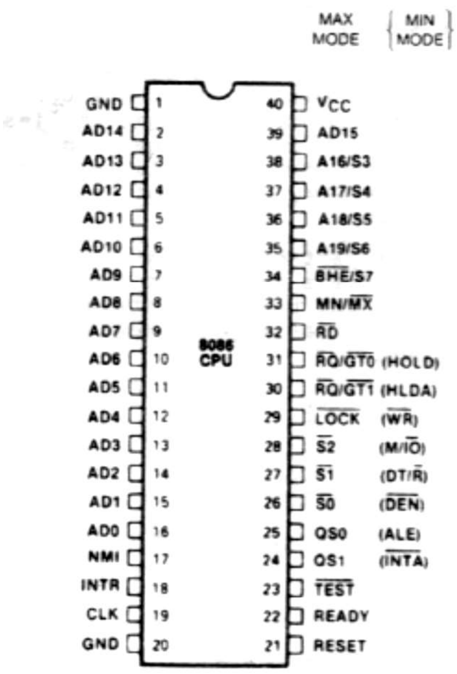
UNIT 2
8086 SYSTEM BUS STRUCTURE

8086 signals – Basic configurations – System bus timing –System design using 8086 – IO programming – Introduction to Multiprogramming – System Bus Structure – Multiprocessor configurations – Coprocessor, Closely coupled and loosely Coupled configurations – Introduction to advanced processors.

2.1 8086 SIGNALS


MAX MODE
MIN MODE
8086 CPU
GND 1, AD14 2, AD13 3, AD12 4, AD11 5, AD10 6, AD9 7, AD8 8, AD7 9, AD6 10, AD5 11, AD4 12, AD3 13, AD2 14, AD1 15, AD0 16, NMI 17, INTR 18, CLK 19, GND 20
40 VCC, 39 AD15, 38 A16/S3, 37 A17/S4, 36 A18/S5, 35 A19/S6, 34 BHE/S7, 33 MN/MX, 32 RD, 31 RQ/GT0 (HOLD), 30 RQ/GT1 (HLDA), 29 LOCK (WR), 28 S2 (M/IO), 27 S1 (DT/R), 26 S0 (DEN), 25 QS0 (ALE), 24 QS1 (INTA), 23 TEST, 22 READY, 21 RESET

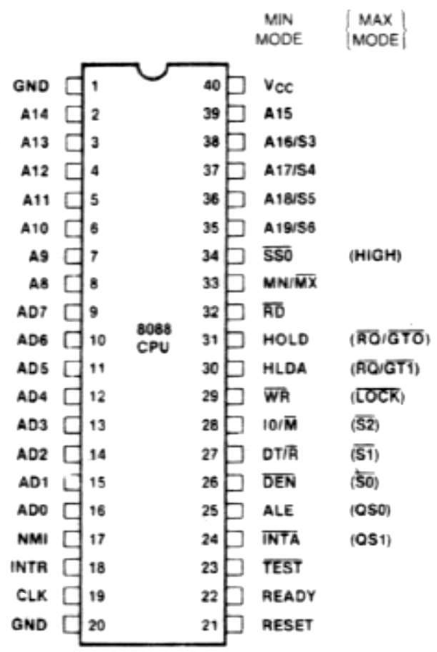

MIN MODE
MAX MODE
8088 CPU
GND 1, A14 2, A13 3, A12 4, A11 5, A10 6, A9 7, A8 8, AD7 9, AD6 10, AD5 11, AD4 12, AD3 13, AD2 14, AD1 15, AD0 16, NMI 17, INTR 18, CLK 19, GND 20
40 Vcc, 39 A15, 38 A16/S3, 37 A17/S4, 36 A18/S5, 35 A19/S6, 34 SS0 (HIGH), 33 MN/MX, 32 RD, 31 HOLD (RQ/GT0), 30 HLDA (RQ/GT1), 29 WR (LOCK), 28 IO/M (S2), 27 DT/R (S1), 26 DEN (S0), 25 ALE (QS0), 24 INTA (QS1), 23 TEST, 22 READY, 21 RESET

Fig 2.1: 8086 pin diagram **8088 pin diagram**

Pin(s)	Symbol	In/Out 3-State	Description
1	GND		Ground
2-16	AD14-AD0	I/O-3	Outputs address during the first part of the bus cycle and inputs or outputs data during the remaining part of the bus cycle.
17	NMI	I	Nonmaskable interrupt request - Positive-going edge triggered.
18	INTR	I	Maskable interrupt request – level triggered
19	CLK	I	Clock - 33% duty cycle, maximum rate depends on CPU model 5 MHz for 8086

			8 MHz for 8086-2 10 MHz for 8086-1
20	GND		Ground
21	RESET	I	Terminates activity, clears PSW, IP, DS, SS, ES, and the instruction queue, and sets CS to FFFF. Processing begins at FFFF0 when signal is dropped. Signal must be 1 for atleast 4 clock cycles.
22	READY	I	Acknowledgment from memory or I/O interface that CPU can complete the current bus cycle.
23	$\overline{\text{TEST}}$	I	Used in conjunction with the WAIT instruction in multiprocessing environments. A WAIT instruction will cause the CPU to idle, except for processing interrupts, until a 0 is applied to this pin
24-31	-	-	Definition depends on mode
32	$\overline{\text{RD}}$	0-3	Indicates a memory or I/O read is to be performed.
33	$\text{MN}/\overline{\text{MX}}$	I	CPU is in minimum mode when strapped to +5 V and in maximum mode when grounded.
34	$\overline{\text{BHE}/\text{S7}}$	0-3	If 0 during first part of bus cycle this pin indicates that at least one byte of the current transfer is to be made on pins AD15-AD8; if 1 the transfer is made on AD7-AD0. Status S7 is output during the latter part of bus cycle, but, presently, S7 has not been assigned a meaning.
35-38	A19/S6 - A16/S3	0-3	During the first part of the bus cycle the upper 4 bits of the address are output and during the remainder of the bus cycle status is output. S3 and S4 indicate the segment register being used as follows: S4 S3 Register 0 0 ES 0 1 SS 1 0 CS or none 1 1 DS S5 gives the current setting of IF. S6 is always 0
39	AD15	I/O-3	Same as AD14-AD0
40	VCC		Supply voltage +5 V ± 10%

2.2 BASIC CONFIGURATIONS

In order to adapt to as many situations as possible both the 8086 and 8088 have been given two modes of operation

1. Minimum mode and
2. Maximum mode.

The minimum mode is used for a small system with a single processor, a system in which the 8086/8088 generates all the necessary bus control signals directly (thereby minimizing the required bus control logic).The maximum mode is for medium-size to large systems, which often include two or more processors. In the maximum mode, the 8086/8088 encodes the basic bus control signals into 3 status bits, and uses the remaining control pins to provide the additional information that is needed to support a multiprocessor configuration.

Pin 33 (MN/$\overline{\text{MX}}$) determines the configuration option. When it is strapped to ground the processor is to be used in a maximum mode configuration and when it is strapped to + 5 V it is to be operated in its minimum mode. Both processors multiplex the address and data signals and both have 20 address pins with address and status signals being multiplexed on the 4 most significant address pins. However, because the 8088 can only transfer 8 bits of data at a time, only eight of its pins are used for data, as opposed to 16 for the 8086. Except for pins 28 and 34 the two processors have the same control pin .definitions. Pin 28 differs only in the minimum mode. For the 8088 this minimum mode signal is inverted from that of the 8086, so that the 8088 is compatible with the Intel 8085 microcomputer chip.

On the 8086, pin 34 $\overline{\text{BHE}}$ designates whether or not at least 1 byte of a transfer is to be made on AD15 through AD8. A 0 on this pin indicates that the more significant data lines are to be used; otherwise, only AD7 through ADO are used. Together the $\overline{\text{BHE}}$ and A0 signals indicate to the interfaces connected to the bus how the data are to appear on the bus. The four possible combinations are defined as follows:

Operation	$\overline{\text{BHE}}$	A0	Data pins used
Write/read a word at an even address	0	0	AD15-AD0
Write/read a byte at an even address	1	0	AD7-AD0
Write/read a word at an odd address	0	1	AD15-AD8
Write/read a byte at an odd address	0	1	AD15-AD8 (First bus cycle: puts the least significant data byte on AD15-AD8)
	1	0	AD7-AD0 (Next bus cycle: puts the most significant data byte on AD7-AD0)

where 0 is low and 1 is high.

Because, on the 8088, only AD7-AD0 can transfer data, this pin is not needed to indicate the upper or lower half of the data bus and is free to provide status information.

Pins 1 and 20 are grounded.

Pins 2 through 16 and 39 (AD15-AD0) hold the address needed for the transfer during the first part of the bus cycle, and are free to transfer the data during the remaining part of the cycle.

Pins 17 and 18 (NMI and INTR) are for interrupt requests.

Pin 19 (CLK) is for supplying the clock signal that synchronizes the activity within the CPU.

Pin 21 (RESET) is for inputting a system reset signal. Most systems include a line that goes to all system components and a pulse is automatically sent over this line when the system is turned on, or the reset pulse can be manually generated by a switch that allows the operator to reinitialize the system. A 1 on the reset line causes the components to go to their "turn on" state. For the processor this state is having the PSW, IP, DS, SS, ES, and instruction queue cleared and CS set to FFFF. With (IP) = 0000 and (CS) = FFFF the processor will begin executing at FFFF0. Normally, this location would be in a read-only section of memory and would contain a JMP instruction to a program for initializing the system and loading the application software or operating system. Such a program is referred to as a bootstrap loader.

Pin 22 (READY) is for inputting an acknowledge from a memory or I/O interface that input data will be put on the data bus or output data will be accepted from the data bus within the next clock cycle. In either case, the CPU and its bus control logic can complete the current bus cycle after the next clock cycle. Pin 23 ($\overline{\text{TEST}}$) is used in conjunction with the WAIT instruction and is employed primarily in multiprocessing situations. Pins 24 through 31 are mode dependent. Pin 32 ($\overline{\text{RD}}$) indicates that an input operation is to be performed and, in minimum mode, is used along with pin 28, which distinguishes a memory transfer from an I/O transfer, and pin 29, which indicates an output operation, to determine the type of transfer.

During the first part of a bus cycle pins 35-38 (AD19/S6-AD16/S3) output the 4 high-order bits of the address, and during the remaining part of the cycle they output status information. Status bits S3 and S4 indicate the segment register that is being used to generate the address and bit S5 reflects the contents of the IF flag. S6 is always held at 0 and indicates that an 8086/8088 is controlling the system bus.

Pin 40 (VCC) receives the supply voltage, which must be + 5 V ± 10%. Systems based on an 8086 or 8088 are ordinarily designed so that only a TTL compatible + 5-V supply voltage and ground are needed, thus simplifying the design of the power supply.

2.3 MINIMUM MODE

2.3.1 Minimum Mode signals

A processor is in minimum mode when its $\text{MN}/\overline{\text{MX}}$ pin is strapped to + 5 V. The definitions for pins 24 through 31 for the minimum mode are given below:

Pin	Symbol	In/Out 3-state	Description
24	$\overline{\text{INTA}}$	O-3	Indicates recognition of an interrupt request. Consists of two negative going pulses in two consecutive bus cycles.
25	ALE	O	Outputs a pulse at the beginning of the bus cycle and is to indicate an address is available on the address pins.
26	$\overline{\text{DEN}}$	O-3	Output during the latter portion of the bus cycle and is to inform the transceivers that the CPU is ready to send or receive data.
27	$\text{DT}/\overline{\text{R}}$	O-3	Indicates to the set of transceivers whether they are to transmit (1) or receive (0) data.
28	$\text{M}/\overline{\text{IO}}$	O-3	Distinguishes a memory transfer from an I/O transfer. For a memory transfer it is .
29	$\overline{\text{WR}}$	O-3	When 0, it indicates a write operation is being performed. It is used in conjunction with pins 28 ($\text{M}/\overline{\text{IO}}$) and 32 ($\overline{\text{RD}}$) to specify

			the type of transfer
30	HLDA	O	Outputs a bus grant to a requesting master. Pins with tristate gates are put in high impedance state while HLDA=1.
31	HOLD	I	Receives bus requests from bus masters. The8086/8088 will not gain control of the bus until this signal is dropped.

2.3.2 Minimum mode system configuration

A typical minimum mode configuration is shown in Fig 2.2

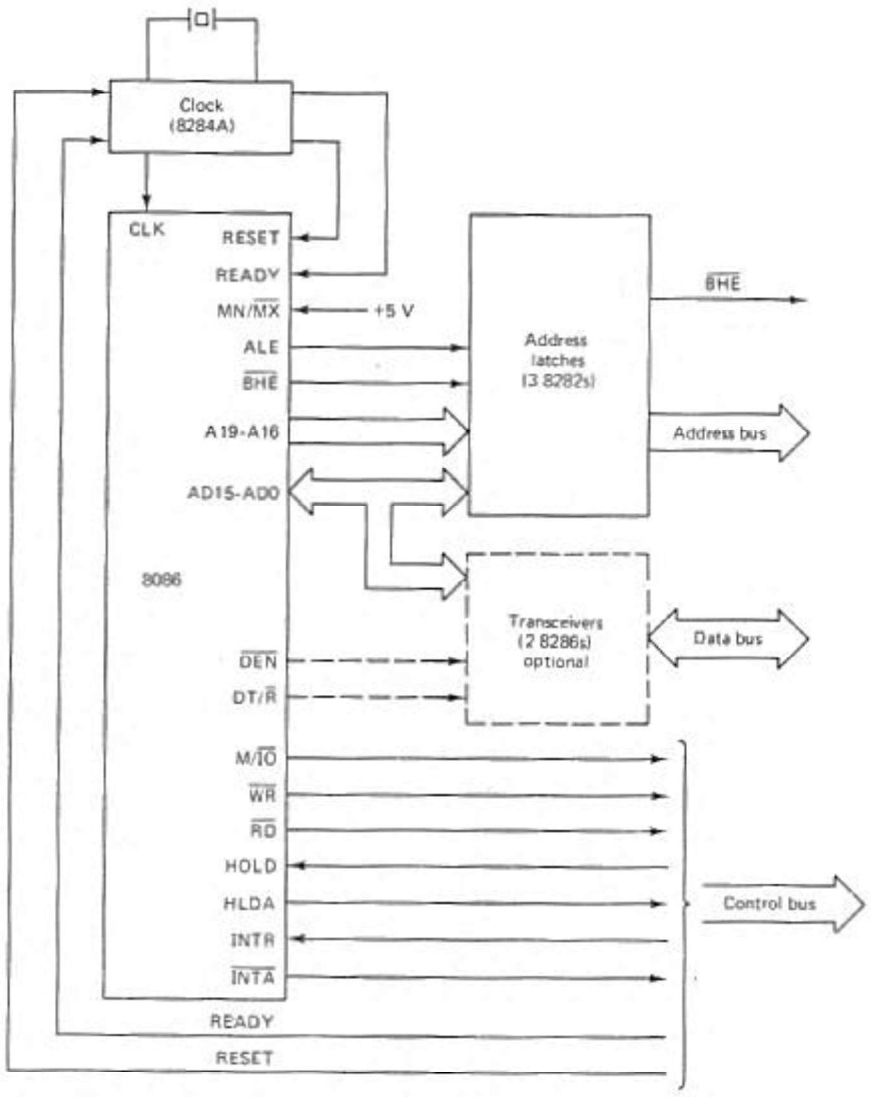
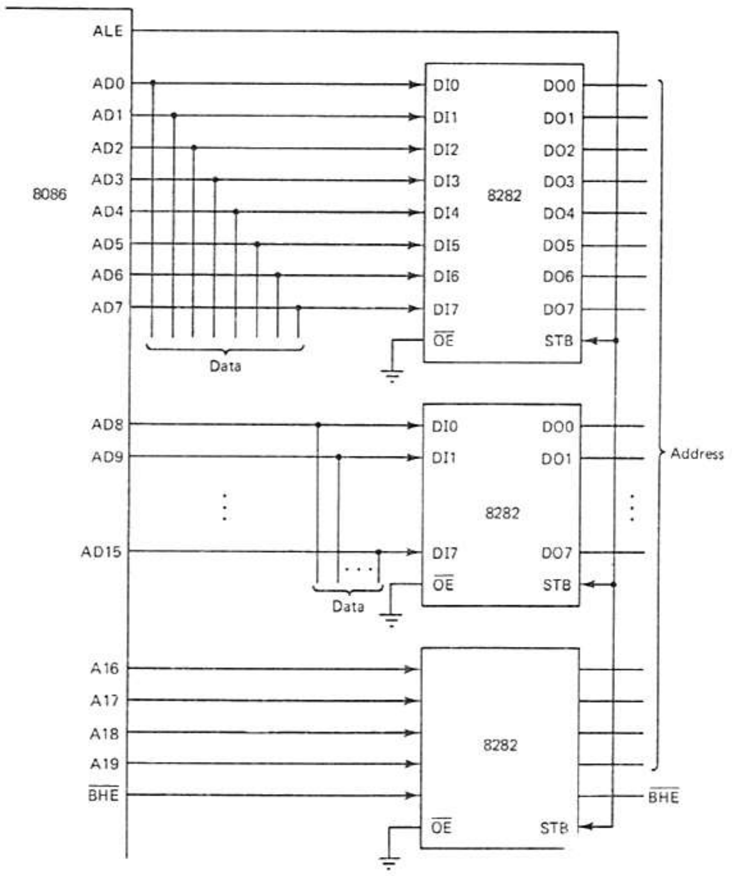


Fig 2.2: Minimum mode system

2.3.3 Address latch (8282):

The address must be latched since it is available only during the first part of the bus cycle. To signal that the address is ready to be latched a 1 is put on pin 25, the address latch enable (ALE) pin. Typically, the latching is accomplished using Intel 8282s, as shown in Fig 2.3.

Because 8282 is an 8-bit latch, two of them are needed for a 16-bit address and three are needed if a full 20-bit address is used. In an 8086 system, $\overline{\text{BHE}}$ would also have to be latched. A signal on the STB pin latches the bits applied to the input data lines DI7-DI0. Therefore, STB is connected to the 8086's ALE pin and DI7-DI0 are attached to eight of the address lines. An active low signal on the $\overline{\text{OE}}$ enables the latch's outputs DO7-DO0, and a 1 at this pin forces the outputs into their high-impedance state.

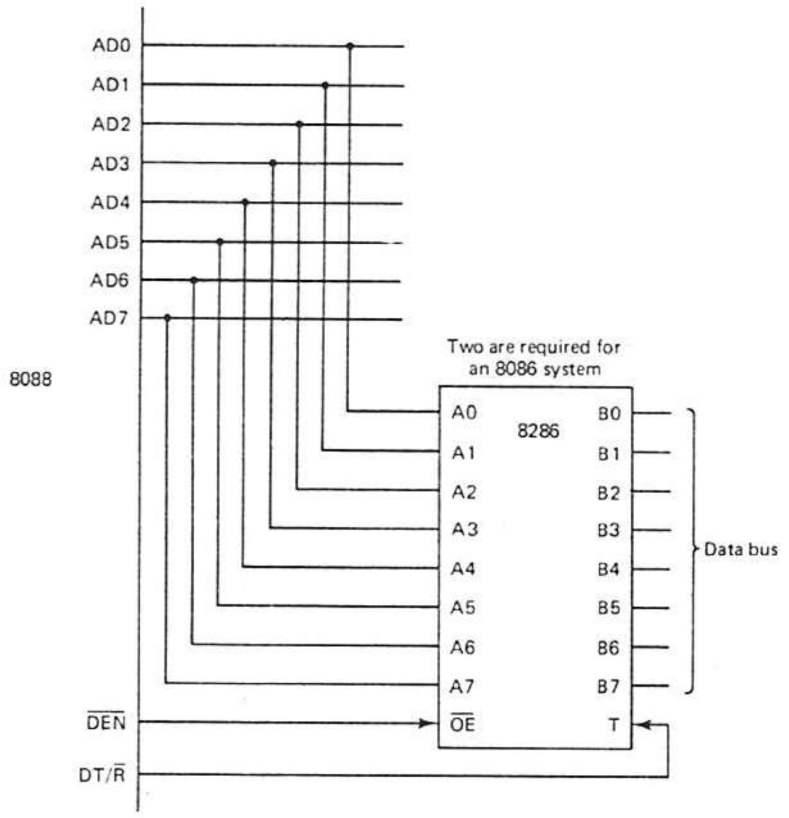


DI – Data input line ; DO – Data output line

Fig 2.3: Application of 8282 latches.

