics. In the MC6800 timing diagrams, the address is presented to memory and I/O during the logic zero portion of the phase two clock. When the phase two clock becomes a logic one, data is transferred into the processor or sent out from it.

The time allowed for a memory access (Tacc) is equal to 540 ns worst case. In other words, the memory, plus the time delay introduced by buffers, should have an access time of no longer than 540 ns. In addition to this time constraint, it is also important to note that data must be held for 10 ns minimum after the phase two clock returns to the logic zero level. If the phase two clock is used as an enable (or E) signal, the amount of time required to enable the memory device must not exceed 350 ns. Since the output buffers in a memory device typically take 120 ns to enable, this is generally ample time.

MC6800 Memory Read and Write Signals

The circuit depicted in figure 4–7 allows the MC6800 or MC6809 to be used with most of this text. It also allows it to be used, without effort, with most of the industrywide standard memory components, such as the 2114 RAM, 2716 EPROM, and others.

By combining the phase two TTL signal or E signal with the VMA output and the R/$\overline{W}$ signal, we obtain the $\overline{MEMR}$ or $\overline{RD}$ and $\overline{MEMW}$ or WR control signals that are used throughout this book. These pulses are approxi-

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	-0.3 to +7.0	Vdc
Input Voltage	V_{in}	-0.3 to +7.0	Vdc
Operating Temperature Range—T_L to T_H MC6800, MC68A00, MC68B00 MC6800C, MC68A00C MC6800BCCS, MC6800CCCS	T_A	 0 to +70 -40 to +85 -55 to +125	°C
Storage Temperature Range	T_{stg}	-55 to +150	°C
Thermal Resistance Plastic Package Ceramic Package	θ_{JA}	 70 50	°C/W

ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0 V ± 5%, V_{SS} = 0, T_A = T_L to T_H unless otherwise noted)

Characteristic		Symbol	Min	Typ	Max	Unit
Input High Voltage	Logic ϕ1, ϕ2	V_{IH} V_{IHC}	V_{SS} + 2.0 V_{CC} - 0.6	–	V_{CC} V_{CC} + 0.3	Vdc
Input Low Voltage	Logic ϕ1, ϕ2	V_{IL} V_{ILC}	V_{SS} - 0.3 V_{SS} - 0.3	– –	V_{SS} + 0.8 V_{SS} + 0.4	Vdc
Input Leakage Current (V_{in} = 0 to 5.25 V, V_{CC} = max) (V_{in} = 0 to 5.25 V, V_{CC} = 0.0 V)	Logic* ϕ1, ϕ2	I_{in}		1.0	2.5 100	μAdc
Three-State (Off State) Input Current (V_{in} = 0.4 to 2.4 V, V_{CC} = max)	D0 - D7 A0 - A15, R/W	I_{TSI}	–	2.0 –	10 100	μAdc
Output High Voltage (I_{Load} = -205 μAdc, V_{CC} = min) (I_{Load} = -145 μAdc, V_{CC} = min) (I_{Load} = -100 μAdc, V_{CC} = min)	D0 - D7 A0 - A15, R/W, VMA BA	V_{OH}	V_{SS} + 2.4 V_{SS} + 2.4 V_{SS} + 2.4	– – –	– – –	Vdc
Output Low Voltage (I_{Load} = 1.6 mAdc, V_{CC} = min)		V_{OL}	–	–	V_{SS} + 0.4	Vdc
Power Dissipation		P_D	–	0.5	1.0	W
Capacitance (V_{in} = 0, T_A = 25°C, f = 1.0 MHz) ϕ1 ϕ2 D0 - D7 Logic Inputs A0 - A15, R/W, VMA		C_{in} C_{out}	 –	 25 45 10 6.5 –	 35 70 12.5 10 12	pF pF

CLOCK TIMING (V_{CC} = 5.0 V ± 5%, V_{SS} = 0, T_A = T_L to T_H unless otherwise noted)

Characteristics		Symbol	Min	Typ	Max	Unit
Frequency of Operation	MC6800 MC68A00 MC68B00	f	0.1 0.1 0.1	–	1.0 1.5 2.0	MHz
Cycle Time (Figure 1)	MC6800 MC68A00 MC68B00	t_{cyc}	1.000 0.666 0.500	– – –	10 10 10	μs
Clock Pulse Width (Measured at V_{CC} - 0.6 V)	ϕ1, ϕ2 MC6800 ϕ1, ϕ2 MC68A00 ϕ1, ϕ2 MC68B00	$PW_{\phi H}$	400 230 180	– – –	9500 9500 9500	ns
Total ϕ1 and ϕ2 Up Time	MC6800 MC68A00 MC68B00	t_{ut}	900 600 440	– – –	– – –	ns
Rise and Fall Times (Measured between V_{SS} + 0.4 and V_{CC} - 0.6)		t_{cr}, t_{cf}	–	–	100	ns
Delay Time or Clock Separation (Figure 1) (Measured at V_{DV} = V_{SS} + 0.6 V @ t_{cr} or t_{cf} ≤ 100 ns) (Measured at V_{DV} = V_{SS} + 1.0 V @ t_{cr} or t_{cf} ≤ 35 ns)		t_d	0 0	–	9100 9100	ns

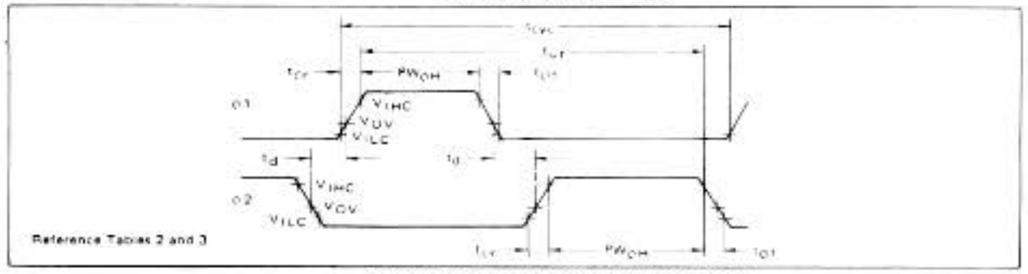
FIGURE 4–6 The read and write timing diagrams and charcteristics of the MC6800 microprocessor.

SOURCE: Courtesy of Motorola, Inc.

READ/WRITE TIMING (Reference Figures 2 through 6)

Characteristic	Symbol	MC6800 Min	MC6800 Typ	MC6800 Max	MC68A00 Min	MC68A00 Typ	MC68A00 Max	MC68B00 Min	MC68B00 Typ	MC68B00 Max	Unit
Address Delay C = 90 pF C = 30 pF	t_{AD}	 –	 –	 270	 –	 –	 180	 –	 –	 150	ns
		–	–	250	–	–	165	–	–	135	
Peripheral Read Access Time $t_{acc} = t_{cyc} - (t_{AD} + t_{DSR})$	t_{acc}	–	–	530	–	–	360	–	–	250	ns
Data Setup Time (Read)	t_{DSR}	100	–	–	60	–	–	40	–	–	ns
Input Data Hold Time	t_H	10	–	–	10	–	–	10	–	–	ns
Output Data Hold Time	t_H	10	25	–	10	25	–	10	25	–	ns
Address Hold Time (Address, R/W, VMA)	t_{AH}	30	50	–	30	50	–	30	50	–	ns
Enable High Time for DBE Input	t_{EH}	450	–	–	280	–	–	220		–	ns
Data Delay Time (Write)	t_{DDW}	–	–	225	–	–	200	–		160	ns
Processor Controls Processor Control Setup Time Processor Control Rise and Fall Time Bus Available Delay Three-State Delay Data Bus Enable Down Time During ϕ1 Up Time Data Bus Enable Rise and Fall Times	 t_{PCS} t_{PCr}, t_{PCf} t_{BA} t_{TSD} t_{DBE} t_{DBEr}, t_{DBEf}	 200 – – – 150 –	 – – – – – –	 – 100 250 270 – 25	 140 – – – 120 –	 – – – – – –	 – 100 165 270 – 25	 110 – – – 75 –	 – – – – – –	 – 100 135 220 – 25	 ns ns ns ns ns ns

CLOCK TIMING WAVEFORM



READ DATA FROM MEMORY OR PERIPHERALS

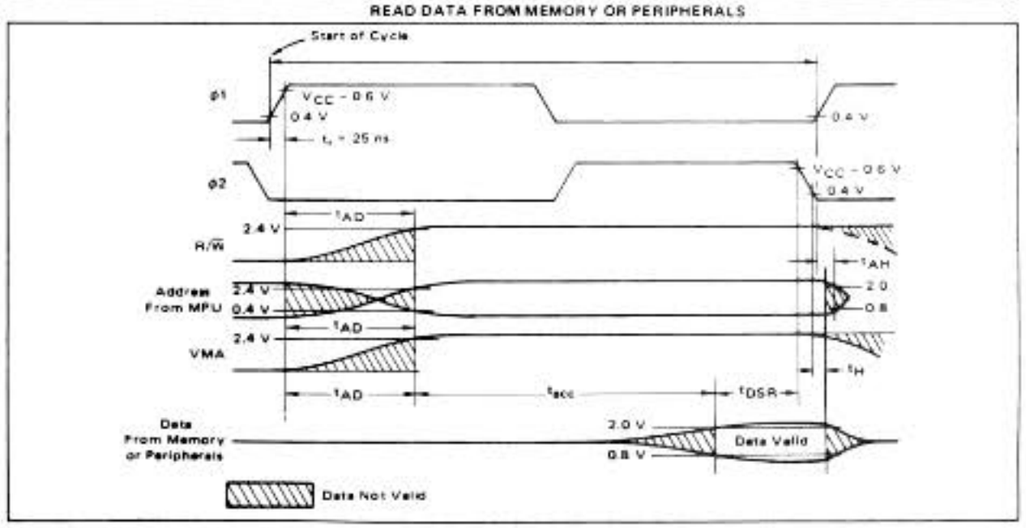
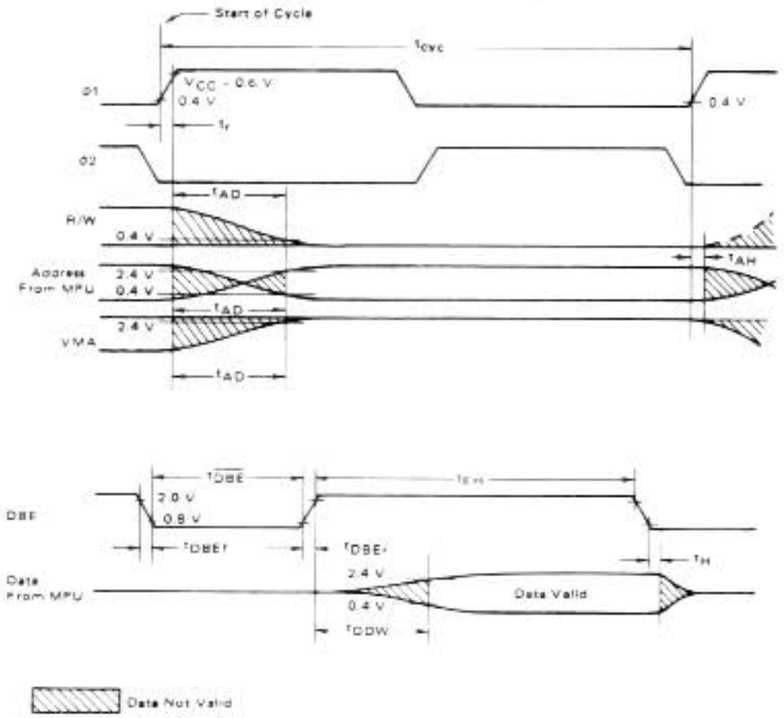
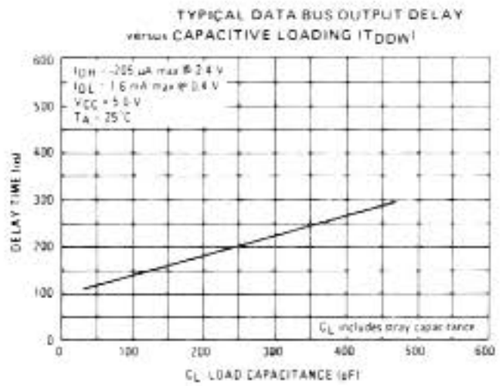


FIGURE 4–6 *continued*

WRITE IN MEMORY OR PERIPHERALS





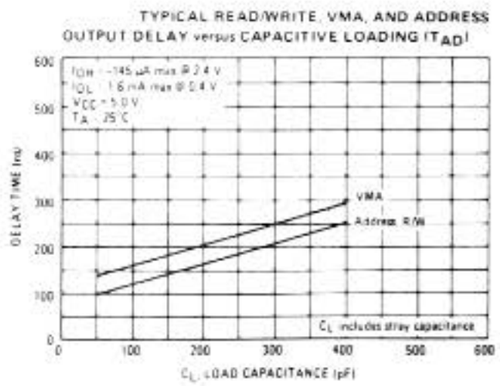
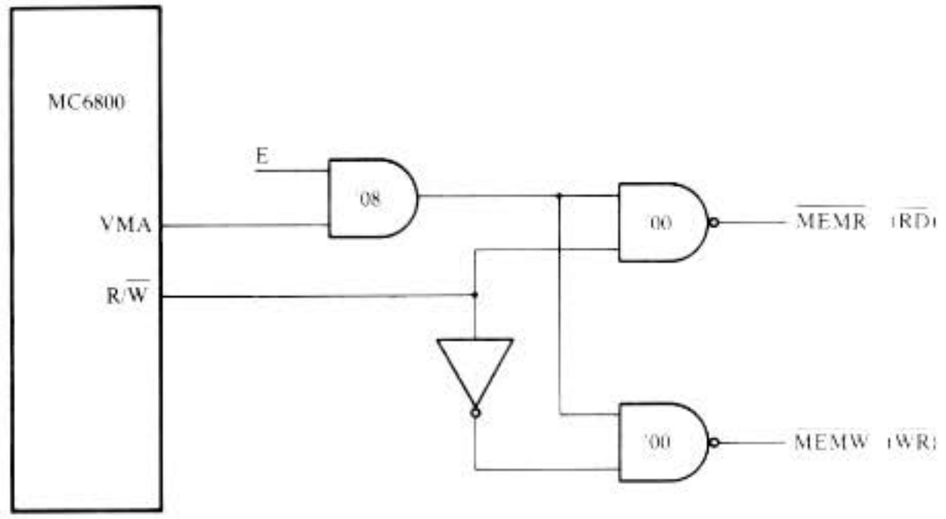


FIGURE 4–6 *continued*



* E is the phase 2 TTL CLOCK

FIGURE 4–7 Using the MC6800 to generate the $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ control signals.

mately 500 ns in width and are compatible with many standard memory components. Since the 680XX series microprocessors do not support isolated I/O, no attempt has been made to develop the I/O control signal $\overline{\text{IOR}}$ and $\overline{\text{IOW}}$. For I/O control and its application, refer to the section in chapter 6 on memory mapped I/O.

MC68000 Read and Write Timing

Figure 4–8 illustrates the timing diagrams for the MC68000 microprocessor. The MC68000 will transfer one word, or 16 bits, of information every 500 ns, since it operates at a basic clock frequency of 8 MHz. The amount of time allowed to the memory component attached to the MC68000 is approximately 300 ns. This means that higher-speed memory components must be selected for use with this processor.

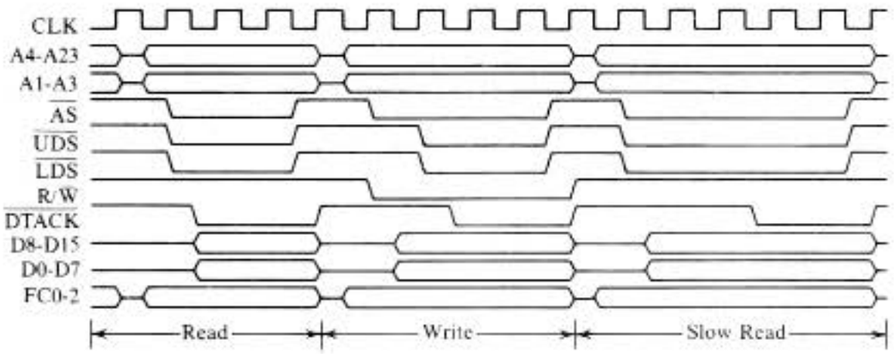


FIGURE 4–8 Basic read and write timing for the MC68000 microprocessor.

SOURCE: Courtesy of Motorola, Inc.

The $\overline{AS}$, or address strobe, signal activates a memory component. It is normal to use this signal to supply the MC68000 with its $\overline{DTACK}$ signal in systems that contain memory that can access data within 350 ns.

The MC6809 MRDY and the MC68000 $\overline{DTACK}$

The MRDY connection on the MC6809 prolongs the processor bus cycle for low-speed memory or I/O devices. This input can be held at a logic zero level for up to 10 μs for these slower devices. If held longer than 10 μs, Motorola will not guarantee the validity of the data stored in the MC6809 internal register array.

The $\overline{DTACK}$ input, or *data acknowledge*, of the MC68000 can serve about the same purpose as the MRDY input of the MC6809. The difference is that the MRDY input is an optional feature that can be ignored by connecting it to a logic one, while the $\overline{DTACK}$ input must be used.

During a read operation, for example, the MC68000 sends out the control signals and waits for the external device (usually memory) to send the $\overline{DTACK}$ signal back to the microprocessor. In fact, if the $\overline{DTACK}$ signal does not occur, the system waits just as it does with MRDY. Once the processor accepts the information, the $\overline{DTACK}$ signal must be returned to its inactive state before another bus cycle can occur. Without this timing, the MC68000 will not function.

