

chip selects will be manufactured into the device. They will be tied to the address bus from the MPU in such a manner that only one ROM is addressed at a time. To select a ROM, a +2-V (or greater) signal must be applied to each positive chip select (CS) and a 0-V level to each negative one ($\overline{\text{CS}}$). The addressing scheme for connecting the chip selects to the address bus will be covered in great detail in Chap. 10. When a ROM is not addressed, the ROM data bus goes into the three-state condition (high impedance).

A functional diagram of the MCM6830 ROM is shown in Fig. 9.13.

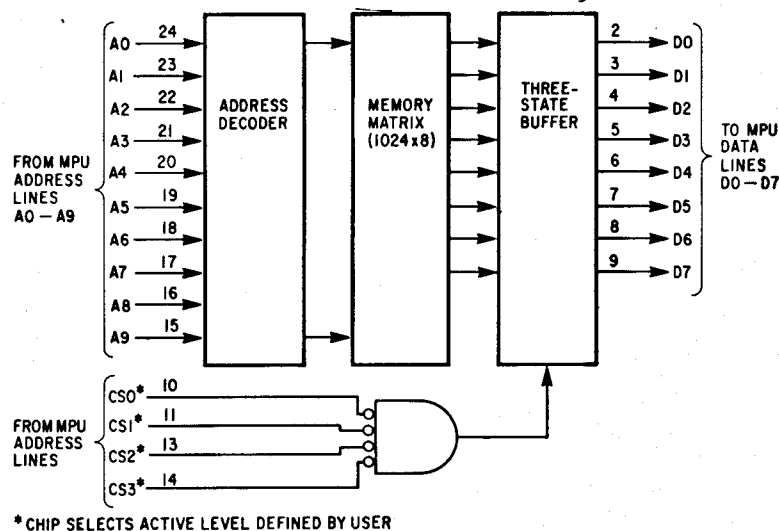


Fig. 9.13 MCM6830 ROM functional block diagram

9.5 Peripheral Interface Adapter (PIA)

The MC6821 Peripheral Interface Adapter (PIA) is an N-MOS device housed in a 40-pin package and used as a means of interfacing peripheral equipment and external signals with the MPU (Fig. 9.14). The PIA communicates with the MPU through the same eight-bit bidirectional data bus that the RAMs and ROMs share. The PIA has two separate eight-bit bidirectional peripheral data busses for interfacing to the outside world. The 16 bidirectional input/output lines may be programmed to act as either input or output lines (Fig. 9.15).

In addition to the 24 lines shown in Fig. 9.15, there are three chip select pins (CS0, CS1, and $\overline{\text{CS2}}$), a reset pin ($\overline{\text{RES}}$), two interrupt pins ($\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$), a read/write Pin (R/W), four control line pins (CA1, CA2, CB1, and CB2), an enable pin (E), two register select pins (RS0 and RS1), and two input power pins (+5 and ground), as shown in Fig. 9.16.

The MC6821 PIA has two sides, an A side and a B side. Each side has a *peripheral data register*, a *data direction register*, and a *control register*.

Each peripheral data register is the interface register between the PIA chip and the outside world. This register is eight bits (one byte) wide.

The data direction register is used by the programmer to define each peripheral line as an input or an output line. When each bit in this eight-bit

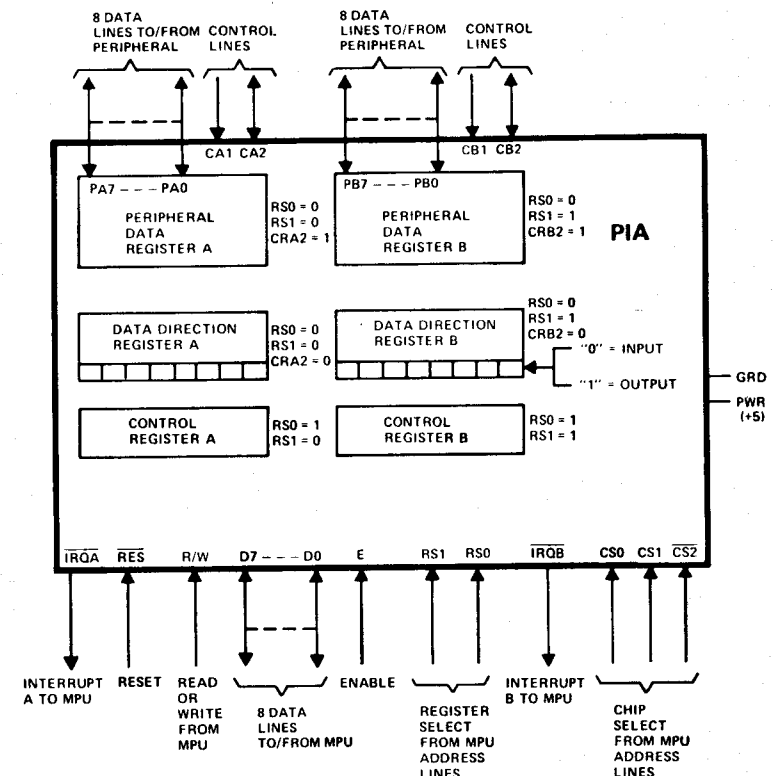


Fig. 9.14 MC6821 PIA

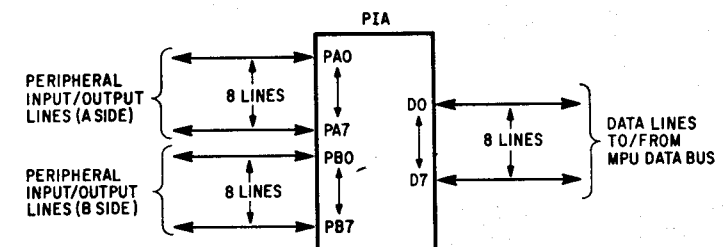


Fig. 9.15 MC6821 PIA input/output lines

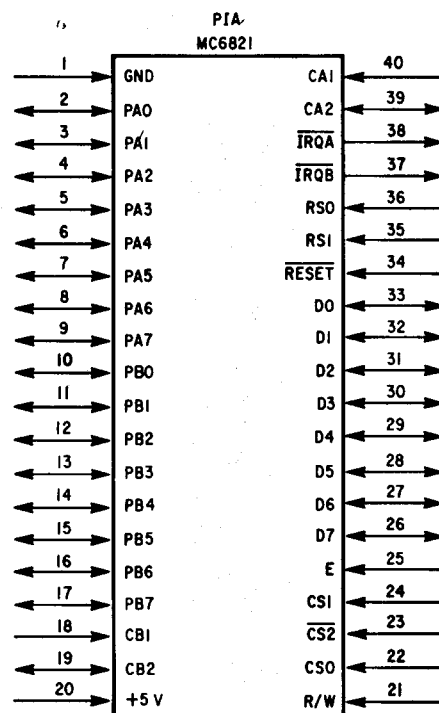


Fig. 9.16 MC6821 package

register is set to a "1", the corresponding peripheral data line is defined as an output line; when set to a "0", the corresponding peripheral data line is defined as an input line.

The control register is used to permit the MPU to control the operation of the four peripheral control lines CA1, CA2, CB1, and CB2. This register is also used to control the interrupt lines and monitor the status of the interrupt flags. Bit 2 of this register is used, in conjunction with the register selects, to determine whether the peripheral data register or the data direction register is to be addressed.

PIA Interface Lines

1. Peripheral Data Lines PA0 through PA7 Each of these eight data lines interfacing with the outside world can be programmed to act as either an input or an output by setting a "1" in the corresponding bit in the data direction register (DDR) if the line is to be an output or a "0" in the DDR if it is to be an input. When the data in the peripheral data lines is read into the MPU by a load instruction, those lines which have been designated as input lines (0 in DDR) will be gated directly to the data bus and into the register

selected in the MPU. In the input mode, each line represents a maximum of 1.5 standard TTL load.

On the other hand, when an output data instruction (STA A PIA) is executed, data will be transferred via the data bus to the peripheral data register. A "1" output will cause a "high" on the corresponding data line, and a "0" output will cause a "low." Data in peripheral register A that has been programmed as output may be read by an MPU "LDA A from PIA" instruction. If the voltage is above 2 V for a logic "1" or below 0.8 V for a logic "0", the data will agree with the data outputted. However, if these output lines have been loaded so that they do not meet the levels for logic "1", the data read back into the MPU may differ from data stored in PIA peripheral register A.

2. Peripheral Data Lines PB0 through PB7 The eight data lines interfacing with the outside world on the B side may also be programmed to act either as an input or output by setting a "1" in the corresponding bit in the data direction register (DDR) if the line is to be an output or a "0" in the DDR if it is to be an input. The output buffers driving these lines have three-state capability, allowing them to enter a high-impedance state when the peripheral data line is used as an input. Data in peripheral register B that has been programmed as output may be read by an MPU "LDA A from PIA" instruction even though the lines have been programmed as outputs. If a line is first programmed as an output line by storing a "1" in data direction register B and then storing a "1" in that same bit position in peripheral data register B, reading the bit status back will indicate a "1" even though excess loading (possibly due to a short) may have occurred at the pin. This is because of the buffering between the register and the output pin.

3. Data Lines (D0 through D7) The eight bidirectional data lines permit transfer of data to/from the PIA and the MPU. The MPU both sends and receives data to and from the outside world through the PIA via these eight data lines. The data bus output drivers are three-state devices that remain in the high-impedance (off) state except when the MPU performs a PIA Read operation.

4. Chip Select Lines (CS0, CS1, CS2) These lines are tied to the address lines of the MPU. It is through them that a particular PIA is selected (addressed). For selection of a PIA, the CS0 and CS1 lines must be high and the CS2 must be low. After the chip selects have been addressed, they must be held in that state for the duration of the E (Enable) pulse, which is the only timing signal supplied by the MPU to the PIA. This enable pulse (E) is normally the $\phi 2$ clock. One of the address lines should be ANDed with the VMA line, with the output of the AND gate tied to a chip select.

5. Enable Line (E) The enable pulse (E) is the only timing signal supplied to the PIA by the MPU. Timing on all other signals is referenced to the leading or trailing edges of the E pulse.

6. Reset Line (RES) This line, which resets all registers in the PIA to a logical zero, is primarily used during a reset or power-on operation.

It is normally in the high state. The transition of high-to-low-to-high resets all registers in the PIA, causing the PA0-PA7, PB0-PB7, CA2 and CB2 to be inputs and disabling all interrupts.

7. *Read/Write Line (R/W)* This signal generated by the MPU controls the direction of the data transfers on the data bus. A low state on the PIA Read/Write line enables the input buffers, and data is transferred from the MPU to the PIA (MPU Write) on the falling edge of the E (ϕ_2) signal if the device has been selected. A high on the Read/Write line sets up the PIA for a transfer of data to the data bus (MPU Read). The PIA output buffers are enabled when the proper address and the enable pulse are present, thus transferring data to the MPU.

8. *Interrupt Request Lines (\overline{IRQA} and \overline{IRQB})* These lines, which interrupt the MPU either directly or indirectly through interrupt priority circuitry, are "open source" (no load device on the chip). They are capable of sinking a current of 3.2 mA from an external source, thereby permitting all interrupt request lines to be tied together in a "wired OR" configuration. Interrupts are serviced by a software routine that sequentially reads and tests, on a priority basis, the two control registers in each PIA for the interrupt flag bits (bits 6 and 7) that are set. (These control registers and the way in which the flag bits get set will be discussed shortly.) When the MPU reads the peripheral data register, the interrupt flags (bits 6 and 7) are cleared and the interrupt request is cleared.

These request lines (\overline{IRQA} and \overline{IRQB}) are active when low.

9. *Interrupt Input Lines (CA1 and CB1)* These lines are input only to the PIA and set the interrupt flag (bit 7) of the control registers in the PIA. Discussion of these lines in conjunction with the control register will follow.

10. *Peripheral Control Line (CA2)* This line can be programmed to act either as an interrupt input or peripheral output. As an output, it is compatible with standard TTL, and as an input, represents 1.5 standard TTL load. The function of this line is programmed with control register A (bits 3, 4, and 5).

11. *Peripheral Control Line (CB2)* This line may also be programmed to act as an interrupt input or peripheral output. As an input, it has greater than 1-megohm input impedance and is compatible with standard TTL. As an output, it is compatible with standard TTL and may also be used as a source of up to 1 mA at 1.5 V and thus to drive the base of a transistor switch directly. The function of this line is programmed with control register B (bits 3, 4, and 5).

