Microproceesor and Peripheral Devices

Unit 1

Architecture of microprocessor :8085 is pronounced as "eighty-eighty-five" microprocessor. It is an 8-bit microprocessor designed by Intel in 1977 using NMOS technology.

It has the following configuration −

- 8-bit data bus
- 16-bit address bus, which can address upto 64KB
- A 16-bit program counter
- A 16-bit stack pointer
- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate at 3.2 MHZ single phase clock

It is used in washing machines, microwave ovens, mobile phones, etc.

8085 Microprocessor – Functional Units

8085 consists of the following functional units −

Accumulator

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.

Arithmetic and logic unit

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

General purpose register

There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data.

These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

Stack pointer

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

Temporary register

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

Flag register

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops −

- Sign (S)
- Zero (Z)
- Auxiliary Carry (AC)
- Parity (P)
- Carry (C)

Its bit position is shown in the following table −

D7	D6	D5	D4	D3	D2	D1	D0
S	Z		AC		P		CY

Instruction register and decoder

It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

Timing and control unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits −

- Control Signals: READY, RD', WR', ALE
- Status Signals: S0, S1, IO/M'
- DMA Signals: HOLD, HLDA
- RESET Signals: RESET IN, RESET OUT

Interrupt control

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the

control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

Serial Input/output control

It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

Address buffer and address-data buffer

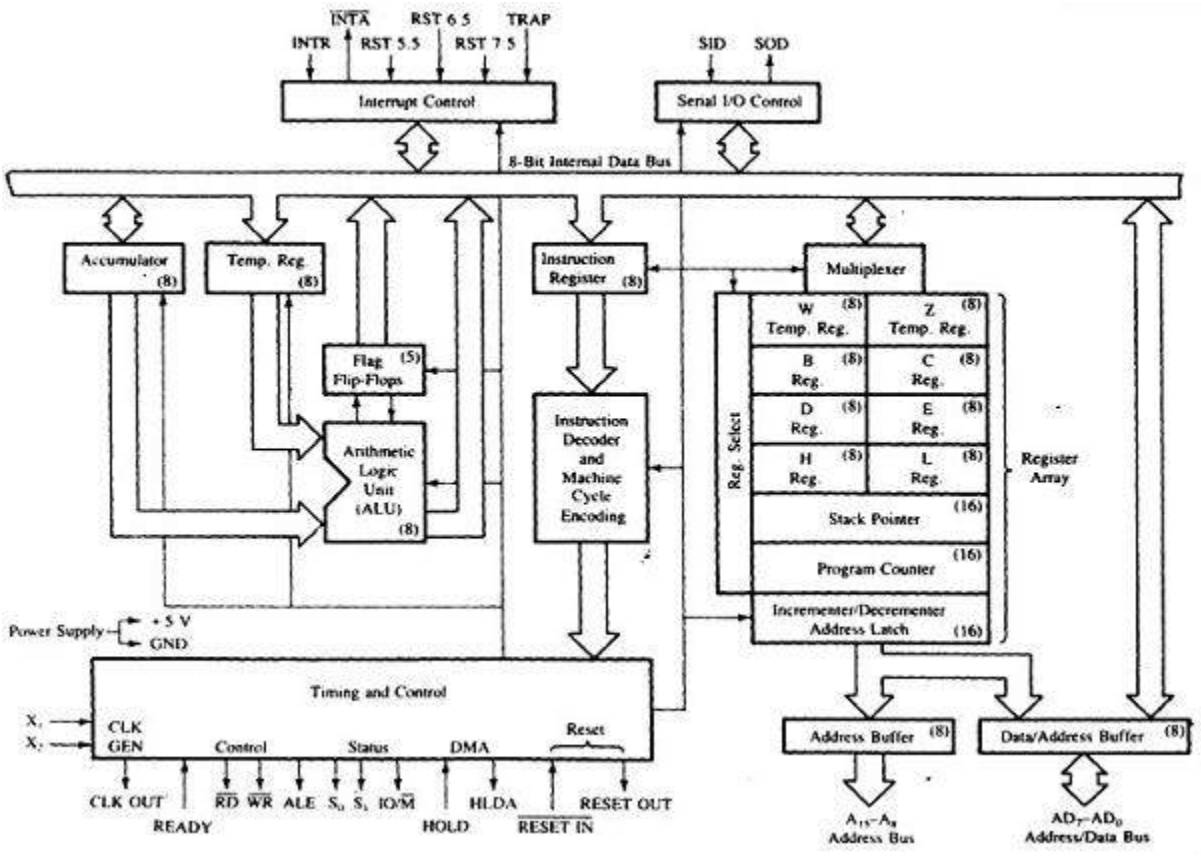
The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.

Address bus and data bus

Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address I/O devices.

8085 Architecture

We have tried to depict the architecture of 8085 with this following image −

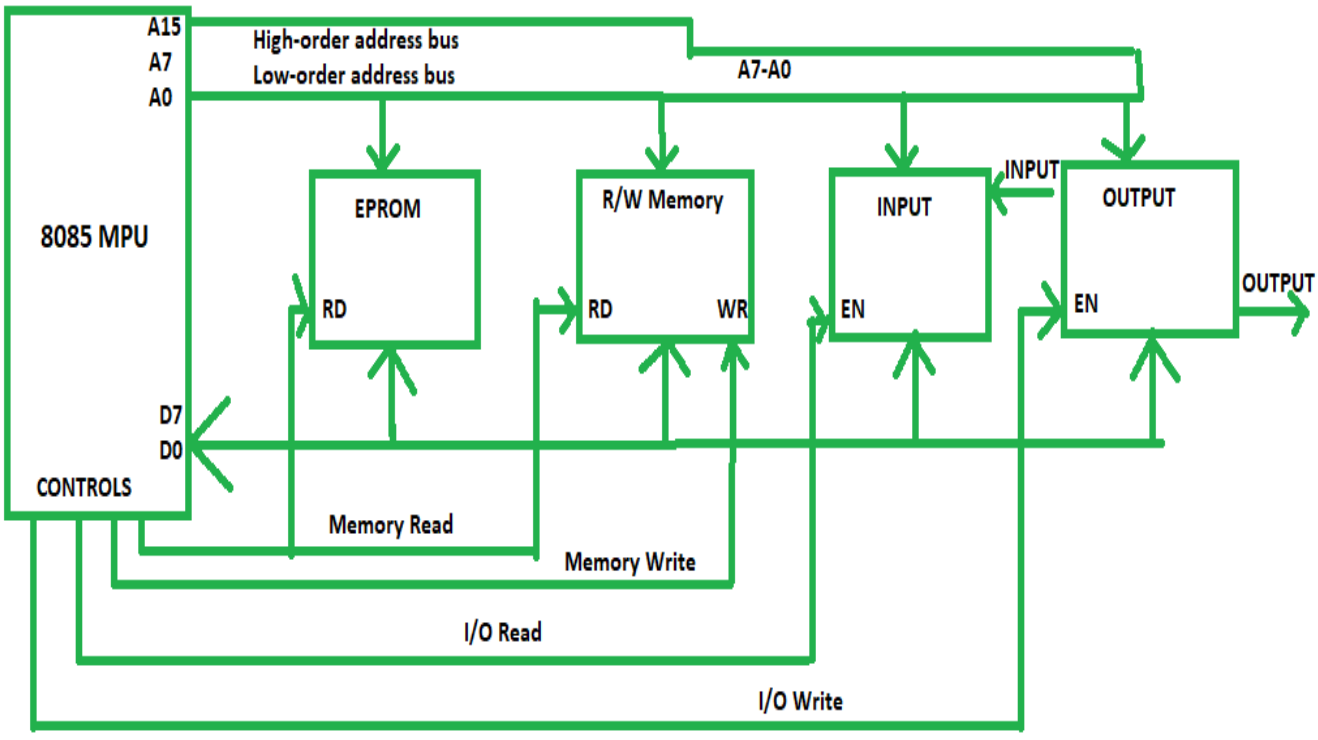


Computer Organization | Microcomputer system

The 8085 microprocessor is an example of Microcomputer System. A microprocessor system contains : two types of memory that are EPROM and R/WM, Input and Output devices and the buses that are used to link all the peripherals (memory and I/Os) to the MPU.
In 8085, we 16 address lines ranging from A0 to A15 that are used to address memory. The lower order **address bus A0-A7** is used in the identification of the input and output devices. This microcomputer system has 8 **data lines D0-D7** which are bidirectional and common to all the devices.
It generates four **control signals**: Memory Read, Memory Write, I/O Read and I/O Write and they are connected to different peripheral devices. The MPU communicates with only one peripheral at a time by enabling that peripheral through its control signal.
For example, sending a data to the output device, the MPU places the device address (or output port number) on the address bus, data on the data bus and enables the output device by using its control signal I/O Write. After that the output device display the result.

The other peripheral that are not enabled remain in a high impedance state called Tri-state. The bus drivers increases the current driving capacity of the buses, the decoder decodes the address to identify the output port, and the latch holds data output for display. These devices are called Interfacing devices. This Interfacing devices are semiconductor chips that are needed to connect peripherals to the bus system.

The block diagram of a microcomputer system is shown below:



A Microcomputer System

Applications of Microprocessors

Microprocessors are a mass storage device. They are the advanced form of computers. They arealso called as microcomputers. The impact of microprocessor in different lures of fields is significant. The availability of low cost, low power and small weight, computing capability makes it useful in different applications. Now a days, a microprocessor based systems are used in instructions, automatic testing product, speed control of motors, traffic light control , light control of furnaces etc. Some of the important areas are mentioned below:

Instrumentation:

---

it is very useful in the field of instrumentation. Frequency counters, function generators, frequency synthesizers, spectrum analyses and many other instruments are available, when microprocessors are used as controller. It is also used in medical instrumentation.

Control:

Microprocessor based controllers are available in home appliances, such as microwave oven, washing machine etc., microprocessors are being used in controlling various parameters like speed, pressure, temperature etc. These are used with the help of suitable transduction.

Communication:

Microprocessors are being used in a wide range of communication equipments. In telephone industry, these are used in digital telephone sets. Telephone exchanges and modem etc. The use of microprocessor in television, satellite communication have made teleconferencing possible. Railway reservation and air reservation system also uses this technology. LAN and WAN for communication of vertical information through computer network.

Office Automation and Publication:

Microprocessor based micro computer with software packages has changed the office environment. Microprocessors based systems are being used for word processing, spread sheet operations, storage etc. The microprocessor has revolutionize the publication technology.