4-7 MC6809 AND MC68000 BUS ARBITRATION (DMA)

MC6809 Bus Arbitration

The MC6809 microprocessor has an input labeled $\overline{DMA}/\overline{BREQ}$ that requests access to the MC6809 system bus. When this pin is active, the microprocessor releases control of the system bus by three-stating the address, data, and control buses. This allows an external device to access the memory and I/O connected to the MC6809 directly.

The BA and BS signals grant or acknowledge the bus request when they are both at logic one levels. This same level also indicates that the microprocessor may be halted.

MC68000 Bus Arbitration

If more than one microprocessor or similar device is to function on the same bus system, a need for bus arbitration arises. The set of connections described in this segment determines which device will control the bus so that no conflict can occur. Bus conflicts will almost always result in a loss of data if they are allowed to occur.

The $\overline{BR}$, or *bus request* signal, is an input to the MC68000 that asks for or requests the system bus. If the MC68000 is at the end of its current bus cycle, it will grant the bus request by sending out the $\overline{BG}$, or *bus grant* signal. Once the requesting device notices the $\overline{BG}$ signal, it returns a $\overline{BGACK}$, or *bus grant acknowledge* signal, back to the MC68000 to indicate that it has taken over the system buses.

This arbitration dialog is normally carried out between the MC68000 and an external DMA controller. During the bus grant, the MC68000 relinquishes control of the system by floating the address, data, and control bus. This of course will allow the external device to gain complete control over the system buses. The typical three wire handshake is illustrated in the timing diagram of figure 4–9.

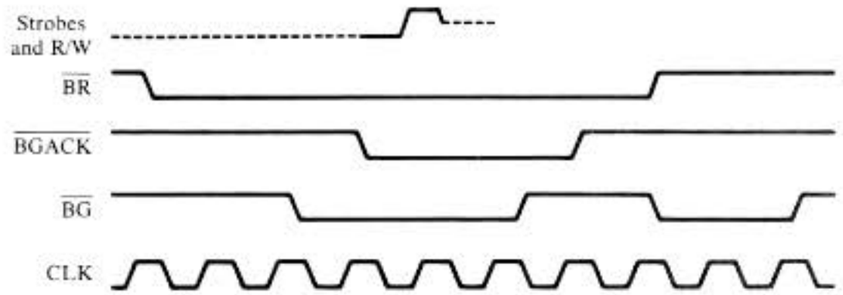


FIGURE 4–9 MC68000 bus arbitration timing.
SOURCE: Courtesy of Motorola, Inc.

MC6800 Interrupt Structure

The MC6800 microprocessor has two hardware and one software interrupt that are vectored through the top part of the memory. Table 4–5 illustrates the interrupt vectors for the MC6800 microprocessor. These vectors contain the location of the software that will be executed in response to these input signals.

TABLE 4–5 MC6800 interrupt vectors.

Vector Location	Signal
$FFFE, $FFFF	Reset
$FFFC, $FFFD	$\overline{NMI}$
$FFFA, $FFFB	SWI
$FFF8, $FFF9	$\overline{IRQ}$

MC6809 Interrupt Structure

The MC6809 microprocessor has three hardware interrupts and three software interrupts that are vectored through the top portion of the memory. A new hardware interrupt, labeled $\overline{FIRQ}$, has been added to the $\overline{IRQ}$ and $\overline{NMI}$ inputs of the MC6800. The only difference between the new interrupt and the two old interrupts is that the $\overline{FIRQ}$ input will only store the program counter and the status register on the stack. The $\overline{IRQ}$ and $\overline{NMI}$ inputs place all of the internal registers, except the hardware stack pointer, on the stack.

Table 4–6 illustrates the vector locations for the interrupt inputs to the MC6809.

MC68000 Interrupt Structure

The interrupt structure for the MC68000 is quite different from the structure for the MC6800 and MC6809. A complete listing of the many different interrupts appears in table 4–7.

TABLE 4–6 MC6809 interrupt vectors.

Vector Location	Signal
$FFFE, $FFFF	Reset
$FFFC, $FFFD	$\overline{\text{NMI}}$
$FFFA, $FFFB	SWI
$FFF8, $FFF9	$\overline{\text{IRQ}}$
$FFF6, $FFF7	$\overline{\text{FIRQ}}$
$FFF4, $FFF5	SWI2
$FFF2, $FFF3	SWI3
$FFF0, $FFF1	Reserved

TABLE 4–7 MC68000 interrupt vectors.

Vector Number	Address	Assignment
0	00000	Reset initial SSP
	00004	Reset initial PC
2	00008	Bus error
3	0000C	Address error
4	00010	Illegal instruction
5	00014	Divide by zero
6	00018	CHK instruction
7	0001C	TRAPV instruction
8	00020	Privilege violation
9	00024	Trace
10	00028	Line 1010 emulator
11	0002C	Line 1111 emulator
12–23	00030–0005F	Reserved by Motorola
24	00060	Spurious interrupt
25	00064	Level 1 interrupt
26	00068	Level 2 interrupt
27	0006C	Level 3 interrupt
28	00070	Level 4 interrupt
29	00074	Level 5 interrupt
30	00078	Level 6 interrupt
31	0007C	Level 7 interrupt
32–47	00080–000BF	TRAP instruction vectors
48–63	000C0–000FF	Reserved by Motorola
64–255	00100–003FF	USER interrupt vectors

This vector table occupies the first 1024 bytes of memory or first 512 words of memory. Seven of these vectors are used for external interrupts; the remaining vectors are used for reset, for various Motorola system functions, and for TRAPS.

TRAPS are used by the system program to call up error handling routines; they may also be used as short form subroutine jumps if so desired. The trap number references a vector in the vector table that indicates the address of the TRAP subroutine.

External interrupts are caused by applying the interrupt device number, one through seven binary, on the three interrupt inputs IPL0, IPL1, and IPL2. Level seven has the highest priority, while level one has the lowest. A zero binary on these 3 pins indicates that no interrupt is being requested.

These interrupts reference the seven vectors listed in table 4–7 if the $\overline{\text{VPA}}$ input is asserted. Notice that these vectors are only 1 byte in length. One-byte vector locations are normally used for interrupts and contain memory address 0000 0000 0000 00XX XXXX XX00, where XXXX XXXX is the vector stored at autovector locations one through seven.

If desired, the external hardware may apply the interrupt vector location by not asserting the $\overline{\text{VPA}}$ input. If an external interrupt vector is supplied through the least significant 8 bits of the data bus, a vector to any of the 256 possible table entries can occur. This is useful if multiple interrupt processed I/O devices exist at each interrupt priority level.

Masking various interrupt levels is accomplished through the status register and the 3 bits assigned to perform this function. Interrupts are prohibited if the masks are the same priority level or greater than the currently requested interrupt level. The level seven interrupt cannot be inhibited or masked by the mask bits. It is equivalent to the $\overline{\text{NMI}}$ interrupt input on the MC6800 and MC6809.

INSTRUCTION TIMING 4-9

This section includes a list of the instructions and the number of clock cycles required to execute them. Only the MC6800 instructions are provided in this chapter. They are given to allow the student to calculate some of the time delays required for homework problems or outside development. The complete instruction set for the MC6800 is listed in table 4–8. To calculate the amount of time required to execute an instruction, multiply the number of instruction cycles by 1 μs. This is, of course, for the standard 1 MHz version of the MC6800.

THE MC6800 AND THE LOGIC ANALYZER 4-10

The logic analyzer is an extremely useful device in microprocessor testing. In fact, it is the only device that can be used to view the timing of a microprocessor while it is functioning in a system. It is even possible to view the pro-

TABLE 4–8 MC6800 instruction timing.

	(Dual Operand)	ACCX	Immediate	Direct	Extended	Indexed	Implied	Relative		(Dual Operand)	ACCX	Immediate	Direct	Extended	Indexed	Implied
ABA		•	•	•	•	•	2	•	INC		2	•	•	6	7	•
ADC	x	•	2	3	4	5	•	•	INS		•	•	•	•	•	4
ADD	x	•	2	3	4	5	•	•	INX		•	•	•	•	•	4
AND	x	•	2	3	4	5	•	•	JMP		•	•	•	3	4	•
ASL		2	•	•	6	7	•	•	JSR		•	•	•	9	8	•
ASR		2	•	•	6	7	•	•	LDA	x	•	2	3	4	5	•
BCC		•	•	•	•	•	•	4	LDS		•	3	4	5	6	•
BCS		•	•	•	•	•	•	4	LDX		•	3	4	5	6	•
BEA		•	•	•	•	•	•	4	LSR		2	•	•	6	7	•
BGE		•	•	•	•	•	•	4	NEG		2	•	•	6	7	•
BGT		•	•	•	•	•	•	4	NOP		•	•	•	•	•	2
BHI		•	•	•	•	•	•	4	ORA	x	•	2	3	4	5	•
BIT	x	•	2	3	4	5	•	•	PSH		•	•	•	•	•	4
BLE		•	•	•	•	•	•	4	PUL		•	•	•	•	•	4
BLS		•	•	•	•	•	•	4	ROL		2	•	•	6	7	•
BLT		•	•	•	•	•	•	4	ROR		2	•	•	6	7	•
BMI		•	•	•	•	•	•	4	RTI		•	•	•	•	•	10
BNE		•	•	•	•	•	•	4	RTS		•	•	•	•	•	5
BPL		•	•	•	•	•	•	4	SBA		•	•	•	•	•	2
BRA		•	•	•	•	•	•	4	SBC	x	•	2	3	4	5	•
BSR		•	•	•	•	•	•	8	SEC		•	•	•	•	•	2
BVC		•	•	•	•	•	•	4	SEI		•	•	•	•	•	2
BVS		•	•	•	•	•	•	4	SEV		•	•	•	•	•	2
CBA		•	•	•	•	•	2	•	STA	x	•	•	4	5	6	•
CLC		•	•	•	•	•	2	•	STS		•	•	5	6	7	•
CLI		•	•	•	•	•	2	•	STX		•	•	5	6	7	•
CLR		2	•	•	6	7	•	•	SUB	x	•	2	3	4	5	•
CLV		•	•	•	•	•	2	•	SWI		•	•	•	•	•	12
CMP	x	•	2	3	4	5	•	•	TAB		•	•	•	•	•	2
COM		2	•	•	6	7	•	•	TAP		•	•	•	•	•	2
CPX		•	3	4	5	6	•	•	TBA		•	•	•	•	•	2
DAA		•	•	•	•	•	2	•	TPA		•	•	•	•	•	2
DEC		2	•	•	6	7	•	•	TST		2	•	•	6	7	•
DES		•	•	•	•	•	4	•	TSX		•	•	•	•	•	4
DEX		•	•	•	•	•	4	•	TSX		•	•	•	•	•	4
EOR	x	•	2	3	4	5	•	•	WAI		•	•	•	•	•	9

NOTE: Interrupt time is 12 cycles from the end of the instruction being executed, except following a WAI instruction. Then it is 4 cycles.

SOURCE: Courtesy of Motorola, Inc.

gram execution path or track with the logic analyzer, which can be extremely useful in debugging complicated software.

Instruction Tracking with the Logic Analyzer

The op-codes of an instruction can be stored with the memory addresses in the memory of the logic analyzer for later viewing as hexadecimal op-codes. In some of the newer logic analyzers, it is even possible to view this information in mnemonic form as a listing on the screen of the analyzer. This, of course, is not a listing of the program; it is a dynamic listing of the instructions as they are actually executed in the system.

To track the program in an operating system with a logic analyzer, three signal components must be connected to the analyzer. The data input connections for the analyzer are connected to the MC6800 data bus, allowing the instructions and data to be displayed as they appear on the data bus. In addition to the data, a logic analyzer needs a clock signal to acquire the information from the data bus. This signal is obtained by logically combining the VMA signal with phase two of the clock. It is important that the analyzer clock is set on the negative edge of the output of the circuit in figure 4–10.

In addition to the data and clock inputs, the analyzer must also be triggered at the proper point by using the beginning address of the software under test as a trigger. Many analyzers have a 16-channel trigger producing circuit for this purpose.

Once the analyzer is triggered, it stores the information from the data bus in its internal memory until it is full. At this time the listing of the program can be viewed and checked for errors.

Displaying the Timing Diagram of the MC6800

To test the entire system, it is a good idea to view the timing of the microprocessor on the logic analyzer. To display the complete timing diagram, you need an analyzer capable of displaying more than 24 signals at one time. In many cases the analyzer that you use may have only 8 or 16 channels. If this is the case, you have to be more selective with the signals viewed on the analyzer.

To display the timing diagram for the MC6800, you may want to use the internal clock set to sample the information at the rate of every 20–50 ns. This procedure generates a fairly accurate timing diagram. The data inputs may consist of the VMA, R/$\overline{W}$, and the clock signal plus a few data bus bits and a few address bus bits. This will not display a complete timing diagram, but at least you can determine if the memory or I/O is functioning properly.

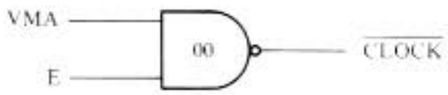


FIGURE 4–10 Circuitry required to generate a clock pulse for the logic analyzer.

Summary

This chapter provides a working knowledge of the hardware signals and major timing of the microprocessor. The MC6800, MC6809, and MC68000 Motorola microprocessors were covered in fair detail, but less comprehen-

sively than in the data sheets from the manufacturer. If more detail is required on any of these or other microprocessors manufactured by Motorola, please refer to *The Complete Motorola Microcomputer Data Library* published by Motorola.

This chapter, of course, does supply enough detail so that you can explore the wonders of these fantastic devices. This book also allows a glimpse of another microprocessor, if you happen to be studying the Intel 8085A microprocessor discussed in chapter 3. In fact, I would strongly urge the student to make a comparative analysis of both Intel and Motorola microprocessors by perusing both chapters.

Glossary

Access time The amount of time required by a memory component to access or retrieve information.

Bus arbitration An access technique used when more than one bus controller or microprocessor exists on the same memory and I/O bus structure.

Bus cycle Whenever information is moved out of or into the microprocessor through its bus.

Direct memory access (DMA) A computer's ability to store or retrieve information directly from the memory without the intervention of the microprocessor.

Instruction cycle Equal to one clocking period in the MC6800, MC6809, and MC68000.

Interrupt An I/O technique that allows a slower external I/O device to interrupt the instruction flow of the microprocessor. This is accomplished through a hardware subroutine jump.

Memory management A technique whereby available memory space can be increased to an unlimited amount.

Parity A technique used to check for the validity of data.

Pinout The pictorial view of an integrated circuit defining each pin connection.

Read cycle Whenever the microprocessor reads data from the memory or an I/O device.

Sink current The amount of current available at an output whenever that output is a logic zero.

Source current The amount of current available at an output whenever that output is a logic one.

Transceiver A digital device that can either drive a bus line or receive data from a bus line.

Vector A number stored in the memory that is used to point to another location in the memory.

Write cycle Whenever the microprocessor writes information to a memory or an I/O device.

Questions and Problems

1 List the number of pin connections on each of the following microprocessors: MC6800, MC6809, and MC68000.
2 How many TTL unit loads can the MC6800 or MC6809 microprocessor directly drive?
3 How many TTL unit loads can the MC68000 microprocessor directly drive? Explain your answer.
4 What is the noise immunity for the MC6800, MC6809, and MC68000?
5 Which crystal frequency would be selected to operate the MC6800 at 1 MHz?
6 Which crystal frequency would be selected to operate the MC6809 at 1 MHz?
7 How many memory locations can the MC6800 or MC6809 directly address?
8 How many memory locations can the MC68000 directly address?
9 How many data bus connections are available on the MC68000 microprocessor?
10 Which MC6800 bus is a bidirectional bus?
11 What is the purpose of the $\overline{AS}$ pin on the MC68000?
12 What is the purpose of the $\overline{LDS}$ and $\overline{UDS}$ strobes on the MC68000?
13 The $\overline{BERR}$ signal on the MC68000 indicates which condition?
14 Which three signals control a DMA action on the MC68000 microprocessor?
15 Which signals control the DMA action of the MC6809 microprocessor?
16 Where must the RESET vector be stored in the MC6800 or MC6809 microprocessor?
17 Where must the RESET vector be stored in the MC68000 microprocessor?
18 How much time is allowed for memory access in a MC6800 based system?
19 How much time is allowed for memory access in a MC68000 based system?
20 Explain the operation of the circuit in figure 4–7.
21 What is the purpose of the $\overline{DTACK}$ signal in the MC68000 microprocessor?
22 List the types of interrupts available for the MC6800 microprocessor.
23 List the types of interrupts available for the MC6809 microprocessor.
24 List the types of interrupts available for the MC68000 microprocessor.
25 What is the difference between the $\overline{FIRQ}$ and the $\overline{IRQ}$ inputs on the MC6809?
26 Which MC68000 interrupt input level has the highest priority?
27 How long does it take the MC6800 to execute the LDAA instruction if a clock frequency of 1 MHz has been selected?

28 Given the following MC6800 program, determine how long it takes to execute if a 1 MHz clock is used.

```
        LDAA   #$10
LOOP    DECA
        BNE    LOOP
```

29 The logic analyzer can monitor the instruction flow in a subroutine or a program. Write a short program to test I/O location $C000.

30 If you were to use the internal clock on the logic analyzer and you set it for a 1 μs sample rate, what would you view on the screen if the data bus were connected to the analyzer's data inputs?

12

MC6800 Application Examples

This chapter collects all of the separate techniques that were learned throughout this textbook. Example problems that include memory interface, various forms of I/O interface, and digital communications have been illustrated. It is very important that the student go through each of the example problems for ideas on hardware and software implementation.

