8085 INTERFACING DEVICES

Address space partitioning

8085 has 16 address lines. So, it can address 2^{16} = 64KB of memory to the maximum.

Now I/0 devices being interfaced can be accessed in 2 ways by means of mapping. It is the way in which the 8085 identifies the device among the whole memory block. It is done in 2 ways

Memory mapped I/0.

Here the MP uses16 address lines to identify on I/0 device. Here, the I/0 device is connected as a memory register itself. The MP uses the same control signal (Memory Read & Memory Write) & instructions as those of the memory. So, in this type of I/0 mapping, the MP follows the same steps as if it is accessing a memory register.

ie, I0/M = 0

Eg:- MOV A,M.

In this example, let HL pair have the value 8000. So, if an I/0 device is connected to the location 8000, the value in the I/0 device is taken to the accumulator.

I/0 mapped I/0

Here, the MP uses & address lines to identify an I/0 device. They are differentiated by the control signals themselves (I/0 Read &I/0 Write). The range of part ids from OOH to FFH is called I/0 map. But here, the devices need interfacing since only one is to be activated at once. This depends on the processor status. The devices use CS (Chip select) to do this. But for memory mapping, due to internal address decoding, this wasn't needed. Also, only half the address bus is used here.

Eg: IN 08H

Here, I0/M = 1.

Usually, a latch is used to interface O/P devices & a tristate buffer to interface I/P devices.

Data transfer schemes

Programmed data transfer

It requires programs & MP assistance. It is further divided as follows:

Synchronous- Both devices have equal speed, The MP doesn't check the readiness of the device. Whenever data is to be transferred, a code is executed & data is sent.

Asynchronous- There is speed mismatch between the devices. Firstly, the device sends a request for data transfer. On acknowledgement, data is transferred by executing special instruction codes.

Interrupt driven- Whenever data is to be sent, an interrupt is given On acknowledgement, data is transferred.

DMA data transfer

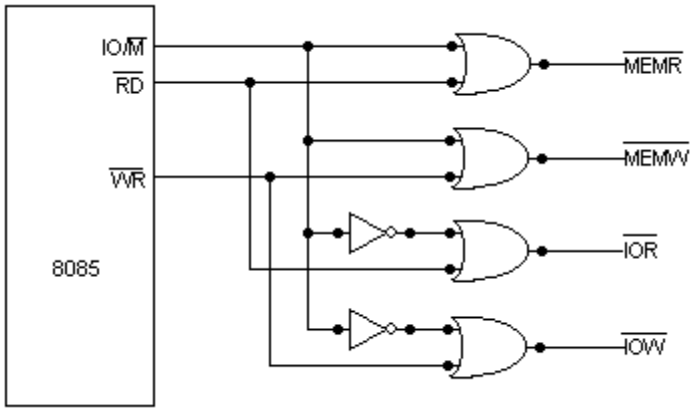
Here, MP isn't involved in data transfer. Instead, DMA controller takes care of the data transfer. It is further divided into three.

Burst mode- Once request is received, DMA controller asks MP to hand over the control of the system bus. If acknowledged, control is transferred and it stays with the DMA controller till the transfer is over.

Cycle stealing mode- Here, a convenient block of data is sent per cycle of the processor. So, the control of the system bus switches between the MP and DMA controller.
Demand mode- Even as data transfer begins, control of system bus is transferred only when a request is given by the DMA controller. During this, a block of data is sent.

Generation of control signals

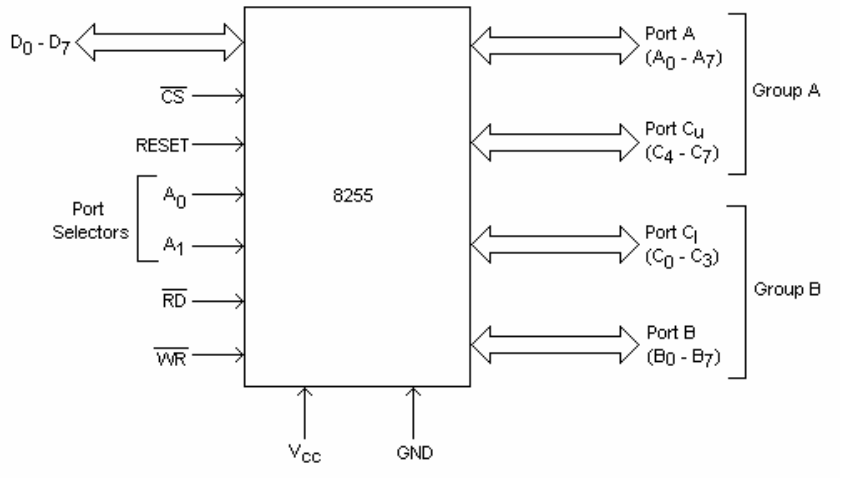
The different R/W signals are generated as follows.



Interfacing chips: Bridging network to connect sophisticated devices to the MP.

PROGRAMMABLE PERIPHERAL INTERFACE (8255)

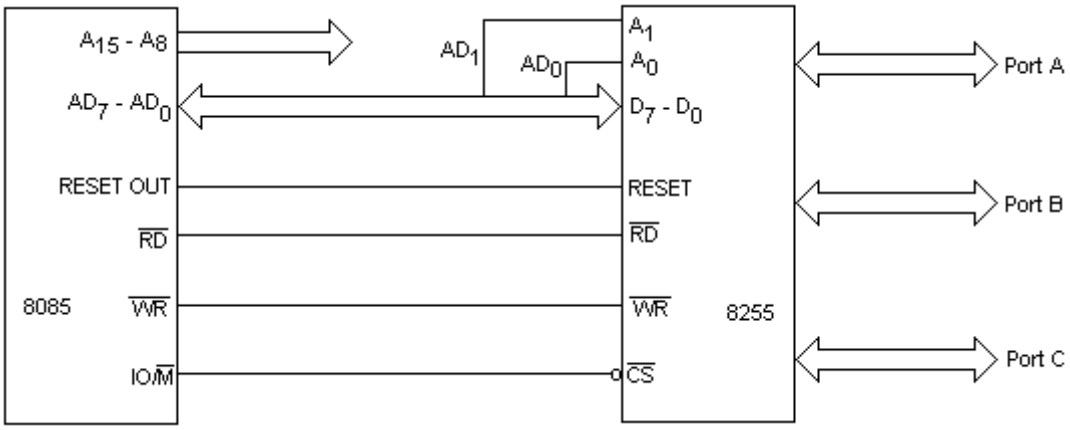
8255 is a 40 pin IC. It is used to interface parallel devices (mainly parallel port) that are used in communication with slower devices like printer. It is a general purpose, programmable, parallel I/O device. It has 24 I/O pins that can be grouped primarily in two 8-bit parallel ports: A and B, with the remaining eight bits as port C. The eight bits of port C can be used as individual bits or can be grouped in two 4-bit ports: C_{upper} & C_{lower}. The functions of these ports are defined by writing a control word in the control register. The 8255 chipset is as follows:



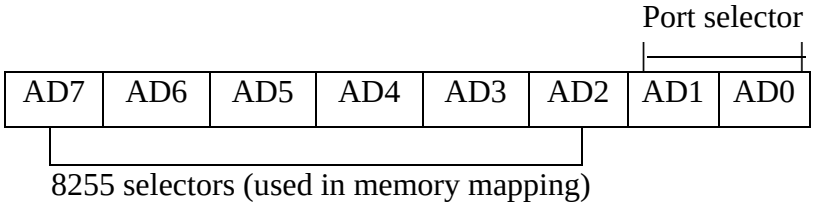
It has three 8-bit ports – Port A, Port B, Port C.
Port C is divided into two 4 bit ports – Port CU (C4-C7) & Port CL (C0-C3).
The ports have been grouped into two.

- Group A = Port A & Port CU
- Group B = Port B & Port CL

The 8255 is interfaced with the 8085 as follows:-



There is a control word register that notifies the different operation for 8255. It is an 8 bit register & is connected to the data bus. The 8bits are integrated as:



The different signals are taken as:

CS	A1	A0	Port selected	Port address
0	0	0	Port A	00H
0	0	1	Port B	01H
0	1	0	Port C	02H
0	1	1	Control Word Register	03H
1	X	X	8255 chip not selected	--

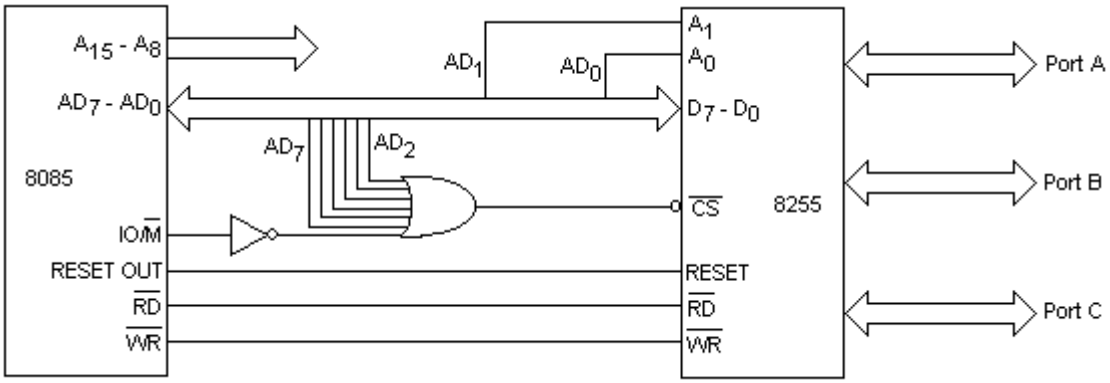
The interfaces mainly used in Traffic Signals.
Execution of IN instruction using the 8255 interface.
Eg:- IN 02H.
Firstly, the operand is retrieved from the address specified by the PC. It is taken to the IR & then decoded by the decoder. The decoder identifies the operation & then retrieves the port address from the next location. On receiving the port address, there are 3 phases in execution.

1. Control – Here, the appropriate control signals are initiated. It is an I/O Read Operation. So, at first, IO/M is set to 1. It indicates on I/O operation. This is connected to the chip selector of 8255, which is active low, through an inverter. So, 8255 is activated. Now, RD signal is set low. This initiates a Read Operation.

2. Address – Here, there is only an 8-bit port address, which is to be fitted into a 16-bit address bus. For that both halves of the address bus will be flushed with the same address. So, only AD0 – AD7 are required to carry the port address to the chip. But of these, only AD0 & AD1 are required to retrieve the data. Here, the value in data bus is 00000010, then AD0=0 and AD1=1
3. Data – The port selected is connected to the required I/O device. The device sends data through the port which is sent to 8085 by the demultiplexed AD0-AD7 bus, which is now D0-D7. After that, the control signals are again set as per the next instructions processes & requirements.

The same procedure applies for OUT instruction also, except for the fact that the connections are for O/P devices.

8255 Memory mapped I/O connection



Here, the device itself is considered as a memory location itself. So, we need not use the I/O instructions to access the devices. We just need to use the memory instructions to change or transfer data. For that, there are specific memory locations allotted. The devices are considered as Memory locations itself.

Eg;- if 8001 is defined as an I/O device, we can read from the device as:

MOV A, M, after initializing HL pair value as 8001. Let us consider MOV A, M for working. Here also, there are 3 phases.

1. Control – Firstly, RD is made low to initiate Read. Since the device is memory mapped, we take it as a memory location & IO/M is set low. But CS is associated with an inverter. So in that case, the chip wont be selected. So, another external inverter is also attached with the IO/M. But if other I/O mapped devices are also attached, there will be problems since the devices can't be selected. For that, the whole setup is modified as above. We can see that the data lines AD2 to AD7 are coupled to an OR gate along with the IO/M pin. So, for the memory mapped devices, any one of the pins AD2 to AD7 should be high to activate the chip or device. But for I/O mapped device, the OR gate needn't be used.

2. Address – Here, the address is 16 bit since it is memory mapped. But the upper half of the address lines is discarded. So, only the bottom half (last 2 nibbles) are used. Of the last two bits, the device ports are selected. The others go to the OR gate for chip selection. If any one is low, the chip is activated.
3. Data – After the address is given, the AD0 – AD7 bus is demultiplexed to data bus and data transfer occurs, coupled with the port lines. After this, control signals are reset.

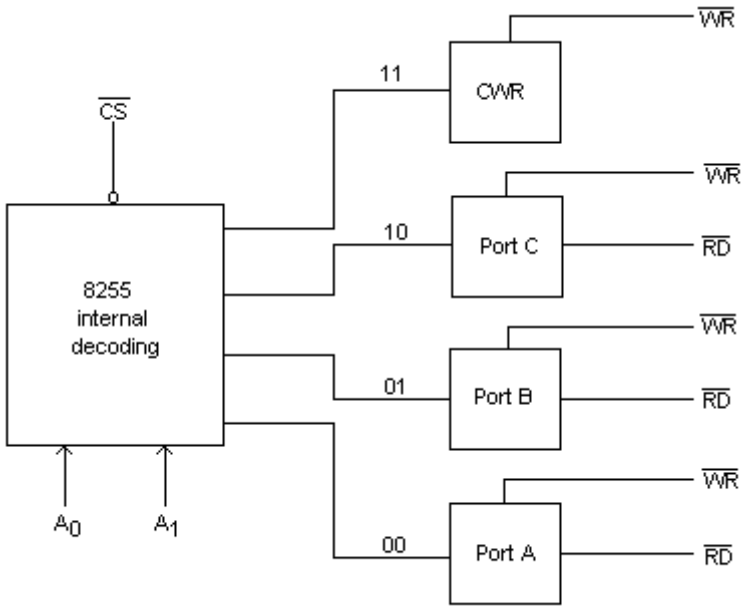
I/O mapping: Instruction based. Specific instructions are needed for execution. The instructions are IN ad OUT.