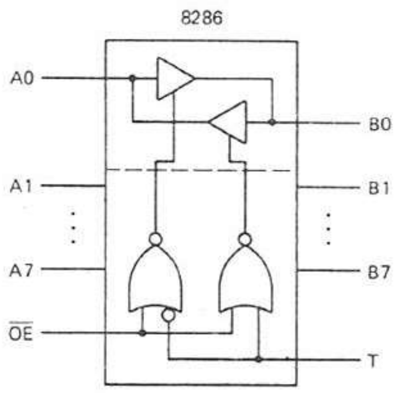
2.3.4 Transceiver (8286):

The transceiver (driver/receiver) 8286 contains 16 tristate elements, eight receivers, and eight drivers. Therefore, only one 8286 is needed to service all of the data lines for an 8088, but two are required in an 8086 system.



A0 to A7 – local bus: B0 to B7 – system bus

(a) 8088 connections



(b) Internal logic

Fig 2.4: Application and internal logic of an 8286

The 8286 is symmetric with respect to its two sets of data pins, either the pins A7-A0 can be the inputs and B7-B0 the outputs, or vice versa. The output enable ($\overline{OE}$) pin determines whether or not data are allowed to pass through the 8286 and the transmit (T) pin controls the direction of the data flow. When $\overline{OE}$ = 1, data are not transmitted through the 8286 in either direction. If it is 0, then T = 1 causes A7-A0 to be the inputs and T = 0 results in B7-B0 being the inputs. In an 8086/8088-based system the $\overline{OE}$ pin would be connected to the $\overline{DEN}$ pin, which is active low whenever the processor is performing an I/O operation. The A7-A0 pins are connected to the appropriate address/data lines and the T pin is tied to the processor's $DT/\overline{R}$ pin. Thus, when the processor is outputting the data flow is from A7-A0 to B7-B0, and when it is inputting the flow is in the other direction. The processor floats the $\overline{DEN}$ and $DT/\overline{R}$ pins in response to a bus request on the HOLD pin.

2.3.5 Clock Generator (8284):

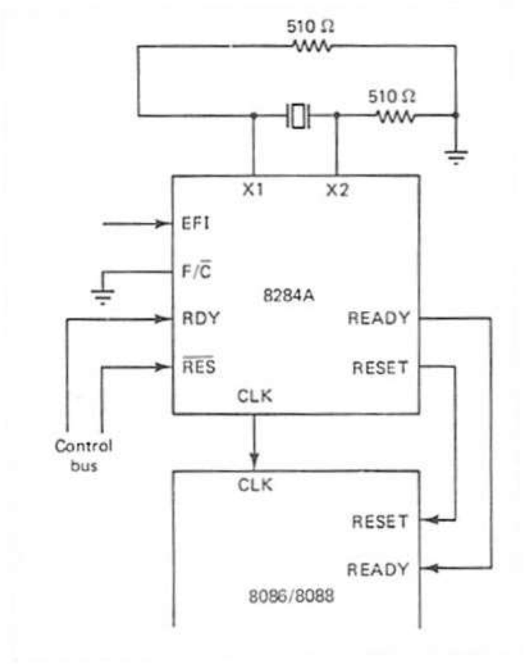


Fig 2.5: Typical 8284 A clock connection

In addition to supplying a train of pulses at a constant frequency it synchronizes ready (RDY) signals, which indicate an interface is ready to complete a transfer, and reset ($\overline{RES}$) signals, which initialize the system, with the clock pulses. Although these two signals may be sent at any time, the 8284A will not reflect them in its READY and RESET outputs until the trailing edge of the clock pulse in which they are received.

The frequency source applied to the 8284A may be from a pulse generator that is connected to the EFI pin or an oscillator that is connected across Xl and X2. If the input to $F/\overline{C}$is 1, then the EFI input determines the frequency; otherwise, it is the oscillator input. In either case the clock output CLK is one-third of the input frequency.

In a minimum system the control lines do not need to be passed through transceivers, but can be used directly. The$M/\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ lines specify the type of transfer according to the following table:

$M/\overline{IO}$	$\overline{RD}$	$\overline{WR}$	
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

where 0 is low and 1 is high.

The interrupt acknowledge ($\overline{INTA}$) signal consists of two negative pulses output during two consecutive bus cycles. The first pulse informs the interface that its request has been recognized, and upon receipt of the second pulse, the interface is to send the interrupt type to the processor over the data bus.

2.4 MAXIMUM MODE

2.4.1 Maximum mode signals:

A processor is in maximum mode when its $MN/\overline{MX}$ pin is grounded. The maximum mode definitions of pins 24 through 31 are given below:

Pin	Symbol	In/Out 3-state	Description
24,25	QS1, QS0	O	Reflects the status of the instruction queue. This status indicates the activity in the queue during the previous clock cycle
26, 27,28	$\overline{S0}, \overline{S1}, \overline{S2}$	O-3	Indicates the type of transfer to take place during the current bus cycle: $\overline{S2}\overline{S1}\overline{S0}$ 0 0 0 Interrupt acknowledge 0 0 1 Read I/O port 0 1 0 Write I/O port 0 1 1 Halt 1 0 0 Instruction fetch 1 0 1 Read memory 1 1 0 Write memory 1 1 1 Inactive - passive (1 represents high and 0 represents low.)The status becomes active prior to the beginning of a bus cycle and returns to inactive during the later part of the cycle.
29	$\overline{LOCK}$	O-3	Indicates the bus is not to be relinquished to other potential bus masters. It is initiated by a LOCK instruction prefix and is maintained until the end of the next instruction. It is also active during and between the two $\overline{INTA}$ pulses.
30	$\overline{RQ}/\overline{GT1}$	I/O	For inputting bus requests and outputting bus grants

31	$\overline{RQ/GT0}$	I/O	Same as $\overline{RQ/GT1}$except that a request on $\overline{RQ/GT0}$has higher priority.

2.4.2 Maximum mode System Configuration:

A typical maximum mode configuration is shown in Fig. 2.6. It is clear from Fig. 2.6that the main difference between minimum and maximum mode configurations is the need for additional circuitry to translate the control signals.

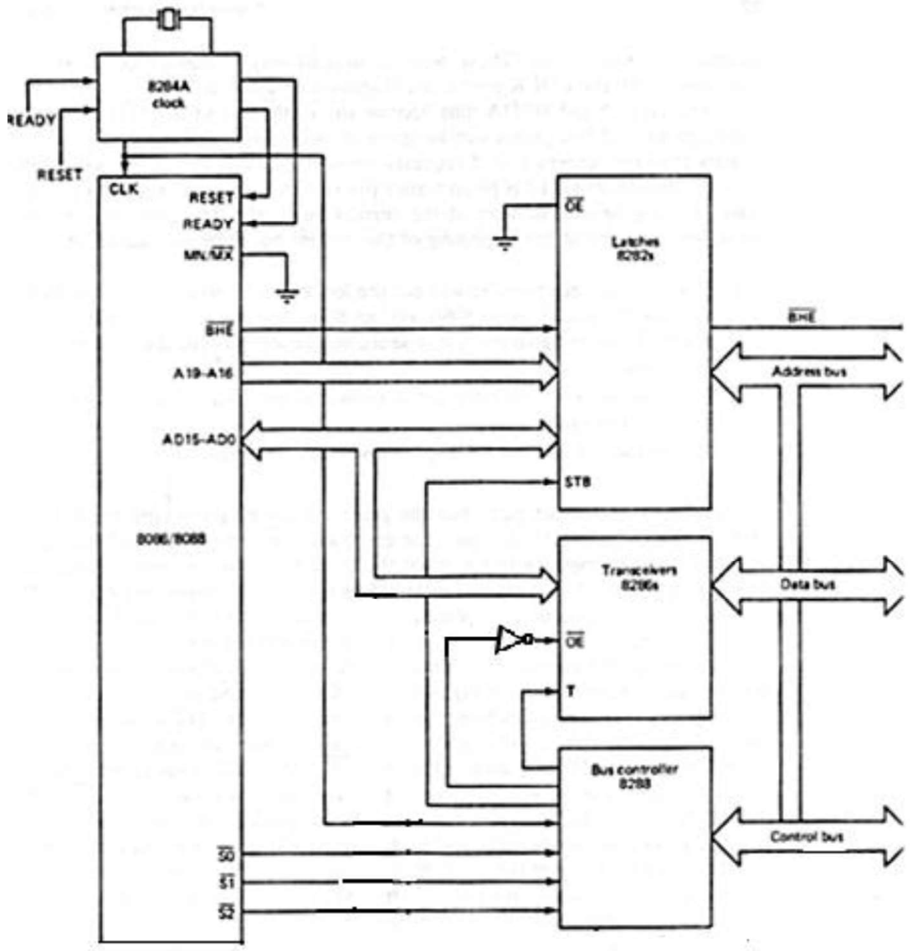


Fig 2.6: Typical maximum mode configuration

This circuitry is for converting the status bits $\overline{S0}, \overline{S1}, \overline{S2}$ into the I/O and memory transfer signals needed to direct data transfers, and for controlling the 8282 latches and 8286 transceivers. It is normally implemented with an Intel 8288 bus controller. Also included in the system is an interrupt priority management device; however, its presence is optional.

The $\overline{S0}, \overline{S1}, \overline{S2}$ status bits specify the type of transfer that is to be carried out and when used with an 8288 bus controller they obviate the need for the $M/\overline{IO}$ (or $IO/\overline{M}$), $\overline{WR}$, $\overline{INTA}$, ALE, $DT/\overline{R}$, and $\overline{DEN}$ signals that are output over pins 24 through 29 when the processor is operating in minimum mode.

Except for the case $\overline{S0} = \overline{S1} = 1$, $\overline{S2}$=0 indicates a transfer between an I/O interface and the CPU and $\overline{S2}$=1 implies a memory transfer. The $\overline{S1}$ bit specifies whether an input or output is to be performed. From the status the 8288 is able to originate the address latch enable signal to the 8282s, the enable and direction signals to the 8286 transceivers, and the interrupt acknowledge signal to the interrupt controller.

The QS0 and QS1 pins are to allow the system external to the processor to interrogate the status of the processor instruction queue so that it can determine which instruction it is currently executing, and the $\overline{\text{LOCK}}$ pin indicates that an instruction with a LOCK prefix is being executed and the bus is not to be used by another potential master. These pins are needed only in multiprocessor systems and, along with the LOCK prefix.

The HOLD and HLDA pins become the$\overline{\text{RQ}}/\overline{\text{GT0}}$and $\overline{\text{RQ}}/\overline{\text{GT1}}$pins. Both bus requests and bus grants can be given through each of these pins. They are exactly the samethat if requests are seen on both pins at the same time, then the one on $\overline{\text{RQ}}/\overline{\text{GT0}}$ is given higher priority. A request consists of a negative pulse arriving before the start of the current bus cycle. The grant is a negative pulse that is issued at the beginning of the current bus cycle provided that:

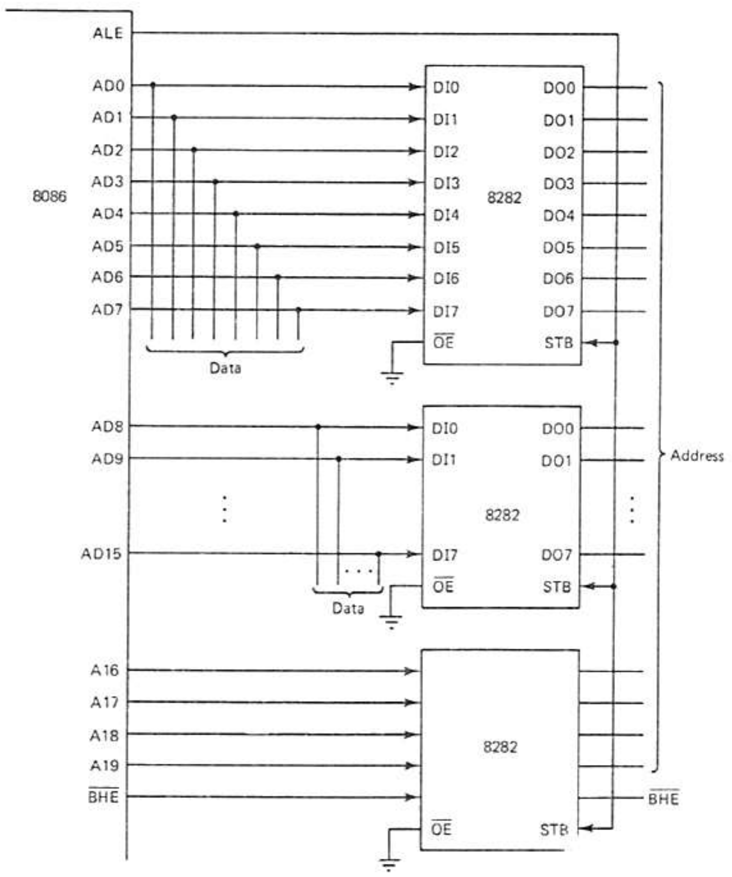
1. The previous bus transfer was not the low byte of a word to or from an odd address if the CPU is an 8086. For an 8088, regardless of the address alignment, the grant signal will not be sent until the second byte of a word reference is accessed.
2. The first pulse of an interrupt acknowledgment did not occur during the previous bus cycle.
3. An instruction with a LOCK prefix is not being executed.

If condition 1 or 2 is not met, then the grant will not be given until the next bus cycle, and if condition 3 is not met, the grant will wait until the locked instruction is completed. In response to the grant the three-state pins are put in their high impedance state and the next bus cycle will be given to the requesting master. The processor will be effectively disconnected from the system bus until the master sends a second pulse to the processor through the $\overline{\text{RQ}}/\overline{\text{GT}}$pin.

2.4.3 Address latch (8282):

The address must be latched since it is available only during the first part of the bus cycle. To signal that the address is ready to be latched a 1 is put on pin 25, the address latch enable (ALE) pin. Typically, the latching is accomplished using Intel 8282s, as shown in Fig 2.7.

Because 8282 is an 8-bit latch, two of them are needed for a 16-bit address and three are needed if a full 20-bit address is used. In an 8086 system, $\overline{\text{BHE}}$ would also have to be latched. A signal on the STB pin latches the bits applied to the input data lines DI7-DI0. Therefore, STB is connected to the 8086's ALE pin and DI7-DI0 are attached to eight of the address lines. An active low signal on the $\overline{\text{OE}}$ enables the latch's outputs DO7-DO0, and a 1 at this pin forces the outputs into their high-impedance state.



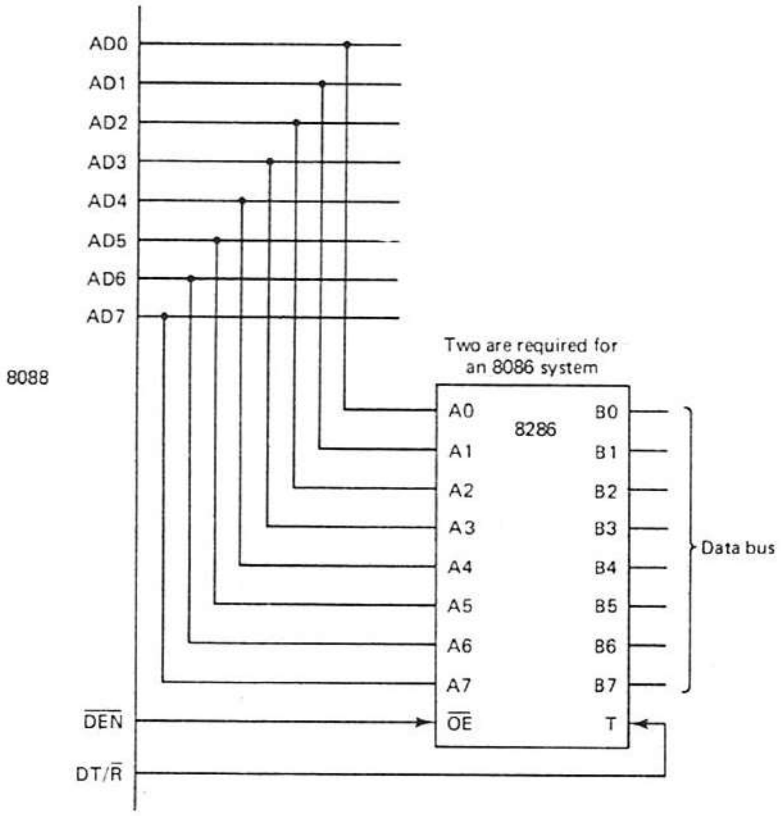
DI – Data input line ; DO – Data output line

Fig 2.7: Application of 8282 latches.

2.4.4 Transceiver (8286):

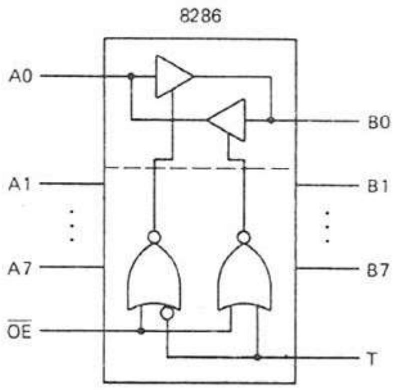
The transceiver (driver/receiver) 8286 contains 16 tristate elements, eight receivers, and eight drivers. Therefore, only one 8286 is needed to service all of the data lines for an 8088, but two are required in an 8086 system.

The 8286 is symmetric with respect to its two sets of data pins, either the pins A7-A0 can be the inputs and B7-B0 the outputs, or vice versa. The output enable ($\overline{\text{OE}}$) pin determines whether or not data are allowed to pass through the 8286 and the transmit (T) pin controls the direction of the data flow. When $\overline{\text{OE}}$ = 1, data are not transmitted through the 8286 in either direction. If it is 0, then T = 1 causes A7-A0 to be the inputs and T = 0 results in B7-B0 being the inputs. In an 8086/8088-based system the $\overline{\text{OE}}$ pin would be connected to the $\overline{\text{DEN}}$ pin, which is active low whenever the processor is performing an I/O operation. The A7-A0 pins are connected to the appropriate address/data lines and the T pin is tied to the processor's $\text{DT}/\overline{\text{R}}$ pin. Thus, when the processor is outputting the data flow is from A7-A0 to B7-B0, and when it is inputting the flow is in the other direction. The processor floats the $\overline{\mathrm{DEN}}$ and $\mathrm{DT}/\overline{\mathrm{R}}$ pins in response to a bus request on the HOLD pin.



A0 to A7 – local bus: B0 to B7 – system bus

(a) 8088 connections



(b) Internal logic

Fig 2.8: Application and internal logic of an 8286

2.4.5 Clock Generator (8284):

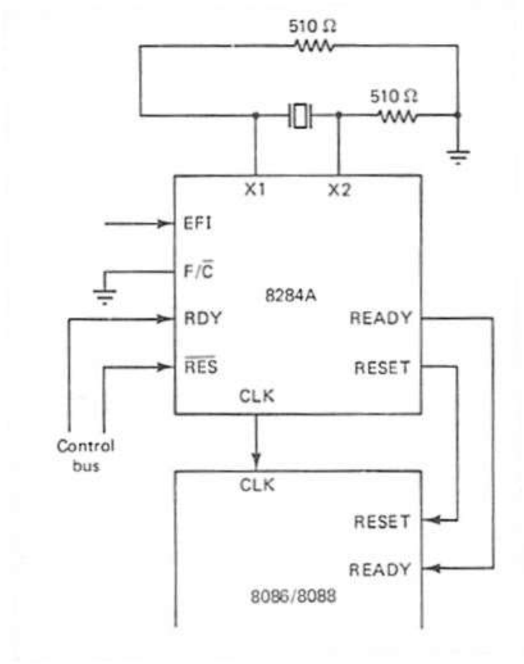


Fig 2.9: Typical 8284 A clock connection

In addition to supplying a train of pulses at a constant frequency it synchronizes ready (RDY) signals, which indicate an interface is ready to complete a transfer, and reset ($\overline{\text{RES}}$) signals, which initialize the system, with the clock pulses. Although these two signals may be sent at any time, the 8284A will not reflect them in its READY and RESET outputs until the trailing edge of the clock pulse in which they are received.

The frequency source applied to the 8284A may be from a pulse generator that is connected to the EFI pin or an oscillator that is connected across Xl and X2. If the input to $F/\overline{C}$is 1, then the EFI input determines the frequency; otherwise, it is the oscillator input. In either case the clock output CLK is one-third of the input frequency.