Addressing

To access a PIA, a high state ("1") must be applied to CS0 and CS1 while a low state ("0") is applied to $\overline{CS2}$. The RS0 and RS1 pins are tied to

MPU address lines A0 and A1, respectively. Once the PIA has been accessed, the RS0 and RS1 input pins are used to select one of the six internal registers in the PIA. How is it possible to select one of six registers with only two input lines? This is the only purpose of bit 2 of the control registers. If bit 2 of control register A (CRA) is a "0", and RS0 and RS1 (from A0 and A1) are also "0", then data direction register A (DDRA) is addressed. The level of each bit in data direction register A (DDRA) defines whether each corresponding line of peripheral data register A is an input (if a "0") or an output (if a "1"). The following sequence of instructions will define bits 0 through 4 of peripheral data register A (PDRA) as inputs, and bits 5, 6, and 7 of this same register as outputs (address of the PIA is 4004 through 4007).

Address	Contents	Description
100	86	LDA A #% 11100000
101	11100000	
102	B7	STA A \$4004
103	40	
104	04	

The above series of instructions would be *part* of the initialization program that would be run after applying power since the control register has selected the DDR as opposed to the output register.

The next step after defining the individual peripheral lines on the A side of this PIA as inputs or outputs is to load a "1" into bit 2 of control register A (CRA). Normally, the remaining bits of this register would have data loaded into them during this same operation, but this matter will be discussed later. Since bit 2 is the only bit of control register A (CRA) that affects addressing, we will look at bit 2 only rather than complicate the issue at this time. If the previous example were continued, it might look as follows:

Address	Contents	Description
105	86	LDA A #% 00000100
106	00000100	
107	B7	STA A \$4005
108	40	
109	05	

This will store a "1" in bit 2 of control register A (CRA). Once it has been loaded, addressing hex location 4004, as we did initially, will access peripheral data register A (PDRA).

To summarize, using the above example, addressing hex location 4004 allows the MPU to communicate with data direction register A (DDRA) if bit 2 of control register A (CRA) is a "0". After this bit is put at a "1", addressing hex location 4004 allows the MPU to communicate with peripheral data register A (PDRA):

Summary of A Side Addressing

RS1	RS0	CRA (Bit 2)	Register Selected
0	0	0	Data direction register A
0	1	Doesn't matter	Control register A
0	0	1	Peripheral data register A

NOTE: CS0 and CS1 must be high while CS2 is low.

Addressing on the B side is handled in much the same way as on the A side. To address data direction register B (DDR B), RS0 is set equal to a "0" and RS1 is set to a "1" while bit 1 of central register B (CRB) is held at a "0" level. After the individual bits in data direction register B (DDRB) are loaded with "1s" or "0s" to define the individual peripheral data register B lines as inputs and outputs, a "1" is stored in bit 2 of control register B by setting RS0 to a "1" and RS1 to a "1". After the "1" has been stored in bit 2 of CRB, peripheral data register B will be addressed whenever RS0 is a "0" and RS1 is a "1".

Summary of B Side Addressing

RS1	RS0	CRA (Bit 2)	Register Selected
1	0	0	Data direction register B
1	1	Doesn't matter	Control register B
1	0	1	Peripheral data register B

To summarize, one of the functions of the initialization program that will be run immediately after powering up is to configure the PIAs. The following program would define lines 0-3 of the A side as inputs and lines 4-7 of the B side as inputs; the remaining lines on both sides would be outputs:

PIA Address: Hex 4004-4007
Output Lines: A Side: lines 4-7
 B Side: lines 0-3
Input Lines: A Side: lines 0-3
 B Side: lines 4-7

Address	Contents	Description
100	C6	LDA B IMMED
101	11110000	
102	F7	STA B EXTENDED
103	40	LOADS 11110000 IN
104	04	DDR A
105	C6	LDA B IMMED
106	00001111	
107	F7	STA B EXTENDED
108	40	LOADS 00001111
109	06	IN DDR B
10A	C6	LDA B IMMED
10B	00000100	
10C	F7	STA B EXTENDED
10D	40	SETS BIT 2
10E	05	IN CRA TO A "1"
10F	F7	STA B EXTENDED
110	40	SETS BIT 2 IN
111	07	CRB TO A "1"

CONTROL REGISTER A (CRA)							
7	6	5	4	3	2	1	0
IRQA1	IRQA2	CA2 Control		DDRA	CA1 Control		

CA1 Control (Bits 0 and 1)

Peripheral control line CA1 is an *input only* line. It may be used to cause an interrupt by setting the interrupt flag IRQA1 (bit 7) of control register A. Bits 0 and 1 of CRA are used to determine how the interrupt is to be handled. The IRQA1 flag (bit 7) of CRA will get set to a "1" *only* under one of the following conditions:







- 1. A negative transition on the CA1 line is detected *and* bit 1 of CRA is a "0".
- 2. A rising transition on the CA1 is detected *and* bit 1 of CRA is a "1".

(All other combinations will be ignored.)

Whether the IRQA1 flag is permitted to pull the $\overline{\text{IRQA}}$ line low, thus interrupting the MPU, depends upon the status of bit 0 of CRA. If this bit is a "0", the interrupt will be masked (disallowed).

Peripheral control line CA1 is summarized in Table 9.2.

Table 9.2 Summary of CA1 Control

Transition of input line CA1	Status of bit 1 in CRA (edge)	Status of bit 0 in CRA (mask)	IRQA1 (interrupt flag) -Bit 7 of CRA	Status of IRQA line (MPU interrupt request)
	0	0	1	Masked (remains high)
	0	1	1	Goes low (processor interrupted)
	1	0	1	Masked (remains high)
	1	1	1	Goes low (processor interrupted)
	1	—	0	Remains high
	0	—	0	Remains High

As seen in Table 9.2, bit 0 of DRA is the IRQA1 interrupt "mask programming bit." If bit 0 is a "0", setting the interrupt flag IRQA1 will not cause the interrupt line $\overline{\text{IRQA}}$ to go low. If bit 0 contains a "1", the $\overline{\text{IRQA}}$ line will be permitted to go low when IRQA1 gets set to a "1".

Bit 1 of CRA is the "edge programming bit." A "0" in bit 1 programs the interrupt flag IRQA1 (bit 7) to get set to a "1" in a *negative* transition of the CA1 line, and a "1" in bit 1 programs the flag to get set to a "1" in a *positive* transition. (NOTE: If the IRQA1 flag was set to a "1" during a period when bit 0 had masked all interrupts, the interrupt will be allowed when bit 0 of CRA is changed to a "1" by the MPU.)

Data Direction Register (DDR A) (Bit 2)

This bit, in conjunction with register select lines RS0 and RS1, is used to select either the peripheral data register or the data direction register, as follows:

RS1	RS0	CRA (Bit 2)	Register Selected
0	0	0	Data Direction Register A
0	1	Doesn't matter	Control Register A
0	0	1	Peripheral Data Register A

CA2 Control (Bits 3, 4, and 5)

As mentioned earlier, this line can be programmed to function as an interrupt input line or a peripheral output line. The status of bits 3, 4, and 5 of control register A determine how it is to function. Bit 5 determines whether the CA2 line is to be an interrupt input line or a peripheral output line. If bit 5 contains a "0", it will be used as an interrupt line; if bit 5 contains a "1", it will be used as an output line.







CA2 As an Interrupt Input Line (Bit 5 = "0") Bits 3 and 4 determine how the interrupt is to be handled. The IRQA2 flag (bit 6) of CRA will be set to a "1" only under one of the following conditions:

1. A negative transition on the CA2 line is detected *and* bit 4 is a "0".
2. A rising transition on the CA2 line is detected *and* bit 4 is a "1".

(All other combinations are ignored.)

The CA2 control is summarized in Table 9.3.

Table 9.3 Summary of CA2 Control

Transition of input CA2	Status of bit 5 in CRA (I/O control)	Status of bit 4 in CRA (edge)	Status of bit 3 in CRA (Mask)	IRQA2 (interrupt flag) Bit 6 of CRA	Status of IRQA line (MPU interrupt request)
	0	0	0	1	Masked (remains high)
	0	0	1	1	Goes low (processor interrupted)
	0	1	0	1	Masked (remains high)
	0	1	1	1	Goes low (processor interrupted)
	0	1	—	0	Remains high
	0	0	—	0	Remains high

As seen in Table 9.3, bit 3 of CRA is the interrupt "mask programming bit." If it is a "0", setting the interrupt flag IRQA2 will not cause interrupt line $\overline{\text{IRQA}}$ to go low. If it is a "1", the $\overline{\text{IRQA}}$ line will be permitted to go low when IRQA2 gets set to a "1".

Bit 4 of CRA is the "edge programming bit." A "0" in bit 4 programs interrupt flag IRQA2 (bit 6) to get set to a "1" on a *negative* transition of the

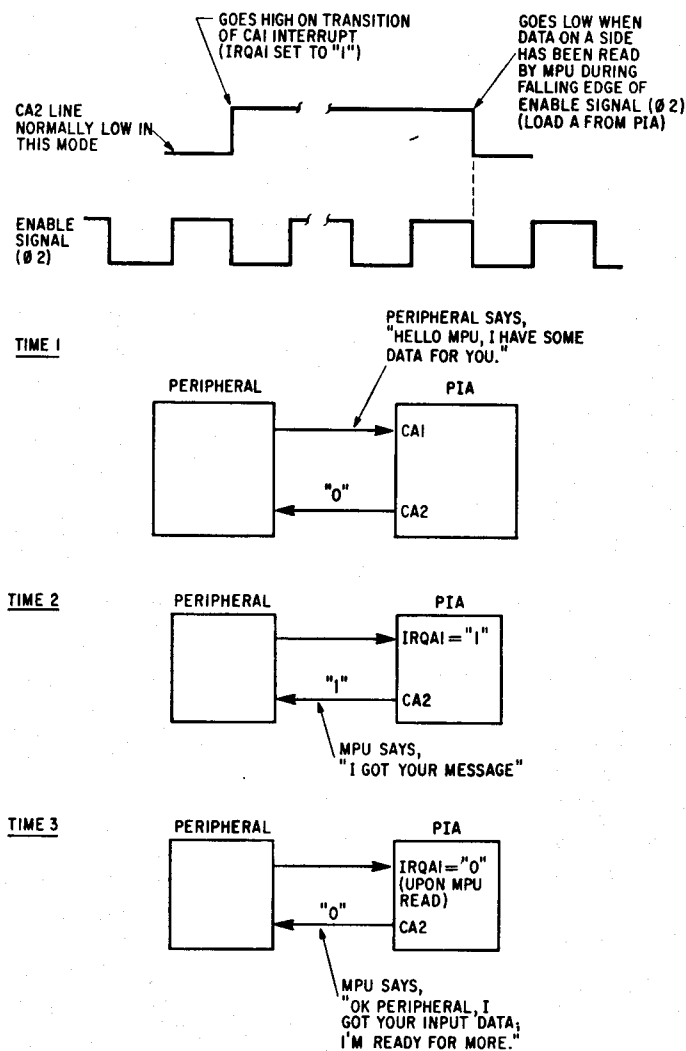


Fig. 9.17 Handshake mode

CA2 line. A "1" in bit 4 programs the flag to get set to a "1" on a rising transition. NOTE: If the IRQA2 flag was set to a "1" during a period when bit 4 had masked all interrupts, the interrupt will be allowed when bit 4 of CRA is changed to a "1" by the MPU.

CA2 As an Output Line (Bit 5 = "1") If bit 5 of CRA is set to a "1", the CA2 line is designated as an output line. Whenever it is used as an output, the IRQA2 flag (bit 6 of CRA) remains a "0" and the $\overline{\text{IRQA}}$ remains high. As an output, it has four options:

1. **Bits 5, 4, and 3 of CRA = 100 (Handshake Mode)** The handshake mode is used when a peripheral is transmitting data to the MPU. The peripheral must tell the MPU when it has some data, and the MPU must tell the peripheral when it has taken the data (see Fig. 9-17). The typical sequence is as follows:

- (1) Peripheral sends signal via interrupt line CA1 to set IRQA1 flag (bit 7) of control register A, which tells the MPU it has data to give to the MPU.
- (2) When the IRQA1 flag gets set to a "1", the CA2 line goes high.
- (3) After the MPU reads the contents of peripheral register A (load A from PIA), the CA2 line will go low. This signals the peripheral that the MPU took the data and is now ready for more.

2. **Bits 5, 4, and 3 of CRA = 101 (Pulse Mode)** This mode, which tells the peripheral that the data on peripheral data register A has been read by the MPU, is used when a complete handshake is not required. The peripheral may make data available to the MPU on a continuing basis but needs to know when the MPU takes the data (see Fig. 9.18).