Memory mapping: Program based. Any memory-based operation can be done with this since the device is like a memory location itself. The execution depends on the HL pair value.

Parallel ports
Three parallel ports are available using 8255.
LPT1: Physical port, accessible to external devices.
LPT2 and LPT3: Logical ports, which are program based. These are virtual ports, which automatically redirect to LPT1.

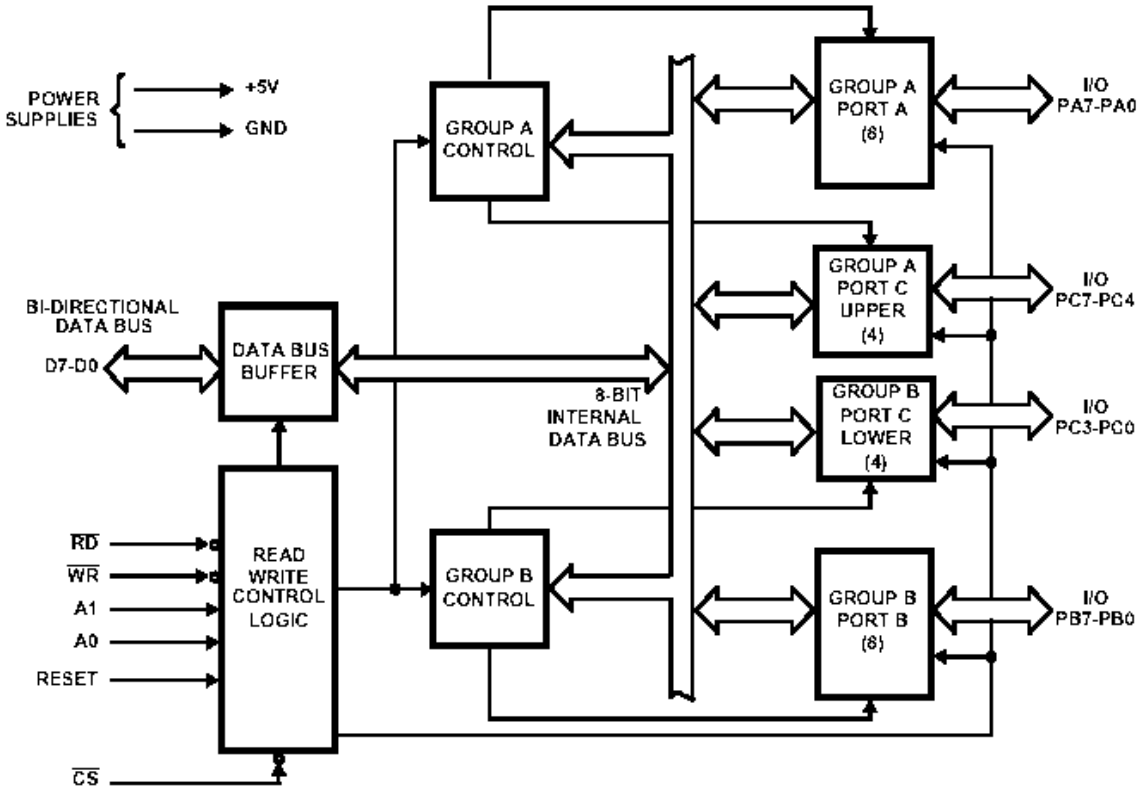
The main application of 8255 – 8085 combination is, as mentioned before, is in traffic signaling systems, since they can be timed using programs to perform delay operations.

8255 programming modes



It is determined by the Control Word Register (CWR).

Architecture of 8255



Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

(CS) Chip Select. A "low" on this input pin enables the communication between the 82C55A and the CPU.

(RD) Read. A "low" on this input pin enables 82C55A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 82C55A.

(WR) Write. A "low" on this input pin enables the CPU to write data or control words into the 82C55A.

(A0 and A1) Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word register. They are normally connected to the least significant bits of the address bus (A0 and A1).

(RESET) Reset. A "high" on this input initializes the control register to 9BH and all ports (A, B, C) are set to the input mode. "Bus hold" devices internal to the 82C55A will hold the I/O port inputs to a logic "1" state with a maximum hold current of 400mA.

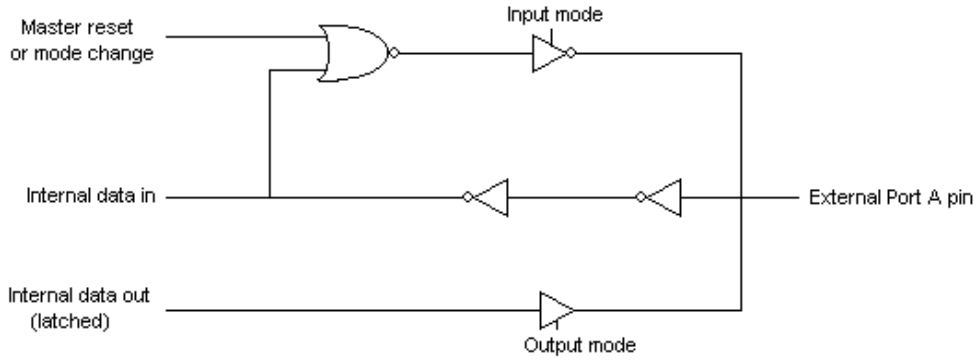
Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 82C55A. The control word contains

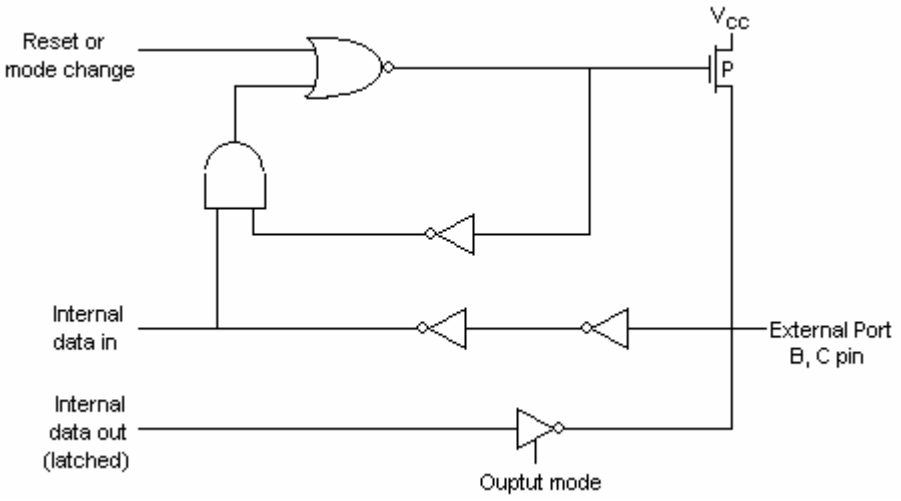
information such as “mode”, “bit set”, “bit reset”, etc., that initializes the functional configuration of the 82C55A. Each of the Control blocks (Group A and Group B) accepts “commands” from the Read/Write Control logic, receives “control words” from the internal data bus and issues the proper commands to its associated ports.
Control Group A - Port A and Port C upper (C7 - C4)
Control Group B - Port B and Port C lower (C3 - C0)
The control word register can be both written and read as shown in the “Basic Operation” table.

A1	A0	RD	WR	CS	INPUT OPERATION (READ)
0	0	0	1	0	Port A to Data Bus
0	1	0	1	0	Port B to Data Bus
1	0	0	1	0	Port C to Data Bus
1	1	0	1	0	Control Word to Data Bus
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	Data Bus to Port A
0	1	1	0	0	Data Bus to Port B
1	0	1	0	0	Data Bus to Port C
1	1	1	0	0	Data Bus to Control
					DISABLE FUNCTION
X	X	X	X	1	Data Bus to Three State
X	X	1	1	0	Data Bus to Three State

Ports A, B, and C
The 82C55A contains three 8-bit ports (A, B, and C). All can be configured to a wide variety of functional characteristics by the system software but each has its own special features or “personality” to further enhance the power and flexibility of the 82C55A.
Port A One 8-bit data output latch/buffer and one 8-bit data input latch. Both “pull-up” and “pull-down” bus-hold devices are present on Port A.



Port B One 8-bit data input/output latch/buffer and one 8-bit data input buffer.
Port C One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal output and status signal inputs in conjunction with ports A and B.



Control Word Register (CWR)

It is an 8 bit register that can be used to set different statuses for 8255. The content of control register is control word, which specifies I/O function for each port. The register can be accessed to write a control word when A0 and A1 are at logic 1. The register is not accessible for read operation. Using CWR, the 8255 can be programmed to operate in 3 different modes.

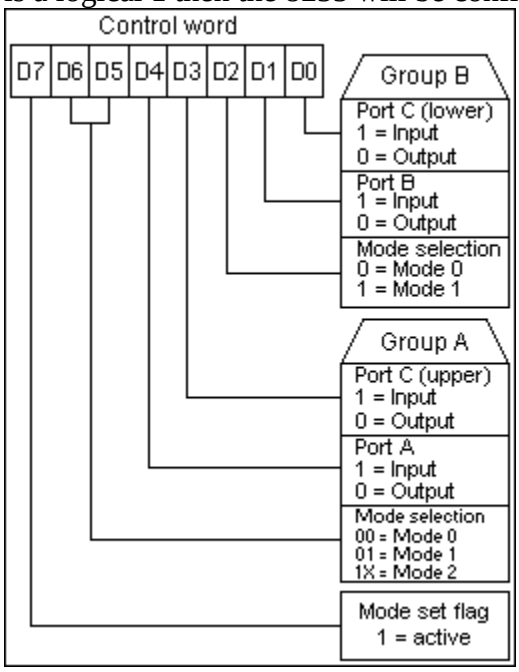
Mode 0 – Basic I/O

Mode 1 – Strobed I/O

Mode 2 – Strobed bidirectional I/O

Each bit of the CWR has its own significance. They are given below.

8255 can operate in two modes Input/Output mode and Bit set/Reset Mode. To make the operation possible in either of these modes, we write control words to control register. If bit 7 of the control word is a logical 1 then the 8255 will be configured in I/O mode



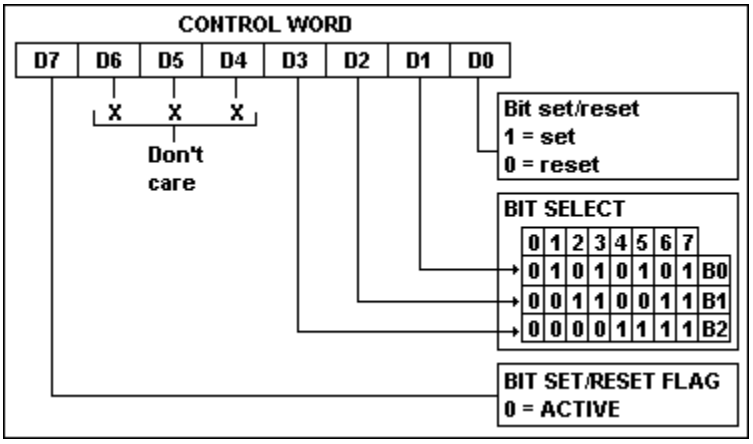
Here, the modes for Ports A and B can be separately defined, while Port C is divided into 2 portions as required by the Ports A and B definitions. All the O/P registers, including status flip flop will be reset whenever mode changes. Modes can also be combined for catering to different purposes simultaneously.

Eg:- Group A in Mode 1 I/P and Group B in Mode 0 O/P

1	0	1	1	0/1	0	0	0/1

Here, the C ports are set as 0/1 since their mode of transfer (input/output) aren't defined.

Now, if bit 7 of the control word is a logical 0 then each bit of the port C can be set or reset



This is called BSR (Bit Set Reset) mode. This allows the user to set or reset any of the Port C bits using just a single instruction. This reduces software requirements in control-based applications.

The 7th bit of the CWR is recognized as BSR flag. When it is 1, BSR mode is activated. This doesn't affect any settings, which had been preset using the 7th bit as 1. The operations of Ports A and B also remain unaffected.

The different BSR combinations are:

Operation	Control Word	MVI A,__H
Reset C0	00000000	00
Set C0	00000001	01
Reset C1	00000010	02
Set C1	00000011	03
Reset C2	00000100	04
Set C2	00000101	05
Reset C3	00000110	06
Set C3	00000111	07
Reset C4	00001000	08
Set C4	00001001	09
Reset C5	00001010	0A
Set C5	00001011	0B
Reset C6	00001100	0C
Set C6	00001101	0D
Reset C7	00001110	0E
Set C7	00001111	0F

So, we can see that the CWR can be initialized by storing the required control word in the accumulator using MVI instruction and then giving the instruction OUT 03H.

Mode 0

This is the basic I/O mode. The functional configuration of provides simple I/O operation for each of the three ports. No handshaking is required. Data is simply written to or read from a specific port. Some basic features are:

- Two 8-bit and two 4-bit ports.
- Any port can be I/P or O/P.
- O/Pare latched, but I/P aren't latched.
- 16 different I/O configurations possible.