In a minimum system the control lines do not need to be passed through transceivers, but can be used directly. The$M/\overline{IO}$, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ lines specify the type of transfer according to the following table:

$M/\overline{IO}$	$\overline{\text{RD}}$	$\overline{\text{WR}}$	
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

where 0 is low and 1 is high.

The interrupt acknowledge ($\overline{\text{INTA}}$) signal consists of two negative pulses output during two consecutive bus cycles. The first pulse informs the interface that its request has been recognized, and upon receipt of the second pulse, the interface is to send the interrupt type to the processor over the data bus.

2.4.6 Bus Controller (8288):

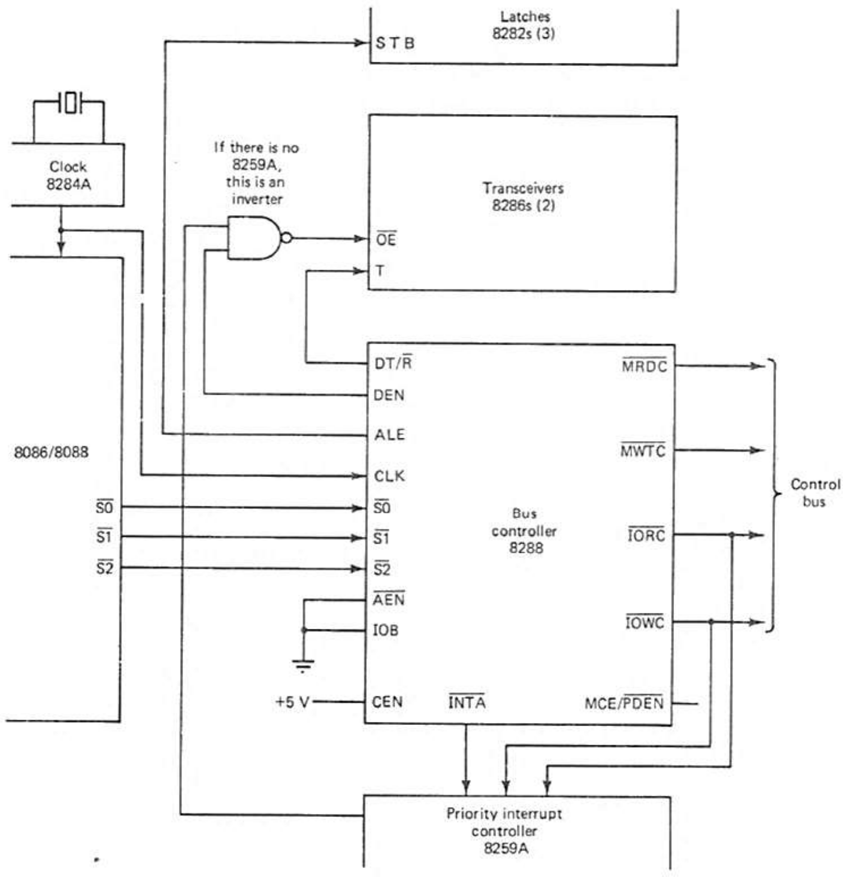


Fig 2.10: Connections to an 8288 bus controller

The $\overline{S0}, \overline{S1}, \overline{S2}$pins are for receiving the corresponding status bits from the processor. The ALE, DT/$\overline{R}$,and DEN pins provide the same outputs that are sent by the processor when it is in minimum mode (except that DEN is inverted from $\overline{DEN}$). The CLK input permits the bus controller activity to be synchronized with that of the processor. The $\overline{AEN}$, IOB and CEN pins are for multiprocessor systems. In a single-processor system $\overline{AEN}$ and IOB are normally grounded and a 1 is applied to CEN. The meaning of theMCN/$\overline{PDEN}$ output depends on the mode, which is determined by the signal applied to IOB. When IOB is grounded it assumes its master cascade enable (MCE) meaning and can be used to control cascaded 8259. In the event that +5 V is connected to IOB, the peripheral data enable ($\overline{PDEN}$) meaning, which is used in multiple-bus configurations, is assumed. The remaining pins have the following definitions:

$\overline{INTA}$-Issues the two interrupt acknowledgment pulses to a priority interrupt controller or an interrupting device when $\overline{S0} = \overline{S1} = \overline{S2}$=0

$\overline{\text{IORC}}$ (I/O Read Command)-Instructs an I/O interface to put the data contained in the addressed port on the data bus.
$\overline{\text{IOWC}}$ (I/O Write Command)- Instructs an I/O interface to accept the data on the data bus and put the data into the addressed port.
$\overline{MRDC}$ (Memory Read Command)-Instructs the memory to put the contents of the addressed location on the data bus.
$\overline{\text{MWTC}}$(Memory Write Command)-Instructs the memory to accept the data on the data bus and put the data into the addressed memory location.These signals are active low and are output during the middle portion of a bus cycle. Clearly, only one of them will be issued during any given bus cycle.

$\overline{\text{AIOWC}}$ (Advanced I/O write command) and $\overline{\text{AMWC}}$ (advanced memory write command) pins serve the same purposes as the $\overline{\text{IOWC}}$ and$\overline{\text{MWTC}}$ pins except that they are activated one clock pulse sooner. This gives slow interfaces an extra clock cycle to prepare to input the data. As with the other 8086 supporting devices, the 8288 requires a +5-V supply voltage and has TIL-compatible inputs and outputs.

2.5 SYSTEM BUS TIMING

The length of a bus cycle in an 8086 system is four clock cycles, denoted T_1 through T_4, plus an indeterminate number of wait state clock cycles, denoted T_W. If the bus is to be inactive after the completion of a bus cycle, then the gap between successive cycles is filled with idle state clock cycles represented by T_1. Wait states are inserted between T_3 and T_4 when a memory or I/O interface is not able to respond quickly enough during a transfer. A typical succession of bus cycles is given in Fig. 2.11.

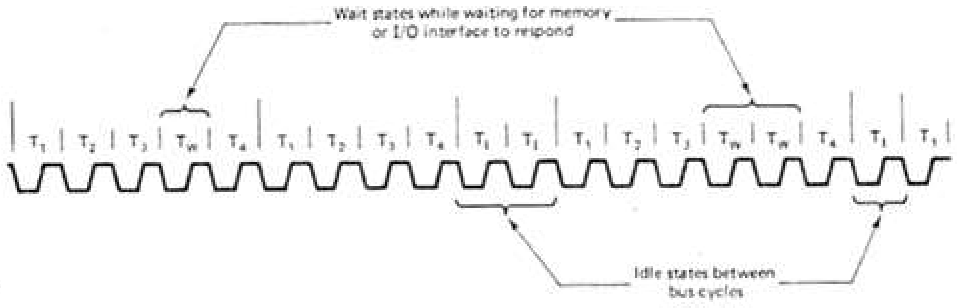
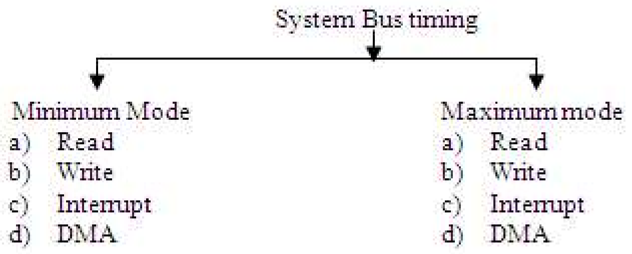

Wait states while waiting for memory or I/O interface to respond
T1 T2 T3 TW T4 T1 T2 T3 T4 T1 T1 T1 T2 T3 TW TW T4 T1 T1
Idle states between bus cycles

Fig 2.11: Typical sequence of bus cycles



2.5.1 Minimum Mode bus timing diagrams

a) Memory/ IO Read for minimum mode

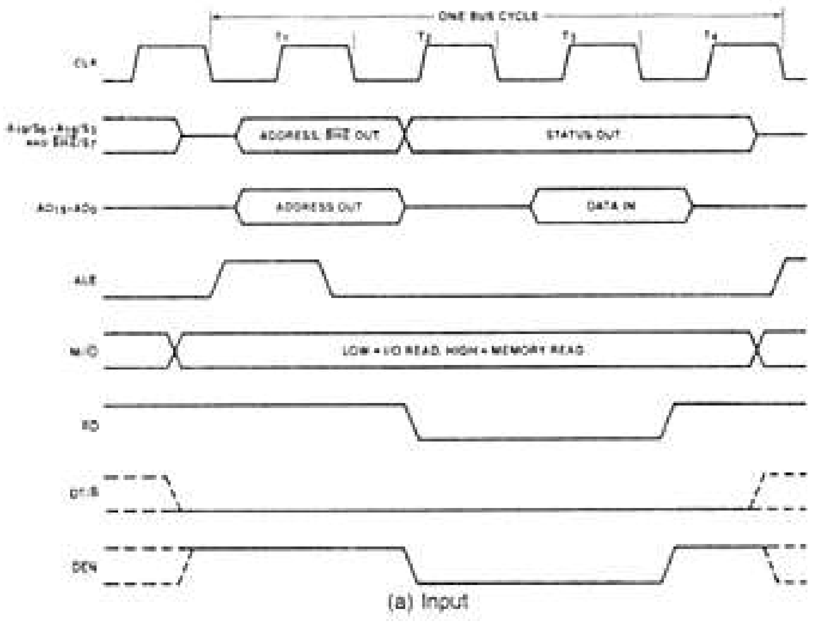


Fig 2.12(a) 8086 minimum mode bus timing diagrams

In minimum mode MN/$\overline{MX}$ is placed at 5 V level. The following activities take place during different clock cycles.

T1:

- ALE is pulsed high.
- $\overline{BHE}$ is made high/low depending on 8/16 bit read at odd/even address boundary.
- M/$\overline{IO}$ is made high to indicate memory operation. It remain high during the entire bus cycle.
- DT/$\overline{R}$ is low and remains low throghout the cycle, to indicate the direction of data transfer as memory to the processor.
- Address is put in the address bus. The falling edge of ALE is used to latch the address from the address bus.

T2:

- Bus is turned around.
- $\overline{RD}$ goes low as read-control signal.
- DEN goes high to enable the 8286 transceiver.
- $\overline{BHE}$ goes high if it was made low in T1.
- Status is put on the A_{16}-A_{19} lines. The activity starts in T2 and continues till T4.

T3:

- DEN goes low.
- Data is put on lines $AD_0 - AD_{15}$

T4:

- $M/\overline{IO}$ goes low.
- $\overline{RD}$ goes low.

The I/O read bus cycle will have only one signal that is different, i.e. $M/\overline{IO}$ will go low in T1and will become high in T4.

b) Memory/ IO Write for minimum mode

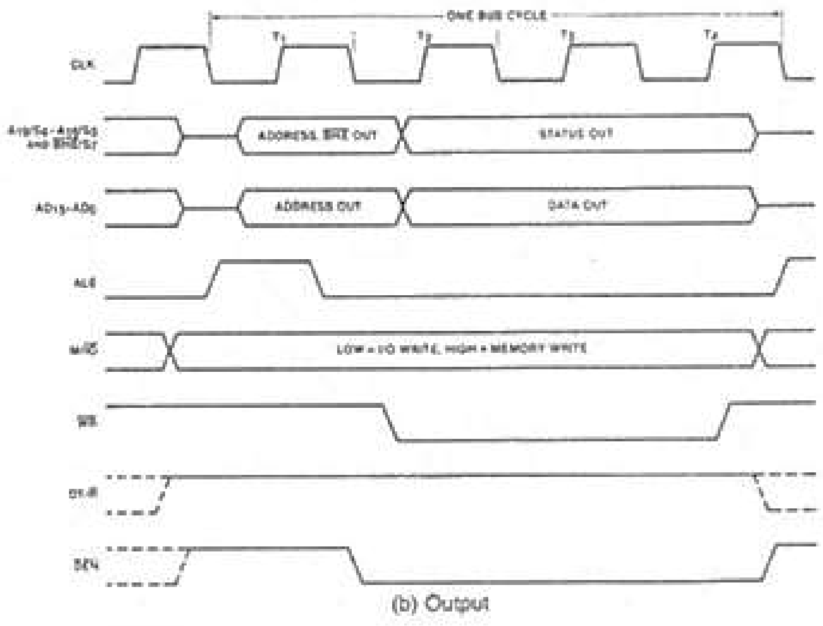


Fig 2.12 (b) : 8086 minimum mode bus timing diagrams

The following activities take place during different clock cycles.

T1:

- ALE is pulsed high.
- $\overline{BHE}$ is made high/low depending on 8/16 bit read at odd/even address boundary.
- $M/\overline{IO}$ is made high to indicate memory operation. It remain high during the entire bus cycle.
- $DT/\overline{R}$ is low and remains low throghout the cycle, to indicate the direction of data transfer as memory to the processor.
- Address is put in the address bus. The falling edge of ALE is used to latch the address from the address bus.
- DEN goes high to enable the 8286 transceivers.

T2:

- $\overline{WR}$ goes low as write control signal.
- $\overline{BHE}$ goes high if it was made low in T1.
- Bus is turned around.
- Status is put on the A_{16}-A_{19} lines. The activity starts in T2 and continues till T4.

T3:

- Data is put on lines $AD_0 - AD_{15}$

T4:

- M/$\overline{IO}$ goes low.
- $\overline{WR}$ goes low.
- DT/$\overline{R}$ goes low.
- DEN goes low.

The I/O read bus cycle will have only one signal that is different, i.e. M/$\overline{IO}$ will go low in T1and will become high in T4.

c) Interrupt Acknowledgement

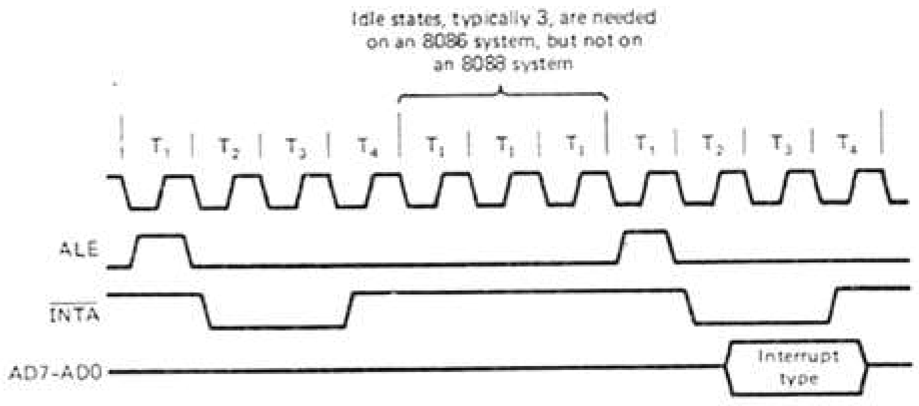


Fig 2.13: Interrupt acknowledgement

The timing diagram for an interrupt acknowledge is shown in Fig 2.13. If an interrupt request has been recognized during the previous bus cycle and an instruction has just been completed, then a negative pulse will be applied to $\overline{\text{INTA}}$ during the current bus cycle and the next bus cycle. Each of these pulses will extend fromT_2 to T_4. Upon receiving the second pulse, the interface accepting the acknowledgment will put the interrupt type on AD7-AD0, which are floated the rest of the time during the two bus cycles. The type will be available from T_2 to T_4·

d) DMA

Figure 2.14 shows the timing of a bus request and bus grant in a minimum mode system.

The HOLD pin is tested at the leading edge of each clock pulse. If a HOLD signal is received by the processor before T_4 or during a T_1 state, then the CPU activates HLDA and the succeeding bus cycles will be given to the requesting master until that master drops its request.

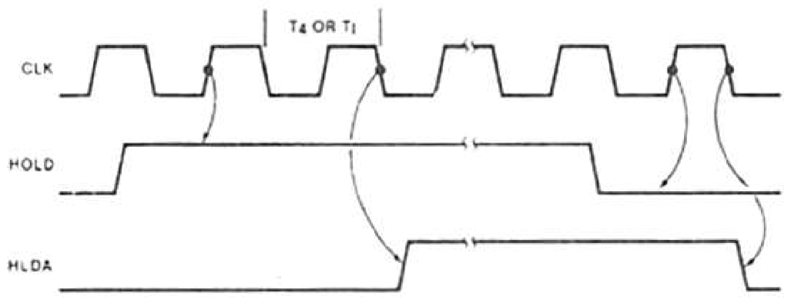


Fig 2.14: Bus request and bus grant timing on a minimum mode system

The lowered request is detected at the rising edge of the next clock cycle and the HLDA signal is dropped at the trailing edge of that clock cycle. While HLDA is 1, all of the processor's three-state outputs are put in their high-impedance state. Instructions already in the instruction queue will continue to be executed until one of them requires the use of the bus.

2.5.2 Maximum mode timing diagrams

The timing diagrams for input and output transfers on a maximum modesystem are given in Fig. 2.15.

a) Memory/ IO Read for maximum mode

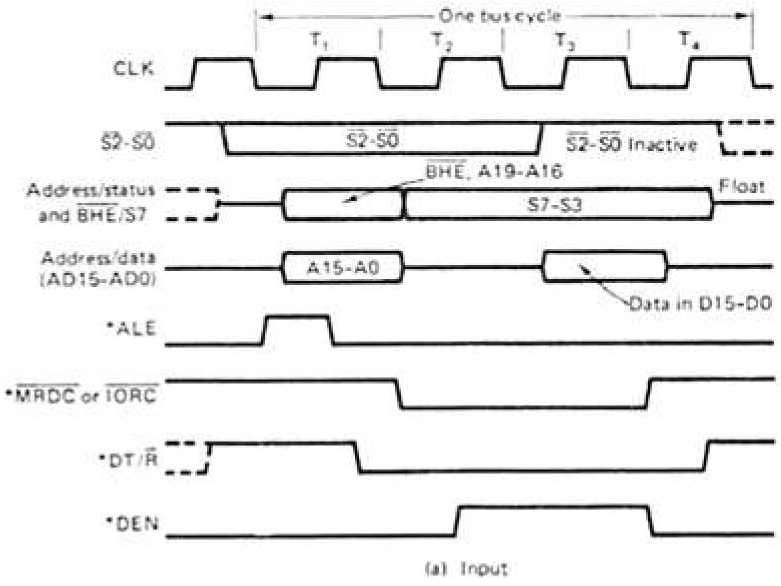


Fig 2.15 (a) : Timing diagrams for a maximum mode system.

T1:

- $\overline{S0}$, $\overline{S1}$, $\overline{S2}$ are set by 8086 in the beginning of clock cycle. It is decoded by the 8288 bus controller.
- ALE is pulsed high.

- $\overline{BHE}$ is made high/low depending on 8/16 bit read at odd/even address boundary.
- M/$\overline{IO}$ is made high to indicate memory operation. It remain high during the entire bus cycle.
- DT/$\overline{R}$ is low.
- Address is put in the address bus.
- ALE is pulsed low. The falling edge of ALE is used to latch the address from the address/data bus.

T2:

- $\overline{BHE}$ goes high if it was made low in T1.
- DEN goes high to enable the 8286 transceiver.
- $\overline{MRDC}$ goes low as memory read-control signal.

T3:

- Data is put on lines $AD_0 - AD_{15}$
- Status lines $\overline{S0}$, $\overline{S1}$, $\overline{S2}$ become inactive.

T4:

- $\overline{MRDC}$ goes high.
- DEN goes low
- DT/$\overline{R}$ goes high

b) Memory/ IO Read for maximum mode

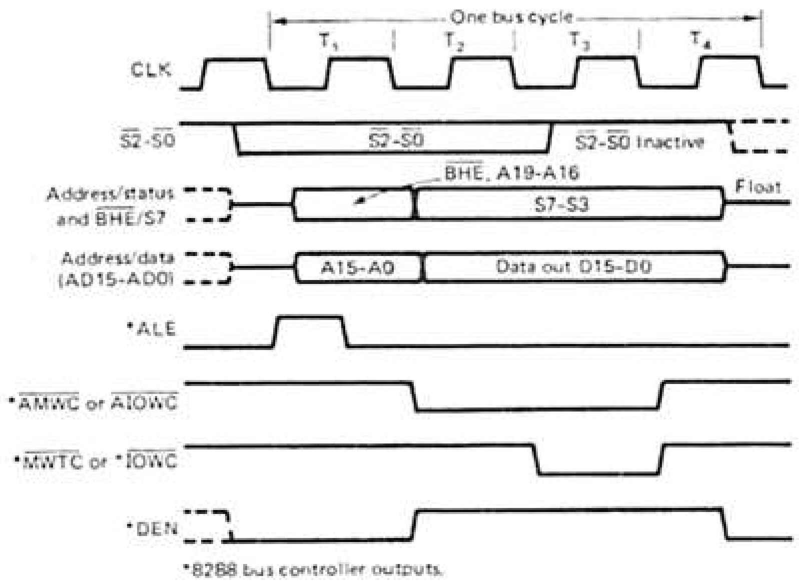


Fig 2.15 (b) : Timing diagrams for a maximum mode system.