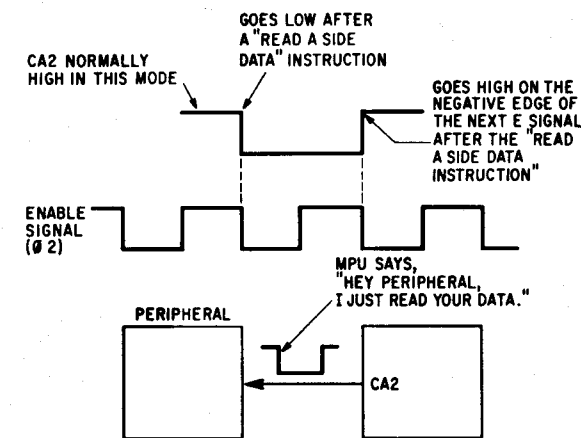


Fig. 9.18 Pulse mode

3. **Bits 5, 4, and 3 of CRA = 110** In this mode, the CA2 output line will always be in the low state.

4. **Bits 5, 4, and 3 of CRA = 111** In this mode, the CA2 output line will always be in the high state.

Interrupt Flag Bits (IRQA1 and IRQA2)

As already seen, bits 6 and 7 of control register A get set when an interrupt occurs. Flag bit IRQA1 (bit 7) is the interrupt bit for the CA1 input line. Flag bit IRQA2 (bit 6) is the interrupt bit for the CA2 input line. The

only way that these bits can get set is via the CA1 and CA2 interrupt input lines. The MPU *cannot* store a "1" in these two locations, but it can read their status. When the MPU reads the status of peripheral data register A, bits 6 and 7 of the control register will be cleared ("0").

Control Register B (CRB)							
7	6	5	4	3	2	1	0
IRQB1	IRQB2	CB2 Control		DDRB	CB1 Control		

CB1 Control (Bits 0 and 1)







Peripheral control line CB1 is an *input only* line that may be used to cause an interrupt by setting interrupt flag IRQB1 (bit 7) of control register B. Bits 0 and 1 of CRB are used to determine how the interrupt is to be handled. The IRQB1 flag (bit 7) of CRB will get set to a "1" *only* under one of the following conditions:

1. A negative transition on the CB1 line is detected *and* bit 1 of CRB is a "0".
2. A rising transition on the CB1 is detected *and* bit 1 of CRB is a "1".

(All other combinations will be ignored.)

Whether the IRQB1 flag is permitted to pull the $\overline{\text{IRQB}}$ line low, thus

Table 9.4 Summary of CB1 Control

Transition of interrupt input line CB1	Status of bit 1 in CRB (edge)	Status of bit 0 in CRB (mask)	IRQB1 (interrupt flag) Bit 7 of CRB	Status of IRQB Line (MPU interrupt request)
	0	0	1	Masked (remains high)
	0	1	1	Goes low (processor interrupted)
	1	0	1	Masked (remains high)
	1	1	1	Goes low (processor interrupted)
	1	—	0	Remains high
	0	—	0	Remains high

interrupting the MPU, depends on the status of bit 0 of CRB. If the bit is a "1", the $\overline{\text{IRQB}}$ line will go low, causing the interrupt. If it is a "0", the interrupt will be masked (disallowed).

The CB1 control is summarized in Table 9.4.

As seen in Table 9.4, bit 0 of CRB is the IRQB1 interrupt "mask programming bit." If bit 0 is a "0", setting the interrupt flag IRQB1 will not cause the interrupt line $\overline{\text{IRQB}}$ to go low. If bit 0 contains a "1", the $\overline{\text{IRQA}}$ line will be permitted to go low when IRQB1 gets set to a "1".

Bit 1 of CRB is the "edge programming bit." A "0" in bit 1 programs the interrupt flag IRQB1 (bit 7) to get set to a "1" on a *negative* transition of the CB1 line. A "1" in bit 1 programs the flag to get set to a "1" on a *positive* transition. (NOTE: If the IRQB1 flag was set to a "1" during a period when bit 0 had masked all interrupts, the interrupt will be allowed when bit 0 of CRB is changed to a "1" by the MPU.)

Data Direction Register B (DDRB) (Bit 2)

This bit, in conjunction with register select lines RS0 and RS1, is used to select either the peripheral data register or the data direction register.

RS1	RS0	CRB (Bit 2)	Register Selected
1	0	0	Data direction register B
1	1	Doesn't matter	Control register B
1	0	1	Peripheral data register B

CB2 Control (Bits 3, 4, and 5)

As mentioned earlier, this line can be programmed to function as an interrupt input line or a peripheral output line. The status of bits 3, 4, and 5 of control register B determine how the CB2 line is to function. Bit 5 determines whether it will be an interrupt input line or an output line. If bit 5 contains a "0", it will be an interrupt line. If bit 5 contains a "1", it will be an output line.

CB2 As an Interrupt Input Line (Bit 5 = "0") Bits 3 and 4 of CRB are used to determine how the interrupt is to be handled. The IRQB2 flag (bit 6) of CRB will get set to a "1" only under one of the following conditions:

1. A negative transition in the CB2 line is detected *and* bit 4 is a "0".
2. A rising transition in the CB2 line is detected *and* bit 4 is a "1".

All other combinations are ignored.

The CB2 control is summarized in Table 9.5.

As seen in Table 9.5, bit 3 of CRB is the interrupt "mask program-

Table 9.5 Summary of CB2 Control

Transition of input CB2	Status of bit 5 in CRB (I/O control)	Status of bit 4 in CRB (edge)	Status of bit 3 in CRB (mask)	IRQB2 (interrupt flag) Bit 6 of CRB	Status of $\overline{\text{IRQB}}$ line (MPU interrupt request)
	0	0	0	1	Masked (remains high)
	0	0	1	1	Goes low (processor interrupted)
	0	1	0	1	Masked (remains high)
	0	1	1	1	Goes low (processor interrupted)
	0	1	—	0	Remains high
	0	0	—	0	Remains high

ming bit". If bit 3 is a "0", setting the interrupt flag IRQB2 will not cause the interrupt line $\overline{\text{IRQB}}$ to go low. If it contains a "1", the $\overline{\text{IRQB}}$ line will be permitted to go low when IRQB2 gets set to a "1".

Bit 4 of CRB is the "edge programming bit". A "0" in bit 4 programs the interrupt flag IRQB2 (bit 6) to get set to a "1" on a *negative* transition of the CB2 line. A "1" in bit 4 programs it to get set to a "1" on a *rising* transition. (NOTE: If the IRQB2 flag was set to a "1" during a period when bit 3 had masked all interrupts, the interrupt will be allowed when bit 3 of CRB is changed to a "1" by the MPU.

CB2 As an Output Line (Bit 5 = "1") If bit 5 of CRB is set to a "1", the CB2 line is designated as an output line. Whenever it is used as an output, the IRQB2 flag (bit 6 of CRB) remains a "0" and the $\overline{\text{IRQB}}$ remains high. As an output, it has four options:

1. **Bits 5, 4, and 3 of CRB = 100 (Handshake Mode)** This mode is used when the MPU sends data to a peripheral device. The peripheral must tell the MPU it is ready for the data. After the MPU sends the data to peripheral data register B, it sends a signal to the peripheral telling it that the data is available at that register. After the peripheral takes the data, it can request more data, and the sequence is repeated. The typical sequence (see Fig. 9.19) is as follows:

- (1) In order to tell the MPU that it wants some data, the peripheral sends a signal via interrupt line CB1 to set the IRQB1 flag (bit 7) of central register B.

- (2) When the IRQB1 flag gets set to a "1", the CB2 line goes high.
- (3) After the MPU sends the data to peripheral data register B, the CB2 line will go low, thereby signaling the peripheral that the data is there for the taking.

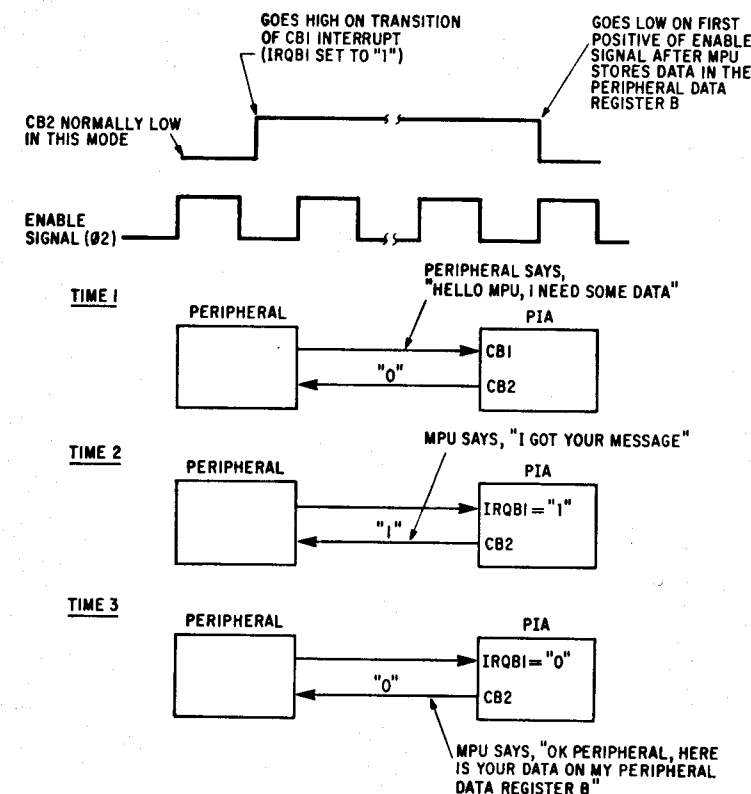


Fig. 9.19 Handshake mode

2. **Bit 5,3 of CRB = 101 (Pulse Mode)** This mode is used to tell the peripheral that data is available in PIA peripheral data register B (see Fig. 9.20).

3. **Bits 5, 4, and 3 of CRB = 110** In this mode, the CB2 output line will always be in the low state.

4. **Bits 5, 4, and 3 of CRB = 111** In this mode, the CB2 output line will always be in the high state.

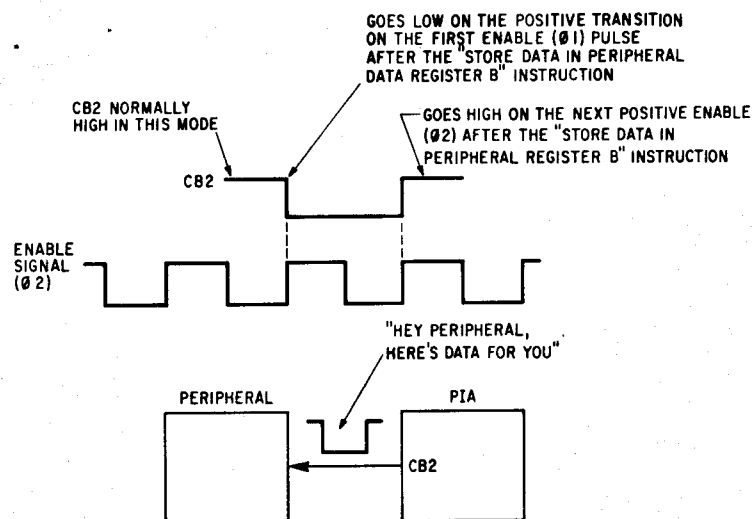


Fig. 9.20 Pulse mode

Interrupt Flag Bits (IRQB1 and IRQB2)

As already seen, these two bits (6 and 7) of control register A get set when an interrupt occurs. Flag bit IRQB (bit 7) is the interrupt bit for the CB1 input line. Flag bit IRQB2 (bit 6) is the interrupt bit for the CB2 input line. The *only* way these bits can get set is via the CB1 and CB2 interrupt input lines. The MPU *cannot* store a "1" in these two locations, but it *can* read their status. When it reads the status of peripheral data register B, bits 6 and 7 of control register B will be cleared ("0").

PIA Summary

1. Register Selects RS1 and RS0

- If RS1 is set to a "0", then the A side of the PIA is selected.
- If RS1 is set to a "1", then the B side of the PIA is selected.
- If RS0 is set to a "0" and CRA (or CRB) bit 2 is a "1", then the peripheral data register is selected.
- If RS0 is set to a "0" and CRA (or CRB) is a "0", then the data direction register is selected.
- If RS0 is set to a "1", then the control register is selected.

2. *CA1 or CB1 Interrupt Lines* If bit 0 of CRA (or CRB) is set to a "0", all interrupts caused by CA1 (or CB1) are masked by the PIA. However, the interrupts flags will still get set if the proper transition occurs on CA1 (CB1). If bit 0 of CRA (or CRB) is set to a "1", all interrupts caused by CA1 (or CB1) will be allowed to interrupt the MPU.

3. *CA2 or CB2 Interrupt Line* If bit 3 of CRA (or CRB) is set to a "0", and bit 5 = "0", all interrupts caused by CA2 (or CB2) are masked by the PIA. However, the interrupt flags will still get set if the proper transition occurs on CA2 (or CB2). If bit 3 of CRA (CRB) is set to a "1", all interrupts by CA2 (or CB2) will be allowed to interrupt the MPU.