A		B		Group A		#	Group B	
D4	D3	D2	D1	Port A	Port Cu		Port B	Port Cl
0	0	0	0	Output	Output	0	Output	Output
0	0	0	1	Output	Output	1	Output	Input
0	0	1	0	Output	Output	2	Input	Output
0	0	1	1	Output	Output	3	Input	Input
0	1	0	0	Output	Input	4	Output	Output
0	1	0	1	Output	Input	5	Output	Input
0	1	1	0	Output	Input	6	Input	Output
0	1	1	1	Output	Input	7	Input	Input
1	0	0	0	Input	Output	8	Output	Output
1	0	0	1	Input	Output	9	Output	Input
1	0	1	0	Input	Output	10	Input	Output
1	0	1	1	Input	Output	11	Input	Input
1	1	0	0	Input	Input	12	Output	Output
1	1	0	1	Input	Input	13	Output	Input
1	1	1	0	Input	Input	14	Input	Output
1	1	1	1	Input	Input	15	Input	Input

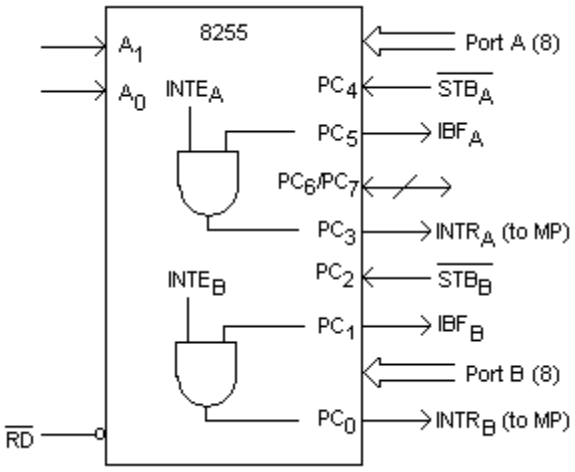
Mode 1

It is strobed I/O. This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or handshake signals. In Mode 1, Ports A and B use the lines on Port C to generate or accept these handshake signals.
The main features are:

- Two groups A and B.
- Each group has an 8-bit data port and a 4-bit control/data port.
- The 8-bit port can either be I/P or O/P. Both I/P and O/P are latched.
- The 4-bit port is used for control and status of the 8-bit port.

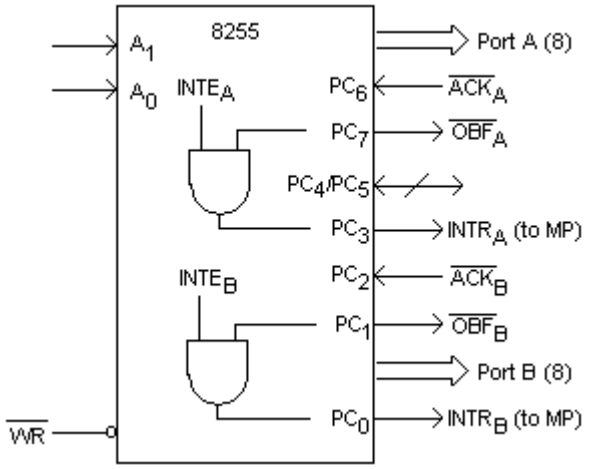
Latches: A buffer capable of storing a bit of information at a time.

I/P control signal definition



STB - Strobe output. A low on this line loads data on to the I/P latch.
IBF – I/P buffer full. A high on this line indicate that data has been loaded into the I/P latch. IBF is set by STB having law & is reset by rising edge of the RD I/P.
INTR – interrupt request to MP. This indicates that there is data present in 8255 to be written into the Processor. This is activated when a device is requesting service. INTR is set when STB = 1, IBF = 1 & INTE = 1. It is reset by falling edge of RD.
INTE – enables interrupts.
INTE A – controlled by PC4.
INTE B - controlled by PC2.

O/P control signal definition.



OBF – O/P buffer full. It will go low to indicate that CPU has written data out to the specified port. This doesn't mean that valid data is sent out of the port. Data is guaranteed valid at rising edge of OBF. It is set by rising edge of WR & reset by ACK being low.
ACK – A low on this inform 8255 that data is ready to be accepted i.e. device is ready for data input.

INTR – When device has accepted the data, it is made high. INTR is set if ACK = 1, OBF = 1 & INTE = 1. It is reset by falling edge of WR.

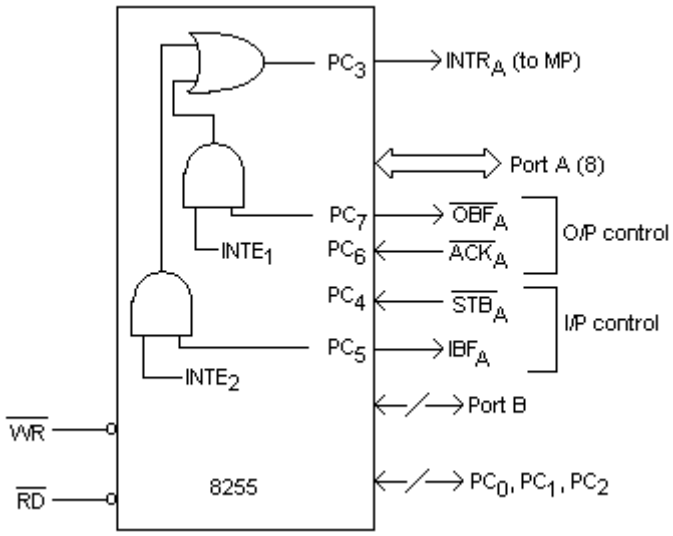
Mode 2

It is strobed bidirectional bus I/O. It provides a means for communication with a peripheral device or structure on a single 8-bit bus for both transmission and reception. Handshaking signals are provided to maintain proper bus flow discipline. Interrupt generation and enable/disable functions are also available.

- Used in Group A only.
- Both I/P and O/P are latched.
- One 8-bit, bidirectional bus port (Port A) and a 5-bit control port (Port C) used.
- The 5-bit Port C used for control and status for 8-bit bidirectional Port A.

Application: Car Air-conditioning.

Control Signal Definition



INTR – A high on this can be used to interrupt the CPU for both I/P and O/P operations.

O/P operations

OBF: It will go low to indicate that CPU has written data out to Port A.

ACK: A low on this enables the Tristate O/P buffer of Port A to send out the data. Otherwise, O/P buffer will be in the high impedance state.

INTE1: INTE associated with IBF.

I/P operations

STB: A low on this loads data into the I/P latch.

IBF: A high on this indicates that data has been loaded into the I/P latch.

INTE2: INTE associated with IBF.

USART (8251)

USART stands for Universal Synchronous Asynchronous Receiver Transmitter. 8251 is a commonly used USART. It is used for serial data transmission. It is compatible with 8085, 8086 and 8088. As peripheral device of a MP system, the 8251 receives parallel data from the CPU and transmits serial data after conversion. This device also receives serial data from outside and transmits parallel data to the CPU after conversion.

Synchronous Serial Transmission

Here, the receiver and transmitter are synchronized. A block of characters is transmitted with a clock signal (synchronizing information) to synchronize the transmission and reception. Synchronous communication is usually more efficient because only data bits are transmitted between sender and receiver, and synchronous communication can be more costly if extra wiring and circuits are required to share a clock signal between the sender and receiver.

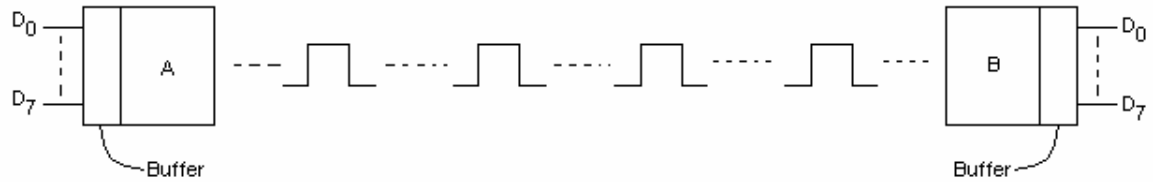
Asynchronous Serial Transmission

Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, the sender and receiver must agree on timing parameters in advance and special bits are added to each word which are used to synchronize the sending and receiving units. It is character oriented mode of transmission. When a word is given to the UART for Asynchronous transmissions, a bit called the "Start Bit" is added to the beginning of each word that is to be transmitted. The Start Bit is used to alert the receiver that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter. Transmission begins with the start bit followed by a character and one or two stop bits. Usually start bits are low signals and stop bits are continuous high signals. So, the format of asynchronously transmitted data will be:

Start bit	Data	Stop bits

This is called bit stuffing.

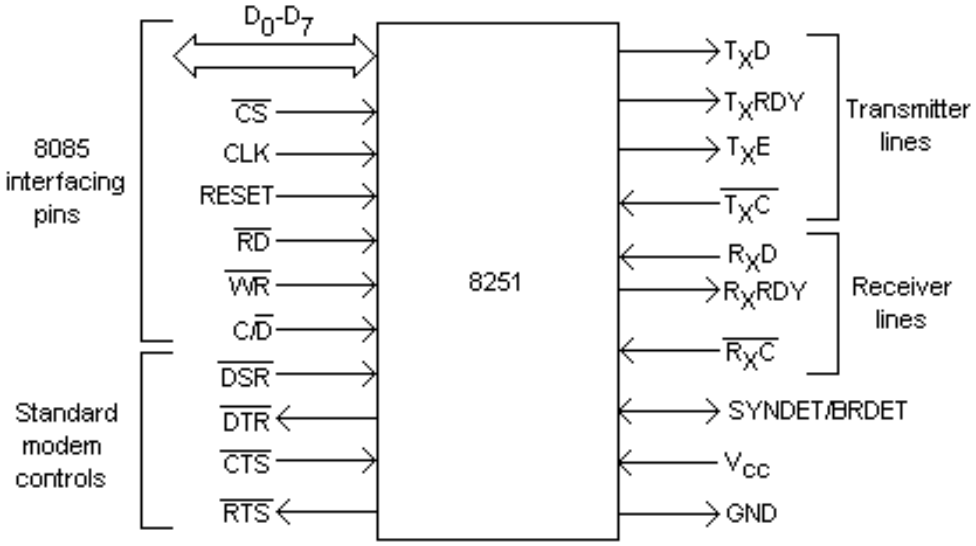
Transfer types



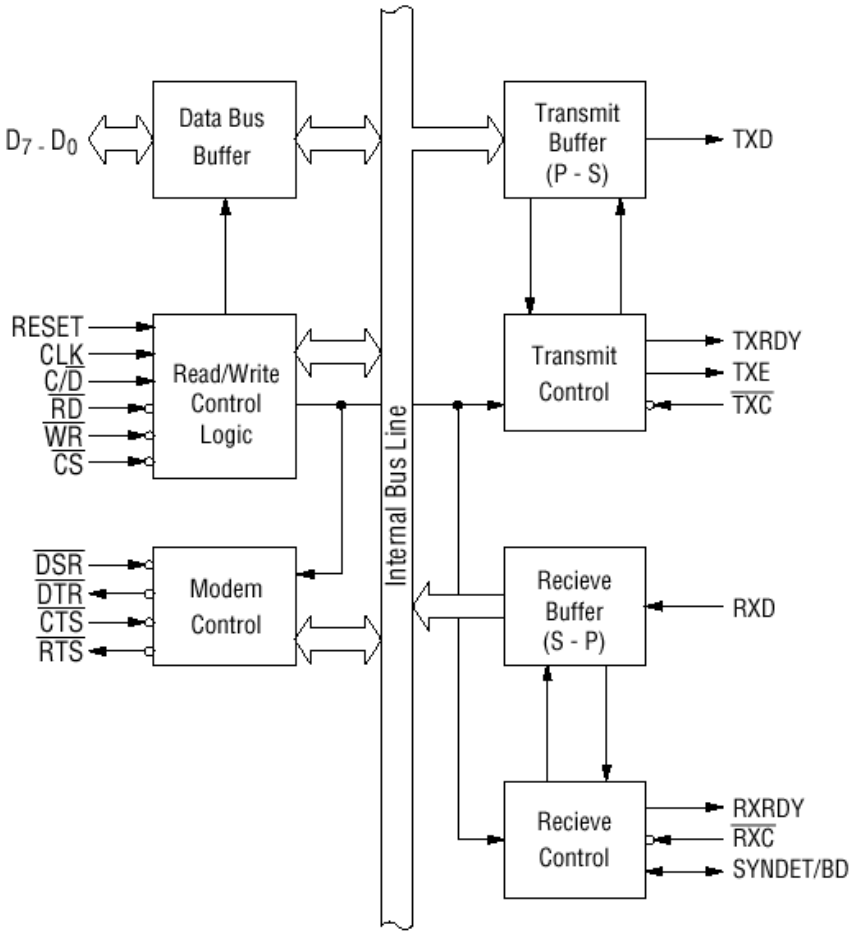
Data transfer between 2 points can be of 3 types.

- Simplex – Here, data transfer is unidirectional. SO, data can be sent only either from A to B or from B to A. So, if bidirectional data transfer is needed, we need atleast 2 cables. So, A→B or B→A.
- Half duplex – Here, data transfer is bidirectional, using one cable. So, obviously, it can't be simultaneous. Thus, we can see that even if 2-way data transfer is possible, it can't be realized fully since there is only one cable. So, A↔B.
- Full duplex – This is a bidirectional implementation of the simplex mode. Here, there are 2 separate cables, one dedicated for each direction. So, data can be transferred in both ways, even simultaneously. So, A ⇔B.

Pinout



Functional block diagram



The different controls in 8251 are:

$C/\overline{D}$	= control or data		
TxD TxRDY TxE $\overline{TxC}$	= data line = ready = empty = clock	TRANSMITTER LINES	
RxD RxRDY $\overline{RxC}$	= data line = ready = clock	RECEIVER LINES	
SYNDET / BRDET = synchronous / break detect (only for asynchronous transmission)			
$\overline{DSR}$ $\overline{DTR}$	= data set ready (acknowledgement) = data terminal ready (indicate data read)	MODEM TO 8251	STANDARD MODEM CONTROLS
$\overline{RTS}$ $\overline{CTS}$	= request to send (intimate data) = clear to send (acknowledgement)	8251 TO MODEM	

So, we can see that 8251 has 5 main sections.