Consumer:

The use of microprocessor in toys, entertainment equipment and home applications is making them more entertaining and full of features. The use of microprocessors is more widespread and popular.Now the Microprocessors are used in :

1. Calculators
2. Accounting system
3. Games machine
4. Complex Industrial Controllers
5. Traffic light Control
6. Data acquisition systems
7. Multi user, multi-function environments
8. Military applications
9. Communication systems

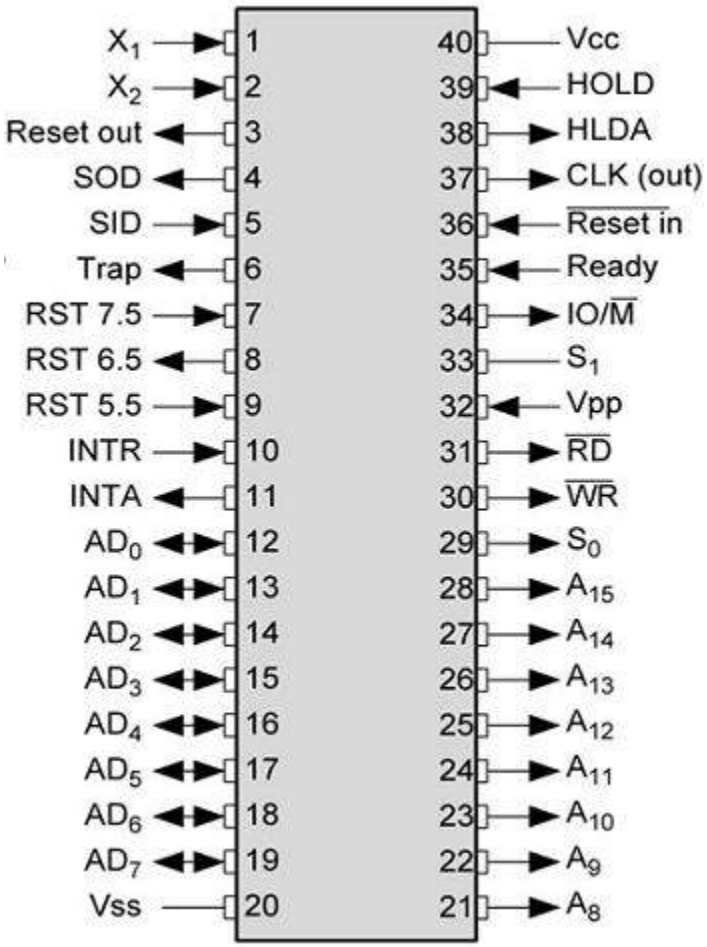
UNIT 2

Microprocessor - 8085 Pin Configuration

Previous Page

Next Page

The following image depicts the pin diagram of 8085 Microprocessor −



The pins of a 8085 microprocessor can be classified into seven groups −

Address bus

A15-A8, it carries the most significant 8-bits of memory/IO address.

Data bus

AD7-AD0, it carries the least significant 8-bit address and data bus.

Control and status signals

These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.

Three control signals are RD, WR & ALE.

- **RD** − This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.
- **WR** − This signal indicates that the data on the data bus is to be written into a selected memory or IO location.
- **ALE** − It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

Three status signals are IO/M, S0 & S1.

IO/M

This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

S1 & S0

These signals are used to identify the type of current operation.

Power supply

There are 2 power supply signals − VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

Clock signals

There are 3 clock signals, i.e. X1, X2, CLK OUT.

- **X1, X2** − A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.
- **CLK OUT** − This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & externally initiated signals

Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.

- **INTA** − It is an interrupt acknowledgment signal.
- **RESET IN** − This signal is used to reset the microprocessor by setting the program counter to zero.

- **RESET OUT** − This signal is used to reset all the connected devices when the microprocessor is reset.
- **READY** − This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.
- **HOLD** − This signal indicates that another master is requesting the use of the address and data buses.
- **HLDA (HOLD Acknowledge)** − It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

Serial I/O signals

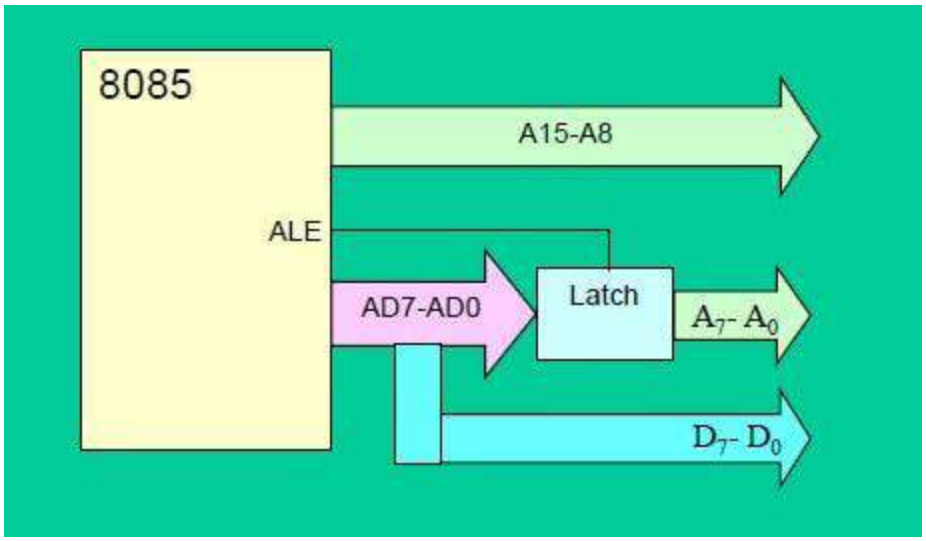
There are 2 serial signals, i.e. SID and SOD and these signals are used for serial communication.

- **SOD** (Serial output data line) − The output SOD is set/reset as specified by the SIM instruction.
- **SID** (Serial input data line) − The data on this line is loaded into accumulator whenever a RIM instruction is executed.

Demultiplexing AD7-AD0

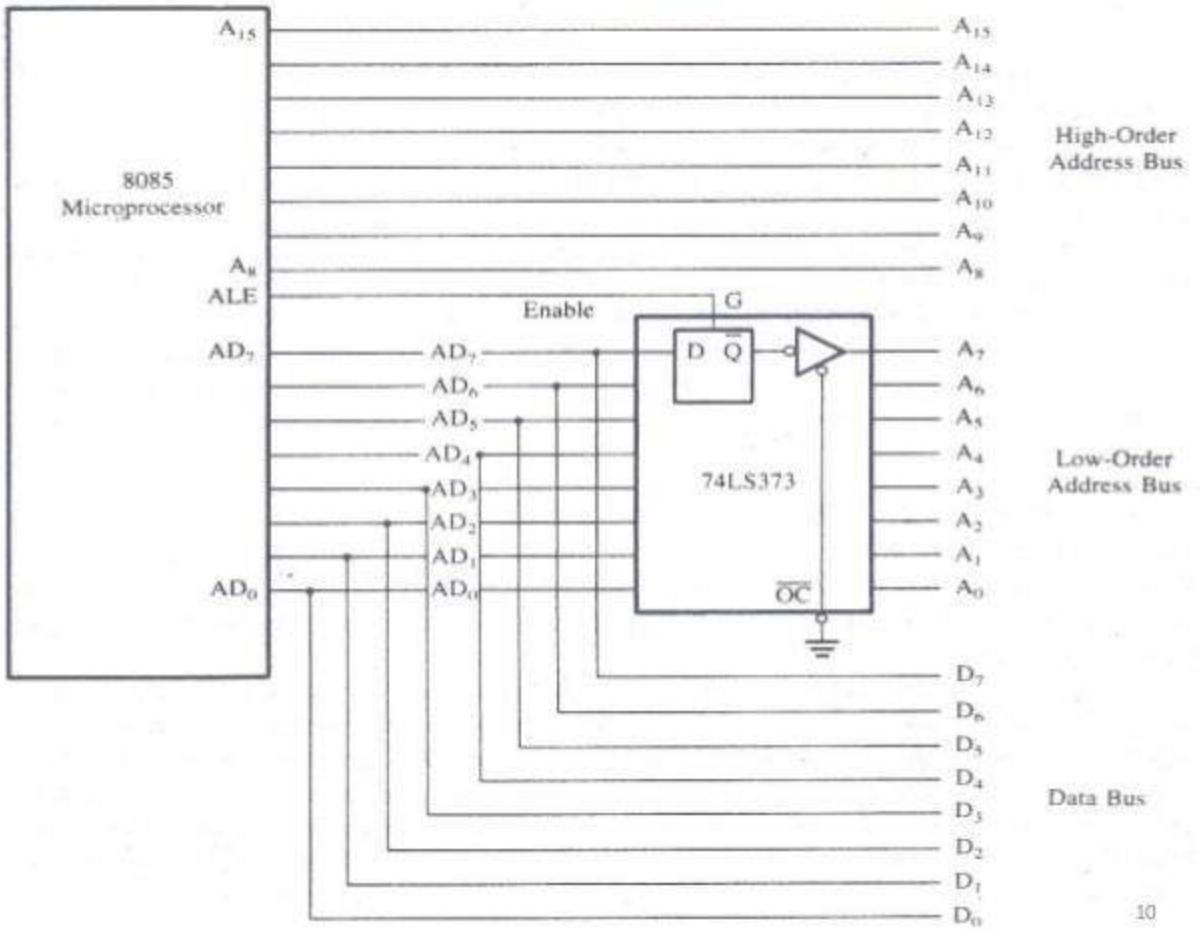
Posted by Mrinal Bag | 10 Dec, 2017 | Microprocessor

– From the above description, it becomes obvious that the AD7–AD0lines are serving a dual purposeand that they need to be demultiplexed to get all the information.
– The high order bitsof the address remain on the bus for three clock periods. However, the low order bitsremain for only one clock periodand they would be lost if they are not saved externally. Also, notice that the low order bitsof the address disappearwhen they are needed most.
– To make sure we have the entire address for the full three clock cycles, we will use an external latchto save the value of AD7–AD0 when it is carrying the address bits. We use the ALEsignal to enable this latch.



– Given that ALE operates as a pulse during T1, we will be able to latch the address. Then when ALE goes low, the address is saved and the AD7–AD0 lines can be used for their purpose as the bi-directional data lines.

• The high order address is placed on the address bus and hold for 3 clk periods,
• The low order address is lost after the first clk period, this address needs to be hold however we need to use latch
• The address AD7 –AD0 is connected as inputs to the latch 74LS373.
• The ALE signal is connected to the enable (G) pin of the latch and the OC –Output control –of the latch is grounded



Microprocessor - 8085 Instruction Sets

Previous Page

Next Page

Let us take a look at the programming of 8085 Microprocessor.

Instruction sets are instruction codes to perform some task. It is classified into five categories.

S.No.	Instruction & Description

1	**Control Instructions** Following is the table showing the list of Control instructions with their meanings.
2	**Logical Instructions** Following is the table showing the list of Logical instructions with their meanings.
3	**Branching Instructions** Following is the table showing the list of Branching instructions with their meanings.
4	**Arithmetic Instructions** Following is the table showing the list of Arithmetic instructions with their meanings.
5	**Data Transfer Instructions** Following is the table showing the list of Data-transfer instructions with their meanings.

8085 – Demo Programs

Now, let us take a look at some program demonstrations using the above instructions −

Adding Two 8-bit Numbers

Write a program to add data at 3005H & 3006H memory location and store the result at 3007H memory location.