For more examples, see the end of this chapter, which contains a series of projects that illustrate many of the techniques discussed in this text.

12-1 DATA CONCENTRATOR

Data concentrators are used in data communications environments to pack many slow channels of digital data onto one high-speed channel. For example, a department store may have 20 point of sales terminals that must be connected to a computer in another city. Instead of leasing 20 telephone lines for the fairly intermittent data from these in-store terminals, a data concentrator can be connected between the POS terminals and the computer in the other city. This connection does not reduce the speed of the system as far as the user is concerned; it only reduces the total system cost by replacing the 20 leased lines with 1.

6800 Data Concentrator Example

In this example two low-speed channels are concentrated onto one higher-speed channel for transmission to another system. The data on the low-speed channels is serial asynchronous data transmitted at 300 baud, and the data on the high-speed channel is asynchronous data transmitted at 4800 baud. For this example, we will only consider one way communications between the two terminals and the remote system.

Figure 12–1 illustrates the protocol between the concentrator and the larger computer system. The data is preceded by an ID byte that indicates which terminal is transmitting the data. The ID byte is always followed by 15 bytes of information, allowing for a fairly efficient means of data transmission between each terminal and the remote computer system.

6800 Data Concentrator Hardware

The hardware for this application is pictured in the schematic of figure 12–2. The 6800 is surrounded by three 6850 ACIAs that receive serial data from the terminals and transmit serial data to the remote computer system. In addition to the ACIAs, a 128-byte RAM for data storage and a 1K-byte EPROM for program storage exist.

The decoder selects the EPROM for memory locations $FXXX, the RAM for locations $0XXX, and the ACIAs for locations $BXXX, $CXXX, and $DXXX. Data channel one uses $BXXX; data channel two uses $CXXX; $DXXX is used as the link between the remote computer and the data concentrator.

Data Concentrator Initialization Dialog

In this system the three ACIAs must be initialized to start the communications between the terminals and the remote system. This dialog resides at the location pointed to by the restart vector in locations $FFFE and $FFFF.

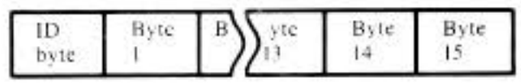


FIGURE 12–1 The protocol for the MC6800 based data concentrator.

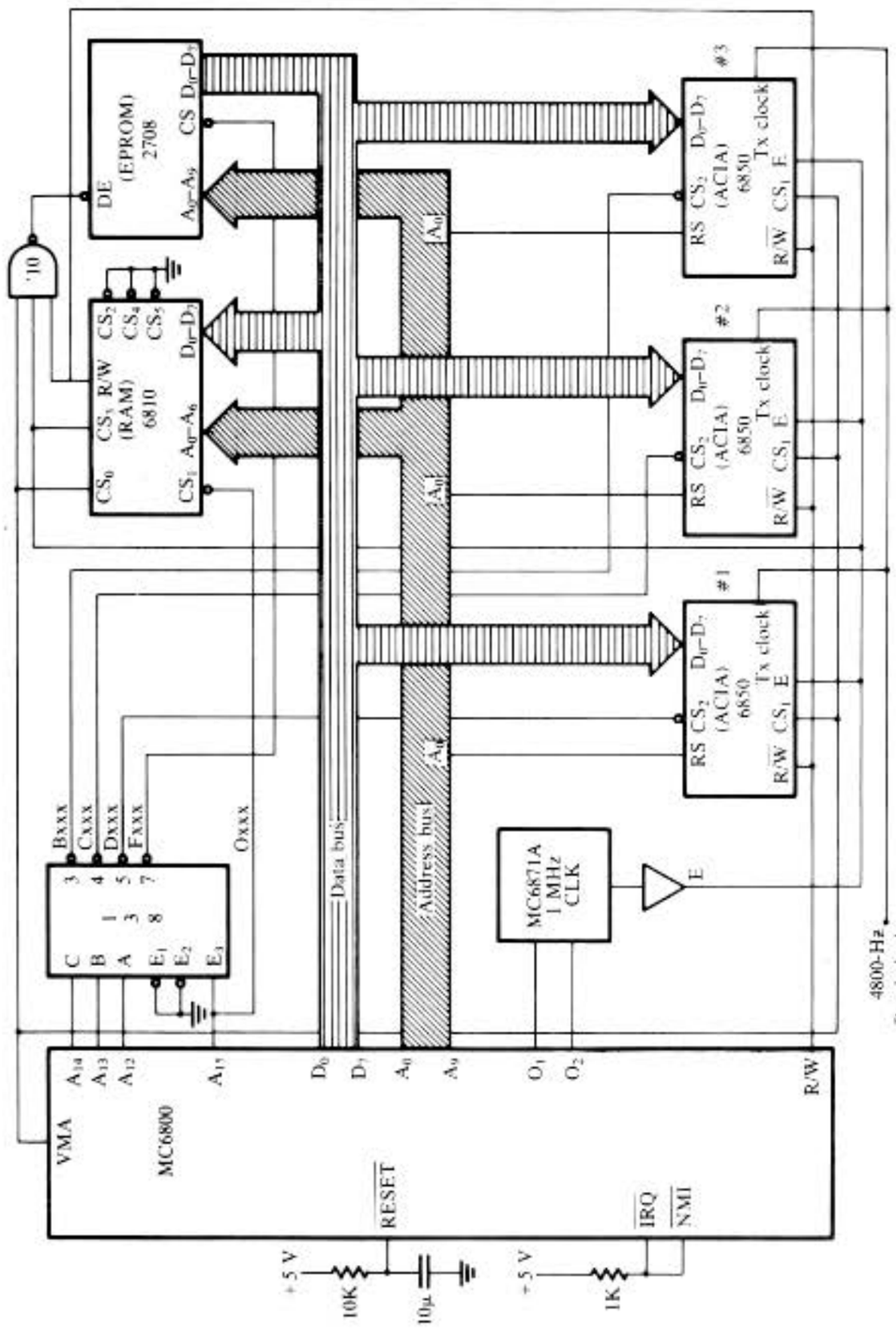


FIGURE 12–2 The schematic diagram of the MC6800 microprocessor based data concentrator.

```
 1 *DATA CONCENTRATOR INITIALIZATION DIALOG
 2 *
 3 START LDDA #$09       SETUP ACIA COMMAND WORD
 4       STAA $B000      PROGRAM THREE ACIAs
 5       STAA $C000
 6       CLR  $D000      SETUP COMMAND FOR ACIA THREE
 7       LDS  #$007F     SETUP STACK AREA
 8       LDX  COUNT      POINT TO BUFFERS, POINTER AND FLAGS
 9 LOOP  CLR  X          CLEAR BUFFERS, POINTERS AND FLAGS
10       DEX
11       BNE  LOOP
12       CLR  X
13       LDAA #32        SETUP QUEUE TWO POINTERS
14       STAA IPNT2+1
15       STAA OPNT2+1
           .
           . (System software begins here)
           .
```

The terminal ACIAs are programmed to divide the external clock source by 16, to transmit 7 data bits with even parity, and to send 1 stop bit. The high-speed ACIA is programmed with the same data format, except that its internal divider is setup to divide by 1. The resulting transmission speed is 4800 baud.

Data Storage for the Data Concentrator

The data storage consists of two separate buffer areas in the memory. These hold data as it comes from the two terminal devices and function as FIFOs, or queue memories.

```
16 *RAM STORAGE
17 *
18 BUF1   RMB  32  TERMINAL ONE BUFFER
19 BUF2   RMB  32  TERMINAL TWO BUFFER
20 IPNT1  RMB  02  QUEUE ONE POINTERS
21 OPNT1  RMB  02
22 IPNT2  RMB  02  QUEUE TWO POINTERS
23 OPNT2  RMB  02
24 FLAG   FCB  00  TRANSMIT FLAG
25 COUNT  FCB  00  BYTE COUNTER
```

The queue pointers are all initialized for the empty condition; that is, both the input and output pointers are equal in value. A full condition is indicated when the IPNT is one less than the OPNT.

ACIA Status Scanning Software

The purpose of this software is to determine when an ACIA has received information or when it is ready to transmit information. This software immediately follows the system initialization dialog presented earlier. Figure 12–3 illustrates the flowchart of this software, which is the main program for the data concentrator.

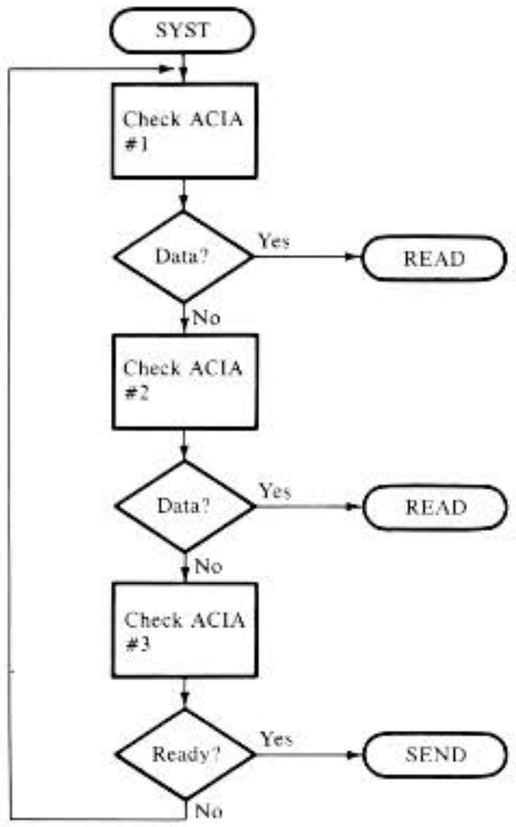


FIGURE 12–3 The flowchart of the SYST program for the MC6800 based data concentrator.

```
26 *STATUS SCANNING SOFTWARE
27 *
28 SYST  LDDB #1        SET ACIA NUMBER
29       LDAA $B000     GET STATUS ONE
30       JSR  CHKI      GO CHECK ACIA STATUS
31       INCB            SET ACIA NUMBER
32       LDAA $C000     GET STATUS TWO
33       JSR  CHKI      GO CHECK ACIA STATUS
34       LDAA $D000     GET STATUS THREE
35       JSR  CHKO      GO CHECK ACIA STATUS
36       BRA  SYST      KEEP CHECKING
```

The software is looped through continually until a ready condition on any receiver is detected or a ready condition in the transmitter is detected. Once detected, data is transmitted or received by subroutines presented later in this text.

```
37 *SUBROUTINE TO CHECK ACIA RECEIVER STATUS
38 *
39 CHKI RARA    RDRF INTO CARRY
40      BCS     READ
41      RTS     IF NO DATA IN THE RECEIVER
```

The subroutine illustrated above checks the receiver of the selected ACIA to determine if it is ready with data. If it is not ready, a return from the subroutine occurs. If it is ready, the subroutine continues at location READ.

Reception Software

As the data is read from each terminal, it is stored in the terminal's queue, where it is held for later transmission by the high-speed ACIA. No attempt is made to detect a queue full condition, since this can never happen because of the speeds involved. Refer to figure 12–4 for a complete flowchart of this subroutine.

```
42 *SUBROUTINE TO READ DATA FROM AN ACIA
43 *AND SAVE IT IN THE APPROPRIATE QUEUE
44 *
45 READ   CMPB #2        CHECK FOR ACIA 2
46       BEQ             READ2
47       LDX  IPNT1      POINT TO QUEUE ONE
48       LDAA $B000      GET DATA
49       BRA             READ3
50 READ2 LDX  IPNT2      POINT TO QUEUE TWO
51       LDAA $C000      GET DATA
52 READ3 STAA X          SAVE DATA
53       JSR  UPDAT      INCREMENT AND WRAP POINTER
54       STX  IPNT2      SAVE IPNT2
55       RTS
```

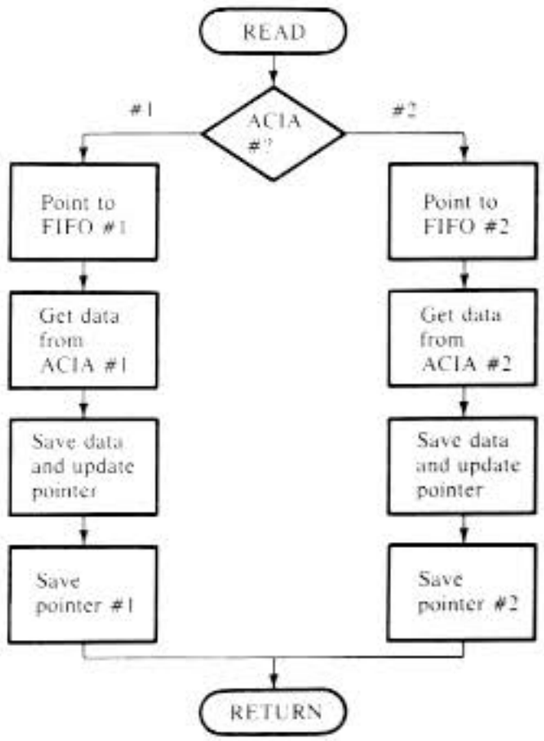


FIGURE 12–4 The flowchart of the READ subroutine.

Transmission Software

```
56 *SUBROUTINE TO CHECK TRANSMITTER STATUS
57 *
58 CHKO   BITA #$02      TEST TDRE
59       BNE  SEND
60       RTS
```

This short subroutine determines if the transmitter in the high-speed ACIA is ready for another byte of information. If it is not, a return from the subroutine occurs so that the remaining ACIAs can be tested.

The SEND subroutine transmits data to the remote system through the high-speed ACIA. The flowchart for this routine is pictured in figure 12–5, and the program itself follows.

```
61 *SEND SUBROUTINE FOR TRANSMITTING DATA THROUGH
62 *THE HIGH SPEED ACIA
63 *
64 SEND   LDDA FLAG      GET TRANSMITTER BUSY FLAG
65       BNE  BUSY      IF BUSY
66       LDAA IPNT1     GET IPNT1
67       CMPA OPNT1     COMPARE WITH OPNT1
68       BNE  ST1       IF FIFO ONE IS NOT EMPTY
69       LDAA IPNT2     GET IPNT2
70       CMPA OPNT2     COMPARE WITH OPNT2
71       BNE  ST2       IF FIFO TWO IS NOT EMPTY
72       RTS            IF BOTH FIFOS ARE EMPTY
```

This portion of the SEND subroutine checks whether the transmitter is currently sending data; if it is not, it continues on to check whether data is available to transmit. If no data is present to transmit and the transmitter is not busy, it returns to scanning for input data through the two low-speed data channels.

```
73 *CONTINUATION OF SEND WHEN FIFOS ARE NOT EMPTY
74 *
75 ST1    LDAA #01       LOAD TERMINAL ID NUMBER
76       BRA  ST        GO SEND IT
77 ST2    LDAA #02       LOAD TERMINAL ID NUMBER
78 ST     STAA $D001     SEND TERMINAL ID NUMBER
79       STAA FLAG      SAVE TERMINAL NUMBER IN FLAG
80       LDAA #15       SETUP BYTE COUNTER
81       STAA COUNT     SAVE IN COUNT
82       RTS            CONTINUE SCANNING
```

If the transmitter is not busy but data is available to transmit, this portion of the software sends the terminal number through the high-speed ACIA and also sets a byte counter to 15. The byte counter contains the number of bytes that must follow the ID number. After transmitting the ID number, a return to scanning occurs so that additional data may be received.