- Read / Write control logic
- Modem control
- Data bus buffer
- Transmitter
- Receiver

Read/Write Control logic

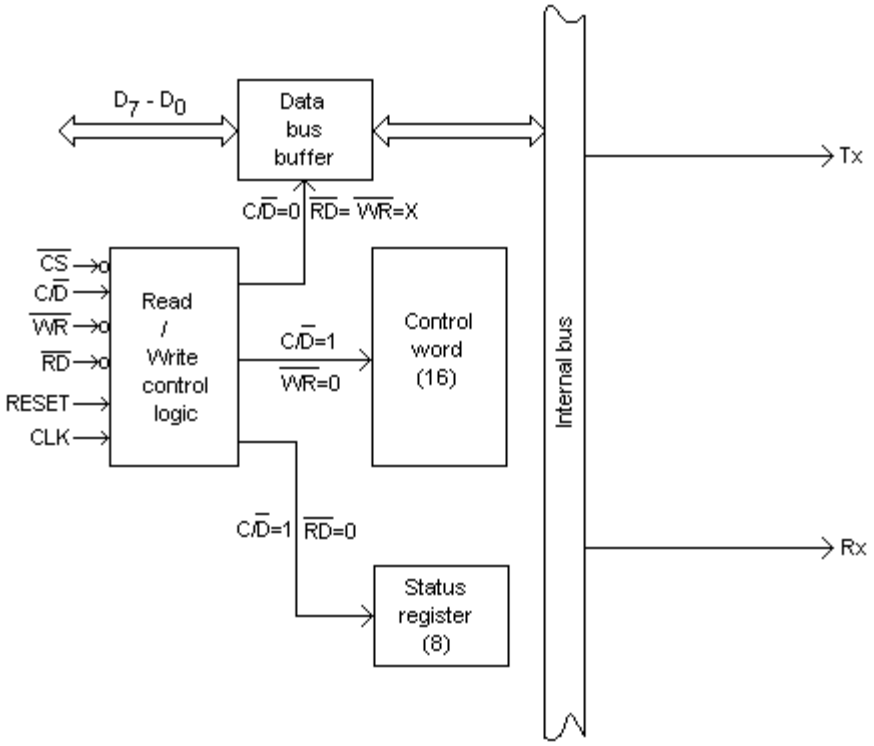
This section includes a control logic, six input signals, and three buffer registers: Data register, control register and status register. The control logic interfaces the chip with MPU, determines the functions of the chip according to the control word in the control register and monitors the data flow.

Input signals

- $\overline{CS}$ – Chip Select: When this signal goes low, the 8251A is selected by the MPU for communication.
- $C/\overline{D}$ – Control/Data: When this signal is high, the control or status register is addressed; when it is low, data buffer is addressed. The control register and status register are differentiated by WR and RD signals.
- $\overline{WR}$: When this signal is low, the MPU either writes in the control register or sends output to the data buffer
- $\overline{RD}$: When this signal goes low, the MPU either reads a status from the status register or accepts data from data buffer.
- RESET: A high on this signal reset 8252A and forces it into the idle mode.

CLK: This is the clock input, usually connected to the system clock for communication with the microprocessor.

The I/P section of the 8251 would look like:



The different control signals work in conjunction to perform different operations.

CS'	C/D'	RD'	WR'	Function
0	1	1	0	MP to control register (write)
0	1	0	1	Status register to MP (read)
0	0	0	1	MP to O/P buffer Tx (write)
0	0	1	0	Data from data buffer to MP (read)

Control word (16 bit)

It is a write only register (ie, reading isn't allowed), that stores the status of 8255 at that time and the operation it is supposed to perform. It has 2 sections:

Mode word (8 bit): Store general characteristics of operation (baud, parity, character length, number of stop bits, etc.).

Control word (8 bit): Enables data transmission and reception.

Baud rate is usually expressed in bps (bits/sec).

Character length defines the no. of bits reqd to rep. a character. For ASCII, it is 7 and for EBCDIC, it is 8.

Parity is defined by the user.

Status word

Stores the register status and transmission errors. Data can only be read from the status reg.

Steps in programming

1. Read status reg. If not busy presently, proceed.
2. Reset control reg.
3. Write mode word (MVI + OUT).
4. Write command word (MVI + OUT).
5. Transmit data.

Status word format

DSR	SYN/BR	FE	OE	PE	TxE	RxRDY	TxRDY

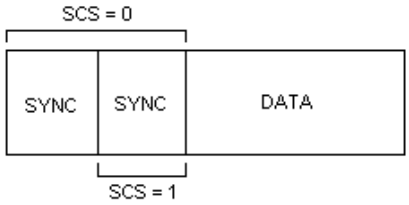
8251 lines

- DSR: If set, Data Set Ready.
- PE: If set, parity error.
- OE: If set, overrun error. ie, CPU cant read from 8251. This occurs during reception. Here, the device sends information to the I/P buffer and than it is sent to CPU. But if CPU doesn't read any bit, that bit is lost forever. If such a situation occurs, OE is set.
- FE: If set, framing error. This is applicable only for async transmission, where it is used to check stop bits. A std. stop bit format is 11. If it is different in the transmitted data, this bit is set.

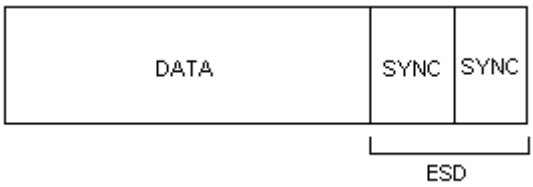
Mode word format

SCS	ESD	EP	PEN	L2	L1	S/A1	S/A0

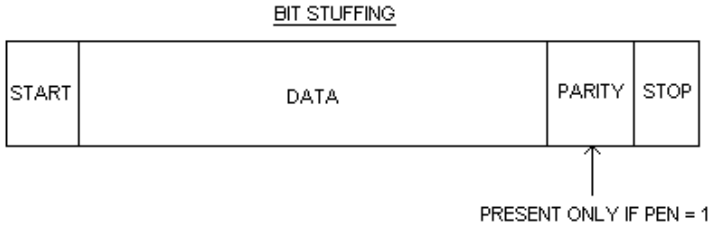
- SCS: Single char sync. If 1, only one sync char. If 0, double sync char.



- ESD: External sync detect. If 1, SYNDET of an I/P device. If 0, SYNDET of an O/P device. Sync bits are sent to the device or read from the device only if ESD is 1. The sync character is decided by the system itself.



- EP: If set, parity is even. Otherwise, odd. It is checked only if PEN is 1.
- PEN: If set, parity is enabled.



- L2, L1: Determines character length.

L2	L1	Character length
0	0	5 bits
0	1	6 bits
1	0	7 bits
1	1	8 bits

Depends on the encoding scheme.

- S/A1, S/A0: determines the speed of transmission and the mode of transmission.

S/A1	S/A0	Transmission mode	Transmission speed
0	0	Sync	Clock speed
0	1	Async	Base speed
1	0	Async	Base speed / 16
1	1	Async	Base speed / 64

So, it determines division of speed (baud).

Command word format

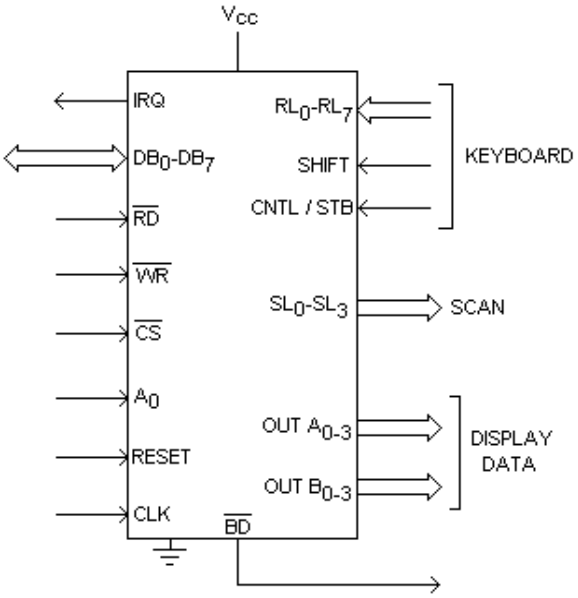
EH	IR	RTS	ER	SBRK	RxEN	DTR	TxEN

- TxEN: If set, transmit enable.
- DTR: If set, Data Terminal Ready. It intimates readiness to send or recv data.
- RxEN: If set, receive enable.
- SBRK: Send Break Character. If 1, break. If 0, normal operation. It is used in sync transmission to indicate end of data block.
- ER: Error flag Reset. If set, PE, OE and FE will reset.
- RTS: If set, Request to Send.
- IR: If set, Internal Reset. It resets the entire registers.
- EH: Enter Hunt Mode. Has no effect on async transmission. It searches for sync character.

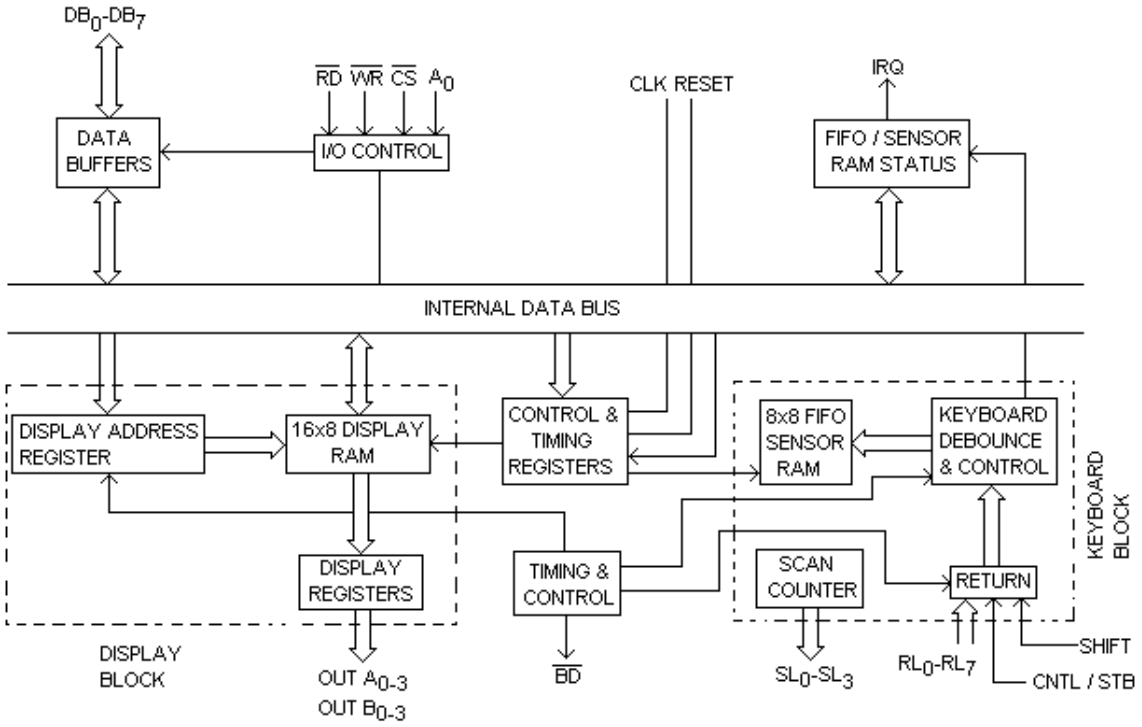
In async transmission, if we have 10 bits of information to be sent and we are using a 5-bit mode, that data will be sent as 2 blocks. But if the same is sent using sync transmission, the whole data will be sent at once, since there is no data limit in sync transmission.

PROGRAMMABLE KEYBOARD/DISPLAY INTERFACE (8279)

Pinout



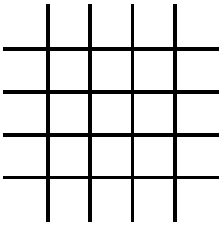
Functional block diagram



It scans and encodes upto 64-key keyboards. It can also control upto a 16-digit 7-segment display. The keyboard section has a built in FIFO 8 character buffer. The display is controlled from an internal 16x8 RAM that stores the coded display information.

Interfacing with 8085, keyboard and display

While interfacing, A0 determines control or data input while IRQ is the interrupt pin of 8279.



RL0 to RL7 are called Return Lines. They forward the row information from the keyboard to 8279. They work in conjunction with 2 other pins.

SHIFT: To detect the function of keys having multiple functions. Eg: a /A

CNTL/STB: Control or Strobe input, which takes inputs for special functions from the keyboard. Eg: Ctrl+B=Bold

Now, to interface display, 8279 has two 4-bit ports. They can be used together as an 8-bit port. The BD pin, if set low, will clear the display.

Now, a group of lines are common to both keyboard and display controls. They are the scan lines. Lines SL0 to SL3 are called Scan lines. They scan the O/P display device or I/P keyboard device.

Keyboard block

8x8 FIFO sensor RAM

- Holds info from keyboard.
- Can hold 8 characters at a time.
- It acts as a keyboard buffer.
- It is a FIFO queue.

RAM status indicator

- Checks if the FIFO RAM is full or not.
- Determines which info is to be forwarded.
- Checks which blocks are free.
- Finds how many blocks are full.

Keyboard debounce control: Controls the whole activity.