T1:

- $\overline{S0}$, $\overline{S1}$, $\overline{S2}$ are set by 8086 in the beginning of clock cycle. It is decoded by the 8288 bus controller.

- ALE is pulsed high.
- $\overline{BHE}$ is made high/low depending on 8/16 bit read at odd/even address boundary.
- M/$\overline{IO}$ is made high to indicate memory operation. It remain high during the entire bus cycle.
- DT/$\overline{R}$ is low and remains low throghout the cycle, to indicate the direction of data transfer as memory to the processor.
- Address is put in the address bus. The falling edge of ALE is used to latch the address from the address bus.
- DEN goes high to enable the 8286 transceivers.

T2:

- $\overline{BHE}$ goes high if it was made low in T1.
- DEN goes high to enable the 8286 transceiver.

T3:

- $\overline{MWTC}$ goes low as memory write control signal.
- Data is put on lines $AD_0 – AD_{15}$
- Status lines $\overline{S0}$, $\overline{S1}$, $\overline{S2}$ become inactive.

T4:

- $\overline{MWTC}$ goes high
- DEN goes low.

The write operation can also be performed by the signals $\overline{AMWTC}$ (for memory write) and $\overline{AIOWC}$ (for I/O write). The signal $\overline{AMWTC}$ is activated one clock cycle earlier than $\overline{MWTC}$

c) Interrupt Acknowledgement

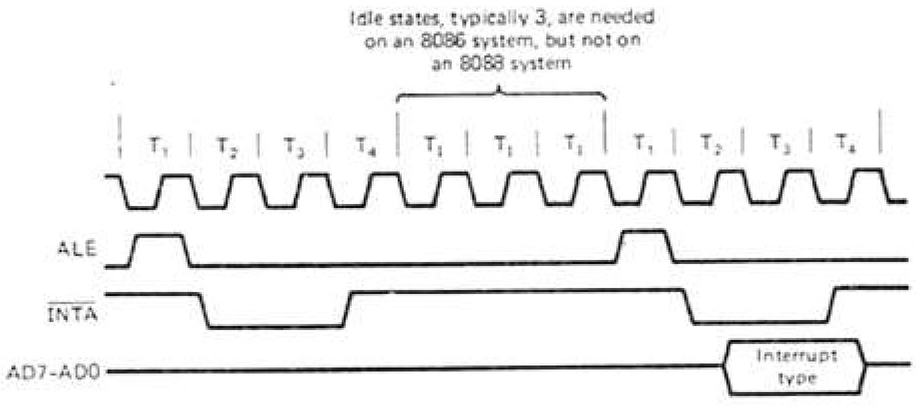

Idle states, typically 3, are needed on an 8086 system, but not on an 8088 system
T1
T2
T3
T4
TI
TI
TI
T1
T2
T3
T4
ALE
INTA
AD7-AD0
Interrupt type

Fig 2.16: Interrupt acknowledgement

The timing diagram for an interrupt acknowledge is shown in Fig 2.16. If an interrupt request has been recognized during the previous bus cycle and an instruction has just been completed, then a negative pulse will be applied to $\overline{\text{INTA}}$ duringthe current bus cycle and the next bus cycle. Each of these pulses will extend fromT$_2$ to T$_4$. Upon receiving the second pulse, the interface accepting the acknowledgment will put the interrupt type on AD7-AD0, which are floated the rest of the time during the two bus cycles. The type will be available from T_2 to T_4·

d)DMA

Bus requests and grants are handled differently, however, and the timing on an$\overline{RQ}/\overline{GT}$pin is shown in Fig. 2.17.

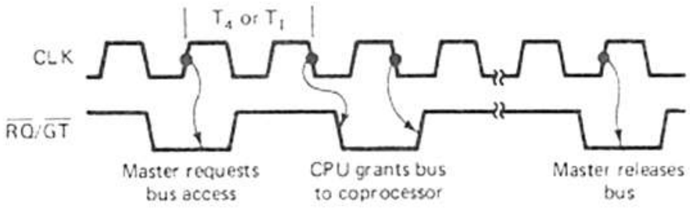


Fig 2.17: Timing for maximum mode bus requests and grants

A request/grant/release is accomplished by a sequence of three pulses. The $\overline{RQ}/\overline{GT}$pins are examined at the rising edge of each clock pulse and if a request is detected (and the necessary conditions discussed previously are met), the processor will apply a grant pulse to the $\overline{RQ}/\overline{GT}$ immediately following the next T_4 or T_1 state. When the requesting master receives this pulse it seizes control of the bus. This master may control the bus for only one bus cycle or for several bus cycles. When it is ready to relinquish the bus it will send the processor the release pulse over the same line that it made its request. $\overline{RQ}/\overline{GT0}$ and $\overline{RQ}/\overline{GT1}$are the same except that $\overline{RQ}/\overline{GT0}$ has higher priority.

2.6 SYSTEM DESIGN USING MICROPROCESSOR

The steps that are involved in the design of microprocessor based system are as follows:

1. Feasibility Study

The feasibility study is the analysis of various aspects such as costing of the product, the technology to be used, time required for the development of the system, maintenance cost, flexibility of the system to adopt modifications and so on. Thus feasibility study gives the overall background for the system development.

There are two basic approaches to design a system.

1. Microprocessor based
2. Logic based

When the system is not complex normally logic based approach is selected. Here ASIC (Application specific integrated circuits) or LSI and MSI chips are used to design the system. For complex systems microprocessor based approach is preferable. This approach is expensive and it involves software development.

Logic approach is preferred when

- System functions are minimal.
- High speed operations are required.
- Less number of inputs and outputs.
- Application specific design is required.

- Memory requirements are less.
- Less flexibility is required to adopt modifications and expansion.

Microprocessor based approach is preferred when:

- System functions are more.
- Large number of inputs and outputs
- Large memory is required.
- More flexibility is required to adopt modifications and expansion.
- Multiple decision paths are required.

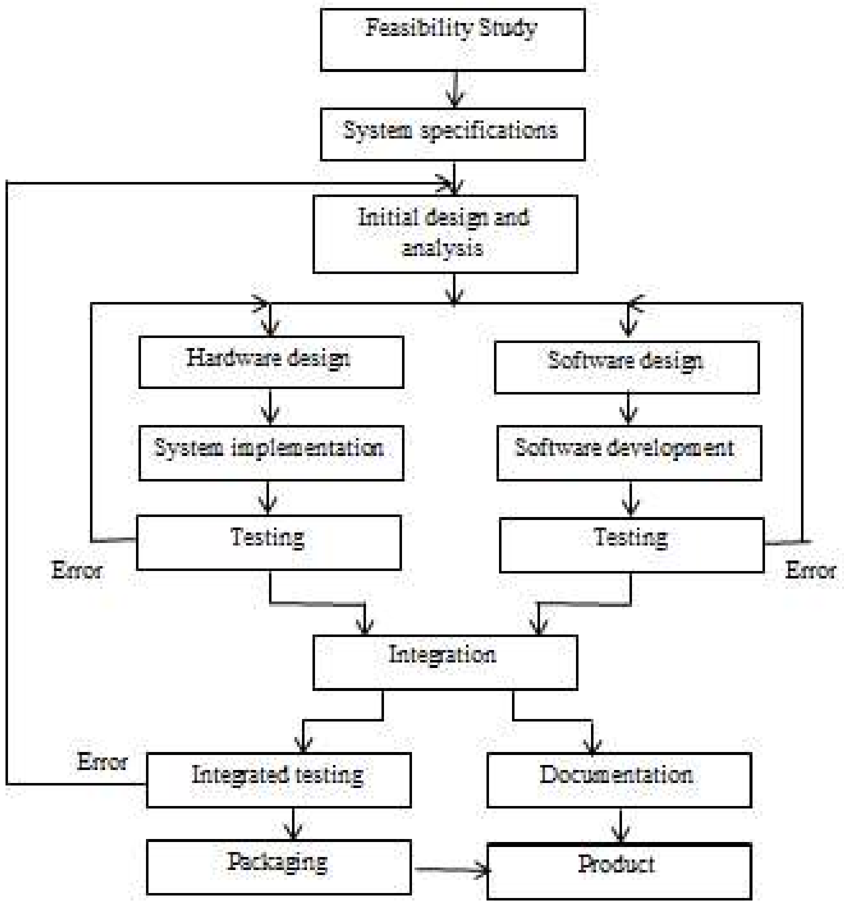


Fig 2.18 : System development cycle

When microprocessor is selected we have to select best suitable microprocessor by analyzing its technical characteristics such as speed, word length, addressing capacity, addressing modes supported, number of general purpose registers and instruction set support. There is one more choice whether to use microprocessor or microcontroller. Microcontrollers are preferred foe embedded systems since they have built-in memory, I/O ports, timer/counters and bit manipulation instructions along with powerful instruction set.

2. System specification:

System specification includes the description of the expected behavior (functionality) of the system. It specifies number of inputs and outputs of the system and functionalities should be provided by the system. System designer usually divides the system into sub-system perform a unique subtask for the system.

3. Initial Design:

Initial design process involves selection of microprocessor and development of block diagram of the hardware, defining basic software routines, estimation of memory requirement and timing considerations, etc. It also involves construction of prototype and its testing with the help of various development tools such as simulators, in-circuit emulators, logic analyzers etc.

4. Hardware Design:

After the prototype has been debugged and tested correctly, the actual hardware is built with microprocessor and peripheral chips.

5. Software Development:

The first step in developing the software is to take the system specifications and write a flowchart to accomplish the desired tasks that will implement the specifications. The source code for different modules (sub-tasks) is then written from the developed flowcharts. The complete source code is then assembled. The assembler will check for syntax error and print error messages to help in the correction of errors. Assembler converts source code into object code. Linker takes the object code generated by the assembler as an input and creates final absolute code that will be executed on the target system.

The emulation phase takes this absolute code and loads it into the development system's RAM. From here, the program may be debugged using breakpoints or single-stepping.

6. Integration:

Once the implementation and testing of hardware and software subsystems is completed, the system performance is checked under real/simulated conditions. For smooth integration of hardware and software and their subsystems, it is necessary to develop hardware and software in co-ordination.

7. System Development:

The following basic steps are carried out in the microprocessor development cycle:

1. The product specifications are finalized in the first stage. This mainly includes the functions which are to be performed by the product and the time required to perform them. The processor should be selected at this page.
2. At this stage, the basic system design is initiated. This includes the complete circuit diagram for hardware and detailed flow-charts for the software.
3. From this stage hardware and software development are carried out simultaneously. Hardware development mainly includes circuit construction, circuit verification, prototype hardware construction and prototype hardware verification. Software development includes code preparation, code compilation and code verification.

4. At the end of the hardware and software development cycles, hardware and software are integrated. This integration includes loading the software into appropriate memory chips.
   Sometimes step-by-step integration methods are adopted. That is, the total system is divided into subsystems. For a completed hardware with the appropriate completed software, integration step is carried out.
5. At last, whole system prototype is tested.

If the errors are detected (hardware and/or software), during the integration or testing procedure, certain hardware and/ or software development steps may have to be corrected and repeated.

Such correction and repetition is easie4r in step-by-step integration method since the modules (subsystems) which are to be tested are smaller and locating possible errors become easier.

Since the software is developed by concentrating the existing hardware, there is a strong interaction and dependence between hardware and software. As the hardware is developed by making the required changes are made in the software, i.e. software is developed. Thus, the hardware and software are developed concurrently. This reduces the overall time required for the development cost.

8. System documentation:

System documentation describes the system's function and how they are implemented. The system must be accurately documented from conception to completion, and although documentation costs may appear to be high, down time will be greatly reduced during trouble shooting, modification or expansion of the system. An accurately documented system should contain the following information.

- A block diagram of the complete system
- A written description of the components used.
- A written description of all functions included.
- Flowcharts and listing of program with proper comments.
- Wiring diagrams and connectors pin descriptions.
- Power supply details
- Memory configuration charts (memory maps)
- Description of processor and connected peripherals
- Description of timing considerations
- Testing procedures used.

2.7 I/O PROGRAMMING

I/O programming describes the ways in which information can be moved between peripheral or mass storage devices and the CPU or memory.

Figure 2.19 shows the basic architecture of a single-bus computer system. In single bus system all peripheral and mass storage devices are connected to the system bus through interfaces. Each interface contains a set of registers, called I/O ports, through which the CPU and memory communicate with the interface's external device. Some of the ports are for buffering data to and from the CPU and memory, some are for holding status information about the device and interface so that it can be examined by the CPU, and some are for retaining the commands sent from the CPU to control the actions taken by the interface and device. All communication with the external world and mass storage is channeled through the VO ports in the interface. Therefore, the CPU must have a means of transferring information to and from these ports as well as memory. Some computers encompass both the memory and port addresses in a single address space and allow all instructions that are capable of accessing memory to access the I/O ports. Others, such as the 8086-based systems, permit the establishment of two address spaces, an I/O space and a memory space.

The latter is done by including control lines in the control bus that indicate whether the address on the address bus is in the I/O space or the memory space. In order to send the correct signals over the control lines, a system that has separate I/O and memory address spaces must have different instructions for communicating with the I/O ports.

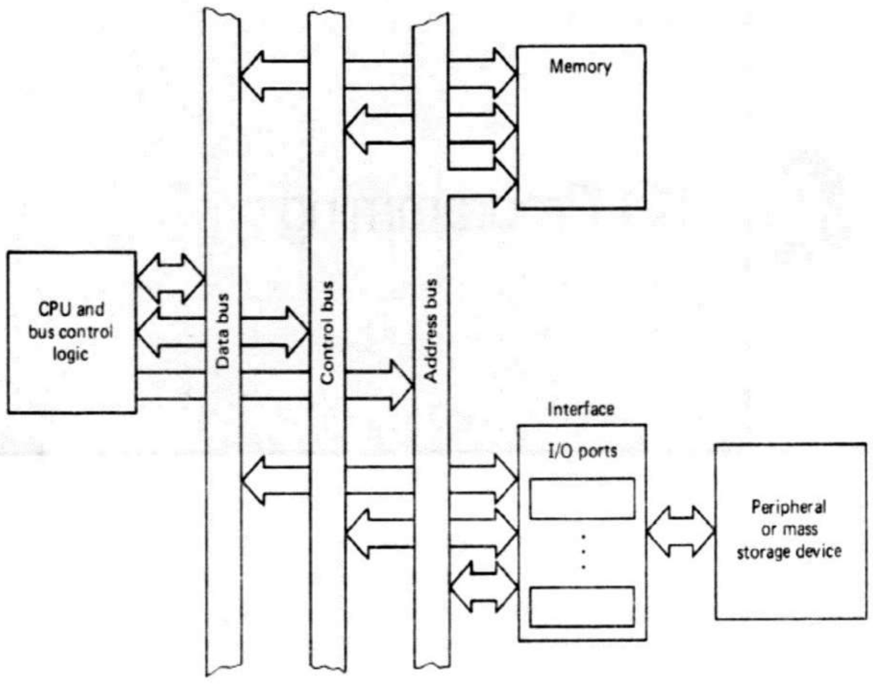


Fig 2.19: Single-bus system

An output from the CPU to a control or buffer port is made by putting the address on the address bus and the proper signals on the control bus, and then putting the data on the data bus. An input from an, input port is accomplished by putting the address and control signals on their respective buses and waiting for the interface to respond by placing the contents of the addressed port on the data bus. It should be emphasized that the addresses are associated with the ports, not the interfaces.

If an interface has four ports, it must be designed to accept four addresses. (However, it is permissible to design an interface so that two or more registers share the same port, provided that the design is such that the interface can properly direct its internal traffic.)

The three principal types of I/O are:

1. Programmed I/O
2. Interrupt I/O
3. Block transfers.

2.7.1 FUNDAMENTAL I/O CONSIDERATIONS

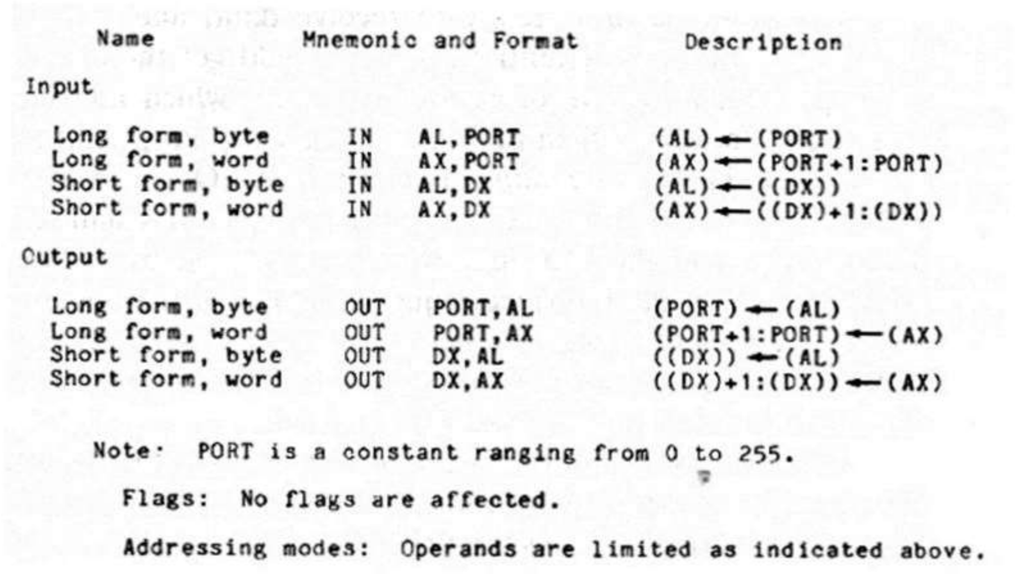
The transfer of data to or from a port can be done in two ways.

1.To execute an instruction that causes a single byte or word to be transferred

2.To execute a sequence of instructions that causes a special system component associated with the interface to transfer a sequence of bytes or words to or from a pre designated block of memory locations.

It is referred to as a block transfer or a direct memory access (DMA) and the special component is called a DMA controller. Byte or word transfers are between the port and the CPU, but block transfers are made directly with memory.

On the 8086, all programmed communication with the I/O ports is done by the IN and OUT instructions defined in fig 2.20..



Name	Mnemonic and Format		Description
Input			
Long form, byte	IN	AL,PORT	(AL)←(PORT)
Long form, word	IN	AX,PORT	(AX)←(PORT+1:PORT)
Short form, byte	IN	AL,DX	(AL)←((DX))
Short form, word	IN	AX,DX	(AX)←((DX)+1:(DX))
Output			
Long form, byte	OUT	PORT,AL	(PORT)←(AL)
Long form, word	OUT	PORT,AX	(PORT+1:PORT)←(AX)
Short form, byte	OUT	DX,AL	((DX))←(AL)
Short form, word	OUT	DX,AX	((DX)+1:(DX))←(AX)

Note: PORT is a constant ranging from 0 to 255.

Flags: No flags are affected.

Addressing modes: Operands are limited as indicated above.

Fig. 2.20 IN and OUT instructions

Both instructions may transfer either a byte or a word and both have a long form and a short form. The first operand in the IN instruction, the destination operand, must be either AL (for a byte transfer) or AX (for a word transfer). As with memory, a word transfer is made from two consecutive addresses with the low-order byte in AX being moved from the port with the lower address.

If the second operand in the IN instruction evaluates to a constant, then the constant is used as the address of the port whose contents are being input. In this case the instruction is 2 bytes long with the port address occupying the second byte. If the second operand is DX, then there is only one byte in the instruction and the contents of DX are used as the port address.

Figure 2.21 shows a possible sequence of events when I/O is handled by the operating system. Ina user's program, whenever I/O is needed the user calls the monitor through a type of software interrupt, or trap, which passes the control to the monitor along with a function code, indicating the I/O operation to be performed, and the necessary parameters required to carry out that operation.