Problem demo −

```
(3005H) = 14H
  (3006H) = 89H
```

Result −

14H + 89H = 9DH

The program code can be written like this −

```
LXI H 3005H   : "HL points 3005H"

MOV A, M      : "Getting first operand"

INX H         : "HL points 3006H"
```

```
ADD M        : "Add second operand"
INX H        : "HL points 3007H"
MOV M, A     : "Store result at 3007H"
HLT          : "Exit program"
```

Exchanging the Memory Locations

Write a program to exchange the data at 5000M& 6000M memory location.

```
LDA 5000M   : "Getting the contents at5000M location into accumulator"
MOV B, A    : "Save the contents into B register"
LDA 6000M   : "Getting the contents at 6000M location into accumulator"
STA 5000M   : "Store the contents of accumulator at address 5000M"
MOV A, B    : "Get the saved contents back into A register"
STA 6000M   : "Store the contents of accumulator at address 6000M"
```

Arrange Numbers in an Ascending Order

Write a program to arrange first 10 numbers from memory address 3000H in an ascending order.

```
MVI B, 09        :"Initialize counter"
START            :"LXI H, 3000H: Initialize memory pointer"
MVI C, 09H       :"Initialize counter 2"
BACK: MOV A, M   :"Get the number"
INX H            :"Increment memory pointer"
CMP M            :"Compare number with next number"
JC SKIP          :"If less, don't interchange"
JZ SKIP          :"If equal, don't interchange"
MOV D, M
MOV M, A
DCX H
MOV M, D
INX H            :"Interchange two numbers"
```

```
SKIP:DCR C        :"Decrement counter 2"
JNZ BACK          :"If not zero, repeat"
DCR B             :"Decrement counter 1"
JNZ START
HLT               :"Terminate program execution"
```

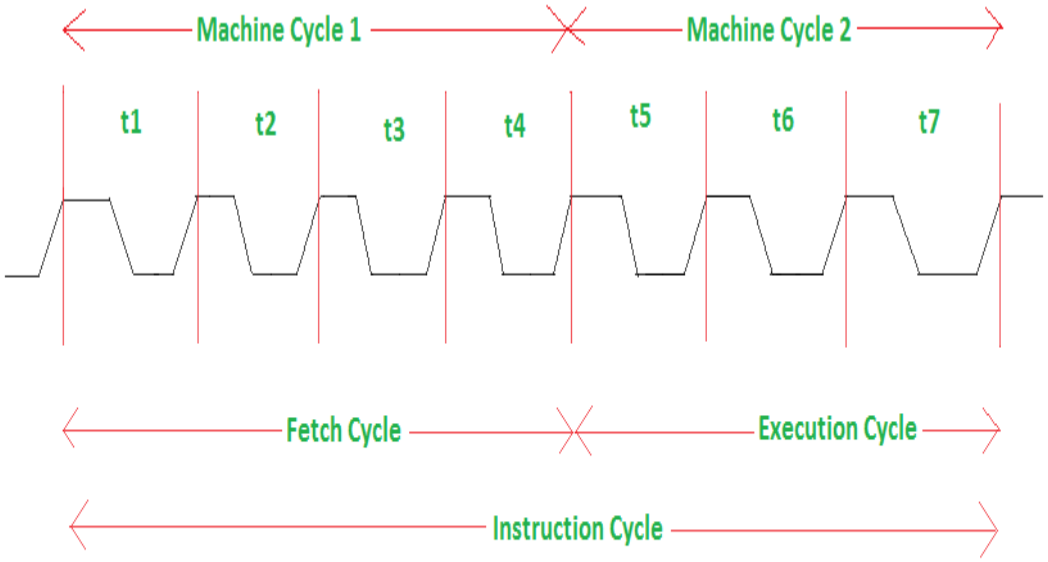
Instruction cycle in 8085 microprocessor

Time required to execute and fetch an entire instruction is called *instruction cycle*. It consists:

- **Fetch cycle –** The next instruction is fetched by the address stored in program counter (PC) and then stored in the instruction register.
- **Decode instruction –** Decoder interprets the encoded instruction from instruction register.
- **Reading effective address –** The address given in instruction is read from main memory and required data is fetched. The effective address depends on direct addressing mode or indirect addressing mode.
- **Execution cycle –** consists memory read (MR), memory write (MW), input output read (IOR) and input output write (IOW)

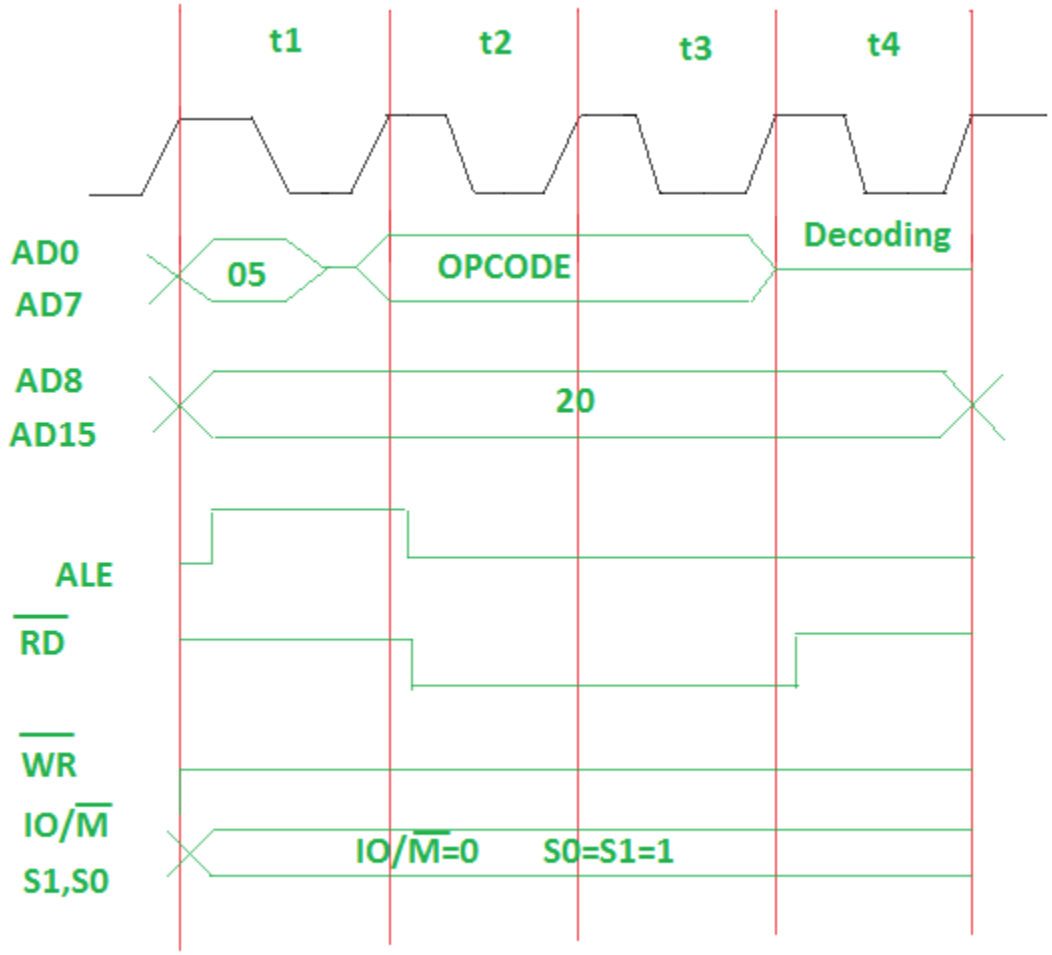
The time required by the microprocessor to complete an operation of accessing memory or input/output devices is called *machine cycle*. One time period of frequency of microprocessor is called *t-state*. A t-state is measured from the falling edge of one clock pulse to the falling edge of the next clock pulse.

Fetch cycle takes four t-states and execution cycle takes three t-states.



Instruction cycle in 8085 microprocessor

Timing diagram for fetch cycle or opcode fetch:



Timing diagram for opcode fetch

Above diagram represents:

- **05 –** lower bit of address where opcode is stored. Multiplexed address and data bus AD0-AD7 are used.
- **20 –** higher bit of address where opcode is stored. Multiplexed address and data bus AD8-AD15 are used.
- **ALE –** Provides signal for multiplexed address and data bus. If signal is high or 1, multiplexed address and data bus will be used as address bus. To fetch lower bit of address, signal is 1 so that multiplexed bus can act as address bus. If signal is low or 0, multiplexed bus will be used as data bus. When lower bit of address is fetched then it will act as data bus as the signal is low.
- **RD (low active) –** If signal is high or 1, no data is read by microprocessor. If signal is low or 0, data is read by microprocessor.
- **WR (low active) –** If signal is high or 1, no data is written by microprocessor. If signal is low or 0, data is written by microprocessor.
- **IO/M (low active) and S1, S0 –** If signal is high or 1, operation is performing on input output. If signal is low or 0, operation is performing on memory.

Machine Cycle	Status			Control Signals		
	$IO/\overline{M}$	S1	S0	$\overline{RD}$	$\overline{WR}$	$\overline{INTA}$
Opcode Fetch	0	1	1	0	1	1
Memory Read	0	1	0	0	1	1
Memory Write	0	0	1	1	0	1
I/0 Read	1	1	0	0	1	1
I/O Write	1	0	1	1	0	1
Interrupt Acknowledge	1	1	1	1	1	0
HALT	Z	0	0	Z	Z	1
HOLD	Z	X	X	Z	Z	1
RESET	Z	X	X	Z	Z	1

Where Z is tri state (pin neither connected to supply nor ground. High impedance) and X represents do not care.

8085 machine cycle status and control signals

Timing Diagram and machine cycles of 8085 Microprocessor

1 Machine cycles of 8085 The 8085 microprocessor has 5 (seven) basic machine cycles. They are ü Opcode fetch cycle (4T) ü Memory read cycle (3 T) ü Memory write cycle (3 T) ü I/O read cycle (3 T) ü I/O write cycle (3 T)

Timing Diagram

Timing Diagram is a graphical representation. It represents the execution time taken by each instruction in a graphical format. The execution time is represented in T-states.

Instruction Cycle:

The time required to execute an instruction is called instruction cycle.

Machine Cycle:

The time required to access the memory or input/output devices is called machine cycle.

T-State:

- The machine cycle and instruction cycle takes multiple clock periods.
- A portion of an operation carried out in one system clock period is called as T-state.

1 Machine cycles of 8085

The 8085 microprocessor has 5 (seven) basic machine cycles. They are

- Opcode fetch cycle (4T)
- Memory read cycle (3 T)
- Memory write cycle (3 T)
- I/O read cycle (3 T)
- I/O write cycle (3 T)

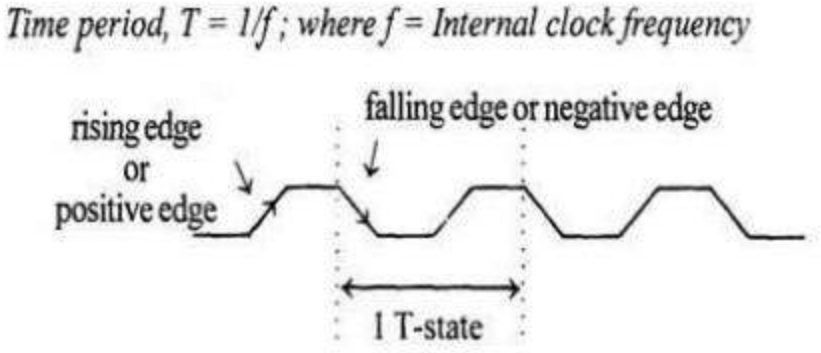


Fig 1.7 Clock Signal

Signal 1.Opcode fetch machine cycle of 8085 :