When some info is to be sent to the computer via the keyboard, the CNTL/STB line is set. This signal is given to the keyboard control. Then the key press is checked. If a key is pressed, it is detected. Then, the debounce control waits for 10 ms to ensure that the

keypress is not accidental. Then, again it checks for key press and this info is sent to the FIFO RAM. After this, it is forwarded to the MP. For this, we use the status reg.
If the RAM has contents in it, status reg is enabled, and the interrupt IRQ is activated. So, the MP is interrupted. Then, the info in the RAM is sent to the data buffer, from where it is taken to the MP's accumulator by the data bus for processing.

Display
Here, the data is taken from the accumulator to the O/P port.
Display address reg: to select the O/P port (A, B, A&B)
16x8 display RAM: To store the data. Can store upto 16 characters at a time.
Display reg: Similar to buffer. It manages speed mismatch between the display and the MP by temporarily storing (queuing) the O/P data for the display.

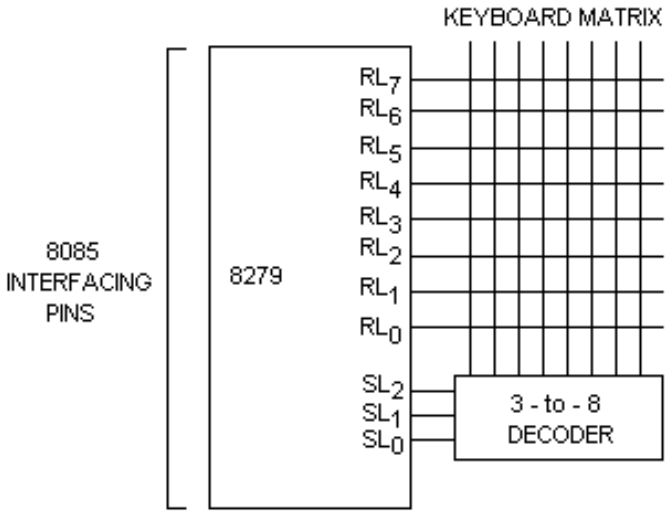
The info from the MPO is taken to the data buffer by the data bus. Then, it is queued up in the 16x8 display RAM. Then, the data is taken out one by one to the display registers, from where they are displayed by the display connected.

Control and timing reg: contains the control word that decides the operation of 8279.

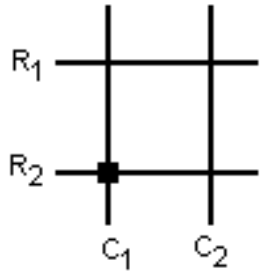
Scan counter: It is a binary counter (counts from 0000 to 1111). For each number, the data is stored for 10 ms and continues with the next number. The scan lines have 2 main functions:

- Scan the keyboard for keypress.
- Refresh the display.

Interfacing the keyboard to 8279 – How is keypress identified?
The keyboard will be interfaced to 8279 as:



Now, consider a simple key matrix.



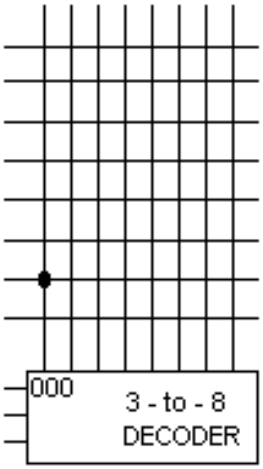
Here, the contact point indicates a keypress. So, R_2C_1 is read to locate the key. The actual matrix will have a much higher number of rows and columns. The columns are charged (5V) while the rows are grounded (0V). When a key is pressed, a row and column comes in contact and the circuit is completed. This position is identified and the corresponding keypress is detected.

Here, the scanning is done by the scan counter (SL0 to SL3). These lines are connected to a 3-to-8 decoder. Since we have 8 rows, we need 8 columns to correctly represent all the 64 key positions. So, we use a 3-to-8 decoder.

Scan counter value	Line activated
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

This process continues. ie, it scans all the lines of the keyboard matrix.

Consider a keypress as shown below.

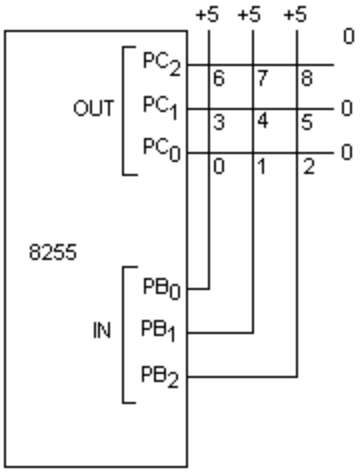


Here, when the counter counts 000, 0^{th} line is set high and the line RL1 becomes active. So, the signal is sent to the debounce control. Then, it waits for 10 ms. By that time, the rest of the lines would have been scanned and the counter would be back to 000. If the

key is still pressed, it is taken as a valid input and the key is identified by the keyboard debounce control.

Keyboard interfacing – Software approach

Here, we use 8255 to interface the keyboard. Consider a 3x3 matrix keyboard as follows, connected to 8255.



PORT C FOR ROWS - XXXXXPC$_2$PC$_1$PC$_0$ (set in O/P mode)

PORT B FOR COLUMNS - XXXXXPB$_2$PB$_1$PB$_0$ (set in I/P mode)

The subroutine for the matrix keyboard interfacing would be like:

REFER APPENDIX A FOR FLOWCHART AND APPENDIX B FOR CODE

I/P port: PB0, PB1, PB2 (5V)
O/P port: PC0, PC1, PC2 (GND)

Initially, PB0, PB1, PB2 = 111
When a key is pressed, that line is grounded. ie, if key press is at pos 0, PB0=0
Algorithm

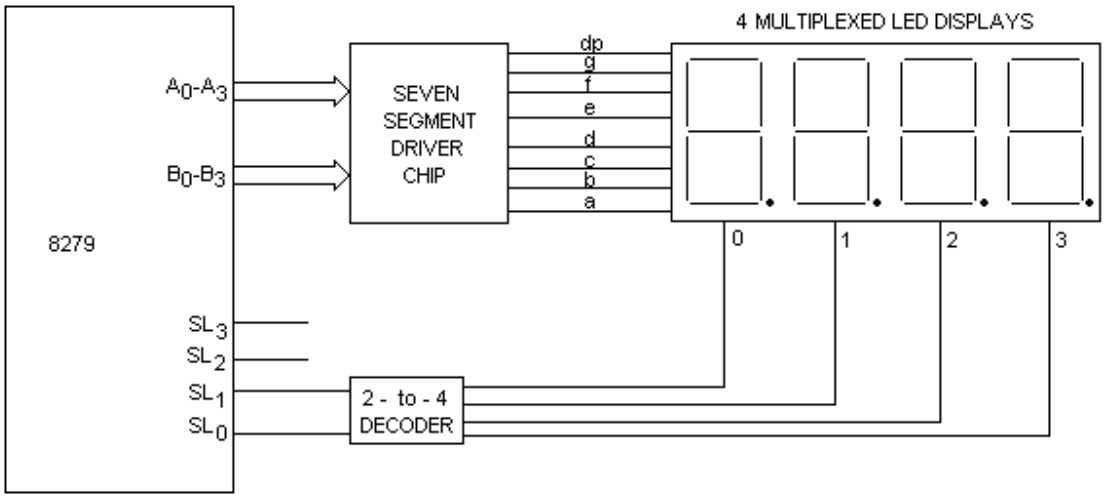
Subroutine

1. Initialize A=0, Column counter=0, Row counter=0.
2. After reading keypress, call delay (debounce). Set scan counter=0, column counter=top value, row counter=top value.
3. Ground one row (top to bottom). Check key status. If key is pressed, move scan code to accumulator. Otherwise, increment scan counter, decrement column counter and decrement row counter. Continue steps 3 and 4.

Exit condition

- If any bit of Port B is 0, exit and move the scan code to accumulator.
- If the row or column counter is 0, exit the loop.

7-segment display interfacing – Hardware approach



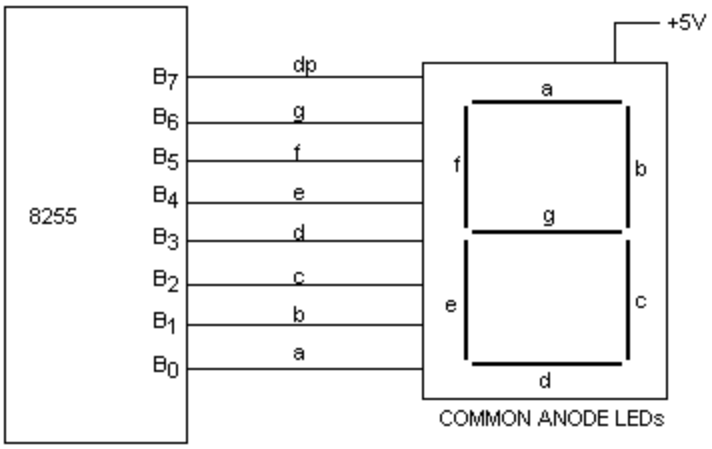
The scan lines form the main part of the interfacing. They are the ones that refresh the display.

Scan counter value	Display refreshed
0000	1
0001	2
0010	3
0011	4
0100	5
0101	6
0110	7
0111	8
1000	9
1001	10
1010	11
1011	12
1100	13
1101	14
1110	15
1111	16

So, we can see that a maximum of 16 displays can be interfaced using 8279.
Consider the 4-display system as given above. Here, it uses a 2-to-4 decoder to realize the display interface. It takes 1.5 ms to count from 0 to 3 for the counter. So, it is designed such that this time is equal to the time for which the LEDs in the display can glow.
Now, to display 32, the value is moved from the accumulator to 8279. 8279 converts it to 7-segment format.
3 = a, b, c, g, d
2 = a, b, e, g, d

Since common anode LED is used, they are activated by giving a low input. When the scan counter is 0, the 8279 traces this and sends the code corresponding to 3 to the activated display. This glows for 1.5 ms. Then, the scan counter increments and moves to the next value (1), which activates the next display. Then, the code corresponding to 2 is sent to the active display. These are controlled by the driver chip.
Refreshing: The continuous charging of the LEDs (one by one) by the scan lines.

7-segment display interfacing – Software approach



Algorithm

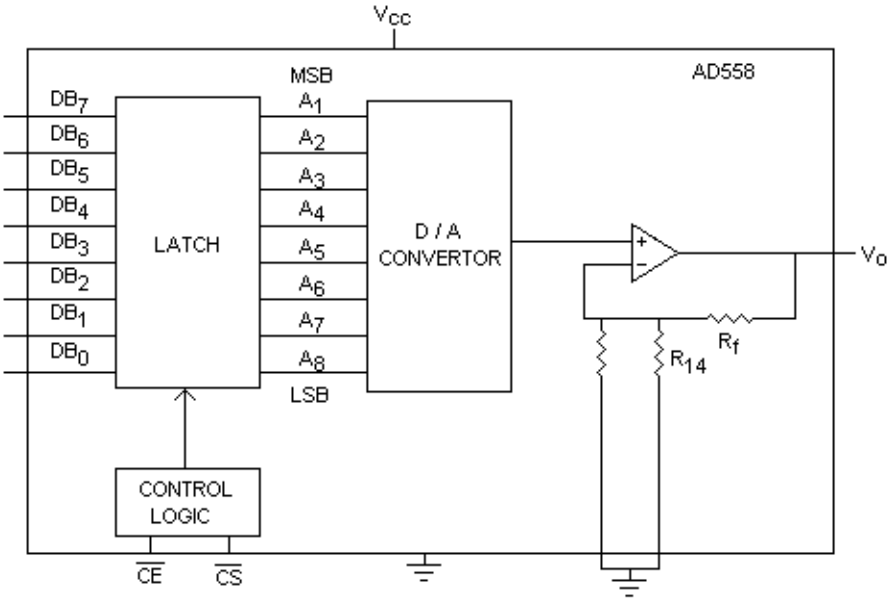
1. Start
2. Define CODE TABLE for the 7-segment display in continuous memory locations.
3. Set HL pair to base address of the code table.
4. Move number to be displayed to the accumulator.
5. Add the accumulator value with the L reg.
6. Move the HEX code from the code table to the accumulator using HL pair.
7. Output accumulator content to the output port.
8. Return.