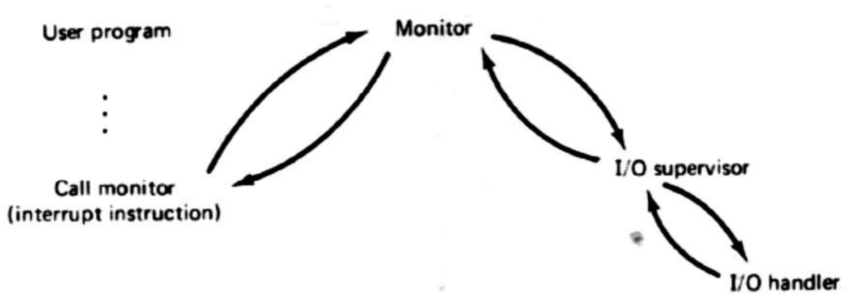


Fig 2.21:I/O handling by the operating system

By examining the function code, the monitor, which handles other services in addition to I/O, knows the request is for an I/O operation and, consequently, dispatches the task to the I/O supervisor. The I/O supervisor further determines which device is involved in the I/O operation so that the corresponding I/O driver, which is typically an interrupt-driven or DMA routine, can be initiated. The I/O handler uses a buffer in the system area for temporarily storing input and output data. The I/O supervisor moves the data to be output (or the input data) from the user's area to the buffer area (or vice versa). Upon completion of the I/O operation, control is transferred back to the user's program through the I/O supervisor and the monitor.

2.7.2 PROGRAMMED I/O

In programmed I/O, the data transfer is controlled by the user program being executed. Depending on the type of device, data transfer may be synchronous or asynchronous. Synchronous data transfer is used when the I/O device matches in speed withg the microprocessor. The microprocessor issues the read/write instruction addressing the device whenever data transfer is required. The actual data transfer takes place in one clock cycle.

When the I/O device speed and the microprocessor speed do not match, i.e. when the I/O device is slower than the microprocessor speed do not match. In this mode the microprocessor checks the status of the device. If the device is not ready, the microprocessor checks the status of the device till it becomes ready. The data transfer instruction is isued by the microprocessor.

Programmed I/O consists of continually examining the status of an interface and performing an I/O operation with the interface when its status indicates that it has data to be input or its data-out buffer register is ready to receive data from the CPU. A typical programmed input operation is flowcharted in fig 2.22.

The flowchart assumes that a sequence of bytes or words is to be input and as each byte or word is brought into the CPU it is modified and transferred to a memory buffer. After all of the data have been brought in and put in the buffer, the buffer is processed.
Suppose that a line of characters is to be input from a terminal to an 82-byte array beginning at BUFFER until a carriage return is encountered or more than 80 characters are input. If a carriage return is not found in the first 81 characters then the message "BUFFER OVERFLOW" is to be output to the terminal; otherwise, a line feed is to be automatically appended to the carriage return. Because the ASCII code is a 7-bit code, the eighth bit, bit 7, is often used as a parity bit during the transmission from the terminal. Let us assume that bit 7 is set according to even parity and if an odd parity byte is detected, a branch is to be made to ERROR.

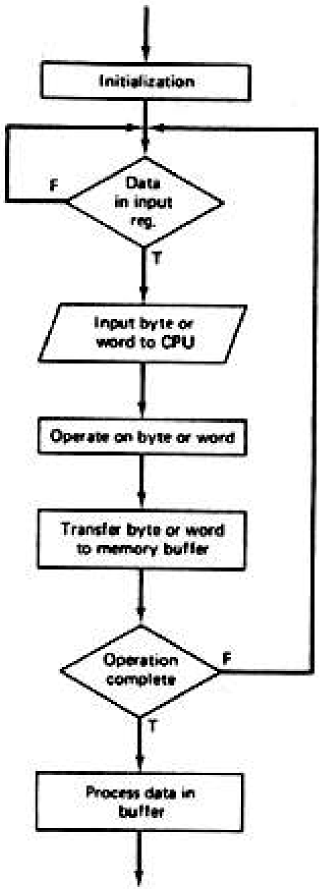


Fig 2.22: Programmed input

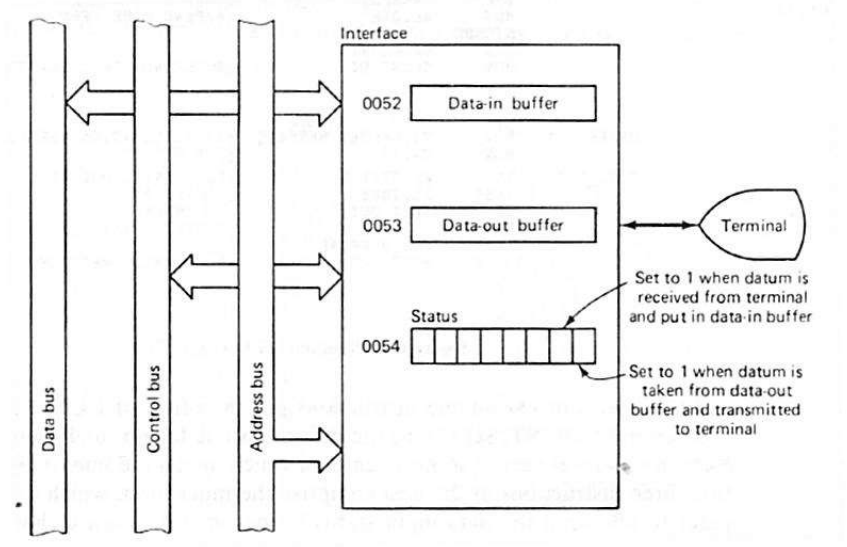

Interface
0052
Data-in buffer
0053
Data-out buffer
Terminal
Set to 1 when datum is received from terminal and put in data-in buffer
Status
0054
Set to 1 when datum is taken from data-out buffer and transmitted to terminal
Data bus
Control bus
Address bus

Fig 2.23: Interface for the programmed I/O example

.In priority polling sequence, the addresses of their status registers have been equated to STAT1, STAT2, and STAT3 and the procedures PROC1, PROC2, and PROC3 are called upon to perform the input. The program sequence gives the devices apriority. The device corresponding to STATl has the highest priority and the other two devices must wait until this device is idle. The device corresponding to STAT3 cannot be serviced until neither of the other two devices is ready.

In round robin polling the devices could be serviced in turn. Such an arrangement essentially gives all three devices the same priority.

2.7.3 INTERRUPT I/O

An interrupt is an event that causes the CPU to initiate a fixed sequence, known as an interrupt sequence. Before an 8086 interrupt sequence can begin, the currently executing instruction must be completed unless the current instruction is a HLT or WAIT instruction. (The WAIT instruction is primarily used to wait for the completion of a coprocessor instruction.)

For the 8086, once the interrupt request has been recognized, the interrupt sequence consists of:

1. Establishing a type N.
2. Pushing the current contents of the PSW, CS, and IP onto the stack (in that order).
3. Clearing the IF and TF flags.
4. Putting the contents of memory location 4*N into IP and the contents of 4*N +2 into CS.

Thus, an interrupt causes the normal program sequence to be suspended and a branch to be made to the location indicated by the double word beginning at four times the type (i.e., the interrupt pointer). Control can be returned to the point at which the interrupt occurred by placing an IRET instruction at the end of the interrupt routine.

There are two classes of interrupts.

1.Internal and

2.External interrupts

External interrupt is caused by a signal being sent to the CPU through one of its pins, for 8086 it is either the NMI pin or the INTR pin.

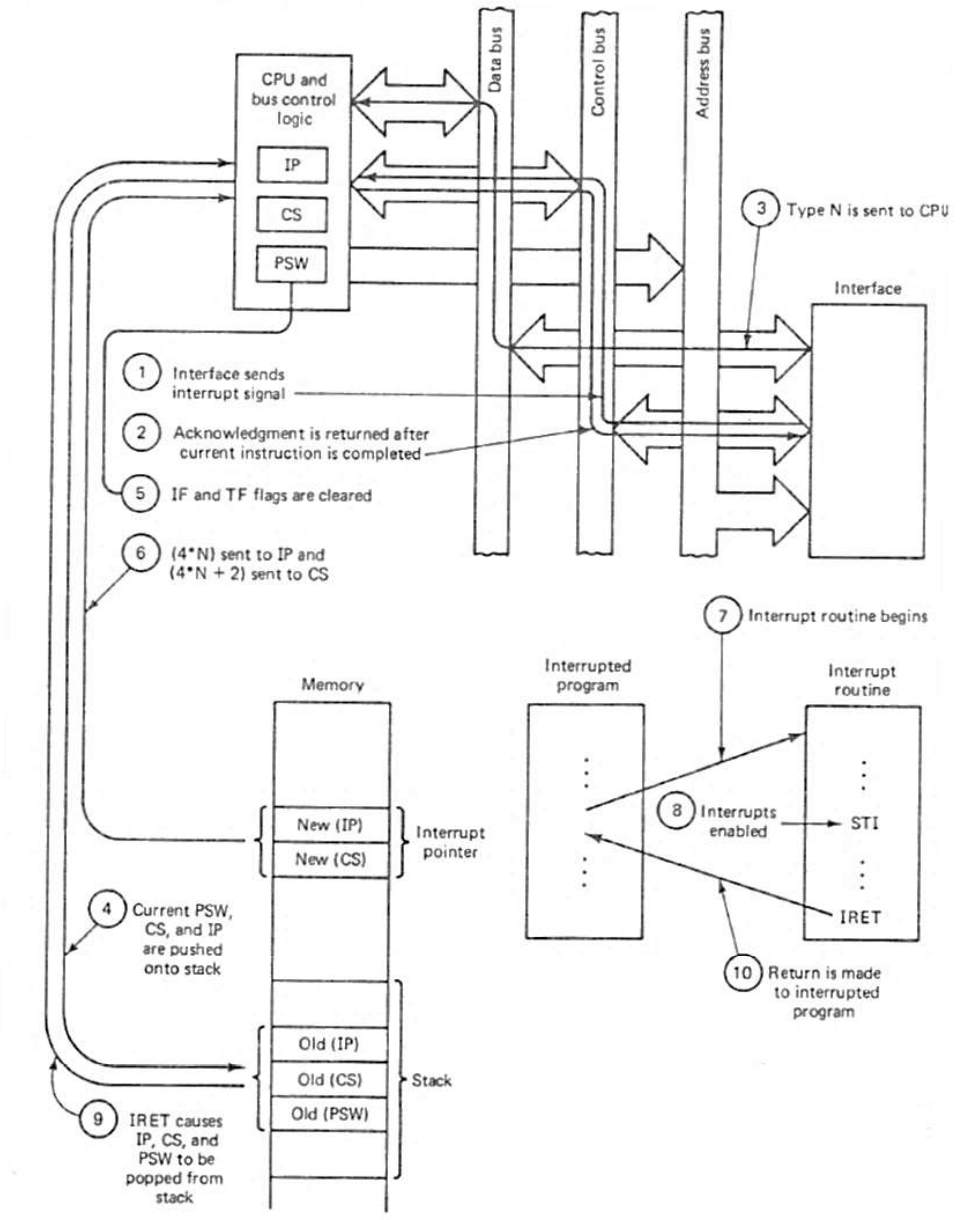

CPU and bus control logic
IP
CS
PSW
Data bus
Control bus
Address bus
3 Type N is sent to CPU
Interface
1 Interface sends interrupt signal
2 Acknowledgment is returned after current instruction is completed
5 IF and TF flags are cleared
6 (4*N) sent to IP and (4*N + 2) sent to CS
7 Interrupt routine begins
Memory
Interrupted program
Interrupt routine
New (IP)
New (CS)
Interrupt pointer
8 Interrupts enabled
STI
IRET
4 Current PSW, CS, and IP are pushed onto stack
10 Return is made to interrupted program
Old (IP)
Old (CS)
Old (PSW)
Stack
9 IRET causes IP, CS, and PSW to be popped from stack

Fig 2.24: Sequence of events during a maskable interrupt and subsequent return

An interrupt initiated by a signal on the NMI pin is called a nonmaskable interrupt and will cause a type 2 interrupt regardless of the setting of the IF flag. Nonmaskable interrupt signals are normally caused by circuits for detecting catastrophic events such as imminent power failure, or by external events or clock pulses that must be processed immediately. An interrupt on the INTR pin is masked by the IF flag so that if this flag is 0, the interrupt is not recognized until IF returns to 1.

When IF = 1 and a maskable external interrupt occurs, the CPU will return an acknowledgment signal to the device interface through its $\overline{\text{INTA}}$ pin and initiate the interrupt sequence. The acknowledgment signal will cause the interface that sent the interrupt signal to send to the CPU (over the data bus) the byte which specifies the type, and hence the address of the interrupt pointer. The pointer, in turn, supplies the beginning address of the interrupt routine.

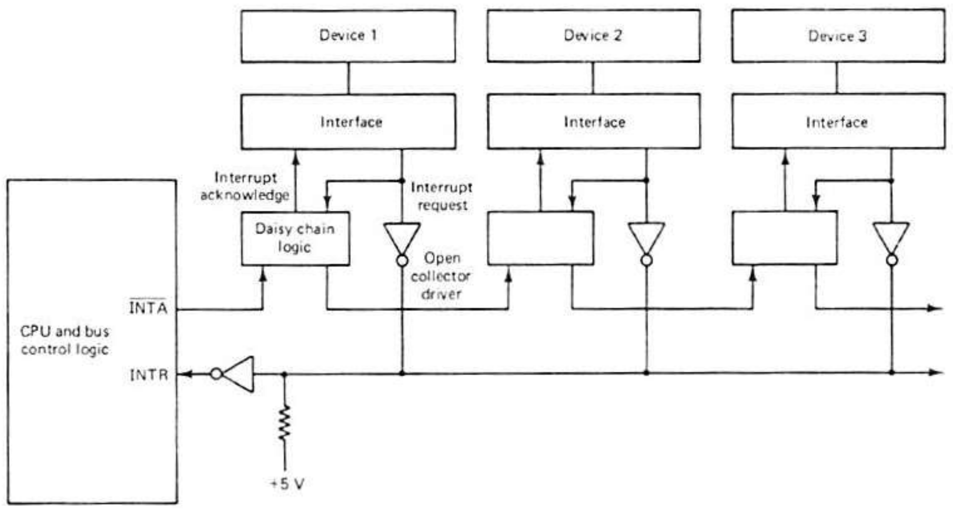
There are several ways of combining priority with interrupt I/O, some involving only' software, some only hardware, and some a combination of the two. Let us consider the following means of giving priority to an interrupt system:

1. Polling.
2. Daisy chaining.
3. Interrupt priority management hardware.

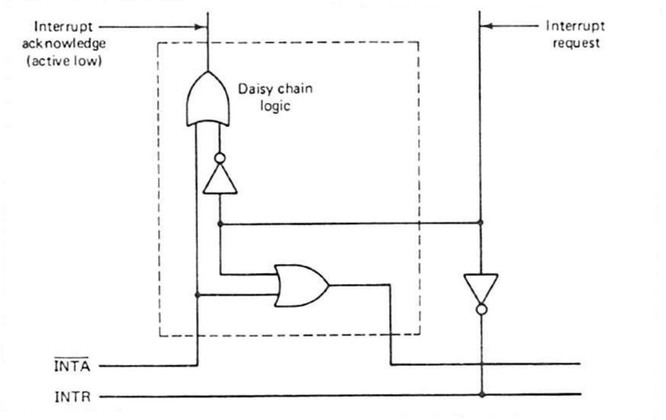
Polling:

Although there are numerous external interrupt types available for use by the interfaces in an 8086 system and it is seldom necessary to assign two interfaces the same type, a set of interfaces may be given the same type so that polling can be used to assign them priorities. By-putting a program sequence at the beginning of the interrupt routine, the priority of the interfaces could be established by the order in which they are polled by the sequence.

Daisy Chaining:



(a) Daisy chain



(b) Logic

Fig 2.25: Daisy chain arrangement.

Daisy chaining is a simple hardware means of attaining a priority scheme. It consists of associating a logic circuit with each interface and passing the interrupt acknowledge signal through these circuits as shown in Fig 2.25 (a) and the details of a daisy chain logic circuit are shown in Fig. 2.25 (b)

Interrupt Priority Management Hardware

The management circuit would contain the logic needed to assign priorities to the incoming requests. For example, the highest priority could be given to IR0, the next highest priority to IRl, and so on. When an interrupt request is recognized by the priority logic as having the highest priority, then the three LSBs of the type register are set to the number of the request line, a bit is set in the in-service register, and an interrupt is sent to the CPU.

If IF= 1, then the CPU returns an acknowledge signal and the management circuit sends the CPU the type. All requests having lower priority are blocked until the bit in the in-service register is cleared, an action which is normally done by the interrupt routine.

Therefore, when IF is re enabled by an STI instruction, higher-priority requests may interrupt the currently executing interrupt routine, but lower-priority requests will be blocked by the priority logic until the bit that was set in the in-service register is cleared.

This gives the interrupt routine control over when lower-priority requests will be recognized. In order for the program to be able to clear bits in the in-service register, this register must be programmable, i.e., it must have an I/O port address so that it can be accessed using the IN and OUT instructions.

In addition to the built -in priority, a 1-byte mask register is included to allow masking of the individual requests. Bit n in this register would be for masking IRn. It is assumed that this register is programmable. In the example, the least significant 3 bits of the type register are determined by the request selected by the priority logic. If this register is programmable, the 5 most significant bits could be initialized when the system is turned on.

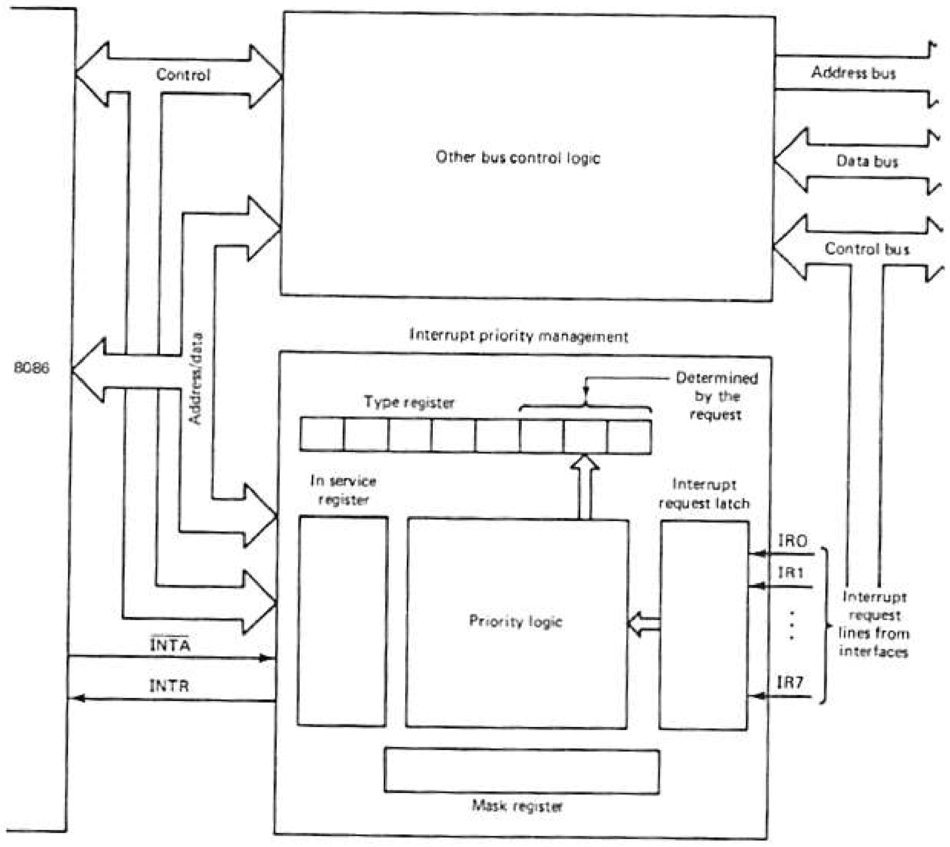


Fig 2.26: Representative interrupt priority management design

2.7.4 BLOCK TRANSFERS AND DMA

If the data transfer rate to or from an I/O device is relatively low, then the communication can be performed using either programmed or interrupt I/O. But executing instructions and performing interrupt sequences take more time than is sometimes available.

For data rates of high magnitude, block transfers, which use DMA controllers to communicate directly with memory, are required. The activity involved in transferring a byte or word over the system bus is called a bus cycle.

During any given bus cycle, one of the system components connected to the system bus is given control of the bus. This component is said to be the master during that cycle and the component it is communicating with is said to be the slave.

The CPU with its bus control logic is normally the master, but other specially designed components can gain control of the bus by sending a bus request to the CPU. After the current bus cycle is completed the CPU will return a bus grant signal and the component sending the request will become the master. Taking control of the bus for a bus cycle is called cycle stealing.