If bit 5 = "1", then the CA2 (or CB2) lines are used as outputs.

4. *IRQA1, IRQA2, IRQB1, IRQB2 Flag Bits* These bits (bits 6 and 7 of CRA & CRB) are read only bits. The MPU *cannot* write into them, and *only* interrupts from the outside world can set them. They will be cleared only when the peripheral data register is read or there is a hardware reset.

5. *Control Registers CRA and CRB* Control registers CRA and CRB have total control of CA1, CA2, CB1, and CB2 lines. The status of all eight bits may be read into the MPU, although it can only write into bits 0 through 5.

6. *Addressing* Before addressing PIAs, the Data Direction Register (DDR) must first be loaded with the bit pattern that defines how each line is to function, that is, as an input or output. A logic "1" in the register defines the corresponding line as an output, and a logic "0" defines it as an input. Since the DDR and the peripheral data register have the same address, control register bit 2 determines which is being addressed. If bit 2 is a logic "0", then the DDR is addressed; if it is a logic "1", the peripheral data register is addressed. Therefore, it is essential that the DDR be loaded before setting bit 2 of the control register.

The above sequence of setting up the PIA assumes that the data outputs of the PIA are active high (True 2.4 V).

7. *PIA—After Reset* When the RES (Reset Line) has been held low for a minimum of eight machine cycles, all registers in the PIA will have been cleared. Because of the reset conditions, the PIA has been defined as follows:

- (1) All I/O lines to the "outside world" are defined as inputs.
- (2) CA1, CA2, CB1, and CB2 are defined as interrupt input lines that are negative-edge sensitive.
- (3) All interrupts on the control lines are masked. Setting of interrupt flag bits will not cause IRQA or IRQB to go low.

Active Low Outputs

When all the outputs of a given PIA port are to be active low (True $\leq 0.4V$), then the following procedure should be used:

1. Set bit 2 in the control register.
2. Store all 1's (\$FF) in the peripheral data register.
3. Clear bit 2 in the control register.
4. Store all 1's (\$FF) in the data direction register.
5. Store control word (bit 2=1) in the control register.

Example The B side of PIA1 is set up to have all active low outputs. CB1 and CB2 are set up to allow interrupts in the handshake mode and CB1 will respond to positive edges (low-to-high transitions). Assume reset conditions. Addresses are set up and equated to the same labels as in the previous example.

- | | |
|-----------------|--|
| 1. LDA A #4 | |
| 2. STA A PIA1BC | Set bit 2 in PIA1BC (control register) |
| 3. LDA B #\$FF | |
| 4. STA B PIA1BD | All 1's in peripheral data register |
| 5. CLR PIA1BC | Clear bit 2 |
| 6. STA B PIA1BD | All 1's in data direction register |
| 7. LDA A #\$27 | |
| 8. STA A PIA1BC | 00100111 Control register |

The above procedure is required in order to prevent outputs from going low—to the active low True state—when all 1's are stored in the data direction register, as would be the case if the normal configuration procedure were followed.

PIA Polling Routine

This routine is one of the various techniques for determining which PIA has generated an interrupt. Recall that every PIA has an A side and a B side that may cause the \overline{IRQ} line to go low, thus generating an interrupt. All PIA interrupt lines are tied together and connected to the one interrupt input pin (\overline{IRQ}) of the MPU. Consequently, when an interrupt is generated, a bit 6 or 7 of a PIA is set. The only way to determine where the interrupt came from is to poll bits 6 and 7 of every PIA control register to see which one is a "1" (and thus an interrupt).

This routine (see Fig. 9.21 for its flowchart) polls the control registers of two PIAs. It reads the contents of each control register and executes the BMI instruction that effectively checks on whether bit 7 is set. If it is not set, a ROL A instruction is executed that shifts bit 6 into bit 7, thus permitting use of the BMI instruction again. Once a set control bit is detected, the processor branches to a subroutine to service that particular interrupt. After the interrupt has been serviced, an RTI instruction is executed that causes the processor to return to whatever it was doing before the interrupt.

The source program for the PIA polling routine is as follows:

```

100 NAM POLL
110 OPT MEM
120 PIA1AC EQU $4005
130 PIA1BC EQU $4007
140 PIA2AC EQU $4009
150 PIA2BC EQU $400B
200 ORG $100
210 POLL LDA A PIA1AC
220 BMI ROUT1

```

```

230 ROL A
240 BMI ROUT2
250 LDA A PIA1BC
260 BMI ROUT3
270 ROL A
280 BMI ROUT4
290 LDA A PIA2AC
300 BMI ROUT5
310 ROL A
320 BMI ROUT6
330 LDA A PIA2BC
340 BMI ROUT7
350 ROL A
360 BMI ROUT8
370 RTI
380 ROUT1 NOP    ♦THIS IS PIA1AC CA1 SERVICE ROUTINE
390 RTI
400 ROUT2 NOP    ♦THIS IS PIA1AC CA2 SERVICE ROUTINE
410 RTI
420 ROUT3 NOP    ♦THIS IS PIA1BC CB1 SERVICE ROUTINE
430 RTI
440 ROUT4 NOP    ♦THIS IS PIA1BC CB2 SERVICE ROUTINE
450 RTI
460 ROUT5 NOP    ♦THIS IS PIA2AC CA1 SERVICE ROUTINE
470 RTI
480 ROUT6 NOP    ♦THIS IS PIA2AC CA2 SERVICE ROUTINE
490 RTI
500 ROUT7 NOP    ♦THIS IS PIA2BC CB1 SERVICE ROUTINE
510 RTI
520 ROUT8 NOP    ♦THIS IS PIA2BC CB2 SERVICE ROUTINE
530 RTI
540 MON

```

The assembled program for the routine is as follows:

Address	Hex	Asm	Obj	Label	Comment
00100		NAM	POLL		
00110		OPT	MEM		
00120	4005	PIA1AC	EQU	\$4005	
00130	4007	PIA1BC	EQU	\$4007	
00140	4009	PIA2AC	EQU	\$4009	
00150	400B	PIA2BC	EQU	\$400B	
00200	0100	ORG	\$100		
00210	0100 B6 4005	POLL	LDA A	PIA1AC	
00220	0103 2B 1C		BMI	ROUT1	
00230	0105 49		ROL A		
00240	0106 2B 1B		BMI	ROUT2	
00250	0108 B6 4007		LDA A	PIA1BC	
00260	0108 2B 1B		BMI	ROUT3	
00270	010D 49		ROL A		
00280	010E 2B 17		BMI	ROUT4	
00290	0110 B6 4009		LDA A	PIA2AC	
00300	0113 2B 14		BMI	ROUT5	
00310	0115 49		ROL A		
00320	0116 2B 13		BMI	ROUT6	
00330	0118 B6 400B		LDA A	PIA2BC	
00340	011B 2B 10		BMI	ROUT7	
00350	011D 49		ROL A		
00360	011E 2B 0F		BMI	ROUT8	
00370	0120 3B		RTI		
00380	0121 01	ROUT1	NOP		♦THIS IS PIA1AC CA1 SERVICE
00390	0122 3B		RTI		
00400	0123 01	ROUT2	NOP		♦THIS IS PIA1AC CA2 SERVICE
00410	0124 3B		RTI		
00420	0125 01	ROUT3	NOP		♦THIS IS PIA1BC CB1 SERVICE
00430	0126 3B		RTI		
00440	0127 01	ROUT4	NOP		♦THIS IS PIA1BC CB2 SERVICE

00450 0129 3B	RTI	
00460 0129 01	ROUT5 NOP	◆THIS IS PIA2AC CA1 SERVICE
00470 012A 3B	RTI	
00480 012B 01	ROUT6 NOP	◆THIS IS PIA2AC CA2 SERVICE
00490 012C 3B	RTI	
00500 012D 01	ROUT7 NOP	◆THIS IS PIA2BC CB1 SERVICE
00510 012E 3B	RTI	
00520 012F 01	ROUT8 NOP	◆THIS IS PIA2BC CB2 SERVICE
00530 0130 3B	RTI	
00540	MON	

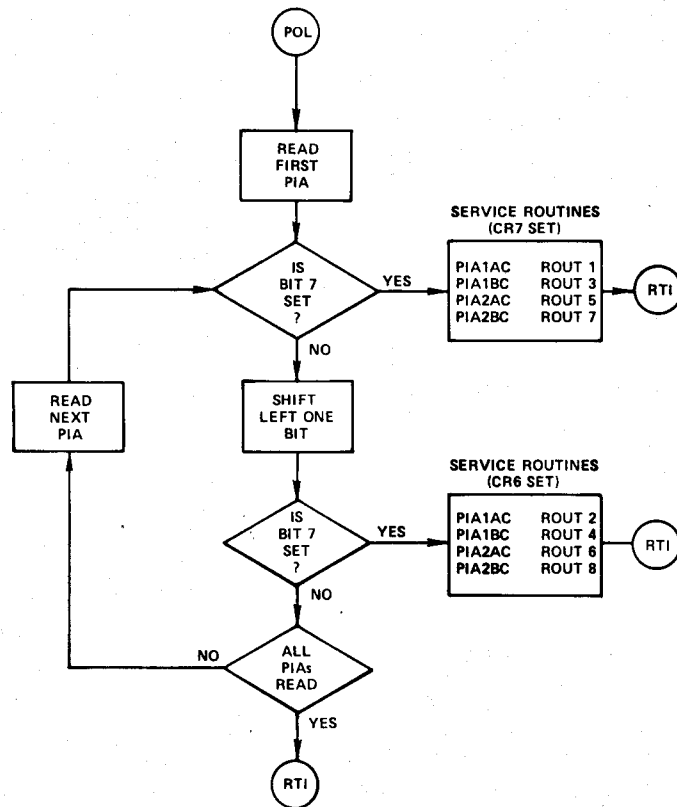


Fig. 9.21 Flowchart for PIA polling routine

9.6 Asynchronous Communications Interface Adapter

In the previous section, it was shown how the MPU communicates with the outside world via the PIA, through which data can be sent to or received from the MPU eight bits at a time in parallel. For this purpose, eight lines are needed between the PIA and the peripheral (data).

When it is necessary to send or receive data over very long distances, eight separate lines must be run from the data service to the PIA. The Asyn-

chronous Communications Interface Adapter (ACIA) permits data to be transmitted in a serial format with only one line, not the eight lines a PIA requires. The ACIA can function either as a serial-to-parallel converter or as a parallel-to-serial converter (Fig. 9.22). Data can be sent to the ACIA over the D0-through-D7 data lines; it is then converted to a series of 1's and 0's in the ACIA and sent out over a single line to a receiver. Likewise, data in the form of 1's and 0's can be received by the ACIA from an external source, converted to a parallel format in the ACIA and sent to the MPU over the D0-through-D7 data lines. As one can well imagine, a great deal of bookkeeping must be done when data bits are transmitted or received in a serial format to resolve such questions as: (1) Where does each group of bits stop and the next group start? (2) How does the ACIA know when it is to receive or send data? (3) How does the ACIA detect if a bit is lost? These questions, and many others, will be answered in this section.

General

In any type of data communications, two terms are encountered—*synchronous* and *asynchronous*—that refer to the type of clocking used to transfer the data. In *synchronous transmission*, the data rate is locked into the system clocking. The receiver and the transmitter must be synchronized with each other since there is no two-way communication. Usually, one device will

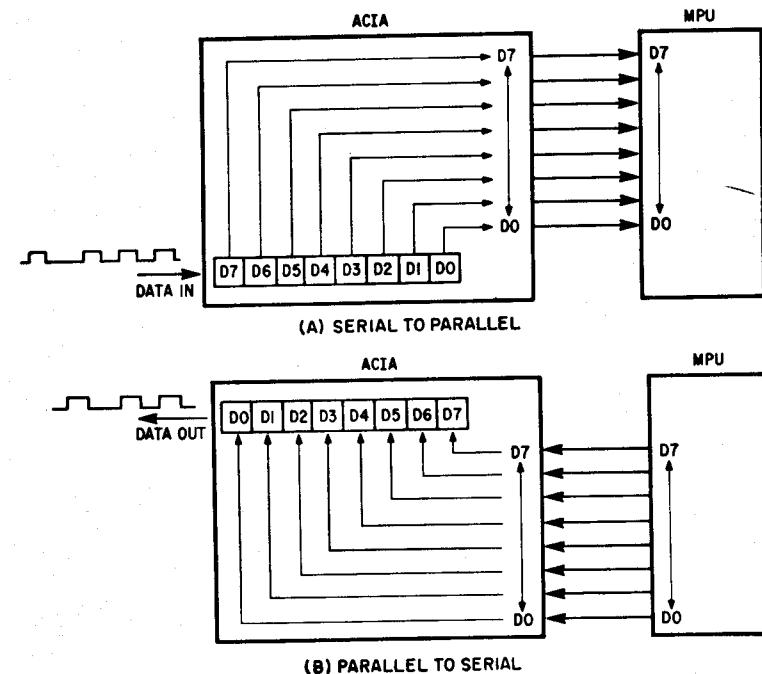


Fig. 9.22 Conversion functions of the ACIA

request some data from the other device, wait a fixed period, and then read the data (assuming that the data was placed on the bus during the waiting period). In *asynchronous transmission*, "start" and "stop" bits are added to the data word to let the receiver know where each word ends and begins. After the receiver detects the stop bit (end of data word), it will then wait for the next data word. The data words are not locked into the system timing.