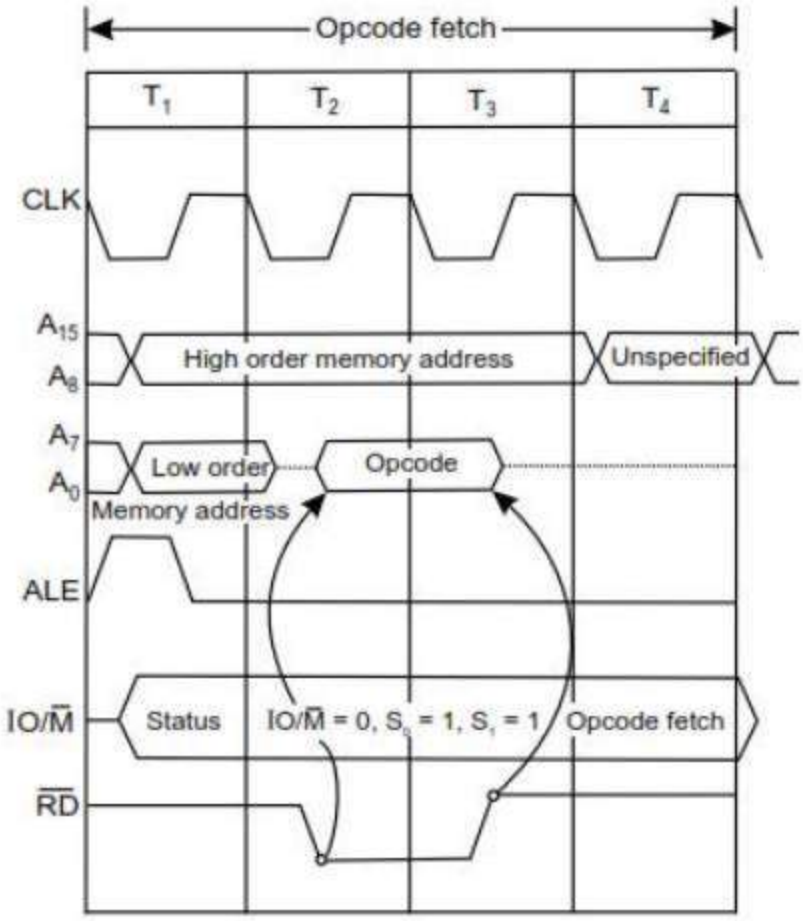


Fig 1.8 Opcode fetch machine cycle

- Each instruction of the processor has one byte opcode.
- The opcodes are stored in memory. So, the processor executes the opcode fetch machine cycle to fetch the opcode from memory.
- Hence, every instruction starts with opcode fetch machine cycle.
- The time taken by the processor to execute the opcode fetch cycle is 4T.
- In this time, the first, 3 T-states are used for fetching the opcode from memory and the remaining T-states are used for internal operations by the processor.

2. Memory Read Machine Cycle of 8085:

- The memory read machine cycle is executed by the processor to read a data byte from memory.
- The processor takes 3T states to execute this cycle.

The instructions which have more than one byte word size will use the machine cycle after the opcode fetch machine cycle.

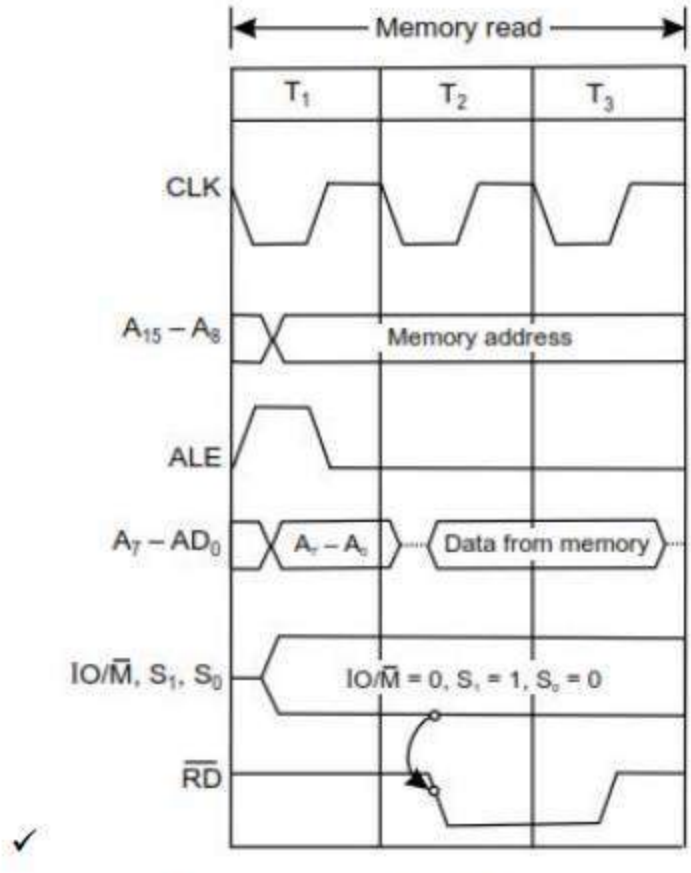


Fig 1.9 Memory Read Machine Cycle

Cycle 3. Memory Write Machine Cycle of 8085

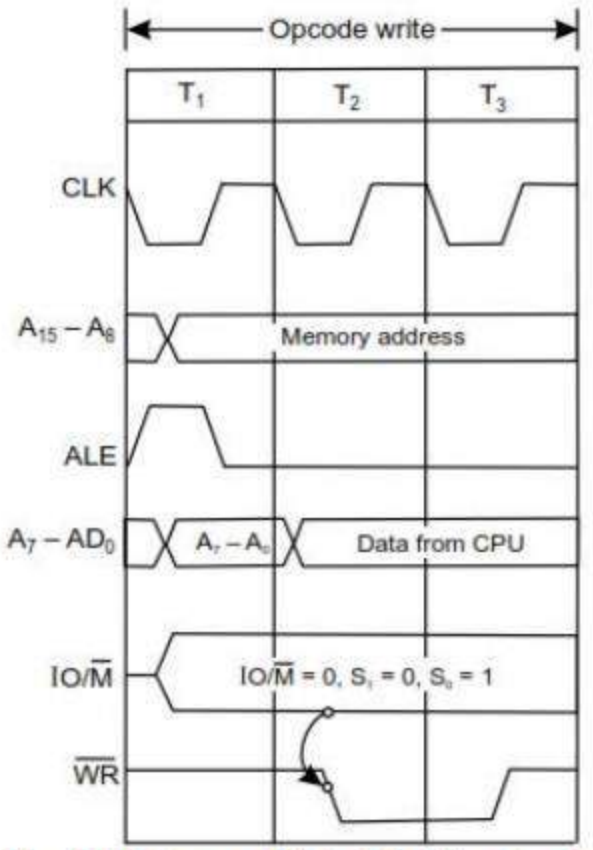


Fig 1.10 Memory Write Machine Cycle

- The memory write machine cycle is executed by the processor to write a data byte in a memory location.
- The processor takes, 3T states to execute this machine cycle.

4. <u>I/O Read Cycle of 8085</u>

- The I/O Read cycle is executed by the processor to read a data byte from I/O port or from the peripheral, which is I/O, mapped in the system.
-
- The processor takes 3T states to execute this machine cycle.
- The IN instruction uses this machine cycle during the execution.

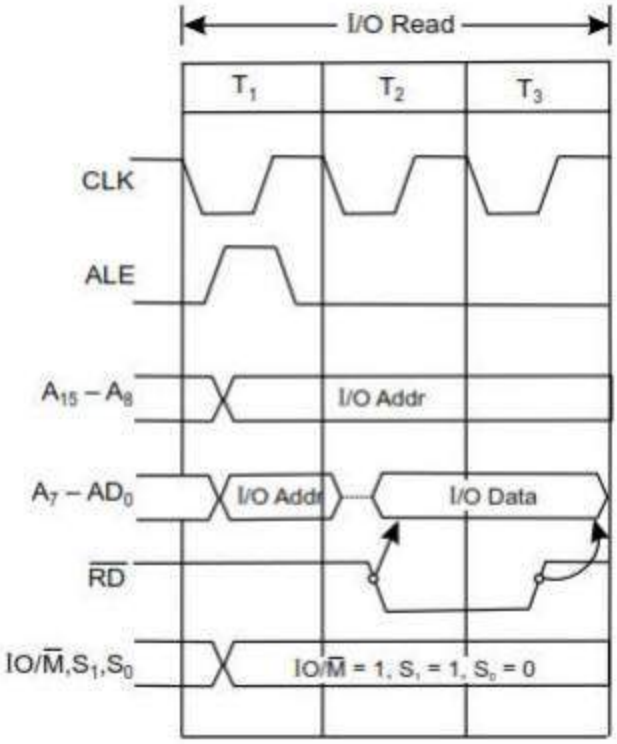


Fig 1.11 I/O Read Cycle

Cycle 1.4.2 Timing diagram for STA 526AH

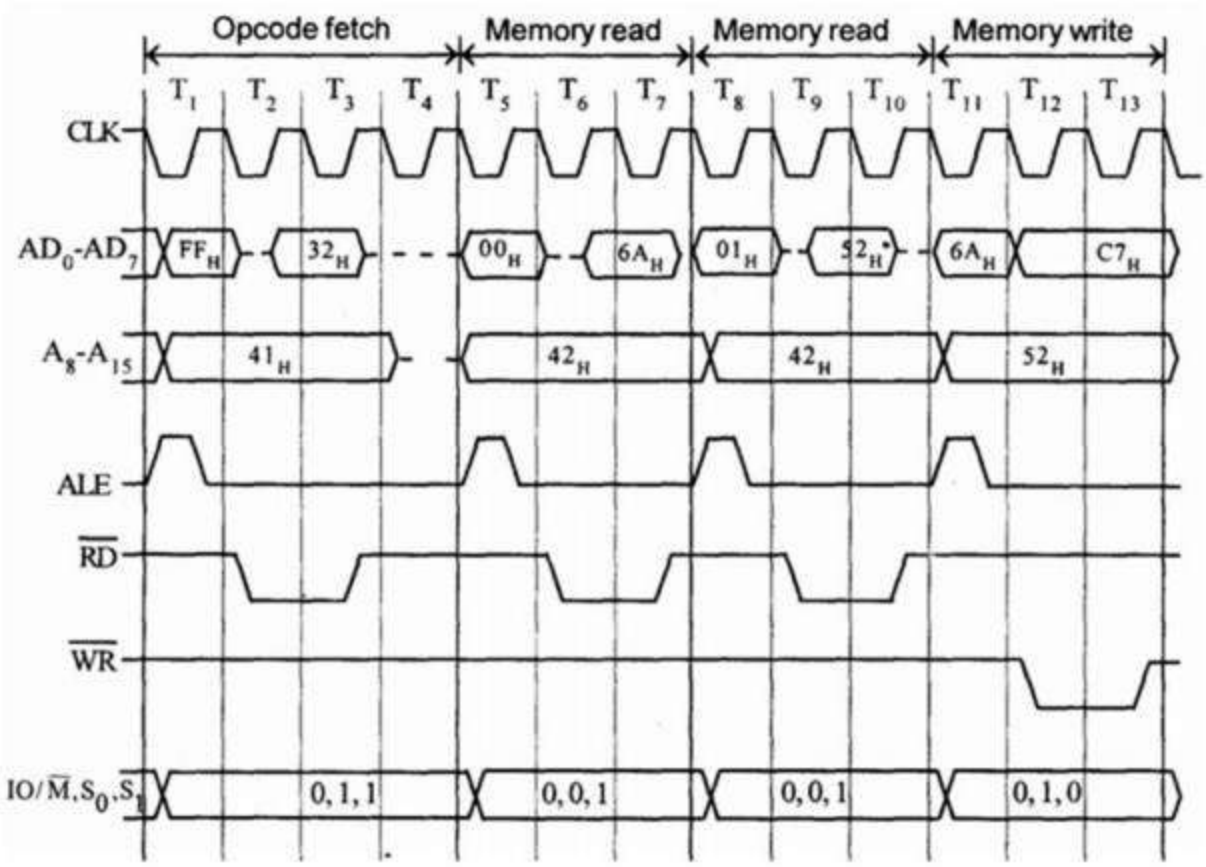


Fig 1.12 Timing Diagram for STA 526A H

Address	Mnemonics	Opcode
41FF	STA 526A$_H$	32$_H$
4200		6A$_H$
4201		52$_H$

- STA means Store Accumulator -The contents of the accumulator is stored in the specified address (526A).