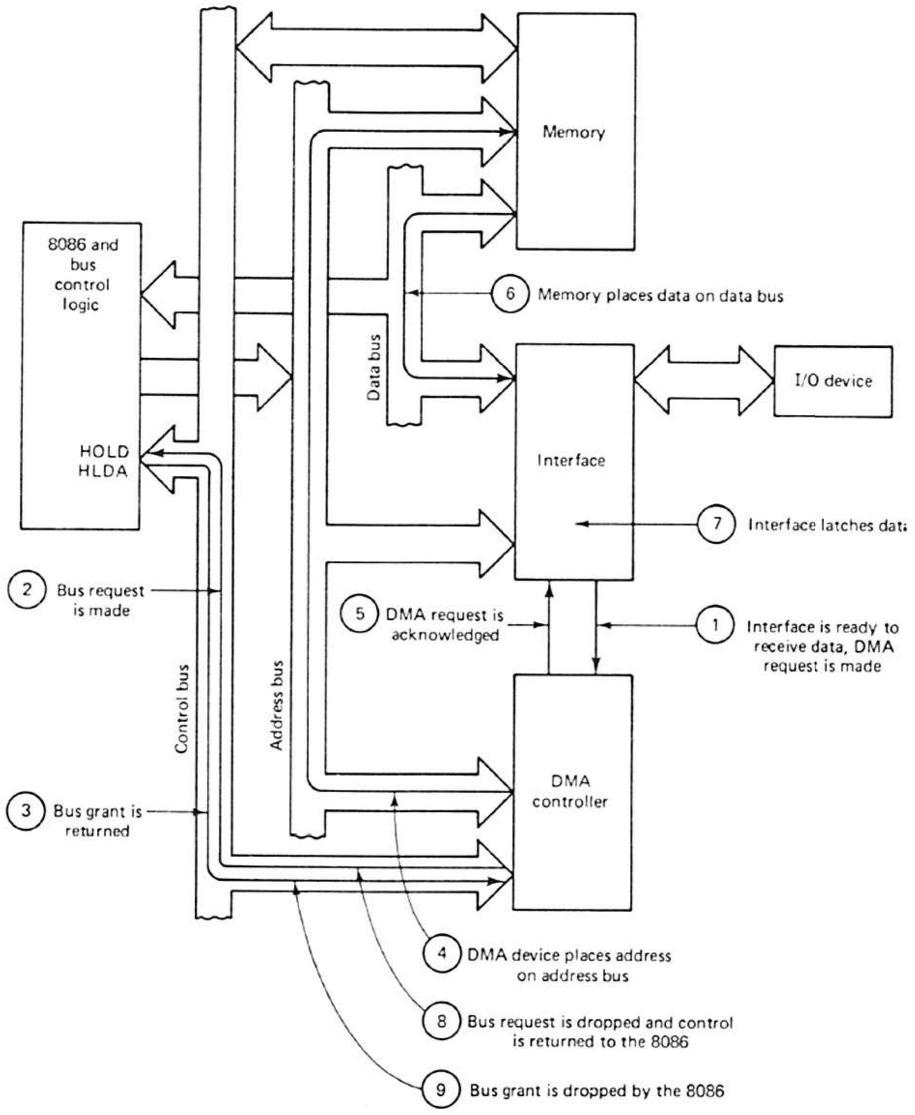

Memory
8086 and bus control logic
HOLD
HLDA
6 Memory places data on data bus
Data bus
I/O device
Interface
7 Interface latches dat
2 Bus request is made
5 DMA request is acknowledged
1 Interface is ready to receive data, DMA request is made
Control bus
Address bus
DMA controller
3 Bus grant is returned
4 DMA device places address on address bus
8 Bus request is dropped and control is returned to the 8086
9 Bus grant is dropped by the 8086

Fig 2.27 Single datum output transfer during a block transfer

Just like the bus control logic, a master must be capable of placing addresses on the address bus and directing the bus activity during a bus cycle. The components capable of becoming masters are processors (and their bus control logic) and DMA controllers. Sometimes a DMA controllers associated with a single interface, but they are often designed to accommodate more than one interface.

The 8086 receives bus requests through its HOLD pin and issues grants from its hold acknowledge (HLDA) pin. A request is made when a potential master sends a 1 to the HOLD pin. Normally, after the current bus cycle is complete the 8086 will respond by putting a 1 on the HLDA pin.

When the requesting device receives this grant signal it becomes the master. It will remain master until it drops the signal to the HOLD pin, at which time the 8086 will drop the grant on the HLDA pin.

One exception to the normal sequence is that if a word which begins at an odd address is being accessed, then two bus cycles are required to complete the transfer and a grant will not be issued until after the second bus cycle.

During a block input byte transfer, the following sequence occurs as the datum is sent from the interface to the memory:.

1. The interface sends the controller a request for DMA service.
2. The controller gains control of the bus.
3. The contents of the address register are put on the address bus.
4. The controller sends the interface a DMA acknowledgment which tells theinterface to put data on the data bus. (For an output it signals the interface to latch the next data placed on the bus.)
5. The data byte is transferred to the memory location indicated by the address bus.
6. The controller relinquishes the bus.
7. The address register is incremented by 1.
8. The byte count register is decremented by 1.
9. If the byte count register is nonzero, return to step 1; otherwise, stop.

Status and control registers :

- Indicates the type of transfer to be conducted.
- It has a "do" bit for initiating the I/O activity (a bit that can be sensed by the I/O device)
- Bit for indicating whether or not the device is currently busy.
- An enable bit which controls whether or not it will recognize DMA requests from the interface. A data direction bit to supervise an input or output transfer.

Byte count register: notes the number of bytes yet to be transferred to initiate a block transfer.
Address register: holds the address of the next memory location.

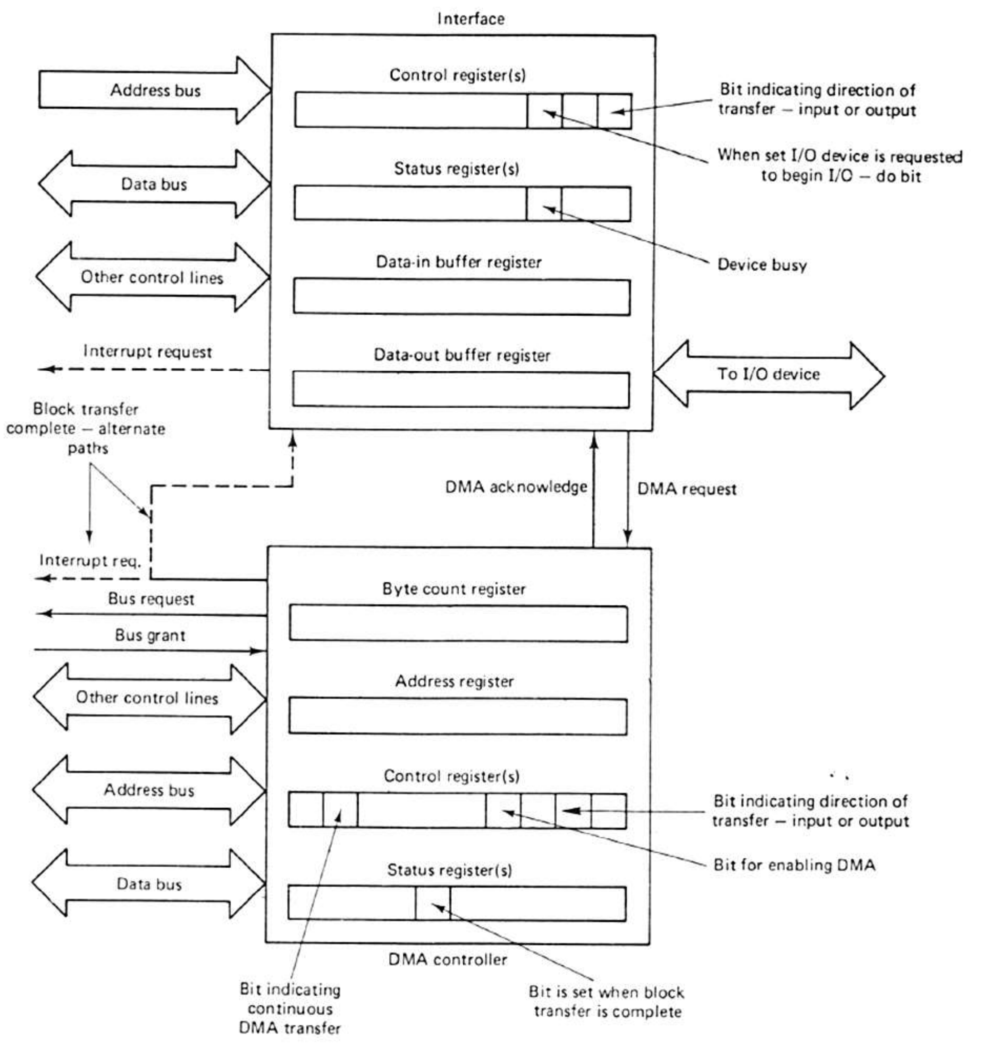


Fig 2.28: Minimal DMA controller/interface configuration

2.8 INTRODUCTION TO MULTIPROGRAMMING

A process is a programming unit which performs an independent task. When processes are executed in a serial fashion, the system is called a uni programming system. In such systems, normally only one process is stored in the memory at a time and the next process is not loaded for execution until the current one is terminated. This is not adequate for a large number of applications, those in which the ability to respond to events in real time is required or which must be able to execute numerous programs rapidly and efficiently.

For example, suppose that a microprocessor-based system is to collect and process data from two independent data devices. If operated in a uni programming environment, either process may miss some of its input data while the other process is executing. This can happen even when the incoming data rate for both processes is low. The problem is not caused by the lack of processing power or by the speed of the interfaces, but is a result of the sequential nature of the overall processing scheme.

For a multiprogramming environment, the code for two or more processes is in memory at the same time and is executed in a time-multiplexed fashion. If, in the above example, the processes were executed in a multi programmed system, they could take turns using the CPU and one process could be performing its computations while the other is doing I/O, and vice versa.

A single CPU can still execute only one instruction at a time. However, through resource sharing a multi programmed single CPU system appears as if it is executing several processes simultaneously.

Let us consider two processes, say 1 and 2. If uni processing is used, the activity is typically as shown in Fig. 2.29.

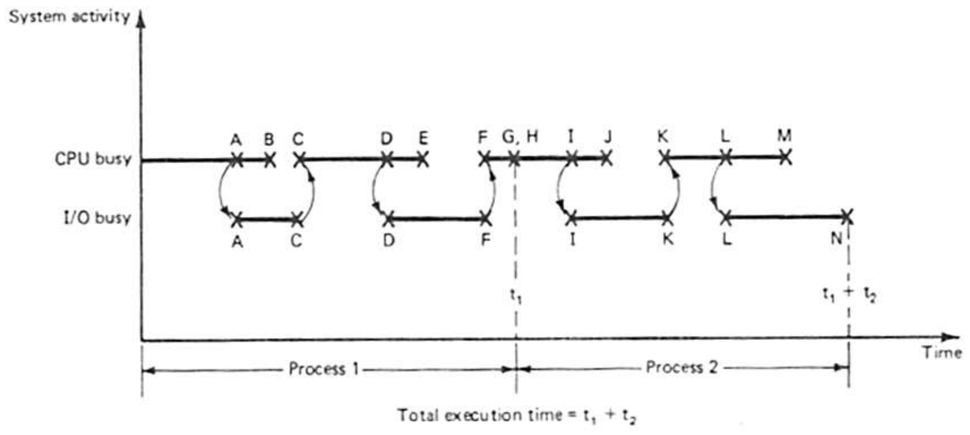


Fig2.29 :Uniprogramming activity

Here process 1 begins and continues until I/O is needed (point A), then the I/O is initiated and the processing continues in parallel with the I/O until the processing needs the input data. At this time it must wait until the I/O is completed (point B). When the I/O is finished (point C)the processing is resumed. A similar description applies to points D, E, and F. At the end of process 1, process 2 can begin and its operation is basically the same as that of process 1.

On the other hand, if multiprogramming is applied, then the activity is as illustrated in Fig. 2.30.It is seen that instead of the CPU being idle while it is waiting for process 1 I/O, in a multiprogramming system process 2 can be begun and can utilize the CPU until it needs to wait for I/O. At that time, if process 1has finished its I/O operation, it can resume its use of the CPU. Thus the overall processing time can be significantly reduced.

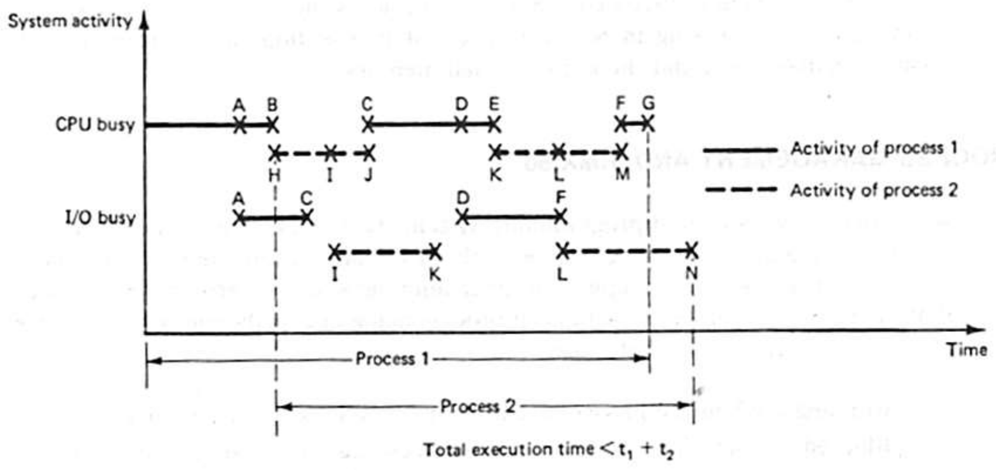


Fig 2.30: Multiprogramming activity utilizing a single CPU and a single DMA channel

A multiprogramming system may be capable of accommodating several users at the same time. When this is the case the users take turns getting control of the CPU, with each user being allocated a time slice, and the system is said to be time-shared.

Multiprogramming can be used in systems that include more than one processor. Such systems, called multiprocessing systems, have the ability to execute more than one instruction at a time.

2.8.1Process Management and iRMX 86

In a single-processor multiprogramming system, two or more processes reside in the memory and share the CPU, but the CPU can execute only one of these processes at a time. In a simple multiprogramming system there are three states that the processes can be in, with each process being in exactly one of these states at any given time. These states are:

1. Running-When the process is currently being executed by the CPU.

2. Blocked-When the execution of the process cannot be continued because it is waiting for an event to occur, e.g., it is waiting for the completion of an I/O operation.

3. Ready-When the execution of the process can be resumed any time. For example, the I/O process has been waiting has finished and the processing is able to continue.

As time passes each process will rotate among these states as shown in Fig 2.31.At the time a process is started it is placed in a waiting queue of ready processes. When the process that is currently executing switches from the running state to the blocked state because it must wait for I/O, the CPU becomes available and a process in the ready queue is selected by the process scheduler and is changed from ready to running. While a process is in the blocked state its execution is suspended. After its I/O is completed, a blocked process becomes ready and it is placed in the ready queue.

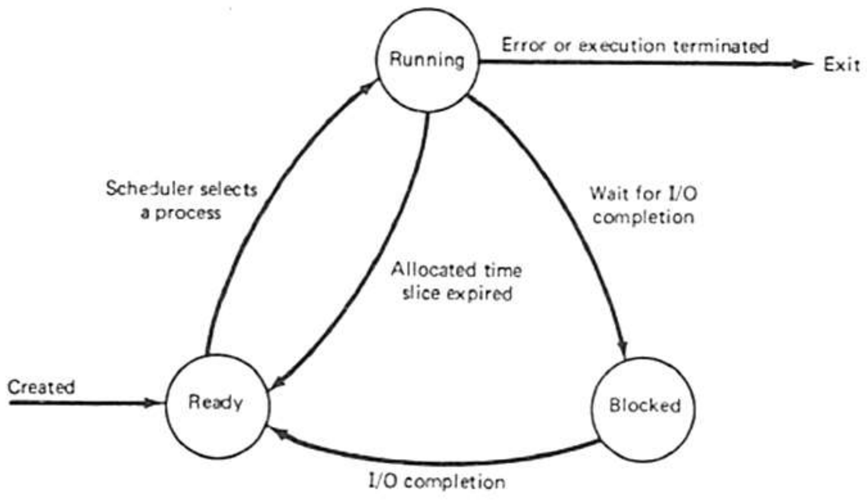

Running
Error or execution terminated
Exit
Scheduler selects a process
Wait for I/O completion
Allocated time slice expired
Created
Ready
Blocked
I/O completion

Fig 2.31: Process states and state changes

In the simplest multiprogramming arrangement a process is allowed to execute until it is terminated or must wait for I/O. However, to prevent a process from dominating the CPU usage, a system can be designed so that a running process must return to the ready state after a specified period of time, thus giving the other processes a chance to execute.

A structure of a ready list that is based on FIFO scheduling and a linked list is shown in Fig. 2.32.

The first-element pointer indicates which process is at the top of the queue and the last-element pointer indicates which process is at the bottom. The last-element pointer is needed so that new processes can be added to the queue without having to search through the entire list. In the example, it is assumed that each process has an ID that is the same as a position in the array containing the linked list. It is also assumed that the queue presently consists of the processes 3, 6, 2, 8, and 5, in that order. Therefore, 3 is in the first element pointer, 5 is in the last-element pointer, and the processes 2, 3, 5, 6, and8 are currently in the ready state. The remaining entries in the process table are occupied by running and blocked processes or are not currently being used. If the number 0 is used to indicate the end of the list, it cannot be used as a process ID. If the process IDs are to be represented by 1 byte and there can be no process 0,then the maximum number of processes that can be in the system at one time is 255.

Each element in the list contains a forward queue pointer (fqp), which indicates the next element in the list, and a process control block pointer for locating a block of memory that is used to store important information related to the process.

The process control block serves as the process's save area for storing the machine status as well as for storing such things as the process ID, the process state, a code indicating how and why the process entered its current state, and so on.

When the CPU switches from one running process to another, it is necessary for the monitor to:

1. Save the machine status of the running process in the process control block.

2. Update the remainder of the process control block.
3. Get the ID of the next process to be run from the first-element pointer.
4. Delete the process that has just entered the running state by setting the first element pointer equal to the fqp of the deleted element.
5. Change the state of the process that was just removed from the ready list to running and restore the machine status of this process.