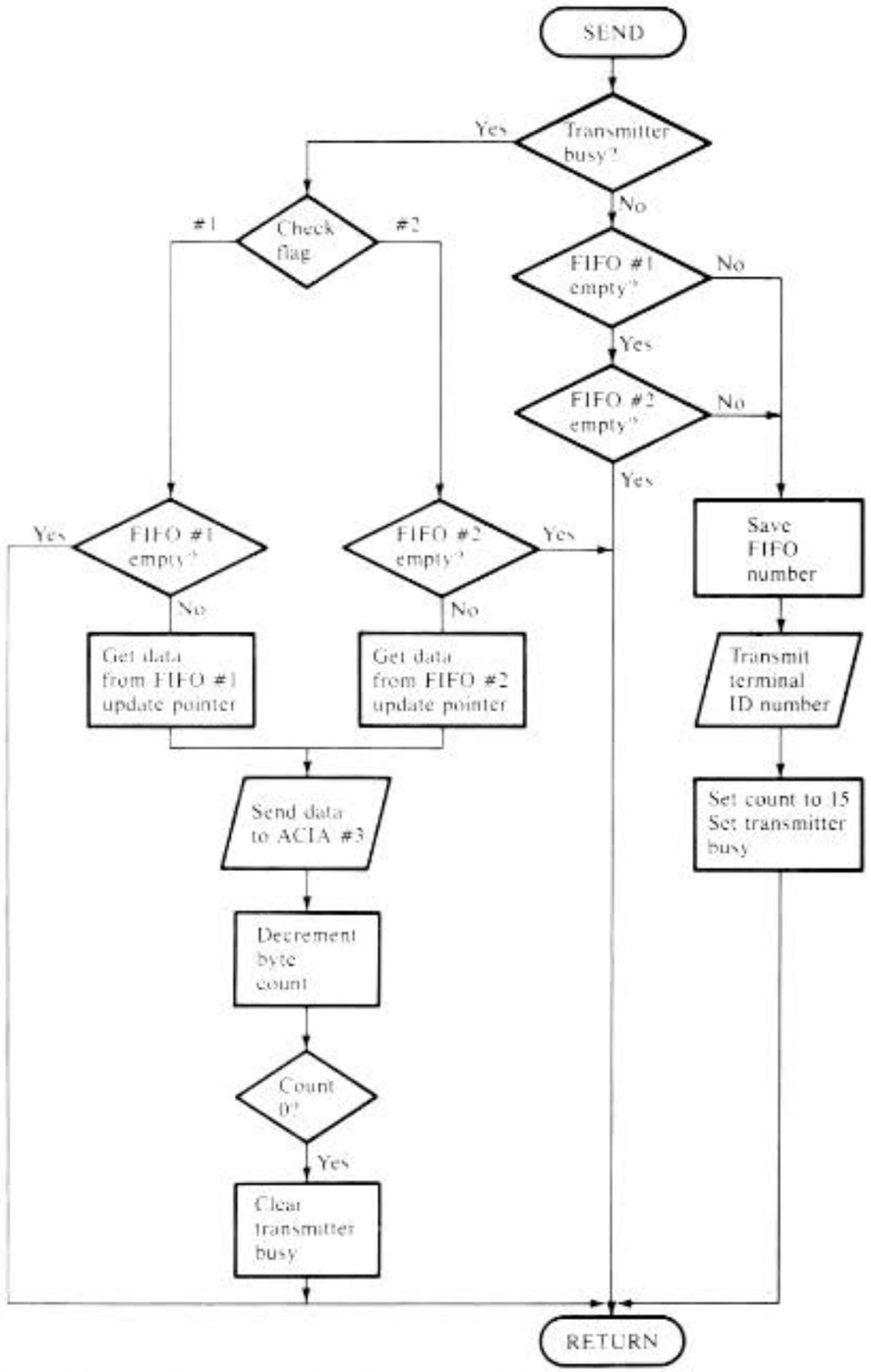


FIGURE 12–5 The flowchart of the SEND subroutine.


```
83 *CONTINUATION OF SEND
84 *
85 BUSY  LDAA FLAG      CHECK TERMINAL NUMBER
86       TSTA #02       CHECK FOR TERMINAL TWO
87       BNE  T1        GO TO TERMINAL ONE
88       LDAA IPNT2+1   GET IPNT2
89       CMPA OPNT2+1   COMPARE WITH OPNT2
90       BNE  SEN2      GO SEND A BYTE FROM TWO
91       RTS            CONTINUE SCANNING
92 T1    LDAA IPNT1+1   GET IPNT1
93       CMPA OPNT1+1   COMPARE WITH OPNT1
94       BNE  SEN1      GO SEND A BYTE FROM ONE
95       RTS            CONTINUE SCANNING
```

If the transmitter has been sending data, it arrives at this section of the software to determine whether any data has been received. If it has, a transfer occurs to either SEN1 or SEN2 to send the information. If it has not, control is returned, and it continues to search for more input data.

```
 96 *DATA TRANSMISSION PORTION OF SEND
 97 *
 98 SEN1 LDX  OPNT1     GET OPNT1
 99      LDAA X         GET A BYTE OF DATA
100      STAA $D001     SEND THE DATA
101      JSR  UPDAT     INCREMENT AND WRAP POINTER
102      STX  OPNT1     SAVE OPNT1
103 SENX DEC  COUNT     DECREMENT BYTE COUNT
104      BNE  RETS      RETURN FROM SUBROUTINE
105      CLR  FLAG      CLEAR BUSY FLAG
106 RETS RTS           CONTINUE SCANNING
107 SEN2 LDX  OPNT2     GET OPNT2
108      LDAA X         GET DATA
109      STAA $D001     SEND DATA
110      JSR  UPDAT     INCREMENT AND WRAP POINTER
111      STX  OPNT2     SAVE POINTER
112      BRA  SENX      FINISH UP
```

This software sends information through the high-speed ACIA and then decrements the byte counter. If the byte counter reaches zero, which indicates that all 15 bytes have been transferred, the FLAG is cleared so that the transmitter can start transmitting the next 15 bytes of data.

```
113 *INCREMENT A POINTER AND WRAP IT IF NEEDED
114 *
115 UPDAT STX  COUNT+1  SAVE TEMP
116      LDAA COUNT+2  GET POINTER
117      ANDA #$E0     STRIP MOST SIGNIFICANT
118      PSHA          SAVE IT
```

```
119         LDAA COUNT+2   GET POINTER
120         INCA           INCREMENT IT
121         ANDA #$1F      MASK MOST SIGNIFICANT
122         STAA COUNT+2   SAVE IT
123         PULA           RESTORE IT
124         ORAA COUNT+2   COMBINE
125         STAA COUNT+2
126         LDX  COUNT+1   LOAD INDEX REGISTER
127         RTS
```

This subroutine increments the index register and stores the result back into the index register. It is a simple task, except that in this case the number must be a 5-bit cyclic number, which requires all of the special coding listed in the above subroutine. Only the least significant 5 bits are incremented in this subroutine.

System Limitations

This system has one important limitation that should be noted. The low-speed data must continue in increments of 15 bytes. If this does not happen, the system hangs up with no output ever for one of the two channels. If this result is not acceptable, the system can be modified to send a byte at a time, preceded with the terminal number. The only problem with this is that the system's efficiency suffers.

12-2 TRAFFIC LIGHT CONTROLLER

Traffic light control by microprocessors is becoming commonplace in many large cities because these units are easily adjusted for different timing sequences and can be controlled by an external computer system. External computer control has increased traffic flow during peak hours and reduced the number of accidents in the cities where it has been tested.

The system illustrated in this text receives its timing sequence through a keyboard located at the controlled intersection. The keyboard also enters the time of day and other information, such as the times the traffic light should flash. This system also includes a set of trip plates to trip the light for one direction.

Traffic Light Controller Hardware

The hardware for this controller includes an MC6821, which scans the keyboard and controls the traffic lamps. In addition to the MC6821, an oscillator is included to provide the MC6800 with its clock and to act as a timing source for the nonmaskable interrupt input ($\overline{\text{NMI}}$). Also included is a trip plate sensor that causes an interrupt to occur whenever a vehicle is in proximity with the trip plate. The trip plate itself is a loop of wire located just below the surface of the roadbed. When a vehicle sits over it, the metal in the vehicle changes the inductance of the loop, which can be sensed by the interface.

Figure 12–6 illustrates the MC6800 controller hardware, including the memory required and the appropriate device selection logic. The outputs of

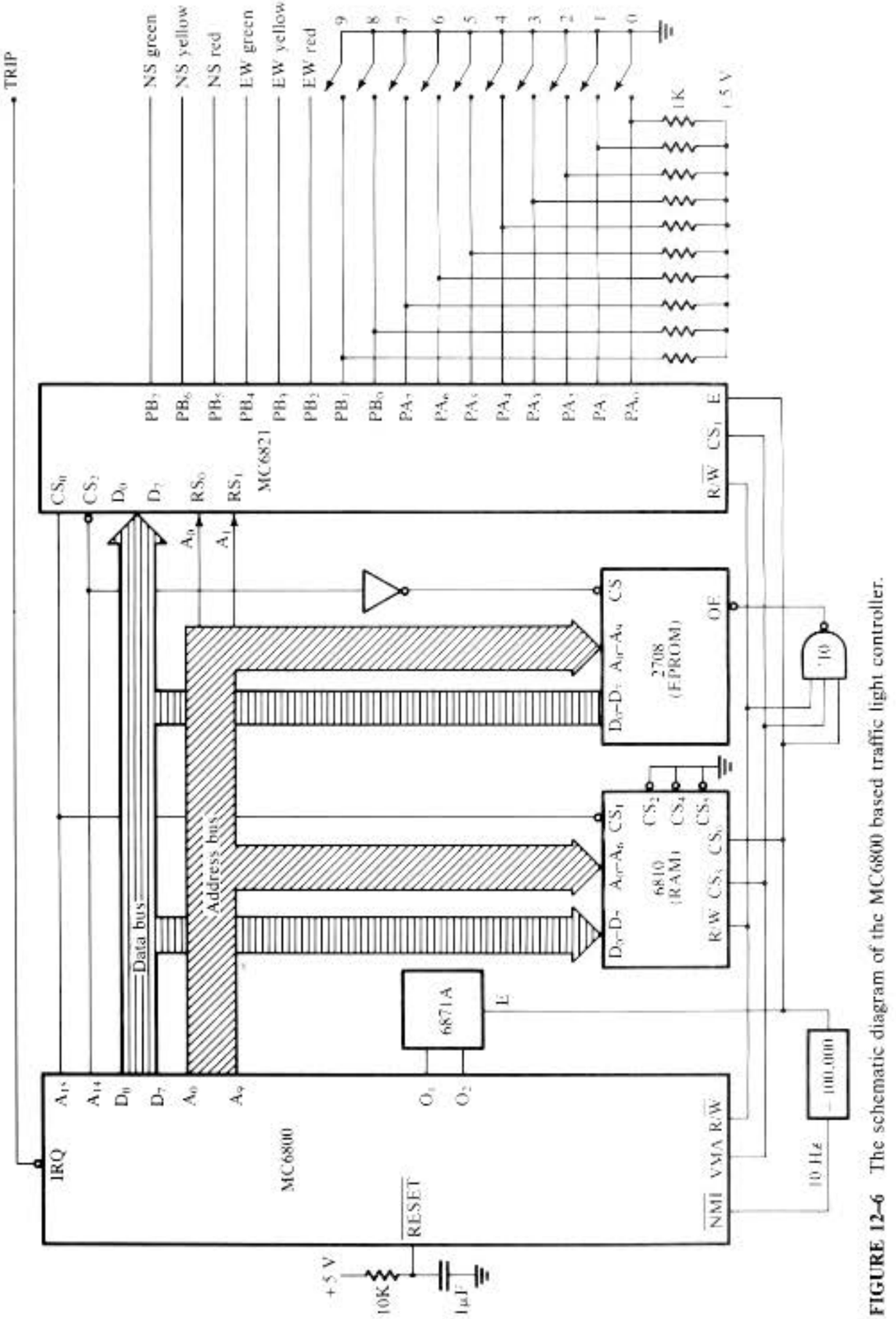


FIGURE 12–6 The schematic diagram of the MC6800 based traffic light controller.

the decoder select the 1K-byte EPROM at address $FXXX, the 128-byte RAM at address $0XXX, and the MC6821 at address $EXXX.

RAM Storage Assignment

```
 1 *RAM STORAGE ASSIGNMENT
 2 *
 3 CLOCK RMB  7       CLOCK STORAGE
 4 TIME  RMB  1       TIMER STORAGE
 5 NSRED RMB  1       NORTH-SOUTH RED
 6 NSGRE RMB  1       NORTH-SOUTH GREEN
 7 NSYEL RMB  1       NORTH-SOUTH YELLOW
 8 EWRED RMB  1       EAST-WEST RED
 9 EWGRE RMB  1       EAST-WEST GREEN
10 EWYEL RMB  1       EAST-WEST YELLOW
11 MAXTR RMB  1       MAXIMUM NUMBER OF TRIPS
12 MINTR RMB  1       MINIMUM TRIP TIME
13 FLSTR RMB  6       FLASH START TIME
14 FLEND RMB  6       FLASH END TIME
```

The basic clock timer is allocated 7 bytes of memory: 6 to keep track of the time (in hours, minutes, and seconds) and 1 to divide the 10 Hz input signal into 1-second pulses. The time is kept in unpacked BCD form for ease in software development.

Memory location TIME is used as a down counter that is decremented once per second. TIME is used by the software to time a particular light and is in standard binary form.

The six locations for light timing, NSRED, NSGRE, and so on, are each programmable for times of up to 255 seconds, which should be more than enough time for a lamp in any direction.

Minimum trip time and maximum number of trips indicate how long a light may remain tripped and the minimum amount of time required to cause a trip. A typical minimum time may be 20 seconds, and a typical maximum number of trips may be five. This, of course, depends on the traffic flow pattern at the intersection.

In many cases it is normal to remove a light from service in the wee hours of the morning by programming the start and end flash times into the 12 bytes of memory allocated for this purpose.

Initialization Dialog

Since this is a programmable device, it must be initialized whenever power is applied or whenever a change in the sequence of the lights is to be effected. The dialog that follows is executed whenever the microprocessor is restarted. The initialization dialog programs the PIA and branches to the keyboard entry portion of the software.

```
15 *INITIALIZATION DIALOG
16 *
17 RESET  LDS  #$007F    SET STACK AREA
18        LDAA #$FC      SETUP PORT B
19        STAA $E002     CONFIGURE PORT B
```



```
20        LDAA #$04      SELECT PERIPHERAL DATA REGISTERS
21        STAA $E001     SEND TO PIA
22        STAA $E003     SEND TO PIA
            .
            .
            .(Continues at the SETUP program)
```

Traffic Controller Setup

The controller must be programmed to function after a restart. Programming is accomplished through the keyboard and consists of entering the time of day, the duration of each light, trip times, and flash times.

Each one of these pieces of information must be entered without visual feedback, since this unit contains no display. A display is unnecessary because the sequence is relatively short and can be entered again if an error is detected.

An example programming sequence is illustrated in figure 12–7.

```
23 *PORTION OF THE SYSTEM PROGRAM THAT SETS ALL OF
24 *THE PROGRAMMABLE FEATURES
25 *
26 SETUP  LDX  CLOCK+1   POINT TO TIME OF DAY
27       JSR  INTIM     GET TIME OF DAY
28       CLR  CLOCK     CLEAR CLOCK
29       LDX  NSRED     POINT TO NSRED
30       JSR  INSEC     GET SECONDS COUNT FOR NSRED
31       JSR  INSEC     GET SECONDS COUNT FOR NSGRE
32       JSR  INSEC     GET SECONDS COUNT FOR NSYEL
33       LDAA NSYEL     GET NSYEL
34       STAA EWYEL     SET EWYEL
35       ADDA NSGRE     DEVELOP EWRED
36       STAA EWRED     SAVE EWRED
37       LDAA NSRED     GET NSRED
38       SUBA NSYEL     DEVELOP EWGRE
39       STAA EWGRE     SAVE EWGRE
40       LDX  MAXTR     POINT TO MAXTR
41       JSR  INSEC     GET COUNT FOR MAXTR
42       JSR  INSEC     GET SECONDS COUNT FOR MINTR
43       LDX  FLSTR     POINT TO FLSTR
44       JSR  INTIM     GET FLASH START TIME
45       JSR  INTIM     GET FLASH END TIME
46       LDAA MAXTR     CHECK FOR A TRIP PLATE
47       BEQ  SYST      IF NO TRIP PLATE
```

Time of day HH/MM/SS	North south red XXX	North south green XXX	North south yellow XXX	Maximum trip count XXX	Seconds for minimum trip XXX	Start flash time HH/MM/SS	Stop flash time HH/MM/SS

FIGURE 12–7 The setup sequence that programs the traffic light controller.

```
47 SET1   CLI              ENABLE TRIP PLATE INTERRUPT
            .
            .              (Continues at SYST program)
            .
```

This software accepts all of the programming data from the keyboard and stores it in the appropriate memory locations. See figure 12–8 for a flowchart. It also calculates the duration of each of the traffic lamps facing east-west and determines whether or not a trip plate is connected to the system. If MAXTR is a zero, it is assumed by the software that no trip plate is connected in the system, and the trip plate interrupt is left disabled.

INTIM Subroutine

The INTIM subroutine reads the time through the keyboard for the time of day, flash starting time, and flash ending time. It stores the time in the format HH/MM/SS. HH is a two digit number for hours, MM is for minutes, and SS

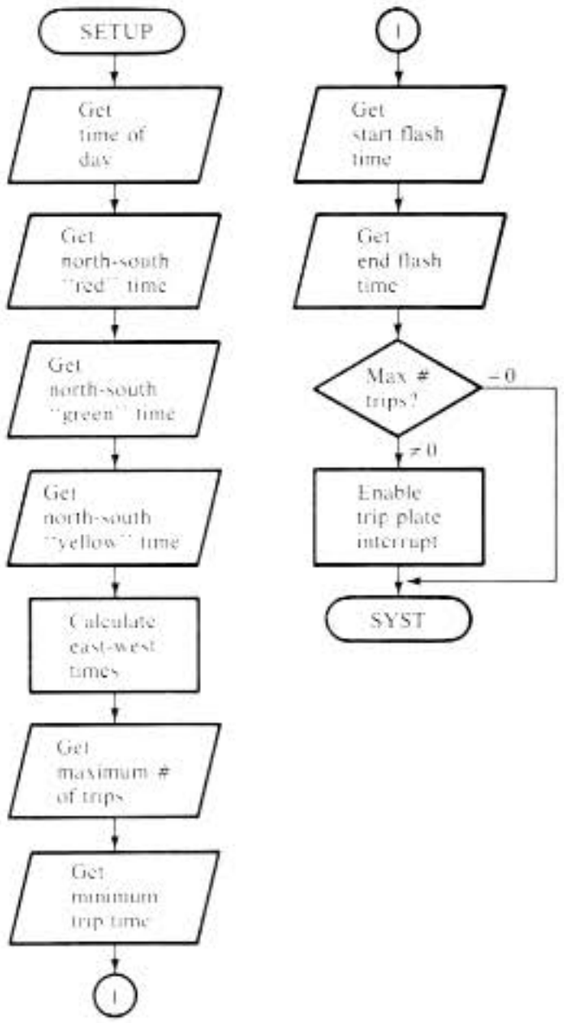


FIGURE 12–8 The flowchart of the SETUP portion of the traffic light controller system program.

for seconds. This six digit number is stored in six contiguous memory locations, which are indexed by the X register in unpacked BCD form.

```
48 *SUBROUTINE TO SAVE THE TIME IN UNPACKED BCD FORM
49 *
50 INTIM  LDAB #$06      SETUP COUNTER
51 INTIS  JSR  INKEY     GET A DIGIT
52        STAA X         SAVE IT
53        INX            POINT TO NEXT LOCATION
54        DECB           DECREMENT COUNT
55        BNE  INTIS     REPEAT UNTIL SIX DIGITS
56        RTS            RETURN FROM SUBROUTINE
```

INSEC Subroutine

The INSEC subroutine, illustrated in the flowchart of figure 12–9, accepts a three digit number from the keyboard and converts it from BCD to binary. It is then stored in the memory location that is indexed by the X register. INSEC is used to get and save data for the timing on the lights and the trip times, if required.