Here itself, we can see the main difference between the hardware and software approaches. In hardware approach, we just need to give the number to be displayed to 8279 and it will take care of the rest (conversion to 7-segment code & display). We can have upto 16 displays like this. But in software approach, we have to define the code table for the numbers (in terms of 0s and 1s) ourselves and send it to the O/P port. Also, we can have only upto a maximum of 3 displays.
To display 3: 3 = a, b, c, g, d.
So, a=b=c=g=d=0.
So, the reqd code is 10110000 = B0H
So, to display 3 using port B, the st. are: MVI A, B0H + OUT 01H

REFER APPENDIX C FOR CODE

DAC INTERFACING (AD558)

DACs convert digital info to analog signals. They form an integral part of communication along with their counterparts, the ADCs.
AD558 is a monolithic DAC, having a latch, control logic, a DAC and an opamp.
The AD558 is interfaced as:



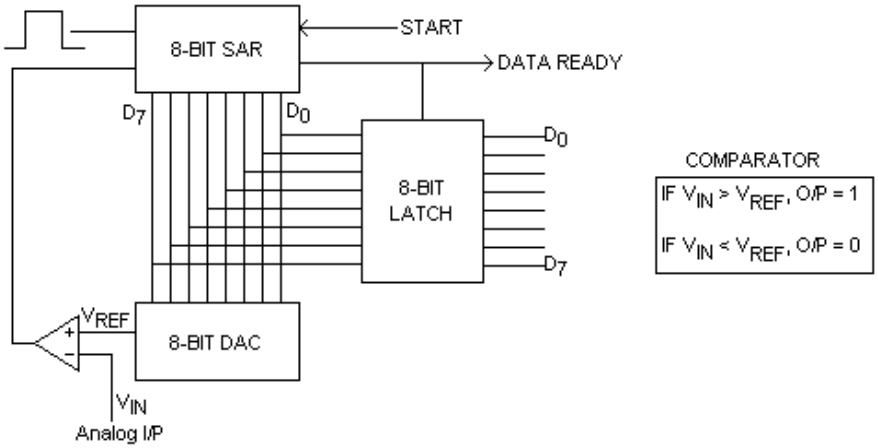
Now, the output will be such that
*Vo=(Rf/R14)*Vref(A1/2+A2/4+A3/8+A4/16+A5/32+A6/64+A7/128+A8/256)*
where Vref=5V
DACs are mainly used in communication (MODEM) and audio (speakers).
In winamp, 128kbps means 128 kbits/sample and 44.1 KHz means 44100 samples/sec.
CE: Chip enable pin, used to enable the chip (not selection). It is connected to IO/M pin of 8085.
CS: Chip select pin, used to select the chip for conversions. It can be selected using any one of the A0 to A7 lines of 8085.
Now, the opamp is in non-inverting configuration. Then, gain=1+Rf/Ri4. But here, the constant factor 1 is omitted. So, the gain or scaling factor=Rf/R14. Vref is kept at 5V to rep. logic 1. A1 to A8 is the input to the DAC, which in turn comes from the DB0 to DB7 input to the latch. This input is given by 8085 using its data bus. But when bits come out of the latch, their order is reversed. ie,

Input to the latch	Corresponding output
DB0	A8=DB0
DB1	A7=DB1
DB2	A6=DB2
DB3	A5=DB3
DB4	A4=DB4
DB5	A3=DB5
DB6	A2=DB6
DB7	A1=DB7

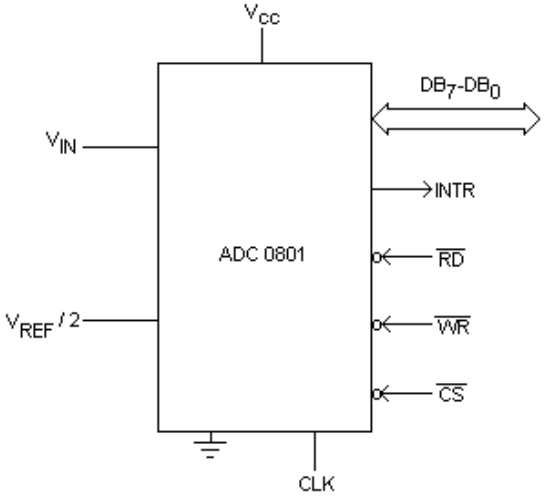
This is since high priority bit (MSB) gets higher weight when converted to analog signal by DAC.
The output of the DAC is:
Vo'=Vref(A1/2+A2/4+A3/8+A4/16+A5/32+A6/64+A7/128+A8/256)
This is given as input to the opamp. Then the output of the opamp, ie, the final output is:
*Vo=(Rf/R14)*Vo'*
Eg: convert 10110010 to analog.
Here, we use OUT instruction. Load 10110010 to the accumulator. Then, using OUT instruction, send it to AD558.
Vo'=5*(1/2+1/8+1/16+1/128)
=3.47V
Now, Rf/R14 is designed to be 3. So, Vo=3*3.47~10V
This is recd as the O/P.
So, the chain is: 8085~AD558~receiver

ADC INTERFACING (ADC0801)

ADC0801 is an 8-bit ADC. It is a Successive Approximation type ADC. A successive approximation type ADC will be like:



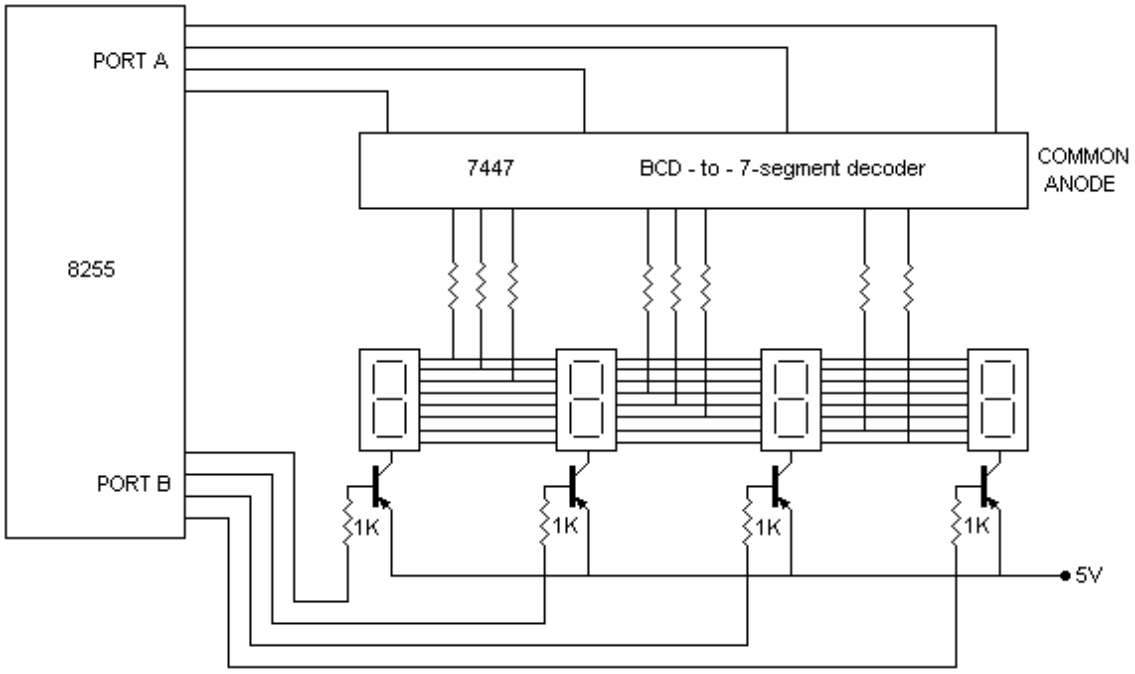
The pinout of 0801 is:



Since SAR is used, 0801 has some additional pins to interface with 8085.
WR: Similar to START signal of SAR.
RD: To read the O/P.
INTR: Interrupt for the MP. Priority is set by the interrupt controller (8259).

When the data is ready, the MP is interrupted. The MP responds by setting low CS and WR signal. This resets the SAR. Then, the ADC performs the conversion using the SAR. Then again, INTR is sent. When the second interrupt is received, the MP responds by setting RD to low to read the O/P from the latch. 8 clock pulses are needed for full conversion of the data.

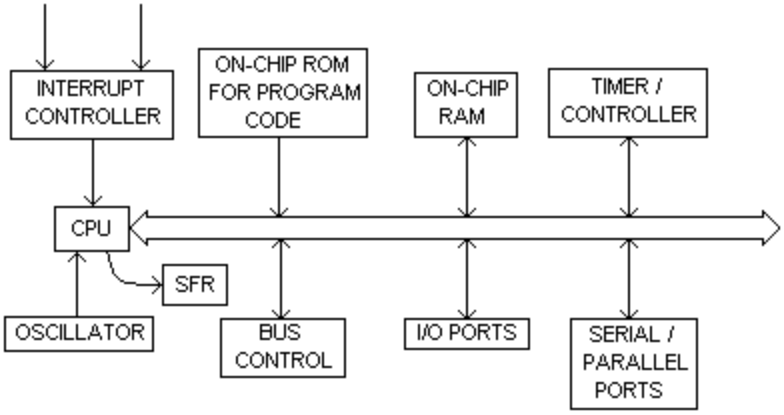
INTERFACING 7-SEGMENT DISPLAY ARRAY



The principle behind this circuit is that human eye can maintain an image for a maximum of 125 ms. Here, each display is activated by applying a high input to the corresponding transistor's base. Also, the circuit uses a 7-segment display driver – 7447, that is a BCD to 7-segment decoder. 4 lines of any suitable port (here, Port A), can be used to transfer the data. 7447 will convert the data in the 4 lines to the corresponding 7-segment code and gives it to the activated display. A maximum of 8 displays can be interfaced.
Now, this interfacing is also a software-based approach. So, a suitable program has to be coded to give the data sequentially, and simultaneously activate the reqd display. Then, the displays should be activated sequentially, one at a time, and the cycle should repeat, each cycle taking less than 125 ms, so that the human eye gives the impression of a continuous illumination.

MICROCONTROLLERS

A microcontroller is a special purpose device (ie, used only for a specific purpose). So, it can't be used as a general-purpose device like microprocessors.
Eg: processors of i-pods, washing machines, etc.



Here, all the components of the chipset are fabricated on to a single chip. This is the main difference from ordinary processors since they have a different chipset for each operation an so, can perform more generalized operations.

Operations performed by the CPU

Arithmetic	Logical	Rotate
ADD	AND	ROTATE RIGHT
SUBTRACT	OR	
INCREMENT	NOT	ROTATE LEFT
DECREMENT	XOR	

The microcontroller also supports all these operations.
Examples of microcontrollers:

- PIC
- Intel 8051

They use different instruction sets.

SFR

It stands for Special Function Register. It is similar to flag register of MP.
The usual SFR flags are:

- Sign
- Carry
- Parity
- Zero

There may be other flags also.

On chip ROM: The hard disk of the MC.
On-chip RAM: The memory of the MC.

Flashing a program

The process of inputting a program to the MC is called flashing the program. Usually, the basic languages used to write the programs are: C, C++, BASIC and JAVA. Also, for compiling, we have to use special compilers. General compilers aren't supported. After compilation, we get HEX CODE or BIN CODE. The steps in flashing a program are:

1. Write the program using the IDE of MC.
2. Compile it.
3. Generate HEX CODE.
4. Flash the HEX CODE to the MC using Parallel Port or USB.

Embedded System OS for SOCs (System On Chip = MC)

- μCOS - free ware
- PALM OS - palm tops
- WINDOWS CE
- Vx WORKS \ Used by Nortel company
- QNX / for telephone switches
- IOS - Internet OS (used by CISCO)
- LINUX
- TINY OS
- SYMBIAN

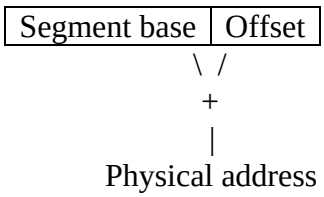
We can also add extra features.

REAL MODE MEMORY ADDRESSING

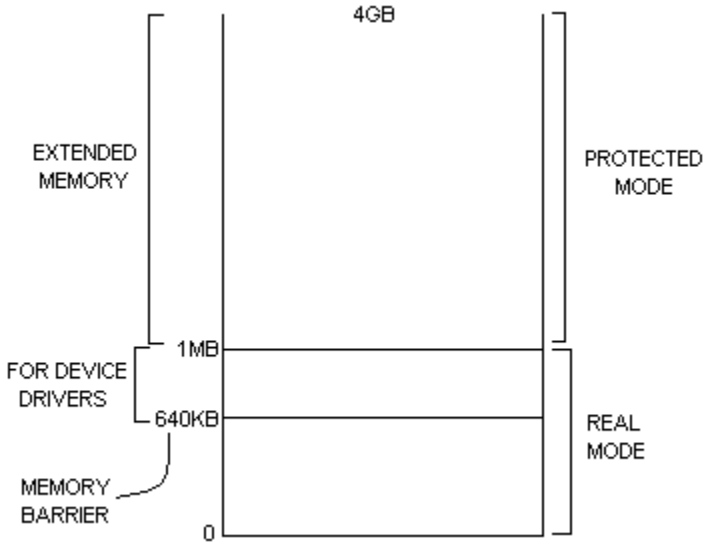
This mode was the only one available in 8086. It can address upto a max of 1MB using 20 address lines. Since the accessible memory is limited, it can't support multitasking.
Eg: C code.

```
if (c==0)
Let the logical address of that instruction be ADDR.
```

Then the equivalent assembly code instruction is: JZ ADDR. So, the logical address is the physical address itself.