- The opcode of the STA instruction is said to be 32H. It is fetched from the memory 41FFH (see fig). - OF machine cycle

- Then the lower order memory address is read (6A). - Memory Read Machine Cycle

- Read the higher order memory address (52).- Memory Read Machine Cycle

- The combination of both the addresses are considered and the content from accumulator is written in 526A. - Memory Write Machine Cycle

- Assume the memory address for the instruction and let the content of accumulator is C7H. So, C7H from accumulator is now stored in 526A.

3 Timing diagram for INR M

- Fetching the Opcode 34H from the memory 4105H. (OF cycle)
- Let the memory address (M) be 4250H. (MR cycle -To read Memory address and data)
- Let the content of that memory is 12H.
- Increment the memory content from 12H to 13H. (MW machine cycle)

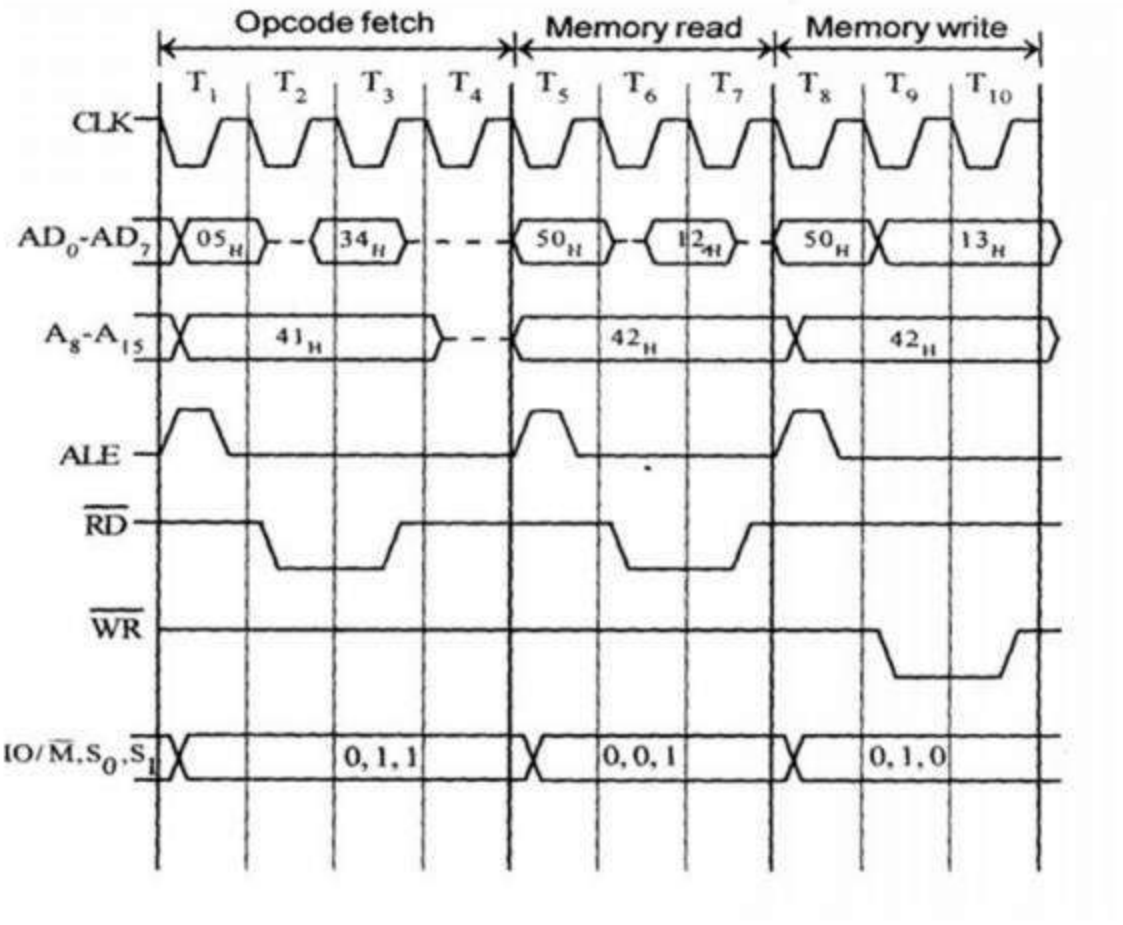
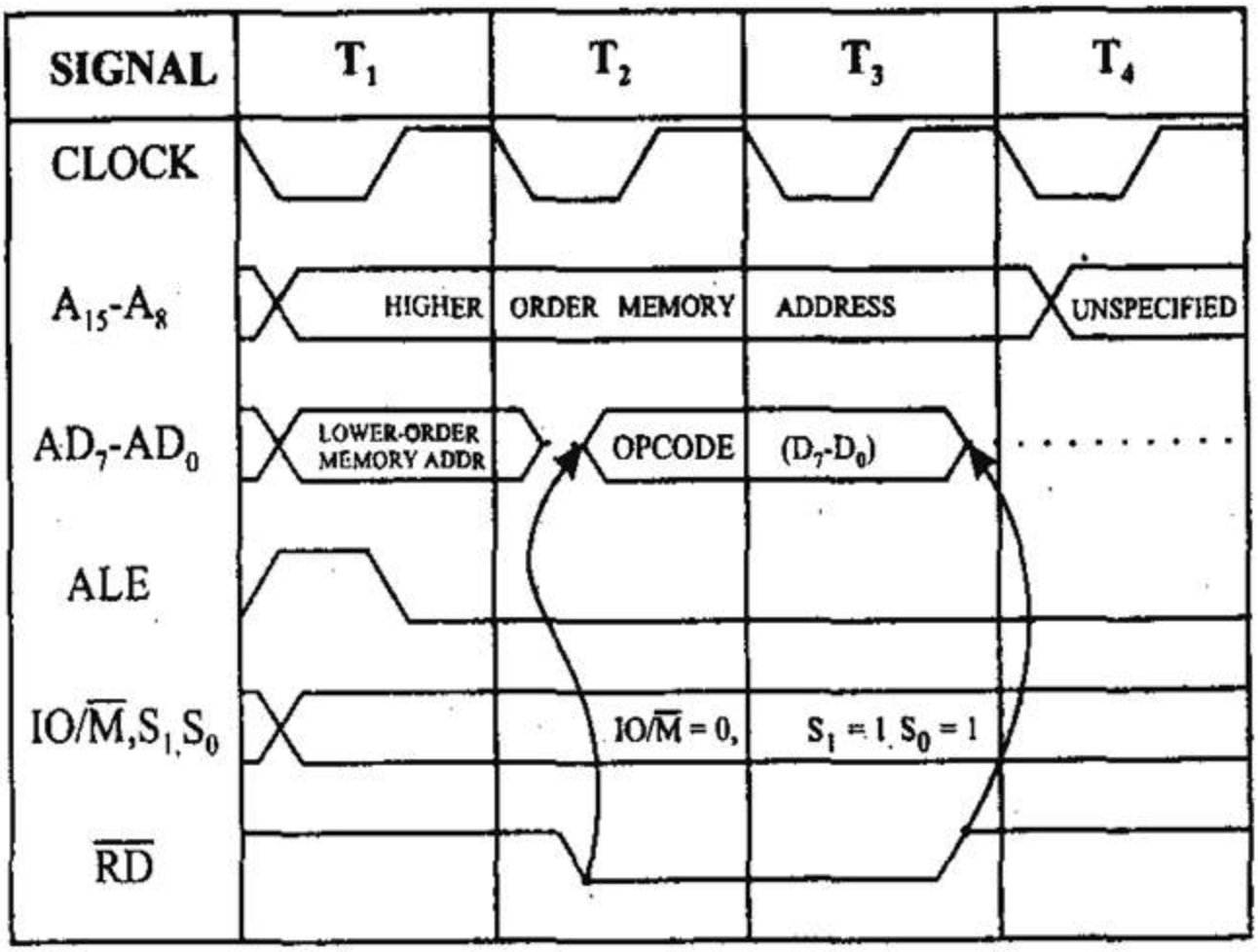


Fig 1.13 Timing Diagram for INR M

Address	Mnemonics	Opcode
4105	INR M	34_H

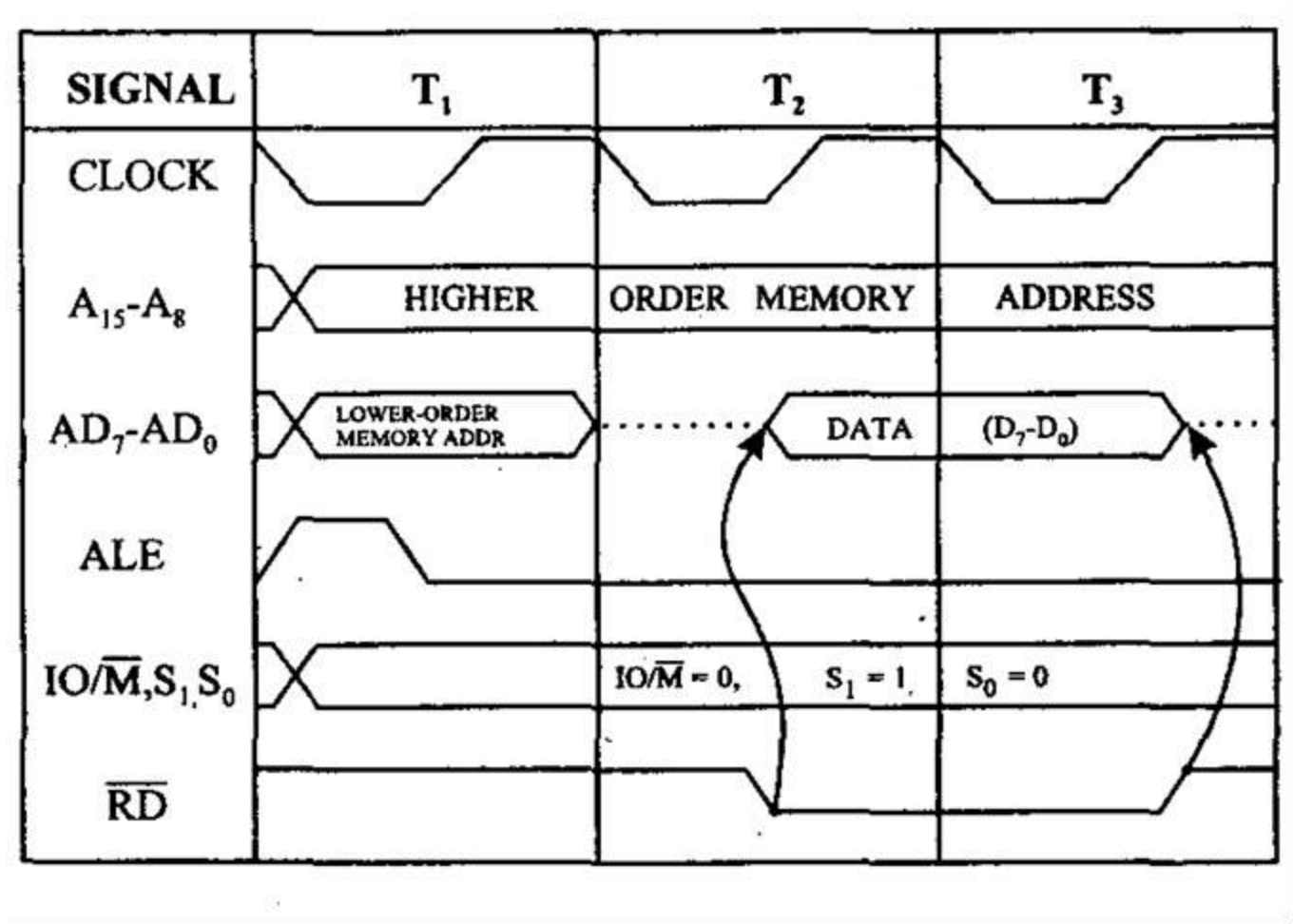
Opcode fetch machine cycle of 8085 :

- **Each instruction of the processor has one byte opcode.**
- **The opcodes are stored in memory. So, the processor executes the opcode fetch machine cycle to fetch the opcode from memory.**
- **Hence, every instruction starts with opcode fetch machine cycle.**
- **The time taken by the processor to execute the opcode fetch cycle is 4T.**
- **In this time, the first, 3 T-states are used for fetching the opcode from memory and the remaining T-states are used for internal operations by the processor.**



Memory Read Machine Cycle of 8085:

- **The memory read machine cycle is executed by the processor to read a data byte from memory.**
- **The processor takes 3T states to execute this cycle.**
- **The instructions which have more than one byte word size will use the machine cycle after the opcode fetch machine cycle.**



The steps involved in doing programs in trainer kit are summarized below.