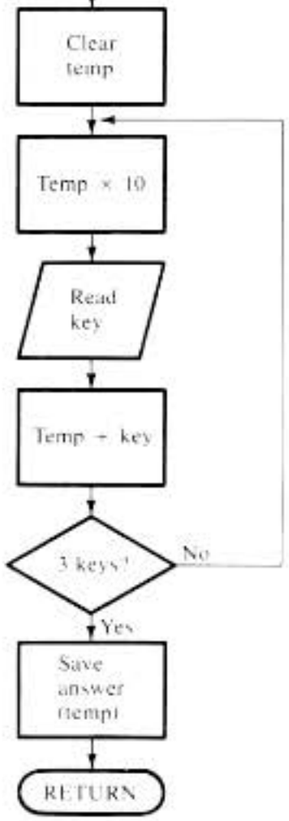


FIGURE 12–9 The flowchart of the INSEC subroutine.

```
57 *INPUTS DATA IN BCD FROM THE KEYBOARD THEN
58 *CONVERTS IT TO BINARY AND SAVES IT
59 *
60 INSEC  LDAB #$03        SETUP COUNTER
61       CLR  FLEND+6      CLEAR TEMP
62 INTI1 LDAA FLEND+6      MULTIPLY BY 10
63       ASLA              DOUBLE ACC
64       STAA FLEND+6
65       ASLA
66       ASLA
67       ADDA FLEND+6
68       STAA FLEND+6
69       JSR  INKEY        GET DIGIT
70       ADDA FLEND+6      CREATE BINARY NUMBER
71       STAA FLEND+6
72       DECB              DECREMENT COUNT
73       BNE  INTI1        REPEAT FOR THREE DIGITS
74       LDAA FLEND+6      GET BINARY VERSION
75       STAA X            SAVE IT
76       INX               POINT TO NEXT
77       RTS               RETURN FROM SUBROUTINE
```

This subroutine converts from BCD to binary by multiplying the previous binary number by ten and then adding in the new BCD digit. This will generate a binary number for a BCD number of up to 255. In example 12–1, a 103 is converted to binary using this algorithm.

EXAMPLE 12–1

FLEND + 6

	× 10	0000	0000	×	0000	1010
First	+ 1	0000	0000	+	0000	0001
	× 10	0000	0001	×	0000	1010
Second	+ 0	0000	1010	+	0000	0000
	× 10	0000	1010	×	0000	1010
Third	+ 3	0110	0100	+	0000	0011
RESULT		0110	0111			

INKEY Subroutine

The INKEY subroutine is used to retrieve information from the ten key numeric keypad interfaced to the MC6800 through a MC6821 PIA. This procedure is accomplished by using the basic INKEY subroutine that was discussed in chapter 7. A flowchart for this subroutine is depicted in figure 12–10.

```
78 *SUBROUTINE TO READ A CHARACTER FROM THE KEYBOARD
79 *
80 INKEY  JSR  CHECK       CHECK FOR A KEYSTROKE
81       BNE  INKEY        IF A KEYSTROKE
```

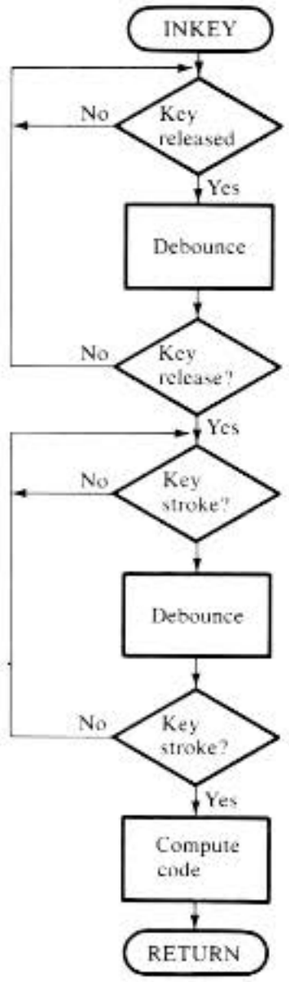



FIGURE 12–10 The flowchart of the INKEY subroutine.

```
82       JSR  DELAY       DEBOUNCE
83       JSR  CHECK       CHECK FOR A KEYSTROKE
84       BNE  INKEY       IF A KEYSTROKE
85 INKEY1 CALL CHECK      CHECK FOR A KEYSTROKE
86       BEQ  INKEY1      IF NO KEYSTROKE
87       JSR  DELAY       DEBOUNCE
88       JSR  CHECK       CHECK FOR A KEYSTROKE
89       BEQ  INKEY1      IF NOISE
90       PSHB             STACK ACC B
91       LDAB #$FF        SETUP BCD CODE
92       LDAA $E000       GET 0 TO 7
93       INCA             CHECK FOR ANY
94       BNE  INKEY3      IF 0 THROUGH 7
95       LDAB #07         IF 8 OR 9
96       LDAA $E002       GET 8 AND 9
97 INKEY3 INCB
```

```
 98          RORA
 99          BCS   INKEY2     IF NOT FOUND
100 OUT      TBA              GET BCD CODE
101          PULB             RESTORE ACC B
102          RTS              RETURN FROM SUBROUTINE
103 CHECK    LDAA $E000       GET 0 TO 7
104          INCA
105          BNE   CHK1       GET A 0 TO 7
106          LDAA $E002       GET 8 AND 9
107          ORAA #$FC
108          INCA
109 CHK1     RTS              RETURN FROM SUBROUTINE
110 DELAY    PSHB             SAVE ACC B
111          LDAB #$14        CAUSE 10 MSEC. DELAY
112          CLRA
113 DEL1     DECA
114          BNE   DEL1
115          DECB
116          BNE   DEL1
117          PULB             RESTORE ACC B
118          RTS              RETURN FROM SUBROUTINE
```

Nonmaskable Interrupt Service Subroutine

This subroutine is used for keeping the correct time by modifying CLOCK; it also, once per second, decrements whichever number happens to be in location TIME. This feature provides the traffic light controller with a real-time clock that not only contains the time of day but can also time events. Location TIME is used as a timer and can time events in 1 second intervals. See figure 12–11 for a flowchart of the interrupt service subroutine.

```
119 *NONMASKABLE INTERRUPT SERVICE SUBROUTINE FOR
120 *THE REAL TIME CLOCK
121 *
122 NMI    INC   CLOCK      GET DIVIDE BY TEN
123          LDAB #$0A      CHECK FOR A TEN
124          CMPB CLOCK
125          BNE   NMI3     EXIT
126          CLR   CLOCK    CLEAR COUNT
127          LDX   TIME     POINT TO TIME
128          DEC   X        DECREMENT TIME
129          DEX            POINT TO SECONDS
130          JSR   INCR     GO INCREMENT SECONDS
131          BNE   NMI3     RETURN FROM INTERRUPT
132          LDAB #$06      SET WRAP
133          JSR   INCR     GO INCREMENT TENS OF SECONDS
134          BNE   NMI3
135          LDAB #$0A      SET WRAP
136          JSR   INCR     GO INCREMENT MINUTES
```

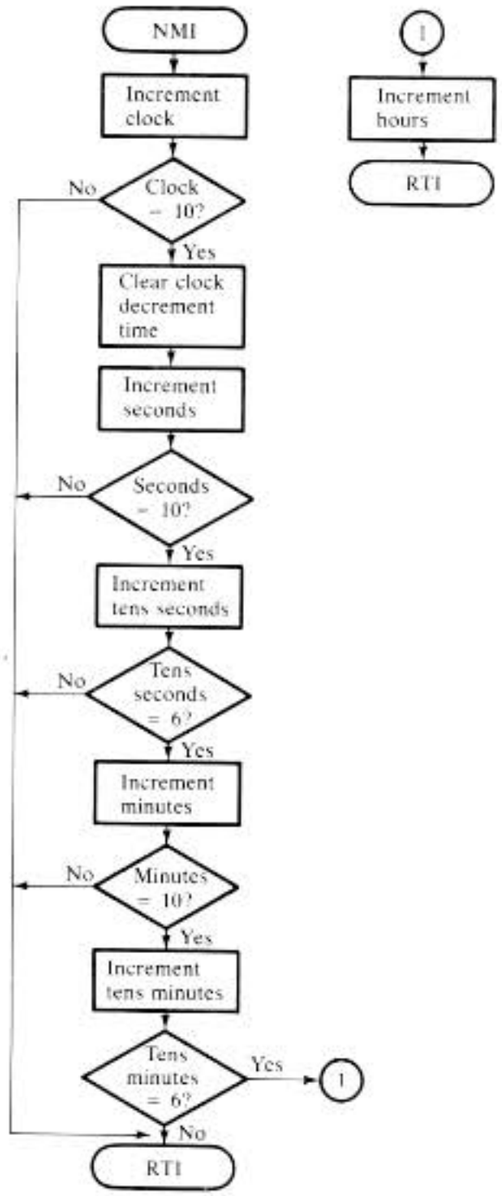



FIGURE 12–11 The flowchart of the nonmaskable interrupt service subroutine.

```
137          BNE   NMI3
138          LDAB #$06      SET WRAP
139          JSR   INCR     GO INCREMENT TENS OF MINUTES
140          BNE   NMI3
141          DEX
142          LDAA X         GET TENS OF HOURS
143          LDAB #$0A      SET WRAP
144          INX
145          CMPA #02       CHECK FOR 20 HOURS
```

```
146          BNE   NMI2       IF NOT 20 TO 23 HOURS
147          LDAB  #$04       SET WRAP
148 NMI2     JSR   INCR       GO INCREMENT HOURS
149          BNE   NMI3
150          LDAB  #$03
151          JSR   INCR       GO INCREMENT TENS OF HOURS
152 NMI3     RTI              RETURN FROM INTERRUPT
153 INCR     INC   X          GET COUNTER
154          CMPB  X          CHECK FOR A WRAP
155          BNE   INCR1      IF NO WRAP AROUND
156          CLR   X
157 INCR1    DEX
158          TST   X
159          RTS              RETURN FROM SUBROUTINE
```

The subroutine INCR has been developed to increment the count in the memory location indexed by the X register. If the count equals the number in ACC B, or wrap, the count is cleared. Number wrap indicates the modulus of the counter to the subroutine. A return with the CCR indicating an equal condition means that the next higher order digit of time must be incremented. If a return with the CCR indicating a not equal condition occurs, it means that no further counters need be updated.

Traffic Light System Software

The purpose of this segment of the software is to change the traffic lights. The software scans through the times programmed into the controller and changes the indicator lamps at the appropriate time. You might call this the system software, since most of the controller's time is spent here. Figure 12–12 illustrates the flowchart for the system software.

```
160 *SYSTEM SOFTWARE
161 *
162 SYST   LDAA #$01       SYNCHRONIZE WITH CLOCK
163          STAA TIME
164 SYST1  TST   TIME
165          BNE   SYST1     WAIT FOR SYNC
166 SYST2  LDAA  #$84       SET NS-GREEN, EW-RED
167          STAA  $E002     CHANGE LIGHTS
168          LDAA  NSGRE
169          JSR   TIMO      GO TIME OUT LIGHT
170          LDAA  #$44      SET NS-YELLOW, EW-RED
171          STAA  $E002     CHANGE LIGHTS
172          LDAA  NSYEL
173          JSR   TIMO      GO TIME OUT LIGHT
174          LDAA  #$30      SET NS-RED, EW-GREEN
175          STAA  $E002     CHANGE LIGHTS
176          LDAA  EWGRE
177          JSR   TIMO      GO TIME OUT LIGHT
178          LDAA  #$28      SET NS-RED, EW-YELLOW
```

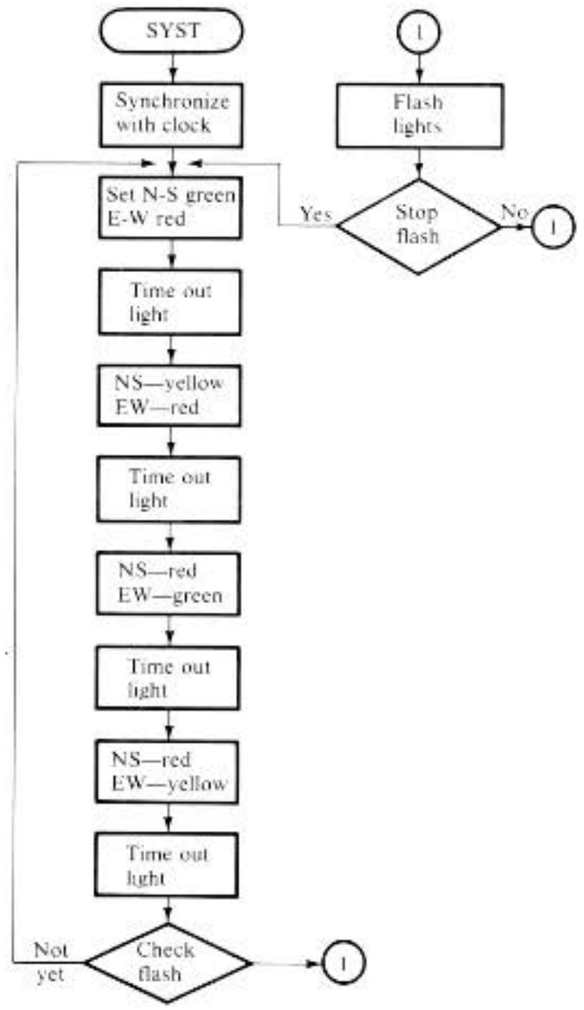



FIGURE 12–12 The flowchart of the main traffic light system program.

```
179          STAA  $E002     CHANGE LIGHTS
180          LDAA  EWYEL
181          JSR   TIMO      GO TIME OUT LIGHT
182          LDX   FLSTR     POINT TO FLASH START TIME
183          JSR   COMPX     COMPARE FLASH START WITH CLOCK
184          BCS   STST2     CONTINUE SEQUENCE
185 SYST3    LDAA  #$28      SET NS-RED, EW-YELLOW
186          STAA  $E002     CHANGE LIGHTS
187          LDAA  #$01      SET COUNT TO ONE SECOND
188          JSR   TIMO      TIME IT OUT
189          LDX   FLEND     POINT TO FLASH END
190          JSR   COMP      CHECK END FLASH TIME
191          BCC   SYST2     CONTINUE NORMAL SEQUENCE
192          CLR   $E002     CHANGE LIGHTS
193          LDAA  #$01      SET COUNT TO ONE SECOND
```

```
194          JSR  TIMO      TIME IT OUT
195          LDX  FLEND     POINT TO FLASH END
196          JSR  COMP      COMPARE FLASH END WITH CLOCK
197          BCC  SYST2     CONTINUE NORMAL SEQUENCE
198          BRA  SYST3     CHECK FOR FLASH END
199 TIMO     STAA TIME      SAVE TIMEOUT TIME
200 TIMOA    TST  TIME      TEST TIME
201          BNE  TIMOA     CHECK FOR TIMED OUT
202          RTS            RETURN FROM SUBROUTINE
203 COMP     LDAA CLOCK+1   GET CLOCK
204          CMPA X         CHECK TIME (SECONDS)
205          BNE  COMP1     IF NOT THE SAME END IT
206          LDAA CLOCK+2   CHECK TIME (TENS SECONDS)
207          CMPA 1,X
208          BNE  COMP1     IF NOT THE SAME END IT
209          LDAA CLOCK+3   CHECK TIME (MINUTES)
210          CMPA 2,X
211          BNE  COMP1     IF NOT THE SAME END IT
212 COMPX    LDAA CLOCK+4   CHECK TIME (TENS MINUTES)
213          CMPA 3,X
214          BNE  COMP1     IF NOT THE SAME END IT
215          LDAA CLOCK+5   CHECK TIME (HOURS)
216          CMPA 4,X
217          BNE  COMP1     IF NOT THE SAME END IT
218          LDAA CLOCK+6   CHECK TIME (TENS HOURS)
219          CMPA 5,X
220 COMP1    RTS            END IT
```

The only feature of the above software that may be a little difficult to understand is the very first portion. Line numbers 162 to 165 are used to synchronize the internal interrupt processed clock with the software listed. If this is not accomplished, timing may be inaccurate by 1 second occasionally.

Trip Plate Software

The trip plate interrupt service subroutine only takes effect if a maximum number of trips is programmed into the controller. If the maximum number is zero, the interrupt remains disabled, and the sequence illustrated in SYST takes complete control. The plate itself produces a pulse on the $\overline{IRQ}$ pin of the MC6800 every time that a vehicle rests on or crosses the plate.

This interrupt service subroutine must be able to determine if the east-west light is in the red condition; if it is, it must then determine how much time remains before the light changes to green. If this time is equal to or less than the minimum time for tripping, no action is taken.

Once tripped, the software must continue to trip the light for up to the maximum amount of time before changing back to red. This is accomplished by counting how many times the light has been tripped during the on cycle. A flowchart for this interrupt service subroutine is illustrated in figure 12–13.