Baud rate is a term used frequently in serial data communications but often is misunderstood. A *baud* is defined as the reciprocal of the shortest pulse duration in a data word (signal), including start, stop, and parity bits. This is often taken to mean the same as "bits per second," a term that expresses only the number of *data* bits transferred per second. Very often, the parity bit is included as an information or data bit. Definitions of various bits follow.

Start Bit The first bit of a serial data word that signals the start of transmission of a series of data bits. This bit is usually detected as a transition from a "1" to "0," referred to as a "mark-to-space" transition.

Stop Bit The last bit of a serial data word that signals the end of that word. This bit is usually a high ("1") signal.

Parity Bit When transmitting a series of bits, it is common for the transmitter to add what is known as a "parity bit" to the regular data bits transmitted. Two types of parity are used. If *odd* parity is used, the sum of the "1s" transmitted, including the parity bit, will be odd. For example, if the data word contains three "1s", the parity bit will be zero. If four "1s" are in the data word, a "1" would be added by the transmitter so that the number of "1s" transmitted is odd. The same principle applies to *even* parity. A "1" or "0" will be added in the parity bit to make the sum of bits transmitted an even number. The receiver, in both cases, will check to make sure that an odd number of "1"s has been received if odd parity is used or an even number of "1"s if even parity is used. It should be pointed out that if two bits change within the transmitted word, it will not be detected by the parity detection circuit. Only when one bit is lost during transmission will the error be detected and an error message presented.

The role of start, stop, and parity bits is graphically displayed in Fig. 9.23. The relationship of the baud rate, the word rate, and the number of bits transmitted per second is shown below:

$$\text{Baud rate} = 1/\text{bit time} = 1/9.09 \text{ msec} = 110 \text{ baud}$$

$$\text{Time to transmit one character word} = (11 \text{ bits}) \times (9.09 \text{ msec/bit}) = .1 \text{ sec}$$

$$\text{Word rate} = 1/.1 \text{ sec} = 10 \text{ characters/sec}$$

$$\begin{aligned} \text{Baud rate of 110} &= (10 \text{ characters/sec}) \times (8 \text{ bits/character}) \\ &= 80 \text{ bits/sec (including parity)} \end{aligned}$$

Notice that the baud rate and the number of data bits transmitted per second are not the same. The baud rate of 110 includes the start and stop bits, whereas the rate of 80 bits per second includes only information bits (including parity). In Fig. 9.23, a seven-bit ASCII word was transmitted. With the ACIA,

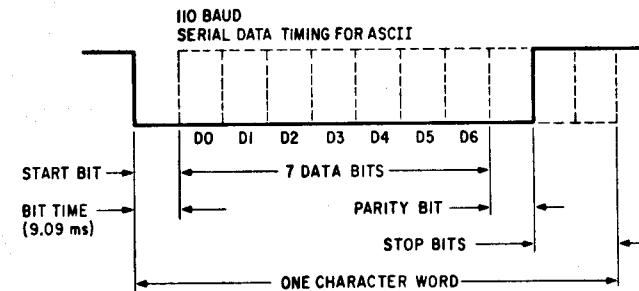


Fig. 9.23 Role of stop, start, and parity bits

several options are available as to the number of data bits (seven or eight), odd or even parity, and the number of stop bits (one or two). Table 9.6 gives bit time, character time, characters/sec, and data bits/sec for various baud rates.

Table 9.6 Baud Rate Data

Baud rate	110	150	300	1200
Bit time(msec)	9.09	6.66	3.33	.833
*Character time	.1 sec	0.73 sec	.0366 sec	.0092 sec
Characters/sec	10	13.7	27.32	108.7
Data bits/sec	80	110	218.6	870

*Assume one start bit, eight data bits (including parity), and two stop bits, or eleven bits per character.

$$\begin{aligned} \text{Bit time} &= 1/\text{baud rate} \\ \text{Character time} &= (\text{total number of bits in word}) \times (\text{bit time}) \\ \text{Characters/sec} &= 1/\text{character time} \\ \text{Data bits/sec} &= 8 \times \text{characters/sec} \end{aligned}$$

To send the ASCII character X (58₁₆) with one start bit, even parity, and two stop bits, the pulse train would be as shown in Fig. 9.24.

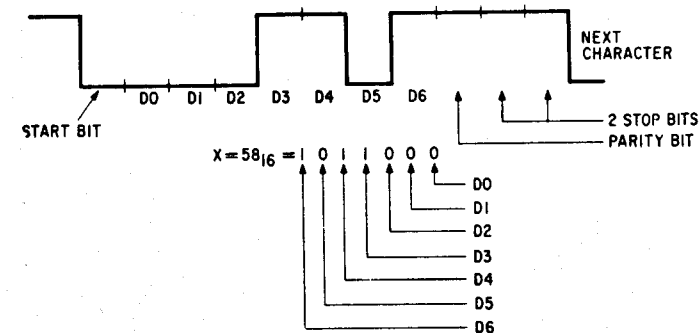


Fig. 9.24 Pulse train

Bit Synchronization

As digital signals are transmitted over a single line, it is possible to read erroneous results because of the noise on the line. To minimize the chance of such error, a sampling technique is used to determine whether the start bit is valid. After it has been proved valid, *each* bit in the character word is sampled at approximately the center of the bit.

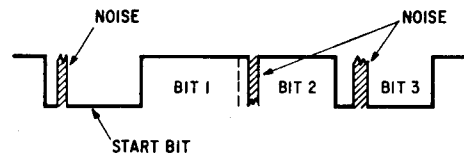


Fig. 9.25 Reading start bit in presence of noise pulse

If the receiving circuit were to read the value of the start bit while a noise pulse is present (Fig. 9.25), it would determine that the start bit is invalid. Likewise, if a reading of bits 2 and 3 were taken during a noise pulse, it would read a "0" for bit 2 and a "1" for bit 3, neither of which would be correct. The erroneous reading of bits 2 and 3 would not be detected by the parity detection circuit since the total number of "1s" remains the same.

A method of minimizing the chance for an erroneous reading is to sample the start bit several times to determine if it is valid and then to sample each bit thereafter with a short pulse at approximately the center of the bit. Sampling at approximately the center of the bit minimizes the chance of error since the noise pulse would have to be present at *precisely* the point where the sampling occurs. This sampling is accomplished by adding an external clock signal. Clock frequencies of 16, 32, and 64 times the baud rate are often used. The higher the clock frequency, the less the chance for a false reading. The MC6850 ACIA, which will be discussed shortly, can accommodate a clock frequency of 1, 16, and 64 times the baud rate. These frequencies are referred to as the $\div 1$, $\div 16$, and $\div 64$ modes.

To illustrate what all this really means, assume that the partial character being received is preceded by the normal start bit. If a clock rate of 16 times the baud rate ($\div 16$ mode) is being used, as shown in Fig. 9.26, the receiver, upon detection of the mark-to-space transition, will start its sampling on the rising edge of the external clock. If the signal remains low for nine separate samplings (in the $\div 16$ mode), the bit is assumed to be a valid start bit and is shifted into the ACIA shift register during the falling edge of the internal clock. After every sixteenth pulse thereafter from the center of the start bit, a reading will be made to determine whether that respective bit is a "1" or "0".

If a clock rate of 64 times the baud rate were used ($\div 64$ mode), the start bit would be determined valid after a sampling of 33 readings. The data bits would be sampled every 64 pulses from the center of the start bit.

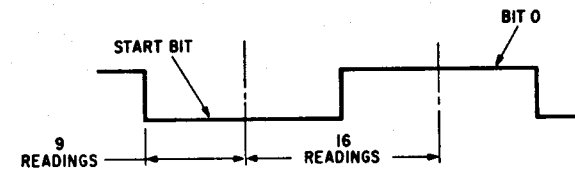


Fig. 9.26 $\div 16$ mode sampling

If the start bit in Fig. 9.26 is expanded, the sampling would appear as shown in Fig. 9.27. Notice the noise pulse on the start bit. Since it did not occur during the sampling period, it goes by undetected, and the start bit is determined to be valid.

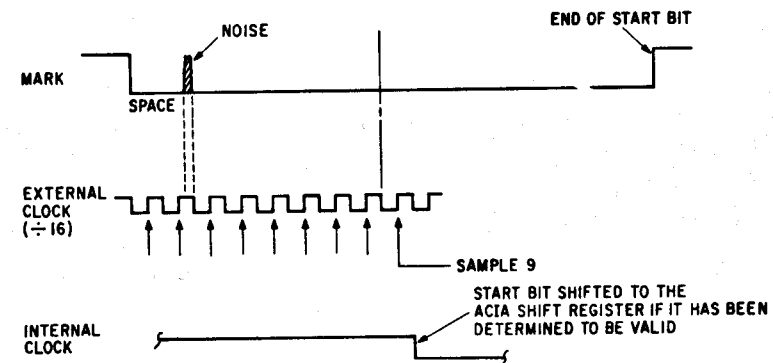


Fig. 9.27 Sampling with expanded start bit

The same argument applies to each data bit. Notice in Fig. 9.28 that on the sixteenth pulse from the center of the start bit, a sampling takes place, yet, despite the noise on the line, the correct data is shifted into the ACIA since the noise does not occur during the sampling period.

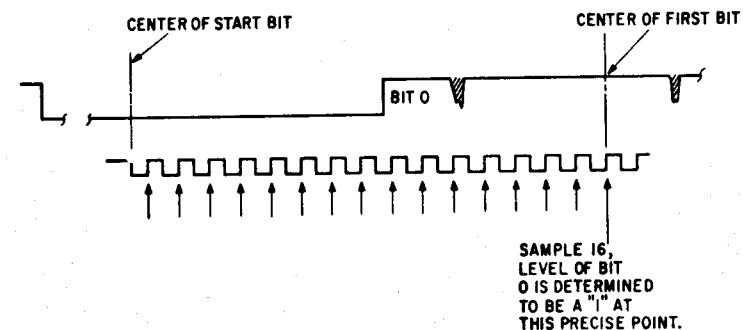


Fig. 9.28 Sampling at sixteenth pulse

Description of the ACIA

The MC6850 Asynchronous Communications Interface Adapter (ACIA) is an N-MOS device housed in a 24-pin package that is used as a means of receiving and transmitting as many as eight bits of data in a serial format (Fig. 9.29). The ACIA communicates (transmits/receives data) with the MPU via an eight-bit bidirectional data bus just as RAMs, ROMs, and PIAs do.

The ACIA has four registers that may be addressed by the MPU. The Status Register (SR) and the Receiver Data Register (RDR) are "read only" registers, meaning that the MPU cannot *write* into them. The Transmit Data Register (TDR) and the Control Register (CR) are "write only" registers, meaning that the MPU cannot *read* them.

In addition to these four registers, the ACIA has three chip select lines (CS0, CS1, CS2), one register select line (RS), one interrupt request line ($\overline{\text{IRQ}}$), one enable line (E), one read/write line (R/W), and seven data and data control lines (RXC, TXC, $\overline{\text{DCD}}$, $\overline{\text{RTS}}$, RXD, TXD, and $\overline{\text{CTS}}$).

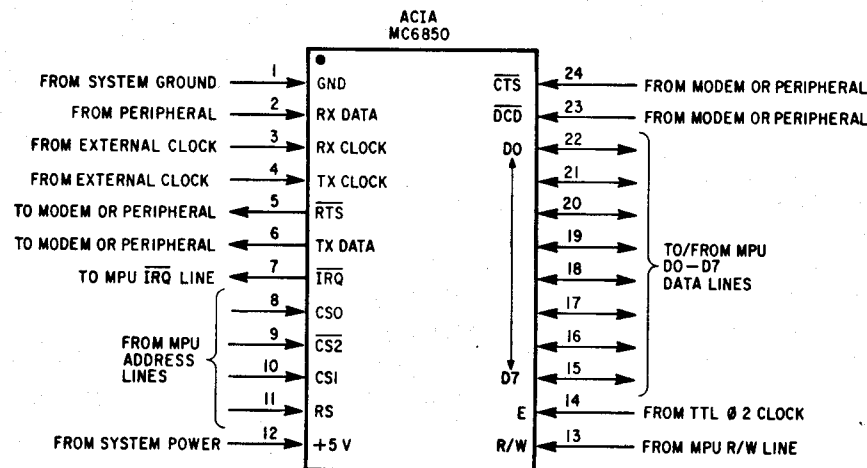


Fig. 9.29 ACIA package

MPU Interface Lines

Bidirectional Data Lines (D0-D7) The eight bidirectional data lines permit transfer of data to and from the ACIA and the MPU. The MPU receives and sends data from and to the outside world through the ACIA via these eight data lines. The data bus output drivers are three-state devices that remain in the high impedance (off) state except when the MPU performs an ACIA read operation.