1.Write the program

2.Prepare the hex code of the program

3.Enter the hex code to trainer kit in suitable locations(eg:From 4100 H)

4.Give sufficient data for program in data memory(eg:From 4200 H)

5.Run the program

6.Check for output

The programming procedure is explained through an example. Consider the case of addition of two 8 bit numbers.

1.**WRITE THE PROGRAM**

```
        MV1 C,00
        LDA 4200
        MOV B,A
        LDA 4201
        ADD B
        JNC LABEL1
        INRC
LABEL1:STA 4202
        MOV A,C
        STA 4203
        HLT
```

2.PREPARE HEX CODE OF THE PROGRAM

Hex code is preparing in accordance with the Intel Hex file format.The hex code table is given below.Each instruction has a hex code. Consider the instruction ADD B.From the hex file format the corresponding hex code is 80.

Consider STA 4203 .The first location should contain the hex code of STA. That is 32.Next location should contain the LSB of 4203, that is 03.The next location should contain the MSB of the address,that is 42.

The hex code table is given below

The hex code of the above program is given below.

LABEL	ADDRESS	MNEMONIC	CODE	COMMENT
	4100	MVI C,00	0E	
	4101	–	00	
	4102	LDA 4200	3A	
	4103	–	00	
	4104	–	42	
	4105	MOV B,A	47	
	4106	LDA 4201	3A	
	4107	–	01	Write the meaning of each instruction in this column
	4108	–	42	
	4109	ADD B	80	
	410A	JNC LABEL1	D2	
	410B	–	0E	
	410C	–	41	
	410D	INR C	0C	
LABEL1	410E	STA 4202	32	
	410F	–	02	
	4110	–	42	
	4111	MOV A,C	79	
	4112	STA 4203	32	

	4113	**–**	**03**	
	4114	**–**	**42**	
	4115	**HLT**	**76**	

3.ENTER THE HEX CODE TO THE TRAINER

The obtained hex code now can be entered to the trainer. Assume the program is entering from location 4100. Now to enter the code,press **SUB key +4100+NEXT key.**Now the location will be 4100. Now start entering the program. After entering each code press the NEXT key to get the next location.

4.ENTER SUFFICIENT DATA

In the above program the two numbers which are to be added are to be supplied. Suppose the data memory is from locations 4200.To write data press **SUB key +4200+NEXT key** . Now the address will be 4200.Enter the first data in 4200.Now press **NEXT** key to get the next location .

5.EXCECUTE THE PROGRAM

Assume the program is from location 4100 and data memory is from 4200.Then the command to execute the program is **SUB key+4100+EXEC key.** Now there will be an **E**symbol on the display . This indicates the running of program.

6. CHECKING DATA MEMORY FOR RESULTS

After few seconds the running is stopping by an interrupt . Then the data memory can be read t get the result of running by the **SUB key +4200+NEXT key.** In our example assume the input numbers are 05 and 04 .The output after running the program is given below.

ADDRESSES	DATA	INPUT/OUTPUT
4200	05	Input
4201	04	Input
4202	09	Output

4203	00	Output

UNIT 3

8085 Addressing Modes & Interrupts

Previous Page

Next Page

Now let us discuss the addressing modes in 8085 Microprocessor.

Addressing Modes in 8085

These are the instructions used to transfer the data from one register to another register, from the memory to the register, and from the register to the memory without any alteration in the content. Addressing modes in 8085 is classified into 5 groups −

Immediate addressing mode

In this mode, the 8/16-bit data is specified in the instruction itself as one of its operand. **For example:** MVI K, 20F: means 20F is copied into register K.

Register addressing mode

In this mode, the data is copied from one register to another. **For example:**MOV K, B: means data in register B is copied to register K.

Direct addressing mode

In this mode, the data is directly copied from the given address to the register. **For example:** LDB 5000K: means the data at address 5000K is copied to register B.

Indirect addressing mode

In this mode, the data is transferred from one register to another by using the address pointed by the register. **For example:** MOV K, B: means data is transferred from the memory address pointed by the register to the register K.

Implied addressing mode

This mode doesn't require any operand; the data is specified by the opcode itself. **For example:** CMP.

Interrupts in 8085

Interrupts are the signals generated by the external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.

Interrupt are classified into following groups based on their parameter −

- **Vector interrupt** − In this type of interrupt, the interrupt address is known to the processor. **For example:** RST7.5, RST6.5, RST5.5, TRAP.
- **Non-Vector interrupt** − In this type of interrupt, the interrupt address is not known to the processor so, the interrupt address needs to be sent externally by the device to perform interrupts. **For example:** INTR.
- **Maskable interrupt** − In this type of interrupt, we can disable the interrupt by writing some instructions into the program. **For example:**RST7.5, RST6.5, RST5.5.
- **Non-Maskable interrupt** − In this type of interrupt, we cannot disable the interrupt by writing some instructions into the program. **For example:** TRAP.
- **Software interrupt** − In this type of interrupt, the programmer has to add the instructions into the program to execute the interrupt. There are 8 software interrupts in 8085, i.e. RST0, RST1, RST2, RST3, RST4, RST5, RST6, and RST7.
- **Hardware interrupt** − There are 5 interrupt pins in 8085 used as hardware interrupts, i.e. TRAP, RST7.5, RST6.5, RST5.5, INTA.

Note − NTA is not an interrupt, it is used by the microprocessor for sending acknowledgement. TRAP has the highest priority, then RST7.5 and so on.

Interrupt Service Routine (ISR)

A small program or a routine that when executed, services the corresponding interrupting source is called an ISR.

TRAP

It is a non-maskable interrupt, having the highest priority among all interrupts. Bydefault, it is enabled until it gets acknowledged. In case of failure, it executes as ISR and sends the data to backup memory. This interrupt transfers the control to the location 0024H.

RST7.5

It is a maskable interrupt, having the second highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 003CH address.

RST 6.5

It is a maskable interrupt, having the third highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 0034H address.

RST 5.5

It is a maskable interrupt. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 002CH address.

INTR

It is a maskable interrupt, having the lowest priority among all interrupts. It can be disabled by resetting the microprocessor.

When **INTR signal goes high**, the following events can occur −

- The microprocessor checks the status of INTR signal during the execution of each instruction.
- When the INTR signal is high, then the microprocessor completes its current instruction and sends active low interrupt acknowledge signal.
- When instructions are received, then the microprocessor saves the address of the next instruction on stack and executes the received instruction.

Instruction Set 8085

---

1. Control
2. Logical
3. Branching
4. Arithmetic
5. Data Transfer

Control Instructions

Opcode	Operand	Explanation of Instruction	Description
NOP	**none**	No operation	No operation is performed. The instruction is fetched and decod However no operation is executed. Example: NOP
HLT	**none**	Halt and enter	The CPU finishes executing the current instruction and halts a

		wait state	further execution. An interrupt or reset is necessary to exit from the h state. Example: HLT
DI	**none**	Disable interrupts	The interrupt enable flip-flop is reset and all the interrupts except TRAP are disabled. No flags are affected. Example: DI
EI	**none**	Enable interrupts	The interrupt enable flip-flop is set and all interrupts are enabled. flags are affected. After a system reset or the acknowledgement of interrupt, the interrupt enable flipflop is reset, thus disabling interrupts. This instruction necessary to reenable the interrupts (except TRAP). Example: EI
RIM	**none**	Read interrupt mas	This is a multipurpose instruction used to read the status of interru 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eig bits in the accumulator with the following interpretations. Example: RIM
SIM	**none**	Set interrupt mask	This is a multipurpose instruction and used to implement the 80 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interpr the accumulator contents as follows. Example: SIM

LOGICAL INSTRUCTIONS

Opcode	Operand	Explanation of Instruction	Description
CMP	**R** **M**	Compare register or memory with accumulator	The contents of the operand (register or memory) are M compar with the contents of the accumulator. Both contents are preserve The result of the comparison is shown by setting the flags of PSW as follows: if (A) < (reg/mem): carry flag is

			if (A) = (reg/mem): zero flag is
			if (A) > (reg/mem): carry and zero flags are reset
			
			Example: CMP B or CMP M

CPI	**8-bit data**	Compare immediate with accumulator	The second byte (8-bit data) is compared with the contents of accumulator. The values being compared remain unchanged. T result of the comparison is shown by setting the flags of the PSW follows:
			
			if (A) < data: carry flag is
			if (A) = data: zero flag is
			if (A) > data: carry and zero flags are reset
			
			Example: CPI 89H

ANA	**R** **M**	Logical AND register or memory with accumulator	The contents of the accumulator are logically ANDed with M contents of the operand (register or memory), and the result is plac in the accumulator. If the operand is a memory location, its addr is specified by the contents of HL registers. S, Z, P are modified reflect the result of the operation. CY is reset. AC is set. Example: ANA B or ANA M
ANI	**8-bit data**	Logical AND immediate with accumulator	The contents of the accumulator are logically ANDed with 8-bit data (operand) and the result is placed in accumulator. S, Z, P are modified to reflect the result of operation. CY is reset. AC is set. Example: ANI 86H
XRA	**R** **M**	Exclusive OR register or memory with accumulator	The contents of the accumulator are Exclusive ORed with M contents of the operand (register or memory), and the result is plac in the accumulator. If the operand is a memory location, its addr is specified by the contents of HL registers. S, Z, P are modified reflect the result of the operation. CY and AC are reset. Example: XRA B or XRA M
XRI	**8-bit data**	Exclusive OR immediate with accumulator	The contents of the accumulator are Exclusive ORed with the 8- data (operand) and the result is placed in the accumulator. S, Z, P modified to reflect the result of the operation. CY and AC are rese Example: XRI 86H
ORA	**R** **M**	Logical OR register or memory with accumulator	The contents of the accumulator are logically ORed with M contents of the operand (register or memory), and the result is plac in the accumulator. If the operand is a memory location, its addr is specified by the contents of HL registers. S, Z, P are modified

			reflect the result of the operation. CY and AC are reset. Example: ORA B or ORA M
ORI	**8-bit data**	Logical OR immediate with accumulator	The contents of the accumulator are logically ORed with the 8- data (operand) and the result is placed in the accumulator. S, Z, P modified to reflect the result of the operation. CY and AC are rese Example: ORI 86H
RLC	**none**	Rotate accumulator left	Each binary bit of the accumulator is rotated left by one position. D7 is placed in the position of D0 as well as in the Carry flag. CY modified according to bit D7. S, Z, P, AC are not affected. Example: RLC
RRC	**none**	Rotate accumulator right	Each binary bit of the accumulator is rotated right by one positi Bit D0 is placed in the position of D7 as well as in the Carry fl CY is modified according to bit D0. S, Z, P, AC are not affected. Example: RRC
RAL	**none**	Rotate accumulator left through carry	Each binary bit of the accumulator is rotated left by one positi through the Carry flag. Bit D7 is placed in the Carry flag, and Carry flag is placed in the least significant position D0. CY modified according to bit D7. S, Z, P, AC are not affected. Example: RAL
RAR	**none**	Rotate accumulator right through carry	Each binary bit of the accumulator is rotated right by one posit through the Carry flag. Bit D0 is placed in the Carry flag, and Carry flag is placed in the most significant position D7. CY modified according to bit D0. S, Z, P, AC are not affected. Example: RAR
CMA	**none**	Complement accumulator	The contents of the accumulator are complemented. No flags affected. Example: CMA
CMC	**none**	Complement carry	The Carry flag is complemented. No other flags are affected. Example: CMC
STC	**none**	Set Carry	Set Carry Example: STC

BRANCHING INSTRUCTIONS

<table>
<tr><th>Opcode</th><th>Operand</th><th>Explanation of Instruction</th><th>Description</th></tr>
<tr><td>JMP</td><td>16-bit address</td><td>Jump unconditionally</td><td>The program sequence is transferred the memory location specified by the bit address given in the operand.