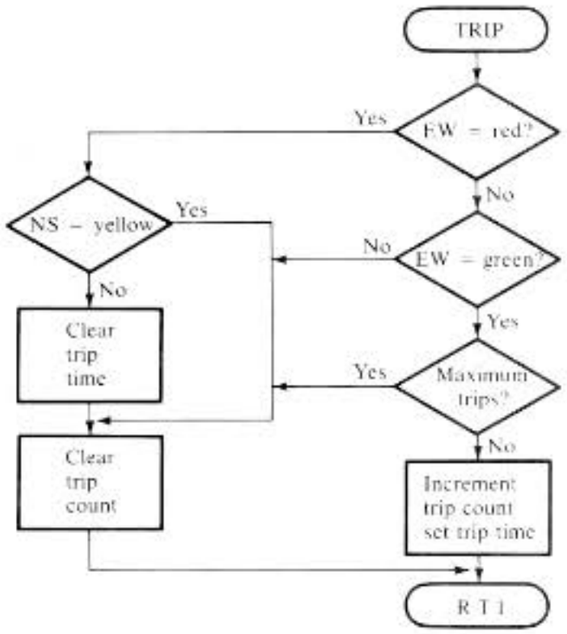


FIGURE 12–13 The flowchart of the trip plate interrupt service subroutine.

```
221 *TRIP PLATE INTERRUPT
222 *
223 TRIP   LDAA $E002      READ LIGHT POSITIONS
224          BITA #$04      TEST EW-RED
225          BNE  TRIP1     IF LIGHT IS RED
226          BITA #$10      TEST EW-GREEN
227          BEQ  TRIP2     IF LIGHT IS YELLOW
228          LDDA FLEND+8
229          CMPA MAXTR     CHECK FOR MAXIMUM TRIPS
230          BEQ  TRIP2     IF DONE
231          INC  FLEND+8
232          LDAA MINTR     GET TRIP TIME
233          STAA TIME      MODIFY TIME
234          RTI
235 TRIP1  BITA #$40        TEST NS-YELLOW
236          BNE  TRIP2     IF YELLOW
237          CLR  TIME      END GREEN FOR NORTH-SOUTH
238 TRIP2  CLR  FLEND+8     CLEAR TRIP COUNT
239          RTI
```

Summary

This chapter applies the Motorola MC6800 series microprocessor in two fairly typical examples, a data concentrator and a traffic light controller. Both appli-

cations are important, since they use some of the fundamental hardware and software techniques presented earlier in this text.

The data concentrator illustrates the use of ACIAs in an actual example problem including programming and a hardware interface. The traffic light controller uses a PIA with a keyboard and some traffic lamps. This second example employs a real-time clock through the nonmaskable interrupt pin on the MC6800 and a trip plate attached to the maskable interrupt input.

The problems at the end of this chapter include several more examples to be worked on as homework or laboratory projects. Whatever their results, the student will learn a great deal about both hardware and software by attempting them.

Suggested Projects

1 Develop the MC6800 hardware and software to implement a coin changer mechanism. It must be able to accept coins in any denomination from 1 cent to 50 cents and dispense change in the fewest number of pennies, nickels, and dimes.

The amount of money to be accepted is programmed through a set of switches located inside the vending machine. The programmable amount can be anything from 1 cent to $1.99.

Your software must accept coins until the amount indicated on the internal switches has been either reached or exceeded. If the amount has been exceeded, dispense the fewest number of coins as change and send an active low pulse out the $\overline{\text{VEND}}$ pin for 20 ms.

As coins are inserted, a mechanical assembly sorts them and signals the microprocessor with a pulse indicating the denomination of the coin. Once your program has detected and remembered the coin, it must drop it into the internal coin box by pulsing the DROP line for 100 ms. There is also a "bent coin" signal in case a defective coin is inserted into the machine. If a bent coin is detected, you must pulse the $\overline{\text{EJECT}}$ line for 120 ms to clear the coin slot.

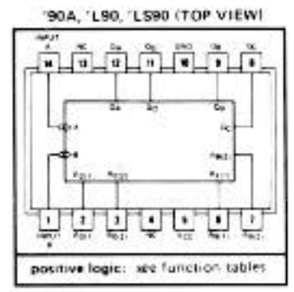
To dispense change, the appropriate $\overline{\text{CH}}$ control line is activated for 100 ms, dropping a coin out of the change slot of the vending machine. You must only return one coin at a time with a pause of at least 50 ms between coins for the mechanical ejection mechanism to function properly. Table 12–1 illustrates all of the TTL input and TTL output connections that are to be interfaced to the MC6800.

2 Develop an IC test fixture that will automatically test the 7490 TTL decade counter. The pinout of this decade counter is pictured in figure 12–14 with a brief description of its operating characteristics.

Your system must completely test this device. If it is found faulty, the red LED must be lit; if good, the green LED must be lit. The test sequence must test the clear to zero, clear to nine, and count sequence of the counter at least 20 times without failure for a good indication. The test socket and two LED indicators are pictured in figure 12–15.

TABLE 12–1 Signal lines for the coin changer.

Signal	Function
$\overline{\text{VEND}}$	Used to vend merchandise from the machine attached to this changer
$\overline{\text{DROP}}$	Used to accept a coin that has been placed into the mechanism
$\overline{\text{EJECT}}$	Used to return a bent or defective coin
1C	One cent program input switch
2C	Two cents program input switch
5C	Five cents program input switch
10C	Ten cents program input switch
20C	Twenty cents program input switch
50C	Fifty cents program input switch
100C	One dollar program input switch
$\overline{\text{CH1}}$	Returns a penny as change if pulsed for 100 ms
$\overline{\text{CH5}}$	Returns a nickel as change if pulsed for 100 ms
$\overline{\text{CH10}}$	Returns a dime as change if pulsed for 100 ms
$\overline{\text{CH25}}$	Returns a quarter as change if pulsed for 100 ms



'90A, 'L90, 'LS90
BCD COUNT SEQUENCE
(See Note A)

COUNT	OUTPUT			
	Q_D	Q_C	Q_B	Q_A
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H

'90A, 'L90, 'LS90
BI-QUINARY (5-2)
(See Note B)

COUNT	OUTPUT			
	Q_A	Q_D	Q_C	Q_B
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	H	L	L	L
6	H	L	L	H
7	H	L	H	L
8	H	L	H	H
9	H	H	L	L

'90A, 'L90, 'LS90
RESET/COUNT FUNCTION TABLE

RESET INPUTS				OUTPUT			
$R_{0(1)}$	$R_{0(2)}$	$R_{9(1)}$	$R_{9(2)}$	Q_D	Q_C	Q_B	Q_A
H	H	L	X	L	L	L	L
H	H	X	L	L	L	L	L
X	X	H	H	H	L	L	H
X	L	X	L	COUNT			
L	X	L	X	COUNT			
L	X	X	L	COUNT			
X	L	L	X	COUNT			

NOTES A. Output Q_A is connected to input B for BCD count.
B. Output Q_D is connected to input A for bi-quinary count.

FIGURE 12–14 The block diagram, pinout, and truth table for the 7490 decade counter.

SOURCE: Courtesy of Texas Instruments, Inc.

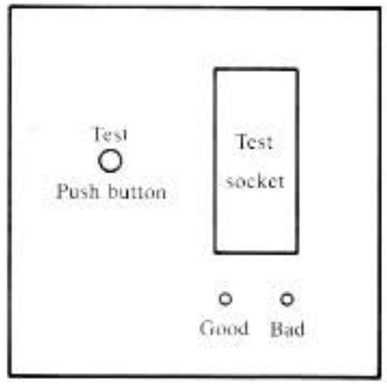


FIGURE 12–15 The control panel of the microprocessor based TTL integrated circuit tester.

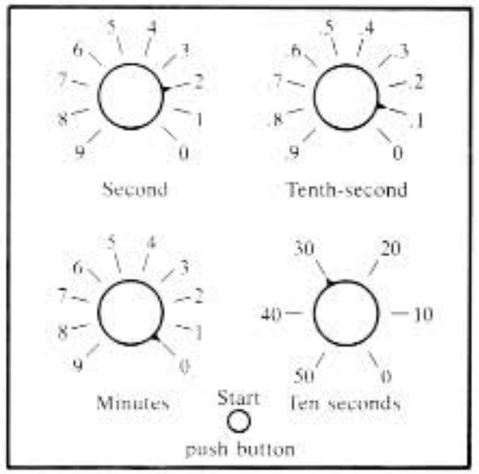


FIGURE 12–16 The control panel of the microprocessor based dark room timer.

3 If the above system is to be able to test any 14-pin integrated TTL circuit, which changes would have to be made to the hardware?

4 Create a darkroom timer that will control the length of time that the enlarger exposes the paper. The timer must be capable of exposing the paper in increments of 0.1 second up to 10 min.

Time settings are dialed in on a series of rotary switches that are labeled in one-tenth seconds, seconds, and minutes (as illustrated in figure 12–16). The push button starts the timing sequence that applies AC power to the lamp in the enlarger for the preset amount of time.

5 Your neighbor's son is a cub scout and wants you to build a timer for the annual pinewood derby. This box must be able to determine who wins each heat and to display the winning time on a set of LED numeric readouts.

Figure 12–17 pictures the ramp, which accommodates two cars at one time, and the location of the beginning and ending trip points.

Your software should start timing when either of the first trip points is tripped and continue timing until either of the second trip points is tripped. The hardware should indicate who has won the race and should light up the elapsed time on a set of displays.

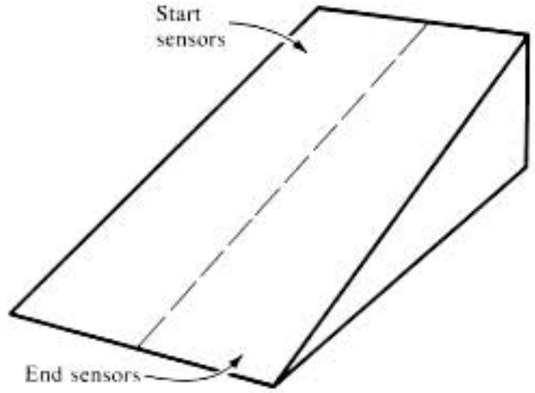


FIGURE 12–17 The ramp for the pine wood derby.

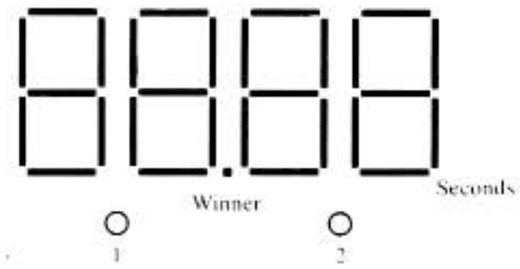


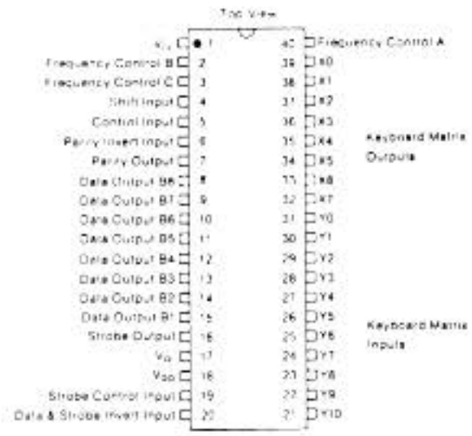
FIGURE 12–18 The display panel on the microprocessor based pine wood derby.

Figure 12–18 pictures the layout of the displays and the winner indicator light.

6 Modify the system developed in question 5 so that it can accommodate a four lane ramp.

7-1 KEYBOARDS

Keyboards are interfaced in two different ways in most microprocessor based systems. The first method uses a keyboard encoder that detects keystrokes, converts the keystrokes into ASCII code, and signals the microprocessor that information is available. The second method requires an input port and an output port, which are used with some software to multiplex the keys in a keyboard matrix.



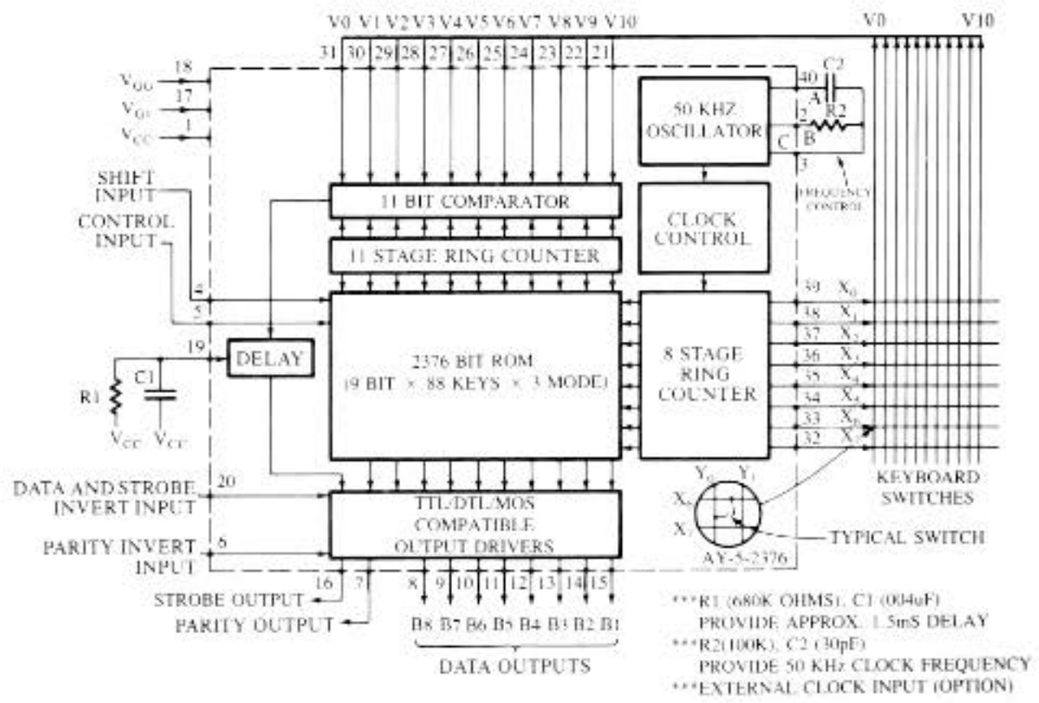


FIGURE 7–1 The pinout and block diagram of the AY-5-2376 keyboard encoder.
SOURCE: Courtesy of General Instruments, Inc.

Keyboard Encoder

The AY-5-2376, a typical keyboard encoder, is illustrated in figure 7–1. The keys are attached to the encoder through an eleven by eight keyboard matrix, which allows 88 keys to be connected to the encoder.

The encoder, under normal operation, scans the keyboard matrix for a key closure. Once the closure is detected, the internal circuitry addresses a ROM, which provides the ASCII address of the key at the data output connections. This information is not considered valid until the AY-5-2376 generates the output strobe signal after time is allowed for the key switch to stop bouncing.

In addition to the 88 key switches connected in the keyboard matrix, 2 additional switch connections accomplish the shift and the control functions. These additional inputs select different ASCII codes for the key switches. The internal ROM is a 264 word read only memory, which provides three sets of ASCII codes, depending upon the conditions of shift and control.

Keyboard Encoder to 8155 Interface

Figure 7–2 pictures the AY-5-2376 connected to an 8155 peripheral interface adapter. The strobe output, which becomes active after a valid keystroke, strobes the keyboard data into the I/O port for use by the microprocessor. Once the software detects this event, data is input to the microprocessor and

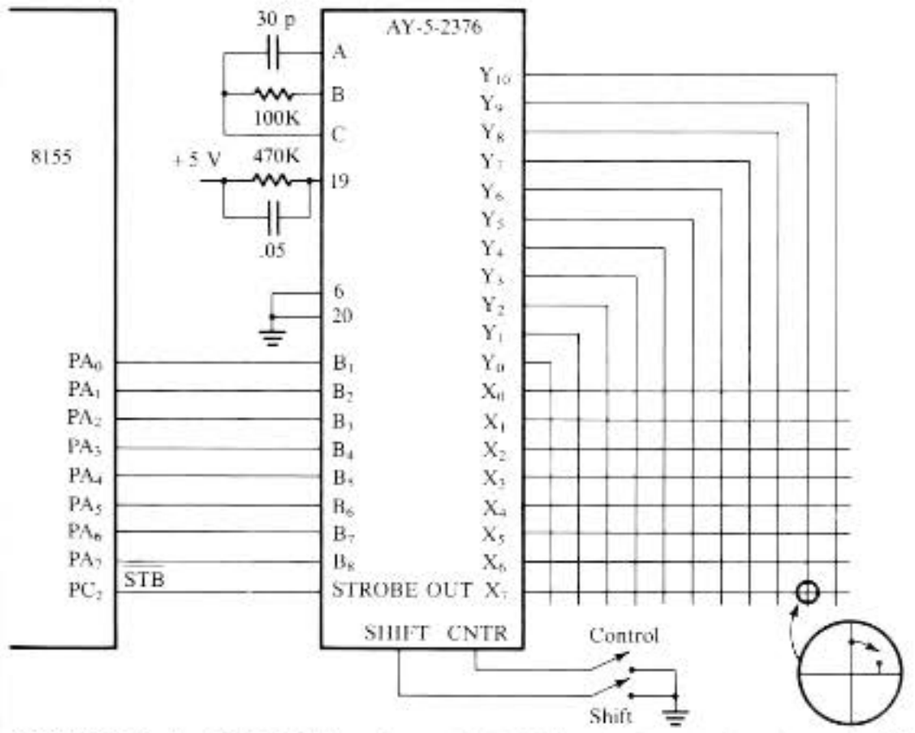


FIGURE 7–2 An AY-5-2376 interface to the 8155 through Port A using the strobed input mode of operation.

the I/O port is again ready for another byte of information from the keyboard.