Chip Select Lines (CS0, CS1, and CS2) It is through these lines, which are tied to the address lines of the MPU, that a particular ACIA is

selected (addressed). For this selection, the CS0 and CS1 lines must be high and the CS2 must be low. After the chip selects have been addressed, they must be held in that state for the duration of the E enable pulse, which is the only timing signal supplied by the MPU to the ACIA.

Enable Line (E) The enable pulse is a high-impedance, TTL-compatible input from the MPU that enables the ACIA input or output buffers and that clocks data to and from the ACIA. This input is usually the TTL ϕ 2 signal from the clock.

Read/Write Line (R/W) The read/write line is a high-impedance, TTL-compatible input that is used to control the direction of data flow between the ACIA's eight-bit parallel data bus and the MPU. When read/write is high (MPU read), the ACIA output driver is turned on, and a selected register is read by the MPU. When the read/write line is low (MPU write), the ACIA output driver is turned off, and the MPU writes into a selected register.

Register Select (RS) The register select line is a high-impedance, TTL-compatible input from the MPU that is used to select, in conjunction with the read/write line, either the transmit/receiver data register or the control/status register in the ACIA (Fig. 9.30). It must be tied to an address line from the MPU. A high or RS selects transmit/receive data registers; a low selects control/status registers.

Modem Control Lines

Serial data to be transmitted over telephone lines must be sent to a modem that prepares the signal for transmission (Fig. 9.31). Three signals between the ACIA and the Modem permit a limited control of the latter.

Clear to Send ($\overline{\text{CTS}}$) This high-impedance TTL-compatible *input* provides automatic control of the transmitting end of the communications link via the modem $\overline{\text{CTS}}$ -signal active-low output by inhibiting the Transmit Data Register Empty (TDRE) status bit. If this line is not used, it should be tied to system ground.

Request to Send ($\overline{\text{RTS}}$) This ACIA *output* enables the MPU to control a peripheral or modem via the data bus. The $\overline{\text{RTS}}$ output corresponds to the state of control register bits 5 and 6. When CR6 equals "0" or both CR5 and CR6 equal "1", the $\overline{\text{RTS}}$ output is low (the active state). This output can also be used for the Data Terminal Ready (DTR) on the 6860 modem.

Data Carrier Detect ($\overline{\text{DCD}}$) This high-impedance, TTL-compatible *input* provides automatic control of the receiving end of a communication link by means of the modem Data Carrier Detect (DCD) output. The $\overline{\text{DCD}}$ *input* inhibits and initializes the receiver section of the ACIA when high. A low-to-high transition of the $\overline{\text{DCD}}$ line initiates an interrupt to the MPU to indicate that a loss carrier has occurred when the Receiver Interrupt Enable (RIE) bit is set. If this line is not used, it should be tied to system ground.

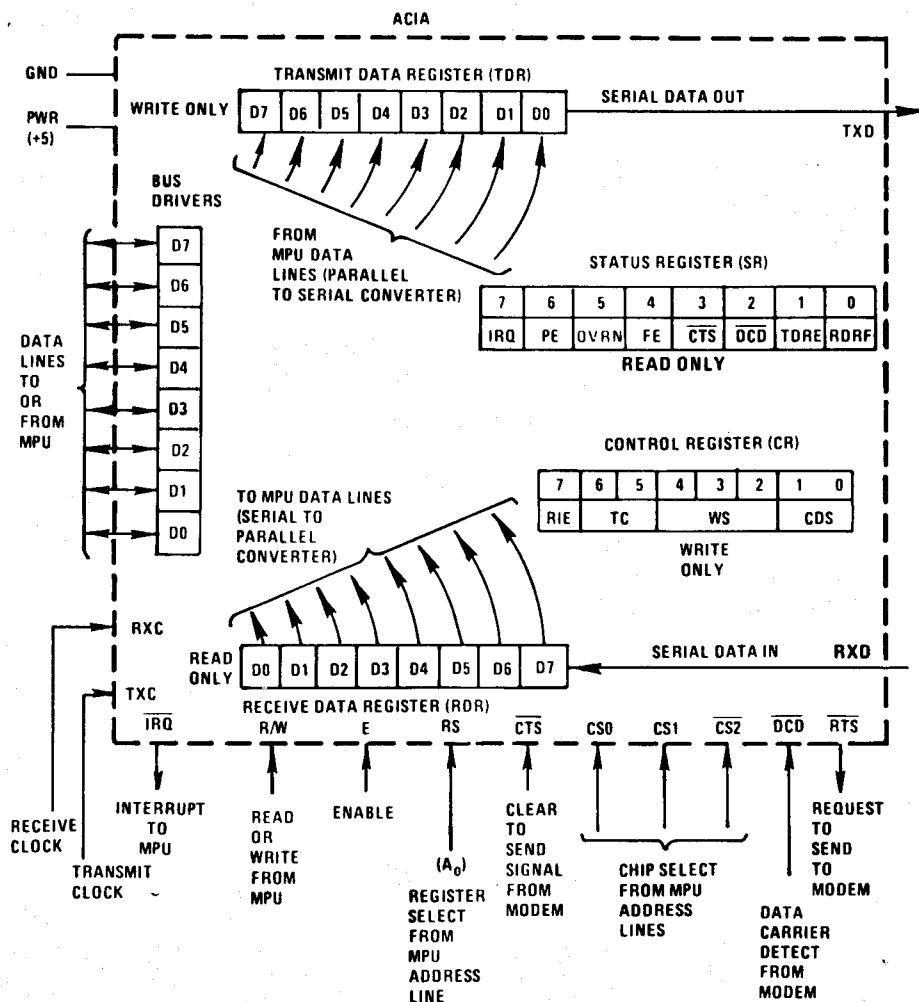


Fig. 9.30 Register select line

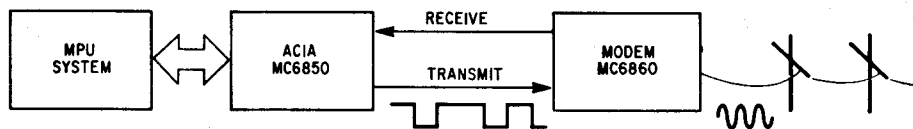


Fig. 9.31 Modem control line

Serial Data Lines

The ACIA has two lines for transfer of data. The Transmit Data (TX DATA) line is used to send, and the Receive Data (RX DATA) line to receive, data from a peripheral. Before transferring data, the ACIA will add the start bit automatically. The number of stop bits and odd or even parity will also be specified in the data word per instructions via bits 2, 3, and 4 of the control register. As data is received over the receive data line, the ACIA will use the parity bit to check the accuracy of the number of "1"s received, and it will strip the start, stop, and parity bits from the data word before converting the data bits to a parallel format for transfer to the MPU over the data bus.

Receive Data (RX DATA) The receive data line is a high-impedance TTL-compatible *input* through which data is received in a serial format. Internal synchronization for detection of data is possible with clock rates of 16 or 64 times the bit rate. Data rates in the range of 0 to 500,000 bits per second are possible with external synchronization (in the $\div 1$ mode).

Transmit Data (TX DATA) The transmit data *output* line is used to transfer data in a serial format to a modem or peripheral. Data rates in the range of 0 to 500,000 bits per second are possible with external synchronization (in the $\div 1$ mode).

External Clock Inputs

Separate high-impedance, TTL-compatible inputs are provided for clocking of transmitted and received data. Clock frequencies of 1, 16, or 64 times the data rate may be selected.

Transmit Clock (TXC) The transmit clock input is used for clocking transmitted data. The transmitter initiates data on the negative transition of the clock.

Receive Clock (RXC) The receive clock input is used for synchronization of received data. The receiver strobes the data on the positive transition of the clock. (In the $\div 1$ mode, the clock and data must be synchronized externally.)

Transmit Data Register (TDR)

The transmit data register, an eight-bit register within the ACIA, is used to hold the data from the MPU (converted from parallel format to serial format) until it is transferred. The data is written into the transmit data register on the negative transition of the enable (E) signal after the ACIA has been addressed through the CS0, CS1, and CS2 lines and the RS line is a "1" and the R/W line is a "0". Writing data into the transmit data register causes the Transmit Data Register Empty (TDRE) bit in the status register to go low ("0"). After the TDRE bit goes low, data will be transmitted. If the transmitter is idling (no character being transmitted), then the transmission will occur

within one bit time of the trailing edge of the write command. If previous data is being transmitted, it will be transferred upon completion of the previous transmission. After the data has been transferred, the TDRE will be changed to a "1", indicating that TDR is empty.

Receive Data Register (RDR)

The receive data register, an eight-bit register within the ACIA, holds the data that is transferred from the modem or peripheral to the ACIA. After the receive data register is full, the data is ready for transfer to the MPU over the parallel data bus, and the Receive Data Register Full (RDRF) bit in the status register will go high ("1"), indicating that the register is full. The RDRF bit's going high may cause the IRQ bit of the status register to go high as well (if enabled) and to remain high until the data is read into the MPU by addressing the ACIA through the CS0, CS1, and $\overline{\text{CS2}}$ lines and by setting the RS and R/W lines to a "1". After the data is read by the MPU, the RDRF and IRQ bits will be reset to a "0", but the data will remain in the RDR.

Status Register (SR)

The Status Register (SR) is an eight-bit register within the ACIA that maintains the current condition of internal ACIA activities (Fig. 9.32). A *read only* register in that the MPU cannot store any data in it, it is used by the MPU to check the status of certain events. To read its contents, the ACIA must be selected through the CS0, CS1, and the $\overline{\text{CS2}}$ lines, with the register select (RS) line being held low ("0") and the R/W line high ("1").

STATUS REGISTERS (SR)							
7	6	5	4	3	2	1	ϕ
IRQ	PE	OVRN	FE	CTS	DCD	TDRE	RDRF

Fig. 9.32 Status register

Bit 0—Receiver Data Register Full (RDRF)

- "1": (a) Indicates that the receiver data register is full.
 (b) The IRQ bit, if enabled, also gets set to a "1" and remains set until the data is read by the MPU (see pg. 198).
 "0": (a) Indicates that the contents of the receiver data register have been read into the MPU. The data is retained in the register.
 (b) If there is a loss of carrier, the DCD line goes high, and the RDRF bit is clamped at "0", indicating that the contents of the RDR are not current.
 (c) A master reset condition also forces the RDRF bit to a "0".

Bit 1—Transmit Data Register Empty (TDRE)

- "1": (a) Indicates that the contents of the transmit data register have been transferred and the register is ready for more data.
 (b) The IRQ bit, if enabled, also gets set to a "1" and remains set until a write operation to the transmit data register (see pg. 198).
 "0": (a) Indicates that the transmit data register is full.
 (b) When a "1" is present on the $\overline{\text{CTS}}$ pin and causes the $\overline{\text{CTS}}$ (bit 3) of the SR to get set to a "1" to indicate that it is not clear to send, bit 0 of the TDRE will be clamped to a "0".

Bit 2—Data Carrier Detect ($\overline{\text{DCD}}$)

- "1": (a) Indicates that there is no carrier from the modem.
 (b) The IRQ bit, if enabled, also gets set and remains set until the MPU reads the status register and the receiver data register or until a master reset occurs (see pg. 198).
 (c) This causes the RDRF bit to be clamped at a "0", inhibiting further interrupts from RDRF.
 "0": (a) The carrier from the modem is present.

Bit 3—Clear to Send ($\overline{\text{CTS}}$)

- "1": Indicates, via the high clear-to-send line from the modem, that the latter is not ready for data.
 "0": Indicates, via the low clear-to-send line from the modem, that the modem is ready for data.

Bit 4—Framing Error (FE)

- "1": Indicates that the received character is improperly framed by the start and stop bit. This error is detected by the absence of the first stop bit and indicates a synchronization error, faulty transmission, or a break condition. The error flag is set or reset during the receiver data transfer time and is therefore present throughout the time that the associated character is available.
 "0": Indicates that the received character is properly framed.

Bit 5—Receiver Overrun (OVRN)

- "1": Indicates that one or more characters in the data stream has been lost, that is, a character or a number of characters has been received, but not read, from the Receiver Data Register

(RDR) prior to subsequent characters being received. The overrun condition begins at the midpoint of the last bit of the second character received in succession without a read of the RDR having occurred. The overrun does not occur in the status register until the valid character prior to overrun has been read. Character synchronization is maintained during the overrun condition. The overrun error flag is reset after the reading of data from the RDR. Overrun is also reset by the master reset.