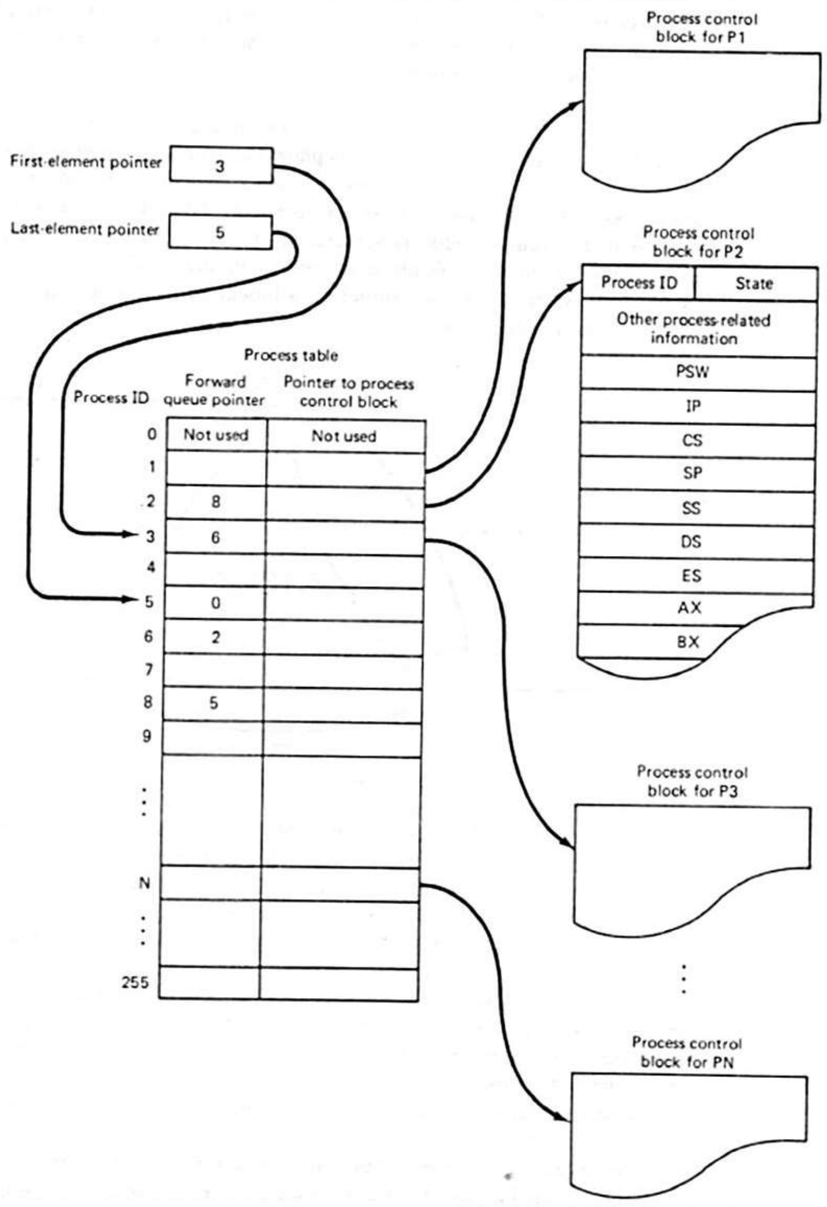
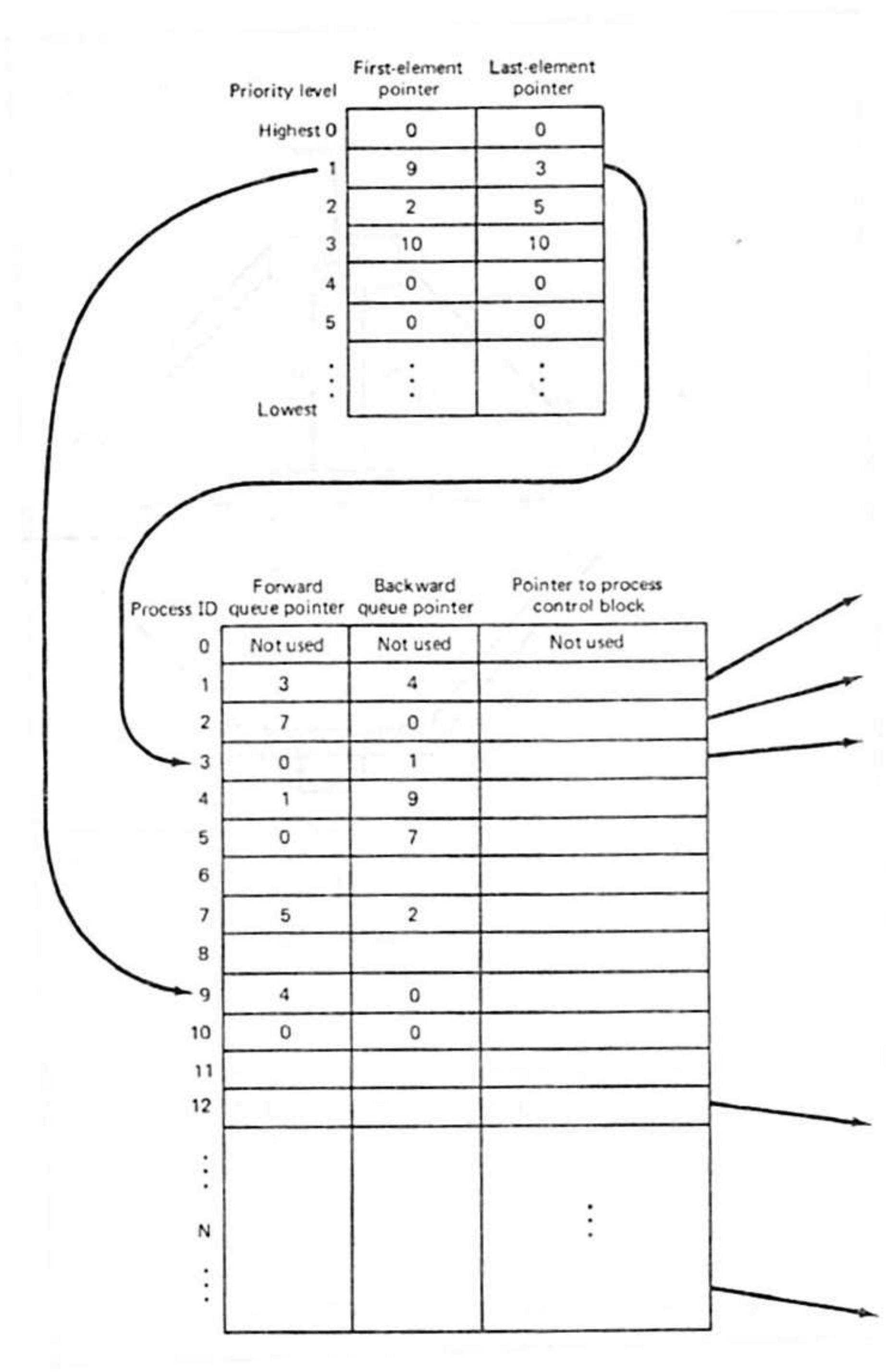

Process control block for P1
First-element pointer
3
Last-element pointer
5
Process control block for P2
Process ID
State
Other process-related information
PSW
IP
CS
SP
SS
DS
ES
AX
BX
Process table
Process ID
Forward queue pointer
Pointer to process control block
0
Not used
Not used
1
2
8
3
6
4
5
0
6
2
7
8
5
9
N
255
Process control block for P3
Process control block for PN

Fig 2.32: Structure of a ready queue



Priority level	First-element pointer	Last-element pointer
Highest 0	0	0
1	9	3
2	2	5
3	10	10
4	0	0
5	0	0
⋮ Lowest	⋮	⋮

Process ID	Forward queue pointer	Backward queue pointer	Pointer to process control block
0	Not used	Not used	Not used
1	3	4	
2	7	0	
3	0	1	
4	1	9	
5	0	7	
6			
7	5	2	
8			
9	4	0	
10	0	0	
11			
12			
⋮ N ⋮			⋮

Fig 2.33: Structure of a prioritized ready queue

As a result of step 5 the newly selected process will continue from the point at which it was suspended. A process that is currently in the system can be added to the ready list by:

1. Storing its process ID into the fqp of the current last element (the element pointed to by the last-element pointer) and into the last-element pointer.
2. Clearing the fqp in its process table entry.
3. Updating its process control block.

For many applications, especially those involving real-time data processing, it is necessary to assign priorities to the various processes by giving the process requiring the fastest service the highest priority. Processes may still be given the same priority, with the FIFO policy being applied within each priority level. If priorities are used, the system must include a priority table.

As shown in Fig. 2.33, each element in this table would represent a priority level and have two fields, with the first field containing the first-element pointer for the priority level and the second field containing the last-element pointer for the level. If the priority of a process is changed, it must be deleted from its current priority chain and appended to the bottom of the chain having its new priority. This action is aided by the inclusion of a backward queue pointer (bqp) so that priority changes can be made quickly. The priority scheduling system illustrated in Fig. 2.38assumes the following, ready processes:

Priority 0: None
Priority 1: 9-4-1-3
Priority 2: 2-7-5
Priority 3: 10
Priority 4: None
Priority 5: None

To select the next running process, the scheduler would search the priority table starting with the highest priority until a nonzero first-element pointer is found, and would then delete that process from the ready queue.

2.8.2 Semaphore Operations

In multiprogramming systems, processes are allowed to share common software and data resources as well as hardware resources. In many situations, a common resource may be accessed and updated by only one process at a time and other processes must wait until the one currently using the shared resource is finished with it. A resource of this type, which is commonly referred to as a serially reusable resource, must be protected from being simultaneously accessed and modified by two or more processes. A serially reusable, resource may be a hardware resource(such as a printer, card reader, or tape drive), .a file of data, or a shared memory area.

For example, let us consider a personnel file that is shared by processes 1and 2. Suppose that process 1 performs insertions, deletions, and changes, and process 2 puts the file in alphabetical order according to last names. If accessed sequentially, this file would both be updated by process 1 and then sorted by process 2, or vice versa. However, if both processes were allowed to access the file at the same time, the results would be unpredictable and almost certainly incorrect. The solution to this problem is to allow only one process at a time to enter its critical section of code, i.e., that section of code that accesses the serially reusable resource.

Preventing two or more processes from simultaneously entering their critical sections for accessing a shared resource is called mutual exclusion.

A flag used to reserve a shared resource is called a semaphore and the operations of requesting and releasing the resource are commonly known as the P and V semaphore operators. If FLAG = 1 indicates that the resource is free and FLAG =0indicates it is busy.

2.8.3Common Procedure Sharing

In a multiprogramming system, it is desirable to allow two or more users to share procedures. These procedures are known as common procedures, and typically are library routines or system programs such as I/O drivers, code conversion routines, and so on. For example, several users may simultaneously want to use a text editor to edit their programs. Without code sharing, several copies of the text editor would need to be loaded into memory. With code sharing, only one copy needs to be kept in memory, thus providing a significant savings in storage.

A common procedure may be shared in serial fashion just like any other shared resource. The application of serially shared procedures is rather limited, and a broader form of sharing is necessary, a form in which a procedure is shared in a time multiplexed fashion or is concurrently reusable. This means that a procedure must be such that it can be called by another process before the execution of the procedure due to a prior call is completed. Such a procedure is said to be reentrant.

A reentrant procedure must consist of code, called pure code, which does modify itself. To meet this requirement a reentrant procedure can store the data to be modified by the routine only in memory locations that are associated with the calling process; it cannot, even temporarily, store such data in locations that are local to itself.

2.8.4 Memory Management

The simplest method for storing more than one job in the memory is the partition allocation scheme.

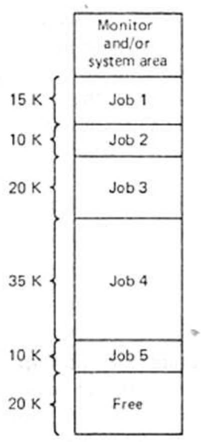


Fig 2.34: Partition memory allocation

As depicted in Fig. 2.34, if this scheme is used, each job is allocated a contiguous area of memory.

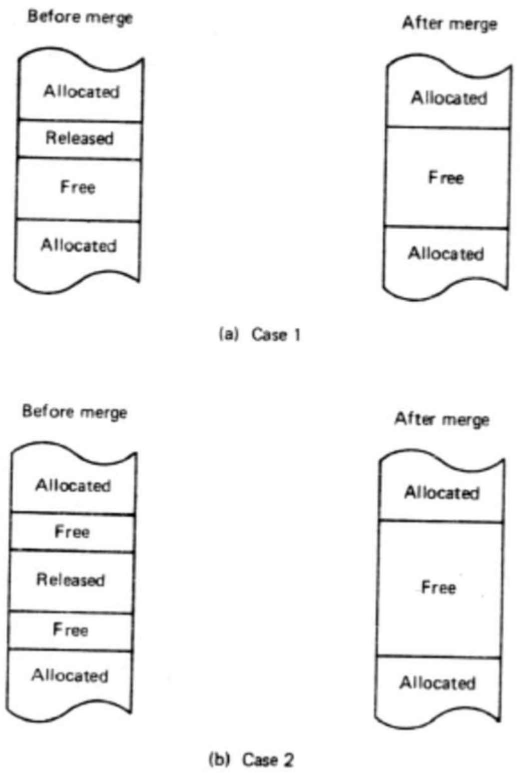


Fig 2.35: Merging adjacent free partitions

When a job needs to be executed, the loader requests the required amount of memory from the memory management routine, which is the part of the operating system that is responsible for storage allocation. If the memory management routine finds a contiguous free area whose size is larger than the job size, the beginning address of that free area is returned to the loader. After adjusting the necessary relative addresses so that they become physical addresses, the loader can load the job into the allocated area. Of course, the initial state of the job is "ready" and the job may not start executing immediately .If there is no single free area large enough, the job must wait before it can be loaded from mass storage.

Each entry would contain the status, size, and beginning address of the partition, with the status being:

Allocated-If the partition is currently allocated to a job.
Free-If the partition is available.
Unused-If the entry is not associated with a partition.
The third possibility may result from two adjacent free partitions being combined into one.

2.8.5 Virtual Memory and the 80286

For each memory reference, the logical address output by the CPU is translated into the physical address, which is the address sent to the memory by the memory management hardware. Logical addresses are the addresses that are generated by the instructions according to the addressing modes.

Because the logical addresses may be different from the physical addresses, the user can design a program in a logical space, also called a virtual space.

2.9 SYSTEM BUS STRUCTURE

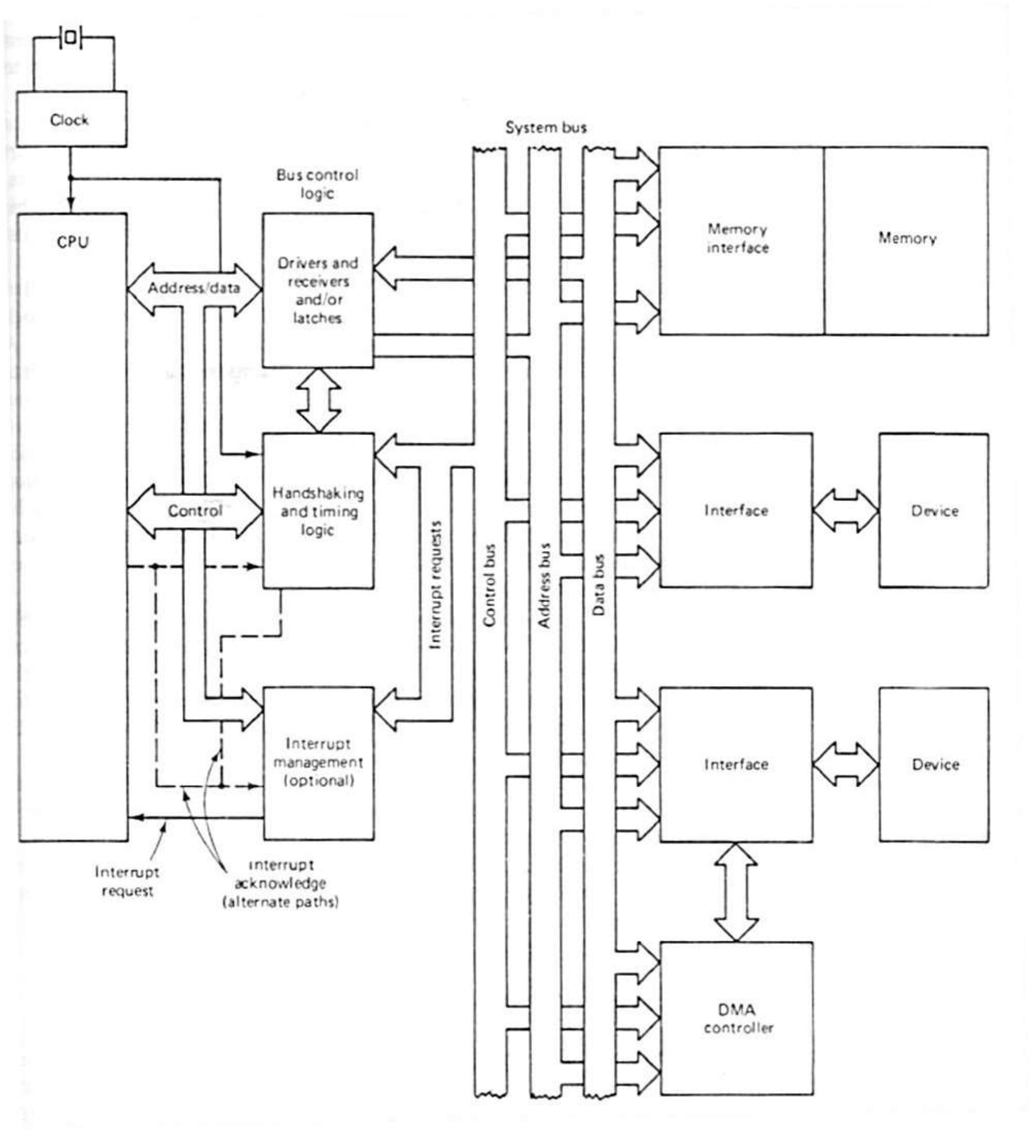


Fig 2.36: Typical system bus architecture

A set of conductors used for communicating information between the components in a computer system is called a bus. If a bus connects two minor components within a major component (e.g., the control unit to the set of working registers within the CPU), it is called an internal bus. When a bus connects two major components, such as a CPU and an interface, it is called an external bus.

Because an internal bus is ordinarily internal to an IC device and its construction is dependent on the device. External buses have common characteristics that must be understood when designing the overall architecture of a computer system. Some systems include more than one external bus. Others contain only one bus, which is referred to as the system bus, and it is the basic structure of system buses, particularly those in 8086 and 8088 based systems.

Figure 2.36 illustrates the fundamental structure of a system bus and its relationships to the various components of the computer system. The complexity of the bus control logic depends on the amount of translation needed between the system bus and the pins on the CPU, the timing requirements, whether or not interrupt management is included, and the size of the overall system. If a system component other than the CPU can be master of the bus, then all of the address and data lines and most of the control lines must be capable of being logically disconnected from the CPU or bus control logic, i.e., most pins connected to the bus must be such that they can enter a high-impedance state.

Generally, the circuits that drive the pins on a single-chip CPU have a quite limited driving capability and can be connected to only a few interfaces. For a small system, many or all of the CPU control pins could be used directly and the handshaking logic could be reduced or, perhaps, eliminated. Similarly, drivers and receivers would not be needed for the data and address lines. However, systems with several interfaces would need bus driver and receiver circuits connected to the bus in order to maintain adequate signal quality.

In addition to drivers and receivers, timing considerations may be such that latches are needed to hold the address that is output by the CPU during one part of the bus cycle but must be maintained throughout the cycle. Some processors, including the 8086 and 8088, use the same pins for both data and addresses. This means that during a data transfer operation the address, which is output first, must be latched before the bus is used to transfer the data.

The timing of the signals within the CPU and bus control logic is controlled by a clock. The bus cycles and CPU activity are controlled by groups of clock pulses. The exact number of clock pulses, or cycles, within a bus cycle varies. Atypical CPU input transaction would proceed by outputting the address of the data during the first clock cycle, signaling that a read is to take place during the second clock cycle, waiting an indeterminate number of clock cycles for the addressed device to put the data on the data lines, inputting the data, and signaling the device that the transfer is complete during the last clock cycle. For an output transaction the address would again be output during the first cycle and the data would be output during the second cycle along with a write signal. After the addressed device has accepted the data it would return a transfer complete acknowledgment which causes the CPU to enter the last cycle.

8088 is fully software compatible with the8086 and, as an 8-bit processor, is hardware compatible with the peripherals in the8080/8085 family. By using the 8088, an existing 8080/8085-based system can be updated with minimal hardware modification.

2.10 MULTIPROCESSOR CONFIGURATION

If a system includes two or more components that can execute instructions simultaneously, it is called a multiprocessing system. The added processors could be special purpose processors which are specifically designed to perform certain tasks efficiently, or other general purpose processors. For example, due to the 8086's limited data width and its lack of floating point arithmetic instructions, it requires many instructions to perform a single floating point operation.

For a system requiring a lot of computations, it would be desirable to perform such calculations with a supporting numeric data processor that is specifically designed to quickly operate on floating point numbers and numbers having larger than usual data widths. Also, it is sometimes advantageous to include in a system an I/O processor with greater capabilities than a DMA controller, one that can perform string manipulations, code conversion, character searching, and bit testing as well as the normal DMA operations. This would permit the CPU to concentrate on higher level functions.

In addition to improving the overall cost/performance ratio of a system, a multiprocessor configuration offers several desirable features not found in a one-processor design.

Several processors may be combined to fit the needs of an application. In a multiprocessor system tasks are divided among the modules. Should a failure occur, it is easier and cheaper to find and replace the malfunctioning processor than it is to find and replace the failing part in a complex processor.

Two problems must be considered in designing a multiprocessor system, bus contention and inter processor communication.

The maximum mode of the 8086 and 8088 is specifically designed to implement multiprocessor systems. Multiprocessing features are provided in maximum mode to accommodate three basic configurations.

They are

1. Coprocessor configuration
2. Closely coupled configurations
3. Loosely coupled configurations.

2.11 Coprocessor configuration:

In coprocessor configuration the CPU and the supporting or external processor share not only the entire memory and I/O subsystem, but also share the same bus control logic and clock generator.