Pins A, B, and C on the AY-5-2376 are used as timing inputs for an internal oscillator. This oscillator times the basic keyboard scanning rate. The SC pin connection develops a time delay internally, which debounces the keys on the keyboard.

The subroutine that is used to test the AY-5-2376 for data follows:

```
1  ;THIS SUBROUTINE CHECKS FOR KEYBOARD DATA
2  ;IF DATA IS FOUND IT RETURNS WITH IT IN THE ACC
3  ;IF NO DATA IS FOUND IT WAITS FOR THE DATA
4  ;
5  ;THE ACC AND FLAGS ARE DESTROYED
6  ;
7  INKEY:  IN    STATUS   ;GET THE BUFFER FULL FLAG
8          ANI   02H      ;ISOLATE ABF
9          JZ    INKEY    ;LOOP IF THERE IS NO DATA
10         IN    PORTA    ;INPUT THE ASCII DATA
11         RET             ;RETURN FROM THE SUBROUTINE
```

Hexadecimal Keypad Interface

The keyboard encoder is only used when a full keyboard is connected to the microprocessor. Most applications do not require a complete keyboard, so this circuit is not found. In its place you would probably find the circuit of figure 7–3 with a small keyboard matrix of 16 keys.

For this interface to fit many different types of parallel interface adapters, the diagram identifies only port A and port B.

This keyboard is organized as a 2-by-8-bit matrix. Port A must be programmed as an input port, while the 2 bits used in port B must be programmed as outputs. This is accomplished with the initialization dialog discussed in the last chapter and with the subroutines for this circuit.

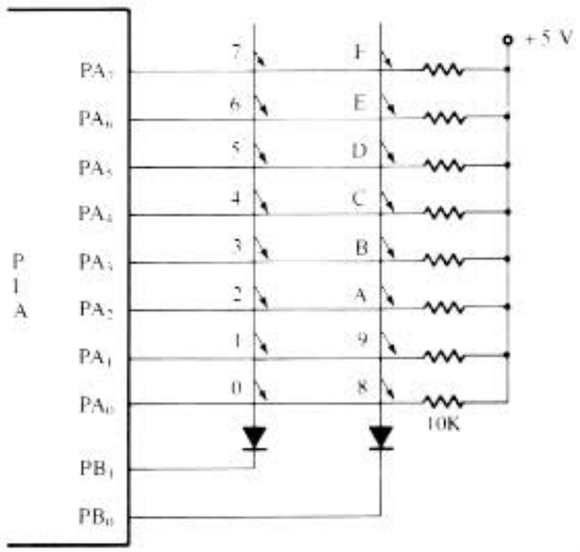


FIGURE 7–3 A hexadecimal keypad connected to Port A of a PIA.

The subroutines that will scan the keyboard must be capable of selecting a column of eight keys, detecting if any of the eight keys is depressed, debouncing the keystroke, and providing a code to identify the key's location. The flowchart provided in figure 7–4 illustrates this sequence of events.

8085A Keypad Software

When developing the software for this application, binary bit patterns 0000 0010 and 0000 0001 are chosen as codes to select the columns, and binary bit patterns 0000 0000 and 0000 1000 are chosen as an indicator for the first key in the selected column.

The time required for debouncing the keys depends upon the type of push button switches selected for the keyboard. In general, push button switches will stop bouncing after 10–20 ms.

The 8155 is initialized by programming the command register so that port A is an input port and port B is an output port. The initialization dialog is placed at the start of the system software at the reset location.

```
1 ;INITIALIZATION DIALOG FOR THE 8155 KEYBOARD INTERFACE
2 ;
3 RESET: MVI   A,00000010B ;SET PORT A = INPUT
4        OUT   COMMAND     ;SET PORT B = OUTPUT
```

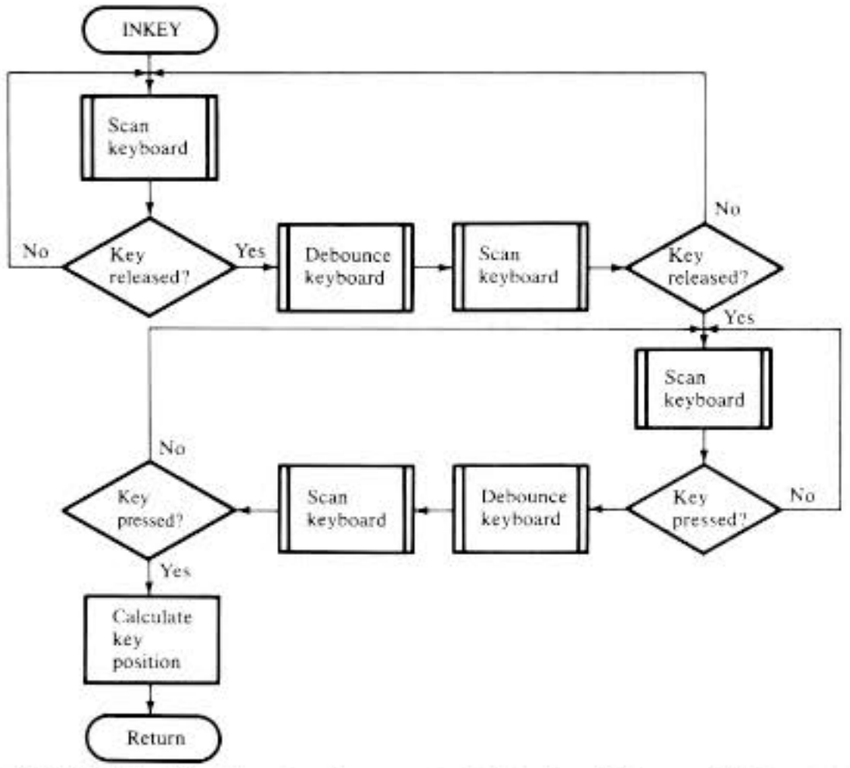


FIGURE 7–4 The flowchart for scanning the keyboard illustrated in figure 7–3.

After the 8155 is initialized, it can be controlled to scan the keyboard. The INKEY subroutine that scans this keyboard follows:

```
 1 ;8085A ASSEMBLY LANGUAGE VERSION
 2 ;SUBROUTINE TO DETECT A KEYSTROKE AND RETURN
 3 ;WITH THE KEY CODE IN THE C-REGISTER.
 4 ;
 5 ;ALL REGISTERS EXCEPT HL ARE DESTROYED
 6 ;USES THE SCAN AND DELAY SUBROUTINES
 7 ;
 8 ;CHECK FOR KEY RELEASE
 9 INKEY: CALL SCAN      ;CHECK ALL KEYS
10        JNZ  INKEY     ;IF KEY IS DEPRESSED
11        CALL DELAY     ;DEBOUNCE
12        CALL SCAN      ;CHECK ALL KEYS
13        JNZ  INKEY     ;IF KEY IS DEPRESSED
14 ;CHECK FOR A KEY
15 LOOP:  CALL SCAN      ;CHECK ALL KEYS
16        JZ   LOOP      ;IF NO KEY IS DEPRESSED
17        CALL DELAY     ;DEBOUNCE
18        CALL SCAN      ;CHECK ALL KEYS
19        JZ   LOOP      ;IF IT WAS NOISE
20 ;DETERMINE WHICH KEY WAS DEPRESSED
21 LOOP1: RRC            ;LOCATE ROW
22        RNC            ;RETURN IF FOUND
23        INR  C         ;MODIFY KEY CODE
24        JMP  LOOP1     ;CONTINUE TO LOOK
```

Lines 9 through 13 in the 8085A version of the keyboard software check whether the previous key has been released. This check is necessary because the software that uses this subroutine may call it before the person using the keyboard has had time to remove a finger from the button. If the key is released, lines 15 through 19 scan, or search, for another key closure. Once a key closure is detected, the subroutine searches the binary bit pattern for the closed contact; as it does, it modifies the key code in the C-register. When the code of the keystroke has finally been calculated, a return occurs with C equal to the key's code number.

```
24 ;20 MSEC. TIME DELAY SUBROUTINE
25 ;CLOCK CYCLE TIME = 333 NSEC.
26 ;ACC, F, D AND E ARE DESTROYED
27 ;
28 DELAY:  LXI  D,1568D  ;LOAD COUNT
29 DELAY1: DCX  D        ;DECREMENT COUNT
30         MOV  A,D      ;TEST DE FOR A ZERO
31         ORA  E
32         JNZ  DELAY1   ;COUNT = ZERO?
33         RET
```


The amount of time used for the contact debounce delay is left up to the user, since it varies with different switches. The count 1568 in the DELAY subroutine is chosen for a 20 ms time delay for this example.

```
34 ;KEYBOARD SCANNING SUBROUTINE
35 ;MODIFIES B AND C, DESTROYS ACC AND F
36 ;RETURN ZERO = NO KEYSTROKE
37 ;RETURN NOT ZERO = KEYSTROKE
38 ;
39 SCAN:  MVI  A,02H     ;SELECT A COLUMN
40       MVI  C,00H     ;SET ROW STARTING KEY CODE
41       OUT  PORTB
42       IN   PORTA     ;CHECK ROWS
43       CPI  0FFH
44       RNZ             ;RETURN ON KEY
45       MVI  A,01H     ;SELECT NEXT COLUMN
46       OUT  PORTB
47       MVI  C,08H     ;SET ROW STARTING KEY CODE
48       IN   PORTA     ;CHECK ROWS
49       CPI  0FFH
50       RET
```

The keyboard scanning subroutine selects a column by modifying the data at port B. Once a column of eight keys is selected, port A is input and checked for a keystroke. If 1 or more bits are logic zeros at this time, it indicates that a key is depressed and the subroutine returns with the accumulator containing the row bit pattern. If no key is depressed, the column selection bit pattern and the row beginning key code are modified and the next column of eight keys is checked.

6800 Keypad Software

To implement the hex keypad with the MC6800 and MC6821 PIA, the PIA must first be programmed or initialized at the reset location for the system program. The dialog that follows programs port A as an input port and port B as an output port.

```
 1 *6821 HEX KEYPAD INITIALIZATION DIALOG
 2 *
 3 RESET  CLR  CRA    SELECT PORT A DDR
 4        CLR  PORTA  PORT A = INPUT
 5        CLR  CRB    SELECT PORT B DDR
 6        LDAA #$FF   PORT B = OUTPUT
 7        STAA PORTB
 8        LDAA #$04
 9        STAA CRA    SELECT PORT A DATA REGISTER
10        STAA CRB    SELECT PORT B DATA REGISTER
            .
            .
            .
```

The keypad scanning subroutine, which follows, checks to see whether a key is released. This is done because the software jumping to this subroutine may execute in a very short period of time. If it jumps to the subroutine before the operator releases the key, multiple keystrokes are entered into the system. Once the key is released, the INKEY subroutine detects which key has been pressed and returns with the code of the key in accumulator B.

```
    *6800 ASSEMBLY LANGUAGE VERSION
    *SUBROUTINE TO DETECT A KEYSTROKE AND RETURN
    *WITH THE KEY CODE IN ACCUMULATOR B.
    *
    *WAIT FOR KEY RELEASE
    INKEY   JSR     SCAN      CHECK ALL KEYS
            BNE     INKEY     IF KEY IS DEPRESSED
            JSR     DELAY     DEBOUNCE
            JSR     SCAN      CHECK ALL KEYS
            BNE     INKEY     IF KEY IS DEPRESSED
    *WAIT FOR A NEW KEYSTROKE
    LOOP    JSR     SCAN      CHECK ALL KEYS
            BEQ     LOOP      IF NO KEY DEPRESSION
            JSR     DELAY     DEBOUNCE
            JSR     SCAN      CHECK ALL KEYS
            BEQ     LOOP      IF NO KEY DEPRESSION
    *DETERMINE KEY CODE
    LOOP1   LSRA              LOCATE KEYSTROKE
            BCC     RET       RETURN WHEN FOUND
            INCB              MODIFY KEY CODE
            JMP     LOOP1     KEEP CHECKING
```

The time delay subroutine uses nested loops to achieve a time delay of 20 ms. This time delay is required to debounce the mechanical key switches in the keyboard matrix.

```
    *20 MSEC. TIME DELAY SUBROUTINE
    *
    DELAY   LDAA    #$14      LOAD COUNT
    DELAY1  LDAB    #$A5
    DELAY2  DECB              DECREMENT B COUNT
            BNE     DELAY2    COUNT B = ZERO?
            DECA              DECREMENT A COUNT
            BNE     DELAY1    COUNT A = ZERO?
            RTS               RETURN FROM DELAY
```

The SCAN subroutine selects a column of eight keys and determines whether or not a key is depressed. If a key is detected, a return equal occurs; if no key is detected, a return not equal occurs.

```
    *CHECK FOR ANY KEY SUBROUTINE
    *RETURN EQUAL = KEYSTROKE DETECTED
    *RETURN NOT EQUAL = NO KEYSTROKE DETECTED
    *
    SCAN    LDAA    #$02      SELECT COLUMN
```



```
            STAA    PORTB
            CLRB              SET KEY CODE
            LDAA    PORTA     CHECK KEYS
            CMPA    #$FF      CHECK FOR KEY
            BNE     RET       RETURN ON KEY
            LDAA    #$01      SELECT COLUMN
            STAA    PORTB
            LDAB    #$08      SET KEY CODE
            LDAA    PORTA     CHECK KEYS
            CMPA    #$FF      CHECK FOR KEY
    RET     RTS               RETURN FROM SUBROUTINE
```

MULTIPLEXED DISPLAYS 7-2

Display devices are normally multiplexed to reduce the component count in a microprocessor based system. In microprocessors, the seven segment code is developed with software to further reduce the amount of external hardware required in the system.

BCD to Seven Segment Code Conversion

Code conversion from binary coded decimal to seven segment code is usually done via a table lookup subroutine. The BCD coded number forms the address of the seven segment coded character stored in a table in the memory. This method of code conversion is widely used because of its speed and relatively low cost. Table 7–1 illustrates the typical lookup table for a common anode seven segment display. The display and driver circuitry is pictured in figure 7–5. When a logic one is applied to the base of the segment driver, it becomes forward biased. This sinks current for the cathode of the display, which then lights.

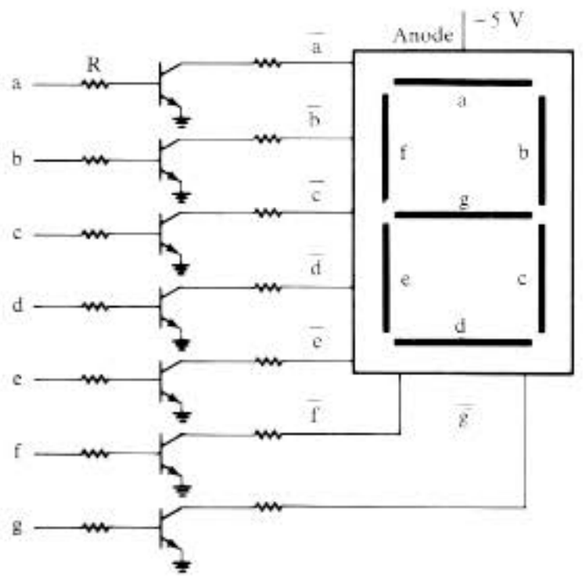


FIGURE 7–5 A seven segment LED display illustrating the segment drivers.

TABLE 7–1 Common anode seven segment lookup table.

Address	Data								Displayed Data
	X	a	b	c	d	e	f	g	
TABLE	0	1	1	1	1	1	1	0	0
TABLE+1	0	0	1	1	0	0	0	0	1
TABLE+2	0	1	1	0	1	1	0	1	2
TABLE+3	0	1	1	1	1	0	0	1	3
TABLE+4	0	0	1	1	0	0	1	1	4
TABLE+5	0	1	0	1	1	0	1	1	5
TABLE+6	0	1	0	1	1	1	1	1	6
TABLE+7	0	1	1	1	0	0	0	0	7
TABLE+8	0	1	1	1	1	1	1	1	8
TABLE+9	0	1	1	1	1	0	1	1	9

8085A Table Lookup Software

Software to convert the unpacked or single BCD digit in the accumulator of an 8085A into a seven segment coded number follows:

```
1   ;8085A ASSEMBLY LANGUAGE PROGRAM
2   ;SUBROUTINE TO CONVERT THE ACCUMULATOR FROM
3   ;BCD TO SEVEN SEGMENT CODE
4   ;HL IS DESTROYED
5   ;REFERENCES TABLE 7-1
6   ;
7   CONVERT: ANI   0FH      ;MASK LEFT NIBBLE
8            LXI   H,TABLE  ;POINT TO LOOKUP TABLE
9            ADD   L        ;ADD BCD TO ADDRESS (HL)
10           MOV   L,A
11           MOV   A,H
12           ACI   00H
13           MOV   H,A
14           MOV   A,M      ;GET SEVEN SEGMENT CODE
15           RET
```

6800 Table Lookup Software

Software to convert the contents of accumulator B in the MC6800 from a single unpacked BCD digit into seven segment code follows:

```
1   *6800 ASSEMBLY LANGUAGE PROGRAM
2   *SUBROUTINE TO CONVERT ACCUMULATOR B FROM BCD
3   *INTO SEVEN SEGMENT CODE.
4   *X IS DESTROYED
5   *REFERENCES TABLE 7-1
6   *
7   CONVERT: ANDB    #$0F    MASK LEFT NIBBLE
8            LDX     #TABLE  GET TABLE ADDRESS
```



```
 9            STX     TEMP    SAVE TABLE ADDRESS
10            ADDB    TEMP+1  DEVELOP ADDRESS
11            STAB    TEMP+1
12            BCC     CONV1
13            INC     TEMP
14   CONV1    LDX     TEMP    GET TABLE ADDRESS
15            LDAB    X       GET 7 SEGMENT CODE
16            RTS
```

Location TEMP in the above software is 2 bytes of memory somewhere in the base page. This reduces the length of this subroutine. The extra work allows this subroutine to be stored in a ROM. If a ROM will not be used, the subroutine can be shortened considerably.