Example: JMP 2034H or JMP XYZ</td></tr>
<tr><td>
<table>
<tr><th>Opcode</th><th>Description</th><th>Flag Status</th></tr>
<tr><td>JC</td><td>Jump on Carry</td><td>CY = 1</td></tr>
<tr><td>JNC</td><td>Jump on no Carry</td><td>CY = 0</td></tr>
<tr><td>JP</td><td>Jump on positive</td><td>S = 0</td></tr>
<tr><td>JM</td><td>Jump on minus</td><td>S = 1</td></tr>
<tr><td>JZ</td><td>Jump on zero</td><td>Z = 1</td></tr>
<tr><td>JNZ</td><td>Jump on no zero</td><td>Z = 0</td></tr>
<tr><td>JPE</td><td>Jump on parity even</td><td>P = 1</td></tr>
<tr><td>JPO</td><td>Jump on parity odd</td><td>P = 0</td></tr>
</table>
</td><td>16-bit address</td><td>Jump conditionally</td><td>The program sequence is transferred the memory location specified by the bit address given in the operand based the specified flag of the PSW described below.

Example: JZ 2034H or JZ XYZ</td></tr>
<tr><td>
<table>
<tr><th>Opcode</th><th>Description</th><th>Flag Status</th></tr>
<tr><td>CC</td><td>Call on Carry</td><td>CY = 1</td></tr>
<tr><td>CNC</td><td>Call on no Carry</td><td>CY = 0</td></tr>
<tr><td>CP</td><td>Call on positive</td><td>S = 0</td></tr>
<tr><td>CM</td><td>Call on minus</td><td>S = 1</td></tr>
</table>
</td><td>16-bit address</td><td>Unconditional subroutine call</td><td>The program sequence is transferred the memory location specified by the bit address given in the operand. Bef the transfer, the address of the n instruction after CALL (the contents the program counter) is pushed onto stack.

Example: CALL 2034H or CALL XY</td></tr>
</table>

CZ	**Call on zero**	**Z = 1**			
CNZ	**Call on no zero**	**Z = 0**			
CPE	**Call on parity even**	**P = 1**			
CPO	**Call on parity odd**	**P = 0**			
RET			**none**	Return from subroutine unconditionally	The program sequence is transfer from the subroutine to the calli program. The two bytes from the top the stack are copied into the progr counter,and program execution begins the new address. Example: RET

Opcode	Description	Flag Status			
RC	**Return on Carry**	**CY = 1**	**none**	Return from subroutine conditionally	The program sequence is transfer from the subroutine to the calli program based on the specified flag the PSW as described below. The t bytes from the top of the stack are cop into the program counter, and progr execution begins at the new address. Example: RZ
RNC	**Return on no Carry**	**CY = 0**			
RP	**Return on positive**	**S = 0**			
RM	**Return on minus**	**S = 1**			
RZ	**Return on zero**	**Z = 1**			
RNZ	**Return on no zero**	**Z = 0**			
RPE	**Return on parity even**	**P = 1**			
RPO	**Return on parity odd**	**P = 0**			

PCHL	**none**	Load program counter with HL contents	The contents of registers H and L copied into the program counter. T contents of H are placed as the high-or byte and the contents of L as the lo order byte. Example: PCHL
RST	**0-7**	Restart	The RST instruction is equivalent to a byte call instruction to one of eig

memory locations depending upon
number. The instructions are genera
used in conjunction with interrupts a
inserted using external hardwa
However these can be used as softw
instructions in a program to trans
program execution to one of the ei
locations. The addresses are:

Instruction	Restart Address
RST 0	**0000H**
RST1	**0008H**
RST 2	**0010H**
RST 3	**0018H**
RST 4	**0020H**
RST 5	**0028H**
RST 6	**0030H**
RST 7	**0038H**

The 8085 has four additional interru
and these interrupts generate R
instructions internally and thus do
require any external hardware. Th
instructions and their Restart addres
are:

Interrupt	Restart Address
TRAP	**0024H**
RST 5.5	**002CH**
RST 6.5	**0034H**

			RST 7.5 **003CH**

Arithmetic Instructions

Opcode	Operand	Explanation of Instruction	Description
ADD	**R** **M**	Add register or memory, to accumulator	The contents of the operand (register or memory) are added to t contents of the accumulator and the result is stored in t accumulator. If the operand is a memory location, its location specified by the contents of the HL registers. All flags modified to reflect the result of the addition. Example: ADD B or ADD M
ADC	**R** **M**	Add register to accumulator with carry	The contents of the operand (register or memory) and M the Ca flag are added to the contents of the accumulator and the result stored in the accumulator. If the operand is a memory location, location is specified by the contents of the HL registers. All fla are modified to reflect the result of the addition. Example: ADC B or ADC M
ADI	**8-bit data**	Add immediate to accumulator	The 8-bit data (operand) is added to the contents of accumulator and the result is stored in the accumulator. All fla are modified to reflect the result of the addition. Example: ADI 45H
ACI	**8-bit data**	Add immediate to accumulator with carry	The 8-bit data (operand) and the Carry flag are added to contents of the accumulator and the result is stored in t accumulator. All flags are modified to reflect the result of t addition. Example: ACI 45H
LXI	**Reg. pair, 16-bit data**	Load register pair immediate	The instruction loads 16-bit data in the register pair designated the operand. Example: LXI H, 2034H or LXI H, XYZ

DAD	**Reg. pair**	Add register pair to H and L registers	The 16-bit contents of the specified register pair are added to contents of the HL register and the sum is stored in the register. The contents of the source register pair are not altered the result is larger than 16 bits, the CY flag is set. No other fl are affected. Example: DAD H
SUB	**R** **M**	Subtract register or memory from accumulator	The contents of the operand (register or memory) are subtrac from the contents of the accumulator, and the result is stored in accumulator. If the operand is a memory location, its location specified by the contents of the HL registers. All flags modified to reflect the result of the subtraction. Example: SUB B or SUB M
SBB	**R** **M**	Subtract source and borrow from accumulator	The contents of the operand (register or memory) and M Borrow flag are subtracted from the contents of the accumula and the result is placed in the accumulator. If the operand i memory location, its location is specified by the contents of HL registers. All flags are modified to reflect the result of subtraction. Example: SBB B or SBB M
SUI	**8-bit data**	Subtract immediate from accumulator	The 8-bit data (operand) is subtracted from the contents of accumulator and the result is stored in the accumulator. All fl are modified to reflect the result of the subtraction. Example: SUI 45H
SBI	**8-bit data**	Subtract immediate from accumulator with borrow	The contents of register H are exchanged with the contents register D, and the contents of register L are exchanged with contents of register E. Example: XCHG
INR	**R** **M**	Increment register or memory by 1	The contents of the designated register or memory) incremented by 1 and the result is stored in the same place. If operand is a memory location, its location is specified by contents of the HL registers. Example: INR B or INR M
INX	**R**	Increment register pair by 1	The contents of the designated register pair are incremented b and the result is stored in the same place. Example: INX H

DCR	**R** **M**	Decrement register or memory by 1	The contents of the designated register or memory are decremented by 1 and the result is stored in the same place. If operand is a memory location, its location is specified by contents of the HL registers. Example: DCR B or DCR M
DCX	**R**	Decrement register pair by 1	The contents of the designated register pair are decremented by and the result is stored in the same place. Example: DCX H
DAA	**none**	Decimal adjust accumulator	The contents of the accumulator are changed from a binary va to two 4-bit binary coded decimal (BCD) digits. This is the o instruction that uses the auxiliary flag to perform the binary BCD conversion, and the conversion procedure is describ below. S, Z, AC, P, CY flags are altered to reflect the results of operation. If the value of the low-order 4-bits in the accumulator is grea than 9 or if AC flag is set, the instruction adds 6 to the low-or four bits. If the value of the high-order 4-bits in the accumulator is grea than 9 or if the Carry flag is set, the instruction adds 6 to the hig order four bits. Example: DAA

Data Transfer Instructions

Opcode	Operand	Explanation of Instruction	Description
MOV	**Rd, Rs** **M, Rs** **Rd, M**	Copy from source(Rs) to destination(Rd)	This instruction copies the contents of the source register i the destination register; the contents of the source register not altered. If one of the operands is a memory location, location is specified by the contents of the HL registers. Example: MOV B, C or MOV B, M