"0": No receiver overruns have occurred.

Bit 6—Parity Error (PE)

"1": Indicates that the number of highs ("1"s) in the character does not agree with the preselected odd or even parity. By definition, odd parity occurs when the total number of "1"s, including the parity bit, is odd. The parity error indication will be present as long as the data character is in the RDR. If no parity is selected, then both the transmitter parity generator output and the receiver parity check results are inhibited.

"0": No parity error occurred.

Bit 7—Interrupt Request (IRQ)

"1": Indicates that there is an interrupt present that has caused the IRQ output line to go low. The interrupt will be cleared by a read operation to the RDR, a write operation to the TDR, or a read of the SR, followed by a read of the RDR if caused by DCD. A master reset always clears this bit.

"0": Indicates no interrupt present.

Control Register (CR)

The Control Register (CR), an eight-bit register within the ACIA, is used by the MPU to control the transmitting and receiving of serial data (Fig. 9.33). It is a write only register since the MPU cannot read it. To write into it, the ACIA must be selected via CS0, CS1, and $\overline{\text{CS2}}$, and both the RS line and the R/W line must be low ("0").

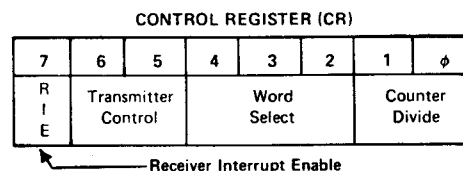


Fig. 9.33 Control register

Bits 0 and 1—Counter Divide Select Bits (CDS)

These two bits determine the divide ratios utilized in both the transmitter and receiver sections of the ACIA. They are also used for master reset of the ACIA, which clears the status register (except for external conditions on CTS and DCD) and initializes both the receiver and the transmitter. Master reset does not affect other control register bits. After a power failure or restart, the ACIA must be reset before setting the clock divide ratio. Bit patterns for the various functions are shown below.

CR1	CR0	Function
0	0	$\div 1$
0	1	$\div 16$
1	0	$\div 64$
1	1	Master reset

Bits 2, 3, 4—Word Select Bits (WS)

The programmer has the option of selecting the word length, number of stop bits, and type of parity by using the proper bit pattern from the chart below.

B4	B3	B2	Word Length	+	Parity	+	Stop Bits
0	0	0	7		Even		2
0	0	1	7		Odd		2
0	1	0	7		Even		1
0	1	1	7		Odd		1
1	0	0	8		None		2
1	0	1	8		None		1
1	1	0	8		Even		1
1	1	1	8		Odd		1

Bits 5 and 6—Transmitter Control Bits (TC)

The status of bits 5 and 6 of the control register provide for control of the interrupt from the Transmit Data Register Empty (TDRE) condition, the Request To Send (RTS) output, and the transmission of a break level (space), as shown below.

CR6	CR5	Function
0	0	The $\overline{\text{RTS}}$ pin is <i>low</i> and Transmit Interrupts are inhibited. This is the code used when requesting that the communications channel be set up. It is not clear to send data yet.
0	1	The $\overline{\text{RTS}}$ pin is <i>low</i> and the communications channel has been set up. Therefore, this code is used to generate IRQs via the TRDE bit in the Status Register.
1	0	The $\overline{\text{RTS}}$ pin is <i>high</i> and transmit interrupts are inhibited. This code can be used to "knock down" the communications channel.
1	1	The $\overline{\text{RTS}}$ pin is <i>low</i> (keep up communications channel) and a break signal (low level on transmit data out line) is transmitted. This is used to interrupt the remote system.

Bit 7—Receiver Interrupt Enable (RIE)

"1": Enables interrupts caused by:

- Receiver Data Register Full (RDRF) going high.
- A low-to-high transition on the Data Carrier Detect (DCD) signal line.

"0": Inhibits interrupts caused by RDRF or by the loss of receive data carrier.

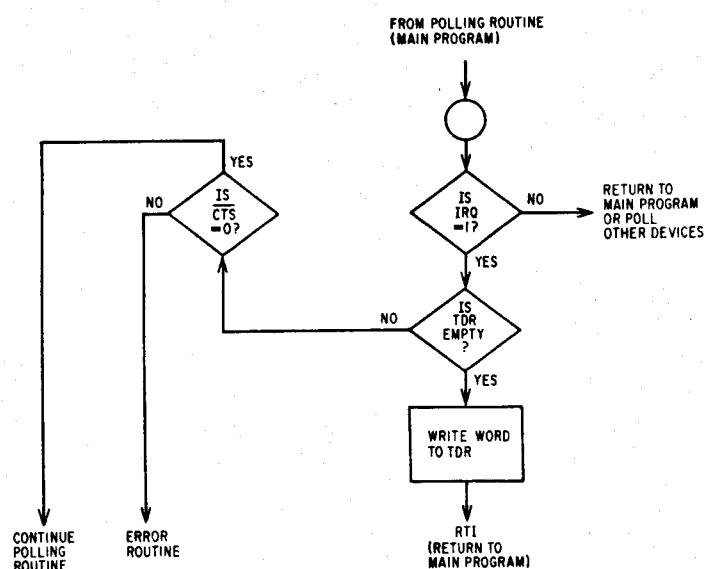


Fig. 9.34 Flowchart of transmit sequence

Power On

After the system power is applied, the ACIA master reset should be set by the initialization program, which stores a "1" in CR0 and CR1 of the ACIA control register. After master reset, the variable clock divide ratio bits, the transmitter interrupt bits, and the receiver interrupt bit of the ACIA control register should be set by the initialization program.

Transmit Sequence

The flowchart in Fig. 9.34 illustrates a typical sequence followed in transmission of serial data (for an example of ACIA programs, see Chap. 11).

Receive Sequence

The flowchart in Fig. 9.35 illustrates a typical sequence followed in

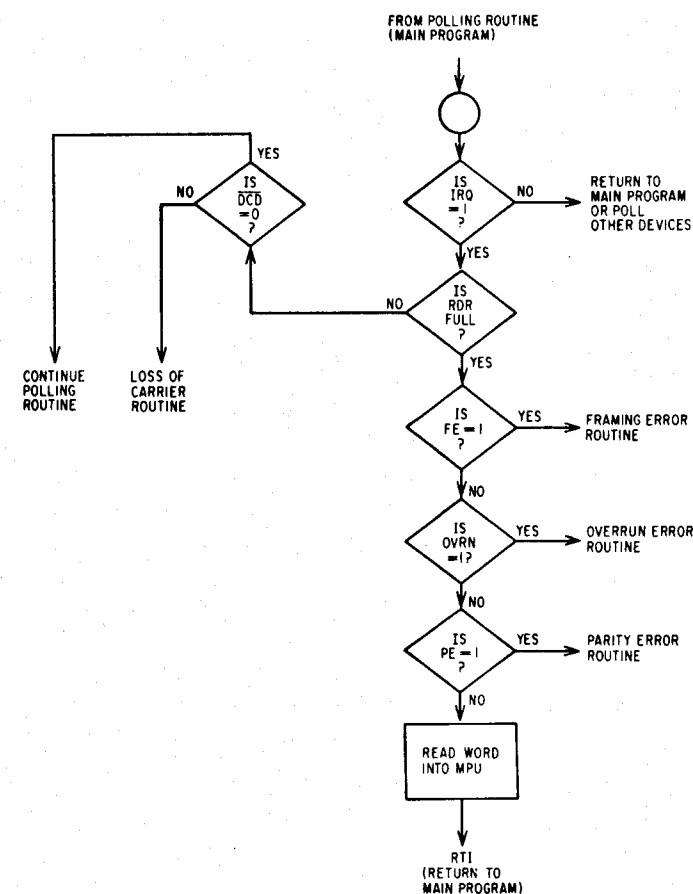


Fig. 9.35 Flowchart of receive sequence

the receiving of serial data by the ACIA (for an example of an ACIA program, see Chap. 11).

Problems

Microprocessor

- How many bits does each of the following registers contain?
 - A _____
 - B _____
 - CC _____
 - X _____
 - PC _____
 - SP _____
- How many bits wide is the data bus? The address bus?
- Running at the maximum rated clock frequency, what is the time required to execute the shortest instructions on the 6800?
- Where does the MPU get the program starting address upon first starting up?
 - It clears the program counter (sets to zero) and therefore always starts a program at 0000.
 - It goes to the program starting address set up by the front panel switches of the microcomputer.
 - It fetches an address for the program start from the two highest bytes in memory corresponding to FFFE and FFFF on the address bus.
 - It always starts executing a program at addresses FFFE and FFFF.
 - The MPU inherently knows where to start a program and will automatically set the program counter to the proper address without being directed.
- What is the state (0 or 1) of each of the following condition code register bits after instruction LDA A #\$FF (load accumulator A with hexadecimal number FF) is executed?
 - Bit N
 - Bit Z
 - Bit V
- If the interrupt bit (I bit) is set to a "1" and a WAI instruction is encountered, how can the MPU exit from this situation?
- Assume the address of the stack pointer is hex 8B. What address will each of the MPU registers be stored in if an interrupt occurs?
- If a system has two PIAs and two ACIAs, how does the MPU know which device causes an interrupt?
- How many locations can be addressed by the MPU in hex? In decimal?
- Which pin can halt the processor for an indefinite amount of time?
- Which interrupt pin can be masked?
- Which pin indicates that a number on the address bus is an arithmetic operand?

- Which pin can halt the microprocessor for a maximum of 9.5 μ sec?
- Which pin refreshes the data bus drivers?
- Which pin indicates that the address bus and the data bus have gone three-state during a halt?
- Which output pins cannot be three-stated?
- Determine the state of the condition code bits after the following operations:

(a) ABA

$$\begin{array}{rcl} A = 1111\ 1111 & A + B = & \text{_____} \quad H = \text{___} \quad N = \text{___} \\ B = 0000\ 0001 & & Z = \text{___} \quad V = \text{___} \quad C = \text{___} \end{array}$$

(b) DECA

$$\begin{array}{rcl} A = 0000\ 0000 & A = & \text{_____} \quad H = \text{___} \quad N = \text{___} \\ & & Z = \text{___} \quad V = \text{___} \quad C = \text{___} \end{array}$$

(c) LDA A #01

$$\begin{array}{rcl} A = & \text{_____} & H = \text{___} \quad N = \text{___} \\ & & Z = \text{___} \quad V = \text{___} \quad C = \text{___} \end{array}$$

(d) INC A

$$\begin{array}{rcl} A = 0111\ 1111 & A = & \text{_____} \quad H = \text{___} \quad N = \text{___} \\ & & Z = \text{___} \quad V = \text{___} \quad C = \text{___} \end{array}$$

- What is the value of the program counter (PC) after completion of the following interrupt sequences:

- After an $\overline{\text{IRQ}}$
- After an $\overline{\text{NMI}}$
- After an SWI
- After restart

FFFF
FFFE
FFFD
FFFC
FFFB
FFFA
FFF9
FFF8

0	0
F	E
8	0
F	C
0	0
F	D
8	0
F	D

Memories

- How many bytes in a MCM6830 ROM?

- 128
- 256
- 512
- 1024
- 4096

- Which pin(s) on the MCM6810 and MCM6830 is (are) three-state?

- Data bus
- Read/Write
- Address bus
- Chip selects
- All of the above

21. Why doesn't the MCM6830 ROM have a R/W pin?
 22. What are the purposes of the positive and negative chip selects on the ROMs and RAMs?
 23. What are the contents of the last eight ROM locations in any system?
 24. What power supply voltages are required for the MCM6810 RAM and MCM6830 ROM?