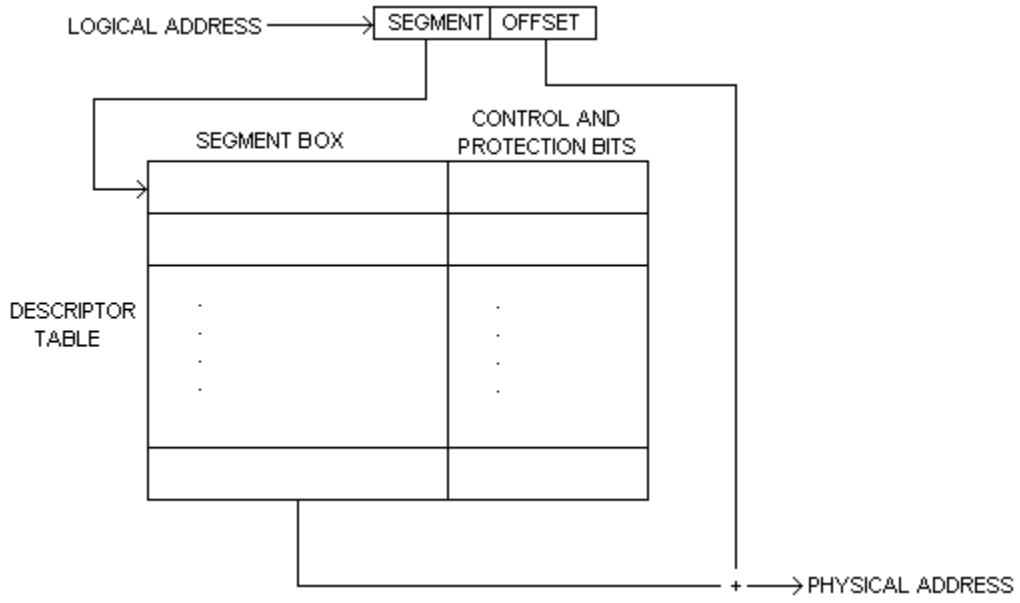
MEMORY ORGANIZATION OF x86 PROCESSORS



The main feature of x86 processors is that it is an open architecture. Initially, when 8088 was introduced, DOS was the operating system used. But after the arrival of UNIX flavours, the whole situation changed. UNIX versions were first introduced by Apple. The main advantage of UNIX was that it supported multitasking, rather than single-task capability of DOS. To support multitasking, UNIX needed a much larger volume of memory, which couldn't be supplied by the real mode operating 8088 (only a max of 1MB). So, later, 80286 was introduced, with an additional protected mode to address above 1MB (uses 24 address lines). To access protected memory, we use special drivers like *himem.sys* in DOS. In real mode, segment and offset values directly give the physical address. But in protected mode, the logical address has to be converted to physical address using the driver *himem.sys*.

PROTECTED MODE MEMORY ADDRESSING

This mode was first introduced in 80286. 80286 can address upto 16MB of memory using 24 address lines. For backward compatibility, 80286 operated in 2 different modes – real and protected. Protected mode supports multitasking. In 80386, protected mode even supports PAGING. The processors from 80286 also have an MMU.



MMU: The main advantage of x86 architecture. It stands for Memory Management Unit. All memory operations are controlled and carried out by MMU.

In protected mode, the logical address provided by the assembly language is sent to the MMU of 80286. The MMU divides it into segment and offset. The segment points to an entry in the descriptor table. The entry in the descriptor table (segment base) is added with the offset to give the physical address. The size of segment can vary from one MP to another. Presently, in most MPs, one segment is of 64kB size. Protected memory is also called extended memory.

Why the name protected?

The MMU calculates the max limit of a segment. If the segment-offset sum exceeds the limit, since the MMU can't access it within the segment, it reports an error and the location is inaccessible to the MP. So, each segment is protected from the others (memory violation error).

Eg: Let segment base be 0000. So, end of segment=0063.

Let logical address be 00064. So, physical address = 0000+64 = 0064.

This lies outside the segment limit and the MMU reports an error.

VIRTUAL MODE

In 80286, when the MP was once switched to protected mode from virtual mode, it was virtually impossible to return to the real mode without resetting the MP. To overcome this, 80386 was introduced (32 bit). In this, a new mode called virtual mode was introduced. Using this, we can switch between real and protected mode quite easily, since the virtual mode is acting as an intermediate state to which both real and protected modes can easily switch to. But virtual mode isn't a new operating environment.

80286

FUNCTIONAL BLOCK DIAGRAM: REFER BREY P.690

Internal structure

BUS UNIT

Address latch: interfaces the address lines.

Prefetcher: fetches the instructions before hand.

Processor extension interface: In 80286, we have a Mathematical Co-processor. It is interfaced with 80286 using this interface.

PEREQ: Processor extension Request.

PEACK: Processor Extension Acknowledgement.

The co-processor is mainly used to perform arithmetic operations for the main MP.

Bus control: to control the buses.

Data transceivers: interface between the external data bus and the internal bus.

6-byte prefetch queue: stores the fetched instructions (not decoded). It is similar to instruction queue of 8086.

INSTRUCTION UNIT

Instruction decoder: decodes the instruction.

3-decoded instruction queue: stores upto 3 decoded instructions.

EXECUTION UNIT

Contains ALU, registers and control unit. ALU performs the reqd operations on the operands.

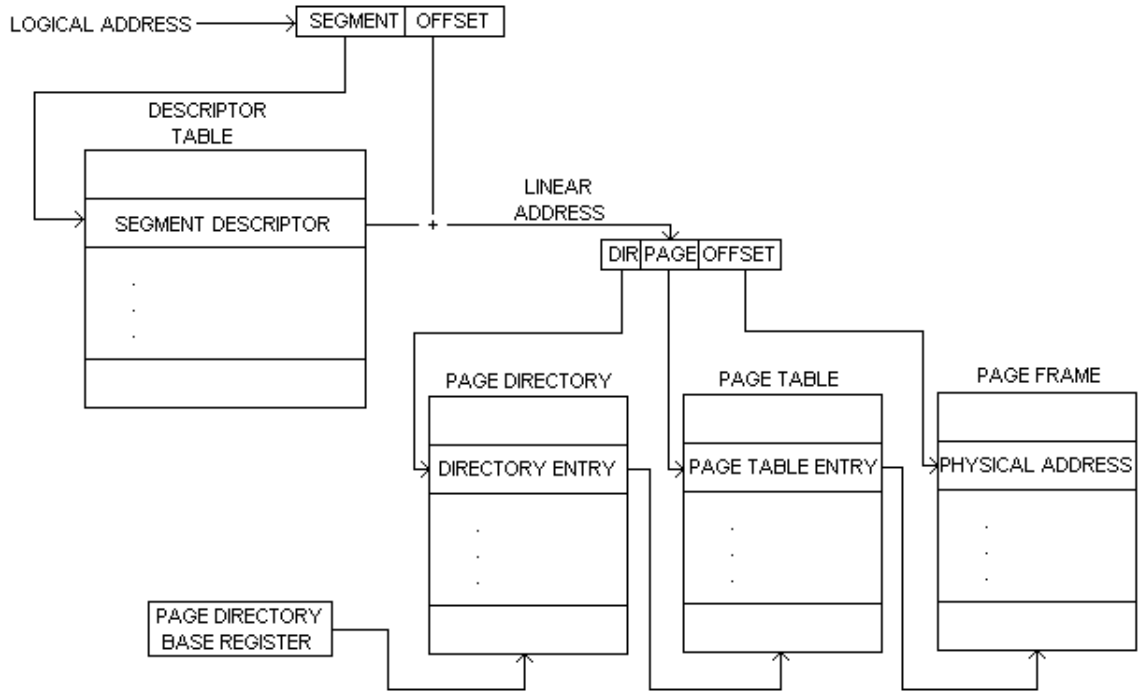
ADDRESS UNIT

The main part of the address unit is the offset adder. It adds the offset value to the address. From the offset adder, it is split into 2 modes – real and protected. For real mode, this address itself is taken as the physical address. But for protected mode, the descriptor tables are also taken into consideration while finding the physical address.

Processing

If a logical address is passed from the instruction queue to the control unit, the control unit splits up the logical address to segment and offset. This is stored in the registers (similar to 8086). There are 2 paths to the offset adder (in the address unit) from the reg. One is for segment and other is for offset. If it is real mode, the offset adder functions as an adder and performs a bit shift of segment and passes it to the physical address adder. If it is protected mode, no shift is done, but passed directly to segment limit checker. After checking the segment limit, if it is found within the limit, the segment base from the descriptor table and the segment offset is added in the physical address adder, which generates the physical address. The result from the physical address adder is passed to the address latch (drivers) from which it gets transferred to the address lines.

80386 MEMORY ADDRESSING – PAGING

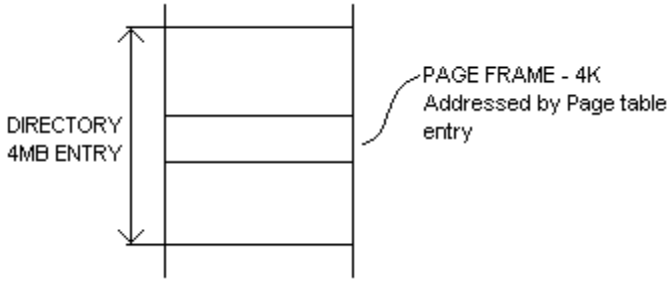


A page frame is a 4KB unit of contiguous addresses of physical memory. All pages are fixed in size. Each directory entry in page directory addresses a 4MB section of the main memory. Each page table entry addresses a 4KB section of the main memory. Offset specifies the byte in the page. Paging unit is controlled by control reg.

	CR4 (Pentium & above)
Page directory Base address	CR3
Most recent page faulting linear address	CR2
Reserved	CR1
	CR0

MSB of CR0 determines paging. If it is 0, paging is disabled (linear address=physical address). If it is 1, paging and virtual mode is enabled.

Memory map



Paging is controlled by the control register. CR4 is used in Pentium and higher MPs. CR3 is page directory base reg. and CR2 is page fault information reg. (if the adder crosses the page limit). CR1 is a reserved reg.

Program invisible registers

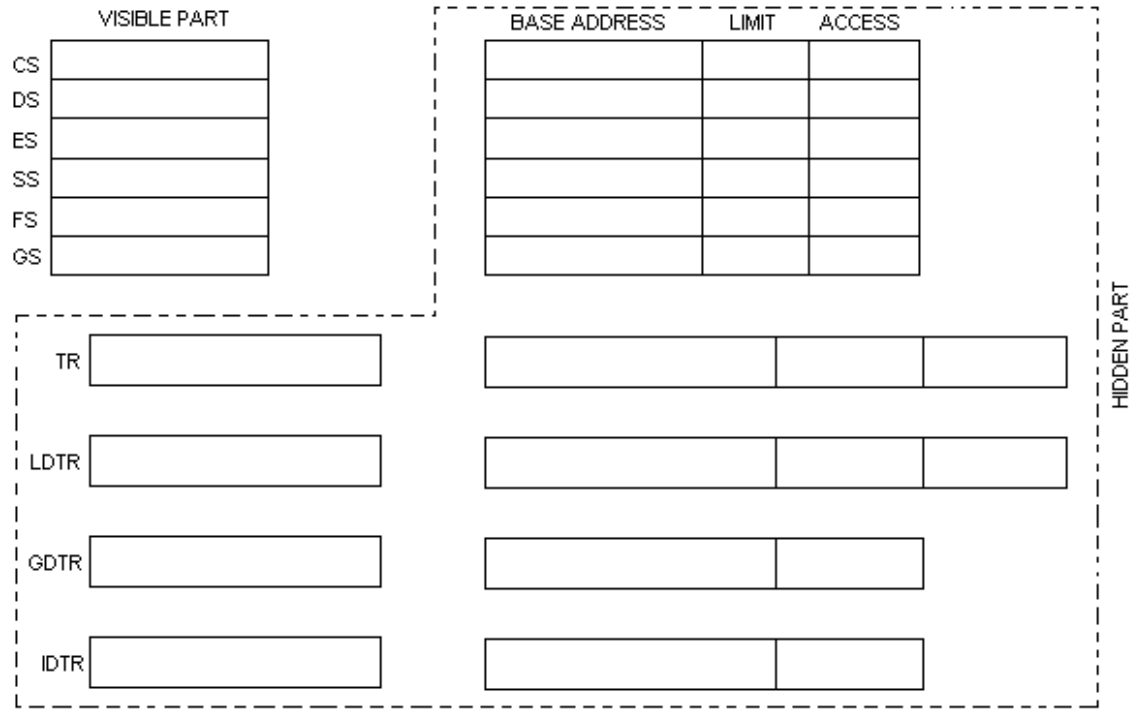
LOGICAL ADDRESS

SELECTOR	OFFSET

SELECTOR

DESCRIPTOR INDEX (12 bits)	TI (I bit)	RPL (2 bits)

RPL – Requested Privilege Level. If it is 00, it is highest and if 11, it is lowest.
TI – Table index. If it is 0, GDT (global descriptor table) is enabled and if 1, LDT (Local descriptor table) is enabled.
Selector – Selects a descriptor from the descriptor table.
Descriptor – Contains info about the reg. base address, limit and access rights.