Multiple Digit Display

The table lookup technique for code conversion, along with other software, multiplexes the two digit display illustrated in figure 7–6. Port A supplies both displays with seven segment code through a set of drivers, and port B selects either digit zero or digit one. Again the type of peripheral interface adapter is not specified, so that any can be utilized.

Port A provides seven segment data for both displays through a set of current amplifiers. These amplifiers are required to provide enough drive current for the displays, which typically require 10 mA per segment. Since this is a two digit multiplexed display, each display segment requires twice this amount of current to remain illuminated at normal intensity. A three digit display requires three times the current, and so forth.

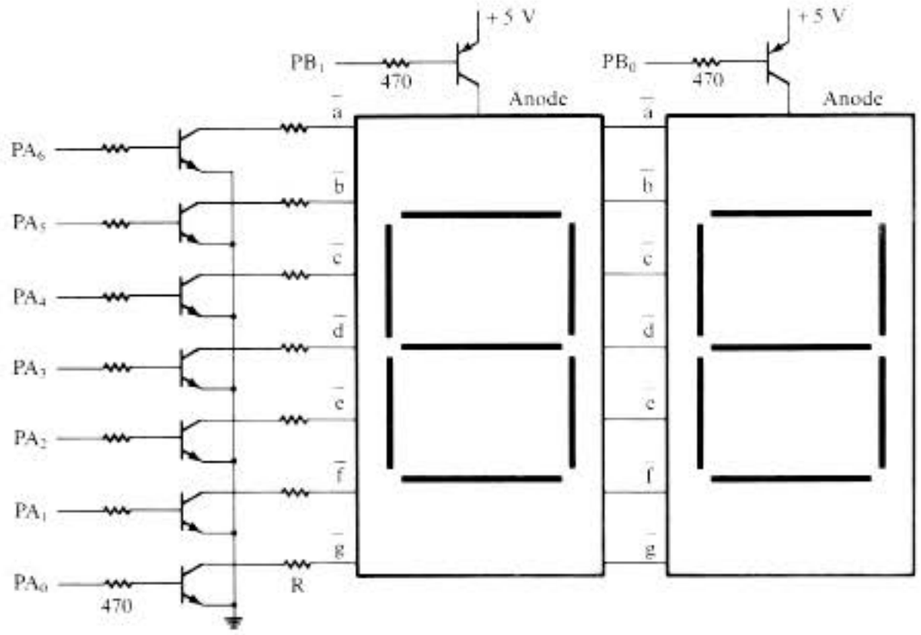


FIGURE 7–6 A two digit multiplexed seven segment LED display.

The software developed to drive the displays will make one pass only; that is, it will display each digit only one time. It is the responsibility of the software using this subroutine to call it continually to maintain a constantly displayed number. If you wish to do quite a bit of processing between calls, it is important to blank the displays to prevent damage. The displays may be blanked by turning off both displays.

Figure 7–7 illustrates the flowchart for the DISP subroutine. Port B selects the digit that displays the information at port A. The two "digit" selection

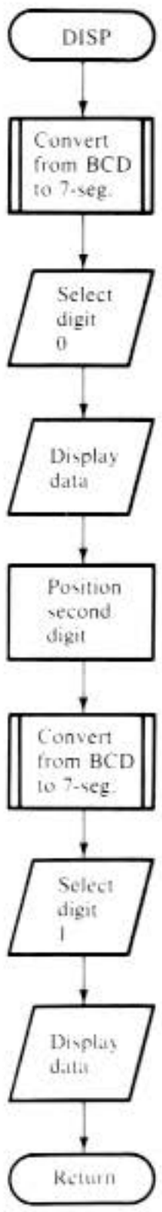


FIGURE 7–7 A flowchart of the subroutine required to multiplex the two LED displays pictured in figure 7–6.

pins at port B are connected to transistor switches that select a digit. These switches must be capable of passing the current from all seven segments in the selected display. In this circuit that amounts to 140 mA peak for each seven segment display, with an average current of 70 mA.

The subroutine that causes the 1 ms time delay is not illustrated but can be developed in the same manner as the DELAY subroutine in the section on keyboards. The DELAY subroutine is included to reduce the switching time to the displays. Without it, RF is generated and propagated from the displays, causing a problem with the Federal Communications Commission (FCC).

8085 Version of the Display Software

Before the display can be used, the 8155 must be programmed. In this application, ports A and B must be programmed as output ports for the display. As with the prior software, the initialization dialog is found at the reset location.

```
1 ;8155 INITIALIZATION DIALOG
2 ;
3 RESET: MVI   A,00000011B  PROGRAM PORT A & B
4        OUT   COMMAND      AS OUTPUT PORTS
           .
           .
           .
```

Once the 8155 is programmed, the DISP subroutine can be used whenever data is to be displayed on the two digit display.

```
 1  ;8085 ASSEMBLY LANGUAGE PROGRAM
 2  ;SUBROUTINE TO DISPLAY THE PACKED BCD NUMBER
 3  ;IN THE ACCUMULATOR ON THE TWO DIGIT DISPLAY.
 4  ;
 5  DISP:   PUSH PSW        ;SAVE BCD
 6          CALL CONVERT    ;CONVERT TO SEVEN SEG.
 7          OUT PORTA       ;SEND DATA
 8          MVI A,02H       ;SELECT DIGIT 0
 9          OUT PORTB
10          CALL DELAY      ;WAIT 1 MSEC.
11          POP PSW         ;GET BCD
12          RRC             ;POSITION NEXT DIGIT
13          RRC
14          RRC
15          RRC
16          CALL CONVERT    ;CONVERT TO SEVEN SEG.
17          OUT PORTA       ;SEND DATA
18          MVI A,01H       ;SELECT DIGIT 1
19          OUT PORTB
20          CALL DELAY      ;WAIT 1 MSEC.
21          RET             ;RETURN FROM DISP
```

6800 Version of the Display Software

Before the display can be used, the MC6821 must be programmed. In this application ports A and B must be programmed as output ports for the display. As with the prior software, the initialization dialog is found at the reset location. Steps 3 and 4 are only required if the MC6821 is not reset. This may be the case in some systems; so it may be better to include these steps as a matter of practice.

```
 1 *6821 INITIALIZATION DIALOG
 2 *
 3 RESET CLR  CRA          SELECT DDR PORT A
 4       CLR  CRB          SELECT DDR PORT B
 5       LDAA #$FF         SET ALL BITS TO OUTPUT
 6       STAA DDRA         PROGRAM PORT A
 7       STAA DDRB         PROGRAM PORT B
 8       LDAA #$04         SELECT DATA FOR PORT A
 9       STAA CRA
10       STAA CRB          SELECT DATA FOR PORT B
            .
            .
            .
```

After the MC6821 is programmed, the DISP subroutine can be used whenever data is to be displayed on the two digit display.

```
 1  *6800 ASSEMBLY LANGUAGE PROGRAM
 2  *SUBROUTINE THAT TAKES THE PACKED BCD FROM
 3  *ACC B AND DISPLAYS IT ON THE DISPLAYS.
 4  *
 5  DISP    PSHB              SAVE BCD DATA
 6          JSR   CONVERT     CONVERT TO SEVEN SEG.
 7          STAB  PORTA       SEND DATA
 8          LDAB  #02         SELECT DIGIT 0
 9          STAB  PORTB
10          JSR   DELAY       WAIT FOR 1 MSEC.
11          PULB              RESTORE BCD
12          LSRB              POSITION NEXT DIGIT
13          LSRB
14          LSRB
15          LSRB
16          JSR   CONVERT     CONVERT TO SEVEN SEG.
17          STAB  PORTA       SEND DATA
18          LDAB  #01         SELECT DIGIT 1
19          STAB  PORTB
20          JSR   DELAY       WAIT FOR 1 MSEC.
21          RTS               RETURN FROM DISP
```

tive for any application requiring this type of sensitivity. The RST 6.5, RST 5.5, and INTR inputs are level sensitive; they must be held at their active levels until they are recognized at the end of the current instruction. The time required to recognize these three inputs varies with different instructions and clock speeds of the 8085A. It is also important to note that the HOLD input causes an interrupt to be delayed until after the HOLD condition has ended.

The INTR Input and $\overline{\text{INTA}}$ Output

The INTR input does not call an interrupt service subroutine directly. Instead the 8085A issues an $\overline{\text{INTA}}$ pulse when this input is acknowledged, as illustrated in figure 8–3. It is the designer's responsibility to add hardware that will force an instruction onto the data bus in response to the $\overline{\text{INTA}}$ output of the 8085A. For most applications, a RST 1 through RST 7 is forced onto the data bus; on occasion, a CALL instruction is. (Note that the RST 0 instruction is normally used for a software and hardware RESET.) Figure 8–4 pictures the application of a RST 5 in response to an INTR interrupt request. The RST 5 instruction, an EFH, is hardwired to the inputs of the eight three-state buffers. Whenever the INTR input is placed at the logic one level requesting an interrupt, the microprocessor responds with an $\overline{\text{INTA}}$ pulse. This procedure enables the buffers and applies the EFH or RST 5 op-code on the data bus. The microprocessor responds by executing the RST 5 or it calls the subroutine that begins at memory location 28H.

8-3 MC6800 AND MC6809 INTERRUPT STRUCTURE

The MC6809 has three interrupt inputs: one is a nonmaskable interrupt input, $\overline{\text{NMI}}$, and the others are maskable interrupts, $\overline{\text{IRQ}}$ and $\overline{\text{FIRQ}}$. The MC6800 has all the same inputs except the $\overline{\text{FIRQ}}$. The $\overline{\text{NMI}}$ input causes the MC6809 to look to memory locations $FFFC and $FFFD for the address of the interrupt service subroutine. The $\overline{\text{IRQ}}$ input uses $FFF8 and $FFF9, and the $\overline{\text{FIRQ}}$ input uses $FFF6 and $FFF7 for their service subroutine vectors.

When the interrupt input is accepted by the MC6809 or MC6800, it automatically saves the contents of all internal registers on the stack and looks to the appropriate interrupt vector for the starting location of the interrupt service subroutine. The exception to this rule is the $\overline{\text{FIRQ}}$, or *fast interrupt request* input, which only saves the contents of the program counter and condition code register.

At the end of the interrupt service subroutine, a special return instruction (RTI) reloads the registers saved on the stack and returns to the program that was interrupted. This return instruction is different from RTS, which does not restore any register but the program counter. An extra flag bit in the status register indicates whether the interrupt is a $\overline{\text{FIRQ}}$ or normal interrupt for the MC6809. This is looked at by RTI to determine which registers must be unloaded from the stack.

In the MC6800 the $\overline{\text{IRQ}}$ interrupt input is enabled by the CLI instruction and disabled by the SEI instruction. These instructions control the interrupt

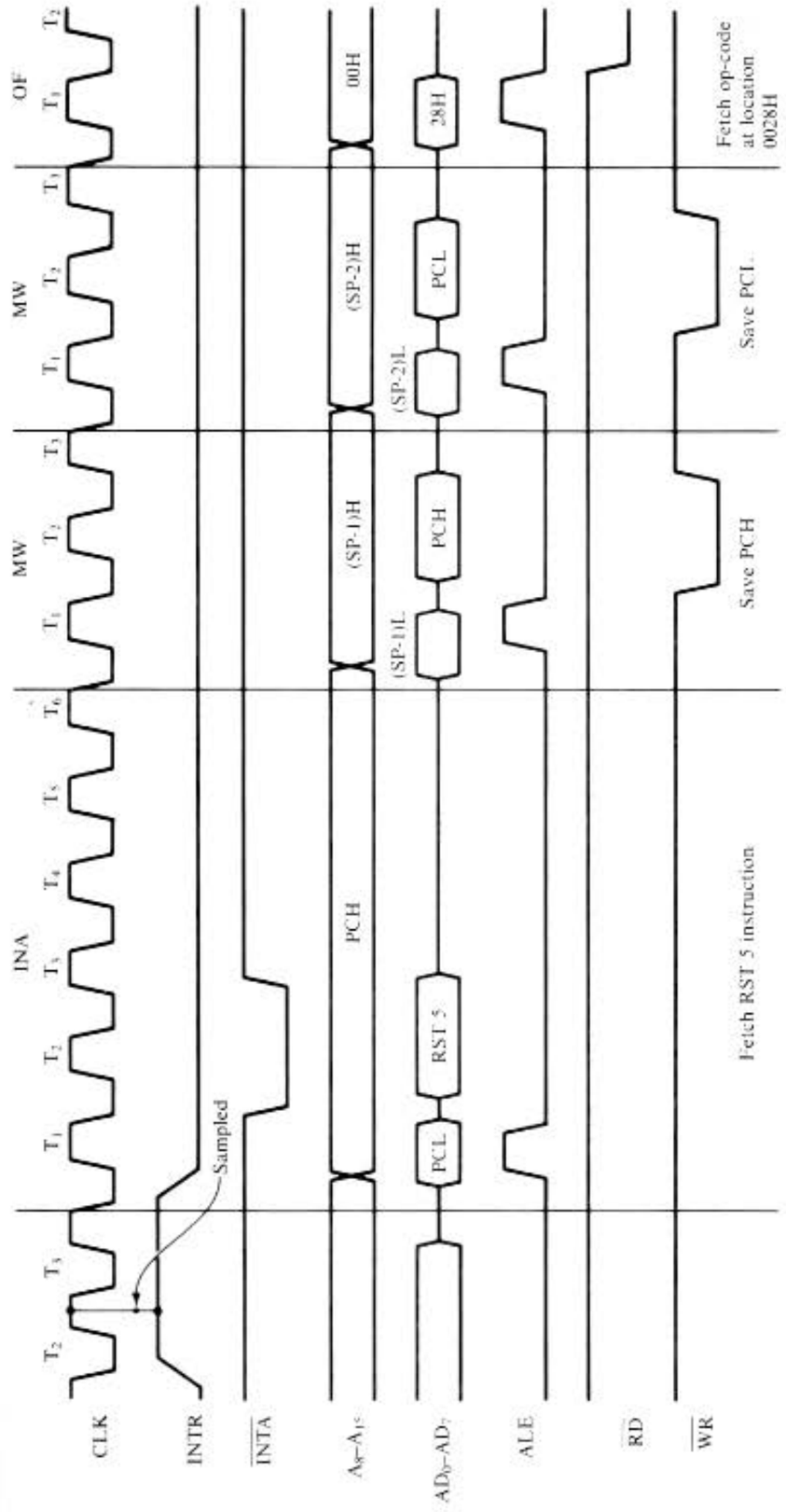


FIGURE 8–3 The timing diagram for an INTR showing the RST 5 instruction in response to the interrupt.

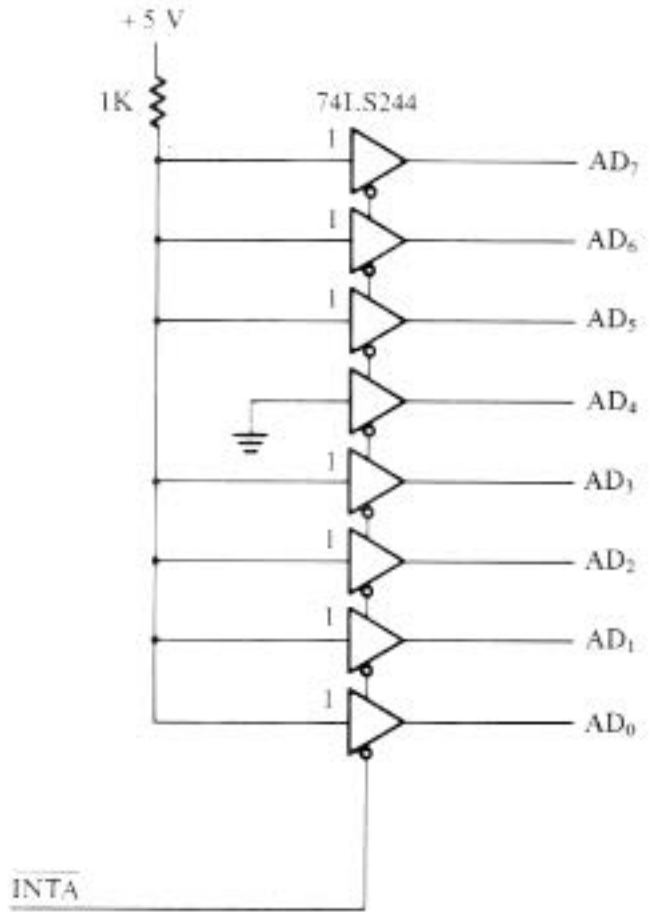


FIGURE 8–4 A circuit that will cause a RST 5 instruction to be gated onto the 8085A data bus in response to an INTR.

enable bit (I) in the condition code register, which in turn controls whether or not the interrupt is accepted by the microprocessor.

In the MC6809, the (I) and (F) interrupt masks are controlled by the ORCC instruction, which sets or disables them, and the ANDCC instruction, which clears or enables these bits. The (F) condition code bit controls the $\overline{\text{FIRQ}}$ input, and the (I) condition code bit controls the $\overline{\text{IRQ}}$ input.