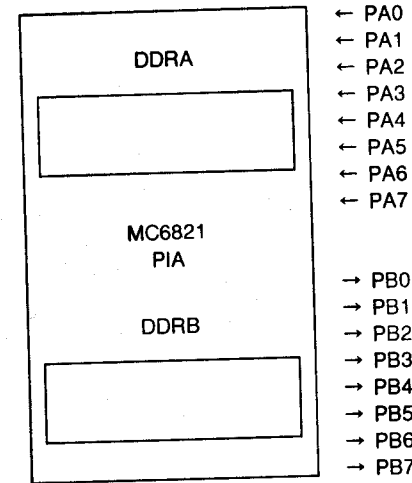
- (a) +5, -12
 (b) +5
 (c) +15, +5, -15
 (d) +5, -5
 (e) None of the above

25. What is the organization of the MCM6810 RAM?

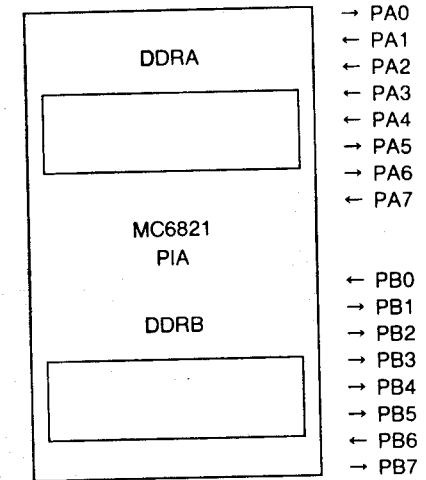
- (a) 1024×1
 (b) 512×4
 (c) 4K bytes (4096×8)
 (d) 128 bytes (128×8)
 (e) 1024 bytes (1024×8)

PIA

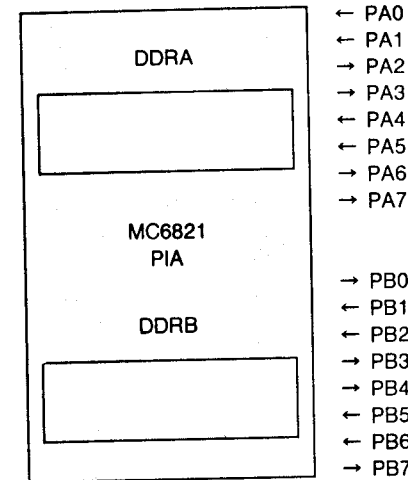
26. How many registers are in the PIA?
 27. How many register select pins are in the PIA?
 28. How many total lines can we have to the "outside world" on a PIA?
 29. Although the A side and B side of the PIA are identical in most respects, there are two differences. One difference is in the internal I/O construction; the other difference is that
- (a) The A side can be either input or output on the eight data lines, whereas the B side can only be output.
 (b) The B side has no control register.
 (c) The A side does not have the "bit following" mode capability.
 (d) Both A and B side interrupts are reset as a result of a read of the A side data register, whereas only the B side interrupts are reset as a result of a read of the B side data register.
 (e) Reading from the A side or writing to the B side causes the handshake or pulse modes on the respective side if it is programmed for one of those modes.
30. Can we have three inputs and five outputs on the PIA A side?
 31. How does the microprocessor know whether it is the A side or the B side which causes an interrupt? (Recall that both the IRQ lines from both sides are normally tied together.)
 32. How can six registers be addressed with only two register selects?
 33. After restart, what steps must be taken before the PIAs can be used?
 34. In each of the following diagrams, fill in the bits of the data direction registers that match the I/O lines.



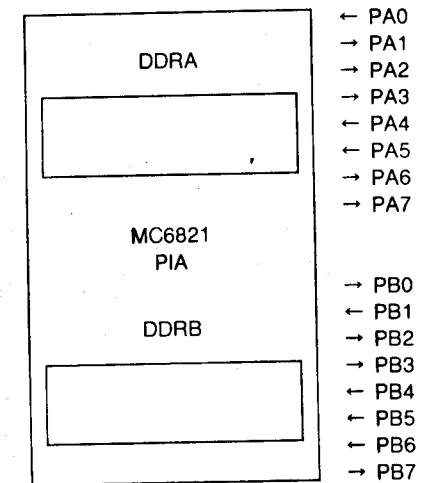
(a)



(b)

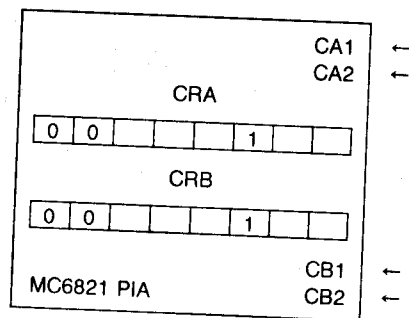


(c)

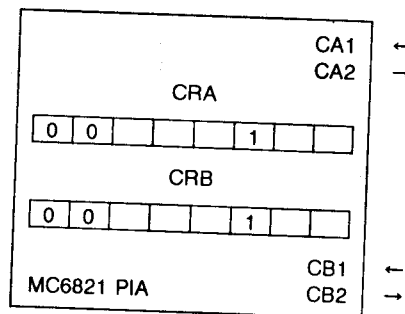


(d)

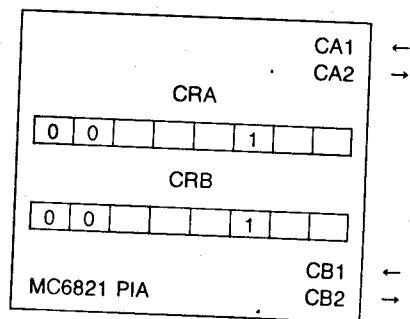
35. In the following diagrams, fill in the bits required to program the interrupt control lines as specified.



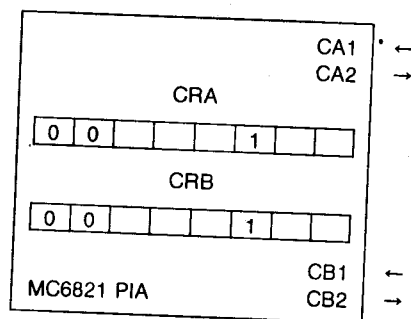
- (a) CA1—Negative edge, masked
CA2—Positive edge, unmasked
CB1—Positive edge, masked
CB2—Negative edge, unmasked



- (b) CA1—Positive edge, masked
CA2—Pulse mode
CB1—Positive edge, unmasked
CB2—Zero



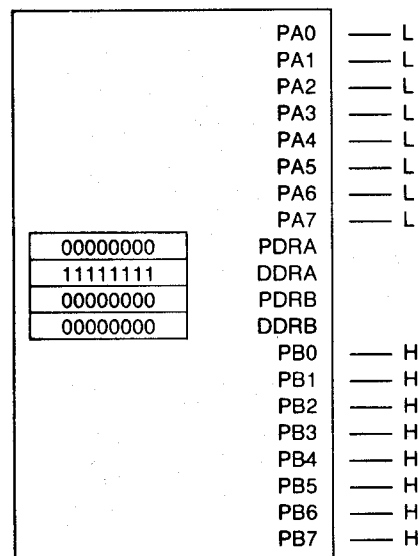
- (c) CA1—Positive edge, unmasked
CA2—One
CB1—Negative edge, masked
CB2—Handshake mode



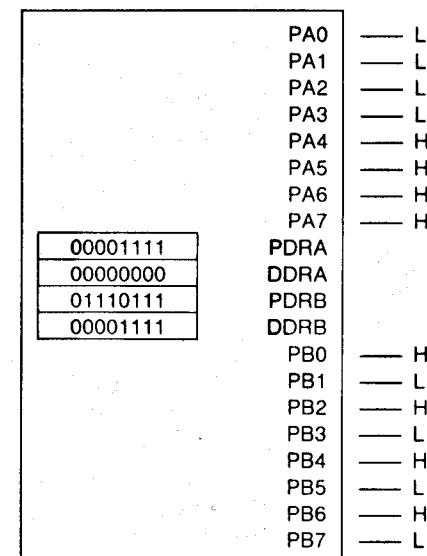
- (d) CA1—Negative edge, unmasked
CA2—Handshake mode
CB1—Negative edge, unmasked
CB2—Pulse mode

36. In the following diagrams, determine the data that would be read from the A and B sides with the logic levels shown on the I/O lines and with the data shown in the registers.

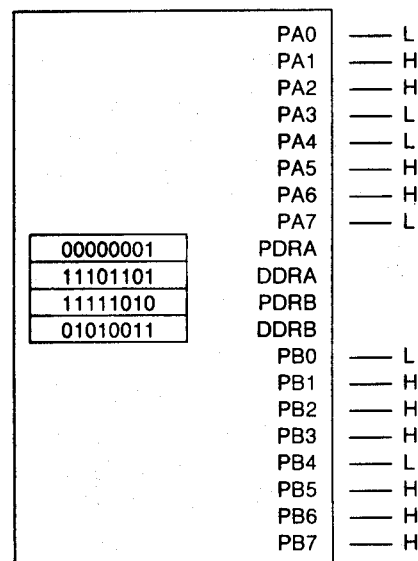
L = Low logic level H = High logic level



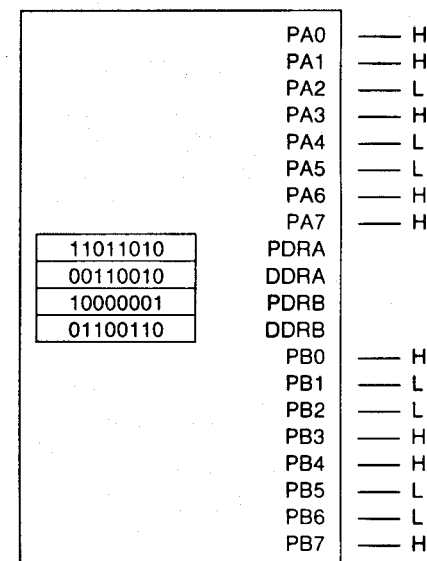
- (a) A side data = _____
B side data = _____



- (b) A side data = _____
B side data = _____

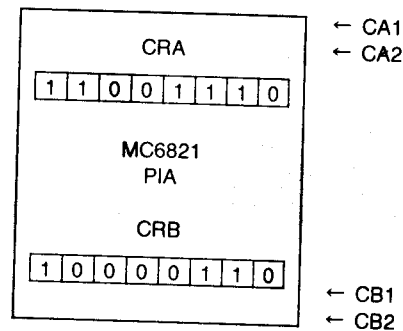


- (c) A side data = _____
B side data = _____

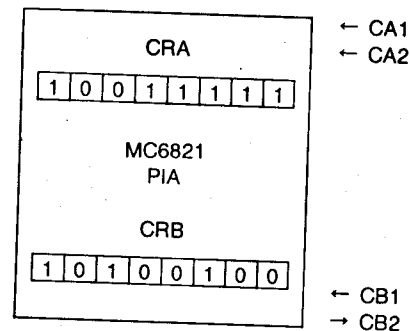


- (d) A side data = _____
B side data = _____

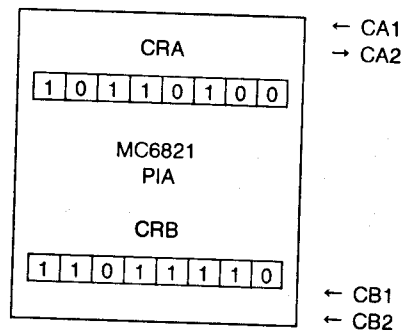
37. Each PIA shown in the following diagrams has interrupted the processor. Determine the control line that caused the interrupt in each case.



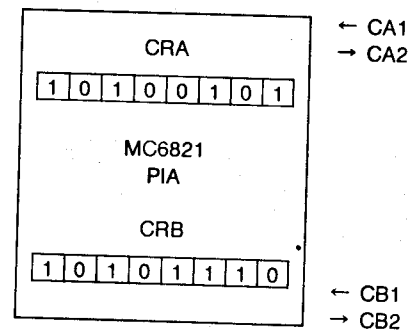
(a)



(b)



(c)



(d)

ACIA

38. During serial transmission at 300 baud, how many information bits are transmitted each second if each character has seven information bits with no parity bit?
39. If the ACIA control register word select bits contain a hex 5, how many external clock pulses are needed in the 16 mode to transfer both the start bit and the information bits?
40. What does ASCII stand for?
- American Standard Code for Information Interchange
 - A Standard Communication—Version 2
 - Asynchronous Standard Code—Version 2
 - Asynchronous Serial Code in Industry

41. Can the MC6850 ACIA be used to transmit eight-bit data other than ASCII code serially?
42. How many registers in the ACIA are accessible from the data bus?
43. How are these registers selected with only one register select line?
44. What is the first thing that must be done to initialize the ACIA?
45. Which of the following bits in the status register could cause an interrupt? (Choose all that apply.)
- RDRF (Receive Data Register Full)
 - TDRE (Transmit Data Register Empty)
 - $\overline{\text{DCD}}$ (Data Carrier Detect)
 - $\overline{\text{CTS}}$ (Clear To Send)
 - FE (Framing Error)
 - OVRN (Receiver OverRun)
 - PE (Parity Error)
46. If we were using odd parity, give the parity bit (0 or 1) for the following data bits:
- 1101100
 - 0000000
 - 1100001
47. What bit pattern (for bits 4, 3, and 2 only) should we write into the control register to send/receive from a standard 10-character/sec. model 33 Teletype machine? (Assume even parity.)

System

48. Assume, as part of your system, that you have two MCM6810 RAMs that start at the very bottom of memory. Your program is to check the first five bytes of the second RAM. If the contents of that RAM location is an odd number, invert each bit and store this result back in that RAM location; if an even number, clear that RAM location. Generate a flowchart and write a program to accomplish the above task. The program is to start at hex location 2000.