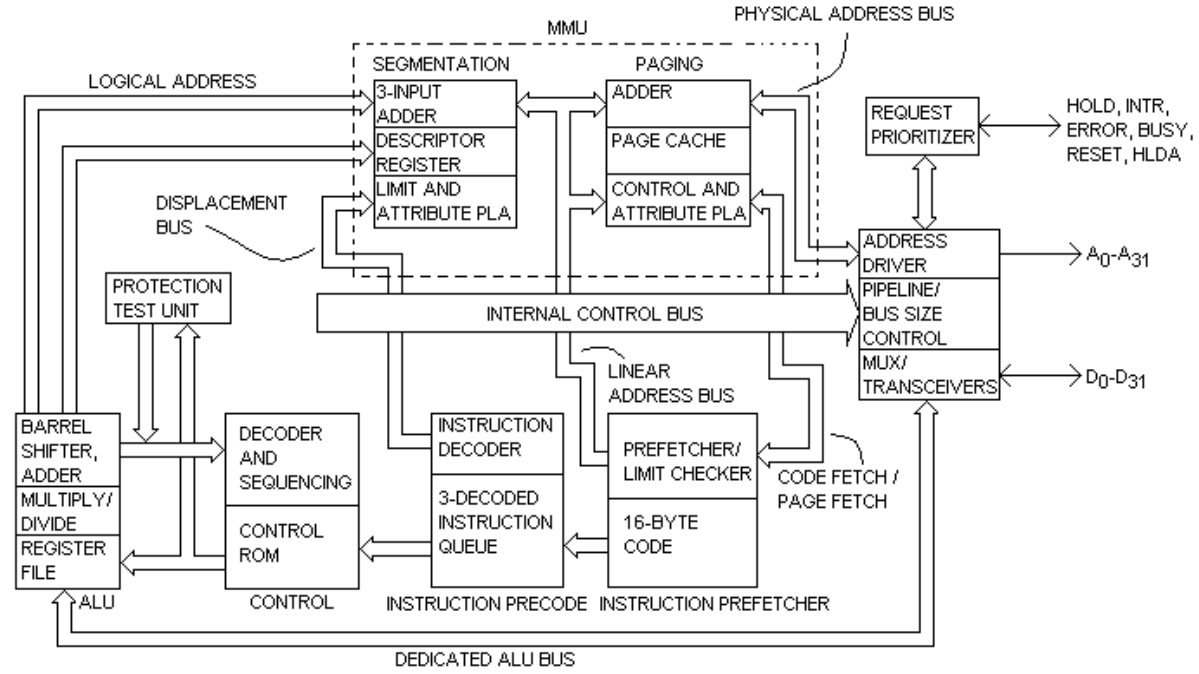
This process is used in segmentation. The linear address is generated as in the case before, using segment and offset values.
TR – Task reg. It holds the descriptor table info. GDT is related to OS operations, while LDT is related to MP operations. Here, we can see that the processor uses 2 different descriptor tables – GDT & LDT. This is since the processor supports multi-tasking. So, for OS-related processes, GDT is used, whose details are stored in GDTR and for user-defined processes, LDT is used, whose details are stored in LDTR.
Now, TR holds the info of the descriptor table currently in use. If it is GDT, TR stores the info of GDTR and if it is LDT, TR stores the info of LDTR. Consider the below process management.

TIME 1	TIME 2	TIME 3	TIME 4
GDT	LDT	GDT	LDT
OS TR=0	USER TR=1	OS TR=0	USER TR=1

Every time, MP fetches the base descriptor table from the TR. The TR info switches according to the currently executing process. The OS is fully responsible for the shifting from one process to another. For this, it uses different scheduling techniques like Round Robin, FIFO, LRU, LFU, etc. The MP has no control over this. It just processes whatever is sent to it.

IDTR – Interrupt Descriptor Table Reg. It is similar to the interrupt vector table of the earlier MPs.

80386 functional block diagram



Firstly, the Prefetcher fetches a group of instructions prior to execution. Then, they are queued up in the instruction queue. From the 16-byte queue, the instruction is forwarded to the decoded instruction queue. From that, the instruction is forwarded to the control ROM. The control ROM forwards it to the Protection Test Unit. There, it checks the segment limit. If the limit isn't crossed, it is forwarded to the register file. Then it is taken to the ALU for processing. The ALU has 2 blocks:

Addition and shifting – Barrel shifter and adder that can shift multiple bits at a time (single clock pulse) and add operands.

Multiply and divide – To perform multiplication and division (for CISC based instructions, which have only a single instruction for multiplication and division).

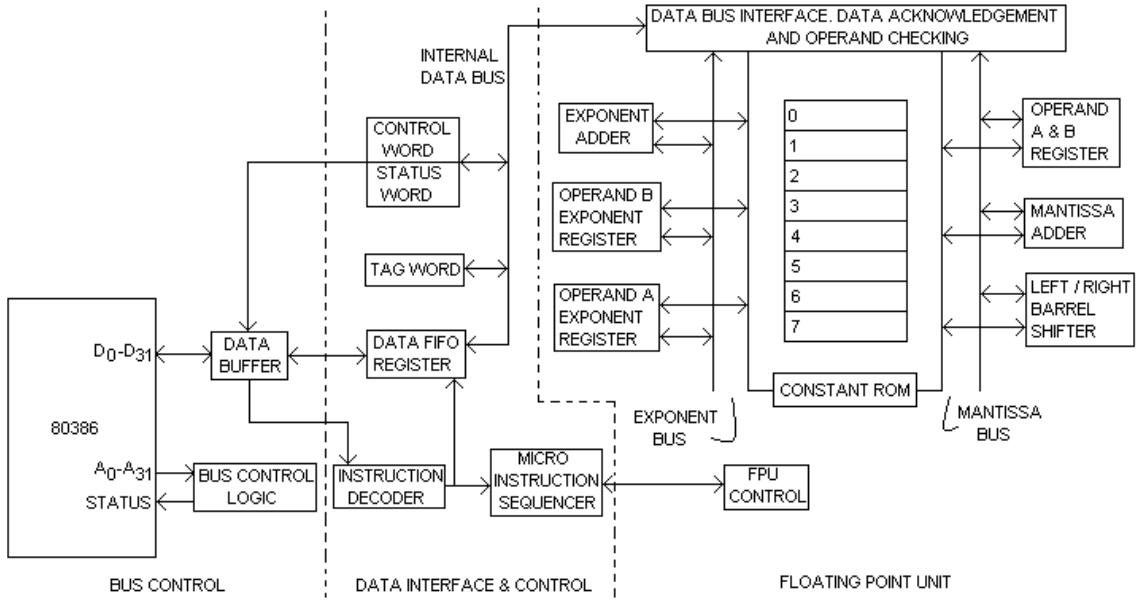
Then, the MMU comes into action. If any operands are to be fetched from memory, the MMU will perform it using its 2 parts. The segmentation block and paging block will

convert the linear address to physical address and fetch the data from those locations using address and data buses. The segmentation block contains the descriptor reg like TR, LDTR, GDTR, etc. (program invisible reg). It adds the segment and offset values and passes it to the paging block. The paging block will check CR0 (to see if paging is disabled. Then, linear address will be the physical address itself) and CR2 (page faults), etc. After all this, the physical address is generated and it is passed to the address driver to pass to the bus for fetching.
The main feature of the address driver is that it stores and sends the address given to it by the paging block. Another main block in the 80386 is the transceiver that controls the external data bus.

COPROCESSOR

Operations performed by the coprocessor may be floating point arithmetic, graphics, signal processing, etc. MATH coprocessors are used for floating point calculations. 8087 was the first coprocessor. Later, Intel introduced 80287, 80387, 80487, etc. This architecture is commonly known as x87 architecture.
Coprocessors use a separate set of instructions, all beginning with F.
Eg: FADD, FMUL, FSUB, etc.

80387 INTERFACING AND INTERNAL ARCHITECTURE



Need of coprocessors

The most advanced MP had only a 32-bit data bus. But when it comes to floating point arithmetic, the space needed is very large. So, coprocessors were introduced, which had an 80-bit register file

BUS CONTROL
Contains the data buffer and the control logic for addition and operational control.
DATA INTERFACE AND CONTROL

Similar to flag reg, we have status word that determines the status of the floating point unit (FPU)
There is also a control word similar to that of 8255 that can be used to program the 387. The tagword holds the status of the registers (stack). This unit also houses data FIFO reg, instruction decoder and a microinstruction sequencer.
FPU
The main part of this is the register file. It is a stack arrangement of the reg. On either side we have registers for exponent and mantissa. For exponent, we have 2 reg. and for mantissa, we have 1 reg.

Let us take an example instruction: FADD 5.2, 4.5
The different steps in its execution are:

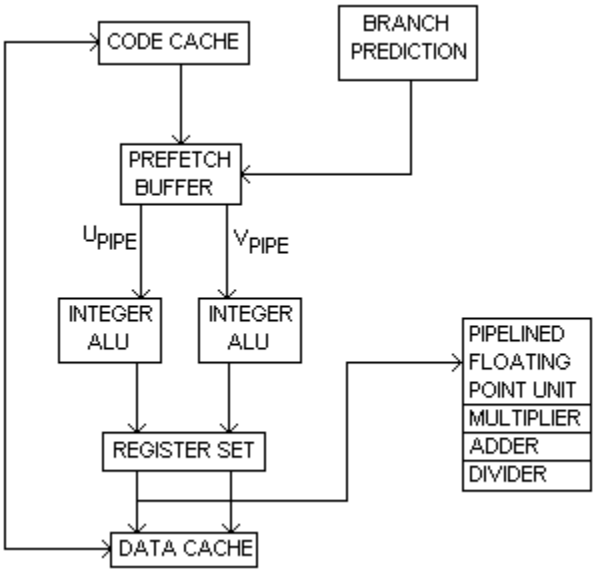
- Instruction (opcode and operands) forwarded to data buffer.
- Operands stored in data FIFO reg.
- Microinstruction sequencer puts the microinstructions in the reqd order.
- By checking the tagword info, we know which reg. is free.
- Operands stored to reg.
- The info. is split up and stored into mantissa and exp. reg. after getting the control signal from FPU.
- Now, the exp. adder and mantissa adder adds the info.
- Result is stored back to reg.
- Then, it is passed back to data buffer, through data FIFO reg.
- From data buffer, the result is passed on to 80386 MP.

80387 has an instruction set of almost 80 instructions. It also has some additional features like instructions for FSQRT (sq. root), SQR (square), SINE, etc. To perform these additional functions, we have the constant ROM. It stores the constants reqd for these special mathematical operations.
To find sq. root, it uses NEWTON-RAFFES method. In gcc type compilers, -lm means LINK TO MATH COPROCESSOR.

FEATURES OF PENTIUM PROCESSORS

- Based on RISC architecture. Upto 80486, Intel used CISC.
- Super scalar execution. It means that more than one instruction can be executed in a single clock pulse. This feature was introduced in Pentium.
- Separate code and data caches to hold instructions and data separately.
- On-chip FPU, eliminating the need for a MATH coprocessor.
- Floating point exception support. ie, instead of executing any errors, it sends back an error msg to the MP.
    1. Divide by zero.
    2. Overflow.
    3. Underflow.
    4. Denormal operand (invalid operand eg: MOV 03,A).
    5. Invalid operation.

Pentium CPU architecture



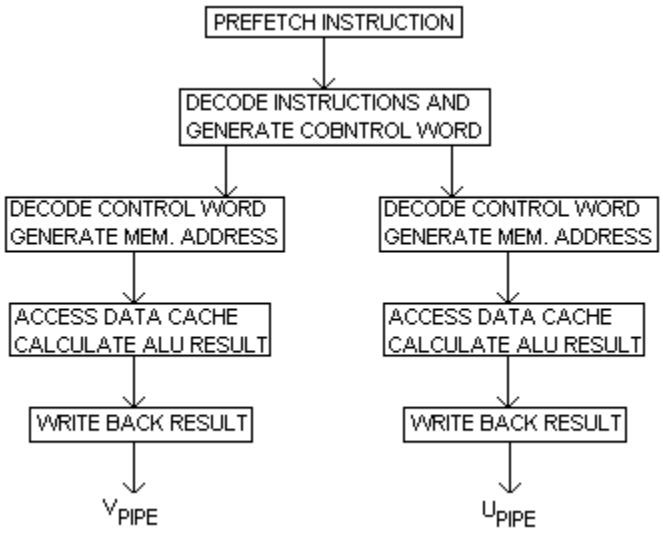
Pentium processors basically consist of the following units (similar to 80386).

- Address unit.
- Execution unit.
- MMU.
- Bus control.

The main feature of the execution unit that makes it different from 80386 is pipelining. It is used for multiple instruction execution. It makes use of the data cache and the code cache for this as the instructions and the data are now completely separated and independent of each other. From the prefetch buffer, by pipelining, the instructions are routed to one of the 2 available ALUs. From the ALUs, the results are put into reg. The internal FPU reduces time delay.

Also, in other MPs, only after executing JMP instruction, we know that there is a memory fetch and jump. But in Pentium series, there is a BRANCH PREDICTION unit that predicts the JMP instruction from the prefetch buffer. This avoids delay in converting the logical address to physical address in the MMU. The architecture in which there are multiple ALUs is called Super Scalar Architecture.

Super Scalar Organization



Here, there are basically 5 stages.

- Prefetch instruction: Pentium is not fully a RISC based MP. So, it can also support The CISC based instructions of 80386. So, we have 2 decoding circuits.
- Instruction decoder: The first one is exclusively for CISC instructions.
- Second stage decoder: The second one is a general decoder circuit. It fetches the info. from the memory after decoding the instruction in the 1st decoding unit.
- Access data from the data cache and perform the calculations.
- Write back the result.

For example, to multiply A and B, in CISC processors, the fetch operations are all implicitly defined. But in RISC processor, all the operations have to be defined to the most basic level.

The biggest disadvantage of pipelining is register-dependent instructions. If we have 2 instructions
ADD B
SUB C
If it is pipelined, we won't get the reqd result since both are being executed based on the same value of the accumulator. But it isn't supposed to happen. The actual execution is supposed to be (A+B)-C.
But in Pentium processors, this won't happen. Pentium processors are usually multi-tasking processors. So, one time slot is reserved for each operation. So, first addition is performed, then another task takes over and only after that the subtraction takes place. So, the instructions will never overlap.

CISC & RISC

CISC	RISC
Emphasis on hardware	Emphasis on software
Includes multi-clock complex instructions	Single-clock, reduced instructions only
Memory-to-memory operations incorporated in the instructions	Memory-to-memory operations are independent instructions
Transistors are used for storing complex instructions	Transistors are used for memory registers

FEATURES OF PENTIUM 3

- L1 cache
- L2 cache
- MMX
  1. Multimedia Extension
  2. Multiple Math Extension
  3. Matrix Math Extension (used in MATLAB)
- Slot 1 (addition like add-on cards) and Socket 370 (flat bed addition).
- SSE: Streaming SIMD support. Original name: KNI (KATNAI instruction set), similar to image processing instructions.
- Dual independent Bus architecture, similar to 2-lane traffic. Here, there are 2 different data buses for to and fro data flow in the MP.