In this configuration, the 8086/8088 is the master or host, and the supporting processor is the slave. The bus access control is provided by the CPU; therefore, the bus request signal from the supporting processor is connected to the CPU. In coprocessor configuration the supporting processor is dependent and must interact directly with the CPU.

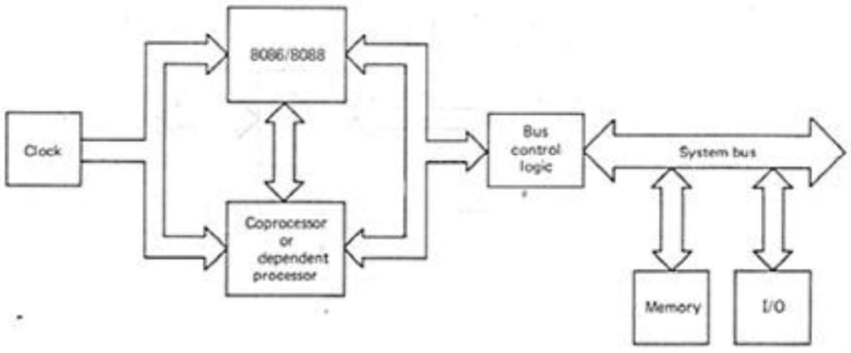


Fig 2.37: Coprocessor configuration

8086 instruction set is not sufficient to effectively satisfy some complex applications. For such applications, the 8086 must be supplemented with coprocessors that extend the instruction set in directions that will allow the necessary special computations to be accomplished more efficiently. For example, the 8086 has no instructions for performing floating point arithmetic, but by using an Intel 8087numeric data processor as a coprocessor, an application that heavily involves floating point calculations can be readily satisfied.

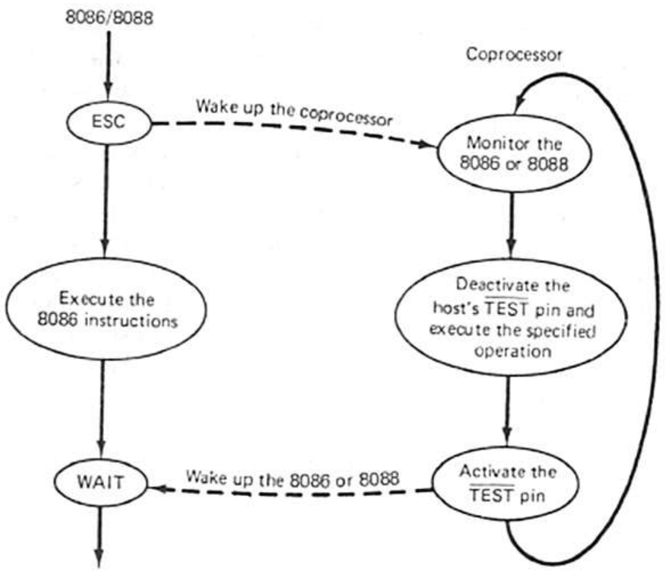

8086/8088
Coprocessor
ESC
Wake up the coprocessor
Monitor the 8086 or 8088
Execute the 8086 instructions
Deactivate the host's $\overline{TEST}$ pin and execute the specified operation
WAIT
Wake up the 8086 or 8088
Activate the $\overline{TEST}$ pin

Fig 2.38: Synchronization between the 8086 and its coprocessor

A coprocessor design does not require any extra logic other than that normally needed for a maximum mode system. Both the CPU and coprocessor execute their instructions from the same program, which is written in a superset of the 8086 instruction set. Other than possibly fetching an operand for the coprocessor, the CPU does not need to take any further action if the instruction is executed by the coprocessor. The interaction between the CPU and the coprocessor when an instruction is executed by the coprocessor is depicted in Fig. 2.38.

An instruction to be executed by the coprocessor is indicated when an escape (ESC) instruction appears in the program sequence. Only the host CPU can fetch instructions, but the coprocessor also receives all instructions and monitors the instruction sequencing of the host. An ESC instruction contains an external operation code, indicating what the coprocessor is to do and is simultaneously decoded by both the coprocessor and the host.

At this point the host may simply go on to the next instruction or it may fetch the first word of a memory operand for the coprocessor and then go on to the next instruction. If the CPU fetches the first word of an operand, the coprocessor will capture the data word and its 20-bit physical address. For a source operand that is longer than one word, the coprocessor can obtain the remaining words by stealing bus cycles.

If the memory operand specified in the ESC instruction is a destination, the coprocessor ignores the data word fetched by the host and later the coprocessor will store the result into the captured address. In either case, the coprocessor will send a busy (high) signal to the host's $\overline{\text{TEST}}$ pin and, as the host continues processing the instruction stream, the coprocessor will perform the operation indicated by the code in the ESC instruction.

This parallel operation continues until the host needs the coprocessor to perform another operation or must have the results from the current operation. At that time the host should execute a WAIT instruction and wait until its $\overline{\text{TEST}}$ pin is activated by the coprocessor.The WAIT instruction repeatedly checks the $\overline{\text{TEST}}$ pin until it becomes activatedand then executes the next instruction in sequence.

2.12 Closely coupled configurations:

In closely coupled configurations configuration the CPU and the supporting or external processor share not only the entire memory and I/O subsystem, but also share the same bus control logic and clock generator. In a closely coupled configuration the supporting processor may act independently.

For instruction fetches and data references, the independent processor accesses the bus through the CPU's $\overline{RQ}/\overline{GT}$lines.

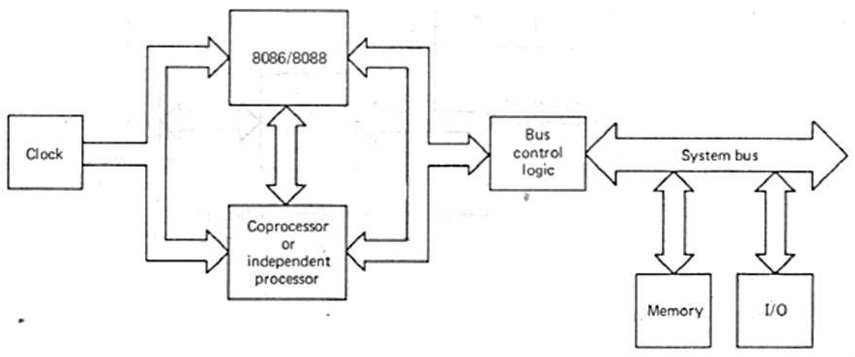


Fig 2.39: Closely coupled configuration

Instead of using special instructions such as ESC and WAIT, communication between the host and the independent processor is accomplished through shared memory space. As illustrated in Fig. 2.40, the host sets up a message in memory and then wakes up the independent processor by sending a command to one of the independent processor's ports.

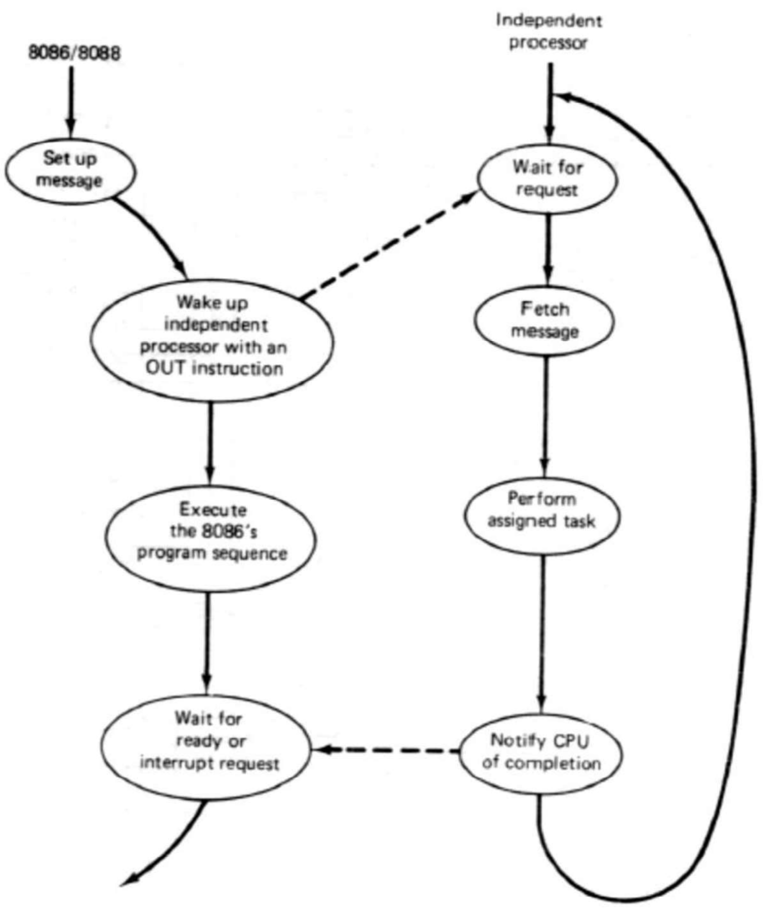

8086/8088
Independent processor
Set up message
Wake up independent processor with an OUT instruction
Execute the 8086's program sequence
Wait for ready or interrupt request
Wait for request
Fetch message
Perform assigned task
Notify CPU of completion

Fig 2.40 Interprocessor communication through shared memory

The independent processor then accesses the shared memory to get the assigned task and executes the task in parallel with the host. After the task is completed, the external processor notifies its host of the completion by using either a status bit or an interrupt request. The message format varies depending on the design of the independent processor and the application. Typically, a message should specify which operation is to be performed, the input parameters, and the addresses of the locations in which to store the results.

2.13 Loosely coupled configurations:

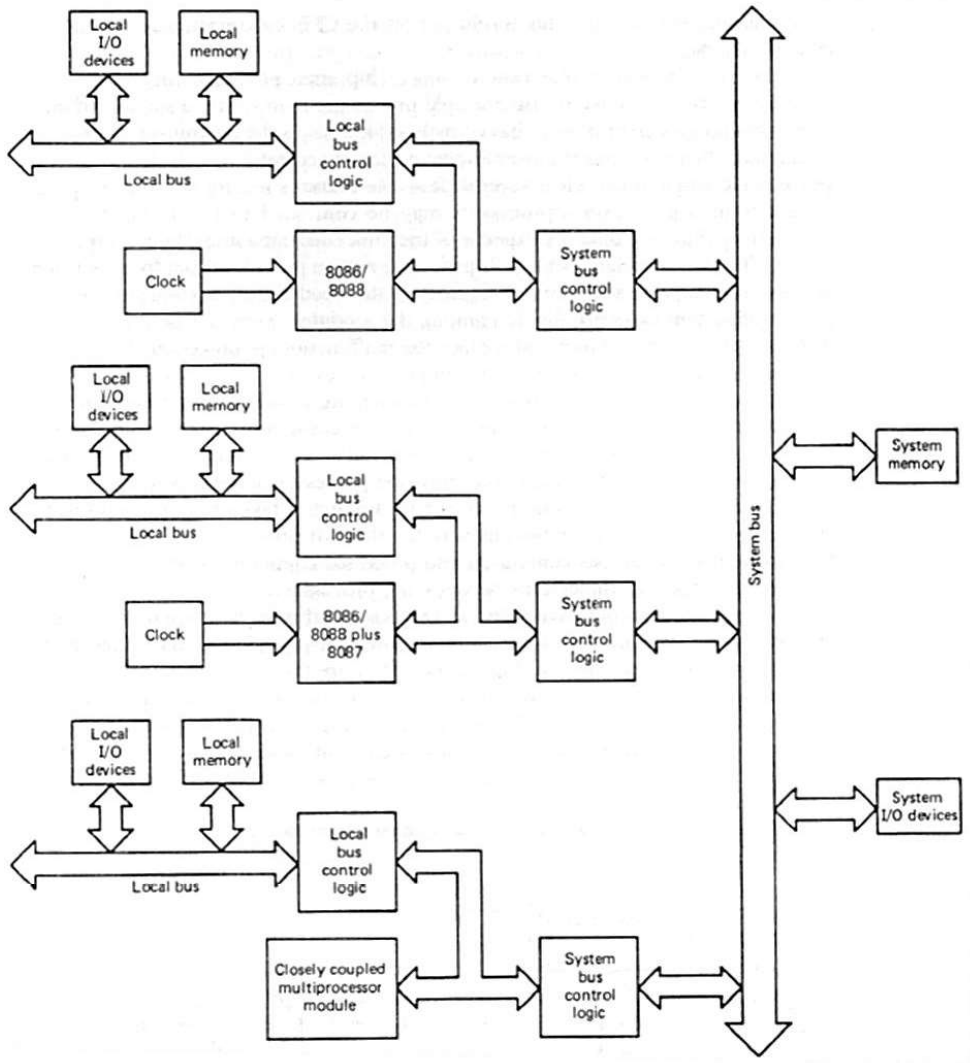


Fig 2.41: Loosely coupled configuration

Loosely coupled configurations are used for medium-size to large systems. Each module in the system may be the system bus master and may consist of an 8086, an 8088, another processor capable of being a bus master, or a coprocessor or closely coupled configuration. Several modules may share the system resources, and system bus control logic must resolve the bus contention problem. As shown in Fig. 2.41, each potential bus master runs independently and there are no direct connections between them.

Interprocessor communication is made possible through the shared resources. In addition to the shared resources, each module may include its own memory and I/O devices. The processors in the separate modules can simultaneously access their private subsystems through their local buses and perform their local data references and instruction fetches independently, thus improving the degree of concurrent processing.

In a loosely coupled multiprocessor system, each CPU has its own bus control logic and bus arbitration is resolved by extending this logic and adding external logic that is common to all master modules. Therefore, several CPUs can form a very large system and each CPU may have independent processors and/or a coprocessor attached to it. A loosely coupled configuration provides the following advantages:

1. High system throughput can be achieved by having more than one CPU.
2. Each bus master module is an independent unit and normally resides on a separate PC board. Therefore, a bus master module can be added or removed without affecting the other modules in the system.
3. A failure in one module normally does not cause a breakdown of the entire system and the faulty module can be easily detected and, replaced.
4. Each bus master may have a local bus to access dedicated memory or I/O devices so that a greater degree of parallel processing can be achieved.

In a loosely coupled multiprocessor system, more than one bus master module may have access to the shared system bus. Since each master is running independently, extra bus control logic must be provided to resolve the bus arbitration problem. This extra logic is called bus access logic and it is its responsibility to make sure that only one bus master at a time has control of the bus. Simultaneous bus requests are resolved on a priority basis. There are three schemes for establishing priority:

1. Daisy chaining.
2. Polling.
3. Independent requesting.

Implementation of the control logic may vary depending on the complexity of the bus access logic included in each module.

2.13.1 Daisy chaining

The daisy chain method is characterized by its simplicity and low cost. All masters use the same line for making bus requests. To respond to a bus request signal, the controller sends out a bus grant signal if the bus busy signal is inactive. The grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This module blocks the propagation of the bus grant signal, activates the busy line, and gains control' of the bus. Therefore, any other requesting module will not receive the grant signal and the priority is determined by the physical locations of the modules. The one located closest to the controller has the highest priority.

Compared to the other two methods, the daisy chain scheme requires the least number of control lines and this number is independent of the number of modules in the system. However, the arbitration time is slow due to the propagation delay of the bus grant signal. This delay is proportional to the number of modules and, therefore, a daisy chain-based system is limited to only a few modules. Furthermore, the priority of each module is fixed by its physical location and the failure of a module causes the whole system to fail.

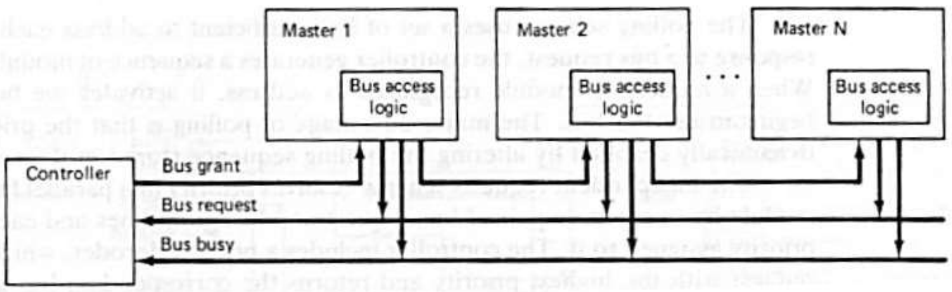


Fig 2.42: Daisy chain method

2.13.2 Polling

The polling scheme uses. a set of lines sufficient to address each module. In response to a bus request, the controller generates a sequence of module addresses. When a requesting module recognizes its address, it activates the busy line and begins to use the bus. The major advantage of polling is that the priority can be dynamically changed by altering the polling sequence stored in the controller.

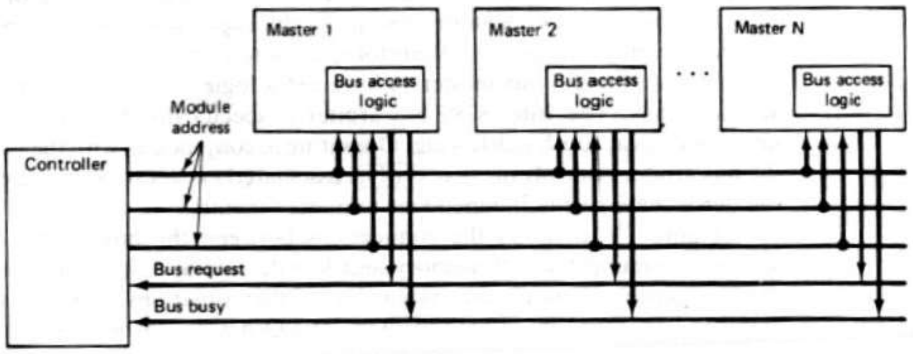


Fig 2.43 Polling method

2.13.3 Independent requesting

The independent requests scheme resolves priority in a parallel fashion. Each module has a separate pair of bus request and bus grant lines and each pair has apriority assigned to it. The controller includes a priority decoder, which selects the request with the highest priority and returns the corresponding bus grant signal Arbitration is fast and is independent of the number of modules in the system. Compared to the other two methods, the independent requests design is the fastest; however, it requires more bus request and bus grant lines (2n lines for n modules).

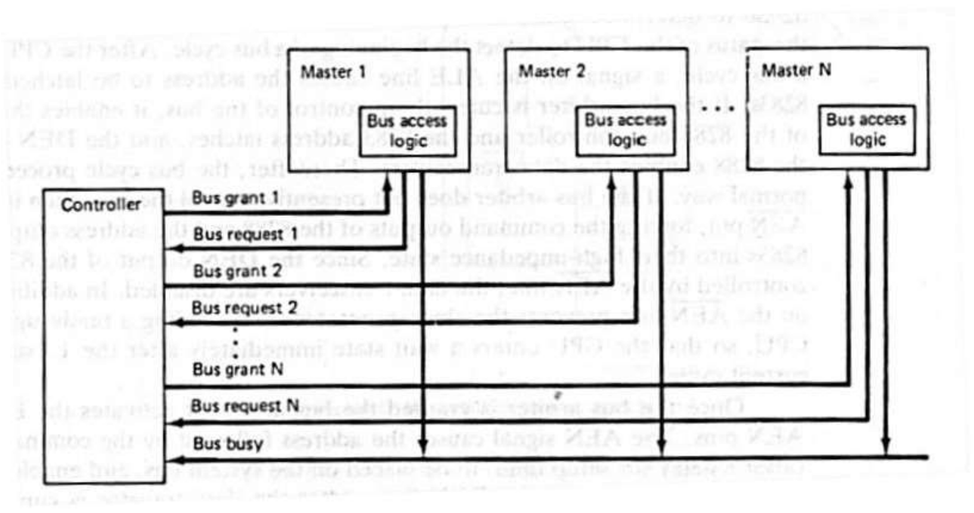


Fig 2.44: Independent requests method

LIST OF QUESTIONS
PART A

1. ***What are the two modes in which 8086 operates?***

   8086 operates in two modes

   3. Minimum mode and
   4. Maximum mode.

2. Explain the difference between minimum mode and maximum mode of operation.

	Minimum mode	**Maximum mode**
1.	When 8086 is operating in minimum mode the pin MN/MX will be connected to VCC	When 8086 is operating under maximum mode the pin MN/MX will be connected to GND.
2.	It is used in single processor environment	It is used in multi processor environment

3. State the functions of queue status lines QS0 and QS1 in 8086 microprocessor.

QS1	**QS0**	**Function**
0	0	Queue is in idle state
0	1	First byte of opcode has entered into the queue
1	0	Queue empty
1	1	Subsequent byte of opcode has entered into the queue