MVI	**Rd, data** **M, data**	Move immediate 8-bit	The 8-bit data is stored in the destination register or memory the operand is a memory location, its location is specified the contents of the HL registers. Example: MVI B, 57H or MVI M, 57H
LDA	**16-bit address**	Load accumulator	The contents of a memory location, specified by a 16- address in the operand, are copied to the accumulator. T contents of the source are not altered. Example: LDA 2034H
LDAX	**B/D Reg. pair**	Load accumulator indirect	The contents of the designated register pair point to a memo location. This instruction copies the contents of that memo location into the accumulator. The contents of either the regis pair or the memory location are not altered. Example: LDAX B
LXI	**Reg. pair, 16-bit data**	Load register pair immediate	The instruction loads 16-bit data in the register pair designa in the operand. Example: LXI H, 2034H or LXI H, XYZ
LHLD	**16-bit address**	Load H and L registers direct	The instruction copies the contents of the memory locati pointed out by the 16-bit address into register L and copies contents of the next memory location into register H. T contents of source memory locations are not altered. Example: LHLD 2040H
STA	**16-bit address**	16-bit address	The contents of the accumulator are copied into the memo location specified by the operand. This is a 3-byte instructi the second byte specifies the low-order address and the th byte specifies the high-order address. Example: STA 4350H
STAX	**Reg. pair**	Store accumulator indirect	The contents of the accumulator are copied into the memo location specified by the contents of the operand (register pa The contents of the accumulator are not altered. Example: STAX B
SHLD	**16-bit address**	Store H and L registers direct	The contents of register L are stored into the memory locati specified by the 16-bit address in the operand and the conte of H register are stored into the next memory location incrementing the operand. The contents of registers HL are altered. This is a 3-byte instruction, the second byte specif the low-order address and the third byte specifies the high-or

			address. Example: SHLD 2470H
XCHG	**none**	Exchange H and L with D and E	The contents of register H are exchanged with the contents register D, and the contents of register L are exchanged with t contents of register E. Example: XCHG
SPHL	**none**	Copy H and L registers to the stack pointer	The instruction loads the contents of the H and L registers i the stack pointer register, the contents of the H register prov the high-order address and the contents of the L register prov the low-order address. The contents of the and L registers are not altered. Example: SPHL
XTHL	**none**	Exchange H and L with top of stack	The contents of the L register are exchanged with the sta location pointed out by the contents of the stack pointer regist The contents of the H register are exchanged with the next sta location (SP+1); however, the contents of the stack poin register are not altered. Example: XTHL
PUSH	**Reg. pair**	Push register pair onto stack	The contents of the register pair designated in the operand copied onto the stack in the following sequence. The sta pointer register is decremented and the contents of t highorder register (B, D, H, A) are copied into that locati The stack pointer register is decremented again and the conte of the low-order register (C, E, L, flags) are copied to t location. Example: PUSH B or PUSH A
POP	**Reg. pair**	Pop off stack to register pair	The contents of the memory location pointed out by the sta pointer register are copied to the low-order register (C, E, status flags) of the operand. The stack pointer is incremented 1 and the contents of that memory location are copied to t high-order register (B, D, H, A) of the operand. The sta pointer register is again incremented by 1. Example: POP H or POP A
OUT	**8-bit port address**	Output data from accumulator to a port with 8-bit address	The contents of the accumulator are copied into the I/O p specified by the operand. Example: OUT F8H

IN	**8-bit port address**	Input data to accumulator from a port with 8-bit address	The contents of the input port designated in the operand read and loaded into the accumulator. Example: IN 8CH

UNIT 4

MACHINE AND ASSEMBLY LANGUAGE OF 8085

Posted on April 22, 2016 by Conceptuality

Word is defined as the number of bits a microprocessor can recognize.
1 byte = 2 nibbles = 8 bits.
Programs written in machine language can not be understood by most of the people, therefore, we use assembly language.
Assembly language has English-like words for a better understanding of the program to common people.
The disadvantage of assembly language is it can not be transferred from one machine to another as it is specific t a given machine.

Machine Language:-
A number of words for a machine is fixed as the number of bits for a machine are fixed.
A machine with8-bitt word length has a maximum of **256 words (2^8)**.
Instructions are formed my the microprocessor design engineer, who selects a particular combination of words to give them a specific meaning by applying the logic.
8085 is the improved version of earlier processor 8080 A.
It has a word length of 8 bits.

An **INSTRUCTION** is a binary pattern entered through an input device in memory to command the microprocessor for performing that specific function.
8085 has 246 bit patterns, amounting to 74 different instructions for performing various operations. 8085 has two more instructions than 8080 A microprocessor.
for example: 0011 1100 is equivalent to 3 C . hence, instead of writing 0011 1100 we directly write 3 C for our convenience.

Assembly Language:-
Mnemonics- is a Greek word meaning memory aid or mindful.

Both the machine language and the assembly language are considered **low level languages** for programming.
We convert the assembly language program written by us in **hexadecimal code** which is then electronically further converted into **binary code** so that computer or processor can comprehend and perform accordingly.
ASCII-American Standard Code Of Information Interchange.

hexadecimal value	decimal value
30H to 39H	0 to 9
41H to 5AH	A to Z

Writing, Assembling and Executing Assembly Language Program(ALP):-

Steps : –

1. Write the Mnemonic provided by the manufacture.
2. Find the hexadecimal Machine code for each of the instructions by searching through the set of instructions.
3. Load the program in the user memory in sequential order by using the hexadecimal keyboard peripheral.
4. Execute the program, result will be displayed on the output(LEDs, LCD or Seven Segmented Display).

Assembler is the mnemonic program that translates the entered ASCII keyboard inputs into the corresponding binary machine code for microprocessor.

Block Diagram Of a High Level Language Machine Code:-

Source Code —> Compiler/ Interpreter —> Object Code.

Interaction between the hardware and the software is managed by a set of programs called **operating system.**

Monitor Program is a code that accepts an input from the keyboard and translate it into its binary equivalent.

Hardware and Programming Model of 8085:-

A **MODEL** is a conceptual representation of a real object.

Hardware Model's Block Diagram:-

The First block includes the ALU and 8 bit register called the Accumulator, instruction decoder and flags.

The Second block includes other 8 bit and 16 bit registers which are available for user.

All the **results are stored in the Accumulator**, while making respective flip-flops (Flags) set or reset.

It has three Buses:-

8 bit bidirectional Data Bus – transfers data.

16 bit unidirectional Address Bus – send out memory Address.

Control Bus – timing signals.

Programming Model's Block Diagram:-

Registers – Six general purpose registers namely B,C ; D, E ; H, L;

They form three register pairs, so that we can perform 16 bit operations.

Register are used to store or copy data using data storing and copying instructions.

Accumulator – it is a special register having 8 bit memory space which performs the ALU operations. Normally, result is stored in this register by default.

Flags – ALU has 5 flags (flip-flops) which are set or reset after the operation, according to the data conditions stored in the registers.

Flag Register's Block Diagram:-

Namely :
D7- Sign Flag ;
D6 – Zero Flag ;
D4 – Auxiliary Carry Flag ;
D2 – Parity Flag ;
D0 – Carry Flag ;
This flags control the flow of program as they set the decisions making processes in the microprocessor.
To do this we have some mnemonics like – JC ; JNC ; JZ ; JNZ ;

Sign Flag is set when D7 is 1.
Zero Flag is set when result is zero.
Auxiliary Carry Flag is set when a carry is generated by the digit D3 and passed to D4. **No jump instruction is associated** with this flag.
Parity Flag is set when it has even number of one's in its binary equivalent.
Carry Flag is set when operation results into a carry.
Program Counter – is used to sequence the execution of the instructions. Its function is to point to the memory address from which the next byte is to be fetched.
Stack Pointer– points to a memory location in RAM.

UNIT 5

Microprocessor - I/O Interfacing Overview

Previous Page

Next Page

In this chapter, we will discuss Memory Interfacing and IO Interfacing with 8085.

Interface is the path for communication between two components. Interfacing is of two types, memory interfacing and I/O interfacing.

Memory Interfacing

When we are executing any instruction, we need the microprocessor to access the memory for reading instruction codes and the data stored in the memory. For this, both the memory and the microprocessor requires some signals to read from and write to registers.

The interfacing process includes some key factors to match with the memory requirements and microprocessor signals. The interfacing circuit therefore should be designed in such a way that it matches the memory signal requirements with the signals of the microprocessor.

IO Interfacing

There are various communication devices like the keyboard, mouse, printer, etc. So, we need to interface the keyboard and other devices with the microprocessor by using latches and buffers. This type of interfacing is known as I/O interfacing.

Block Diagram of Memory and I/O Interfacing

8085 Interfacing Pins

Following is the list of 8085 pins used for interfacing with other devices −

- A_{15} - A_8 (Higher Address Bus)
- AD_7 - AD_0(Lower Address/Data Bus)
- ALE
- RD

- WR
- READY

Ways of Communication − Microprocessor with the Outside World?

There are two ways of communication in which the microprocessor can connect with the outside world.

- Serial Communication Interface
- Parallel Communication interface

Serial Communication Interface − In this type of communication, the interface gets a single byte of data from the microprocessor and sends it bit by bit to the other system serially and vice-a-versa.

Parallel Communication Interface − In this type of communication, the interface gets a byte of data from the microprocessor and sends it bit by bit to the other systems in simultaneous (or) parallel fashion and vice-a-versa.

In a previous tutorial we looked at the digital **Not Gate** commonly called an inverter, and we saw that the NOT gates output state is the complement, opposite or inverse of its input signal.

So for example, when the single input to NOT gate is "HIGH", its output state will NOT be "HIGH". When its input signal is "LOW" its output state will NOT be "LOW", in other words it "inverts" its input signal, hence the name "Inverter".

But sometimes in digital electronic circuits we need to isolate logic gates from each other or have them drive or switch higher than normal loads, such as relays, solenoids and lamps without the need for inversion. One type of single input logic gate that allows us to do just that is called the **Digital Buffer**.

Unlike the single input, single output inverter or NOT gate such as the TTL 7404 which inverts or complements its input signal on the output, the "Buffer" performs no inversion or decision making capabilities (like logic gates with two or more inputs) but instead produces an output which exactly matches that of its input. In other words, a digital buffer does nothing as its output state equals its input state.

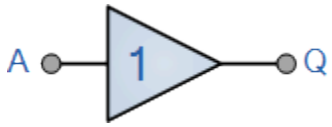
Then digital buffers can be regarded as Idempotent gates applying Boole's Idempotent Law because when an input passes through this device its value is not changed. So the digital buffer is a "non-inverting" device and will therefore give us the Boolean expression of: Q = A.

Then we can define the logical operation of a single input digital buffer as being:

"Q is true, only when A is true"

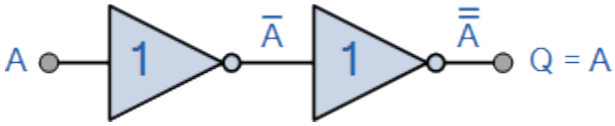
In other words, the output (Q) state of a buffer is only true (logic "1") when its input A is true, otherwise its output is false (logic "0").

The Single Input Digital Buffer

Symbol	Truth Table	
 The Digital Buffer	A	Q
	0	0
	1	1
Boolean Expression **Q = A**	Read as: **A** gives **Q**	

The **Digital Buffer** can also be made by connecting together two NOT gates as shown below. The first will "invert" the input signal A and the second will "re-invert" it back to its original level performing a double inversion of the input.

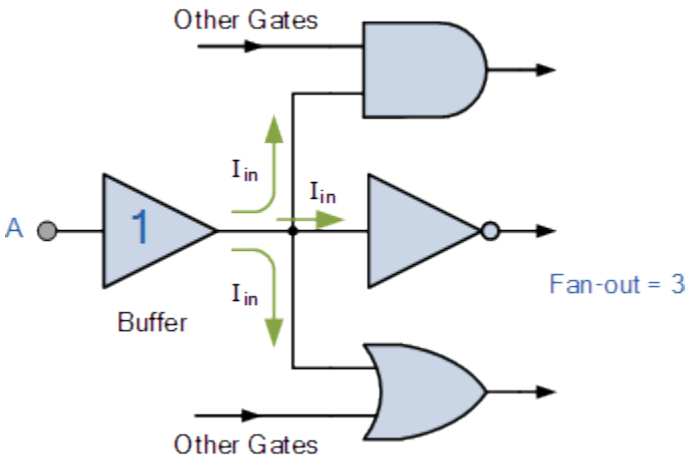
Double Inversion using NOT Gates



You may be thinking, well what's the point of a Digital Buffer if it does not invert or alter its input signal in any way, or make any logical decisions or operations like the AND or ORgates do, then why not just use a piece of wire instead, and that's a good point. But a non-inverting Digital Buffer does have many uses in digital electronics with one of its main advantages being that it provides digital amplification.

Digital Buffers can be used to isolate other gates or circuit stages from each other preventing the impedance of one circuit from affecting the impedance of another. A digital buffer can also be used to drive high current loads such as transistor switches because their output drive capability is generally much higher than their input signal requirements. In other words buffers can be used for power amplification of a digital signal as they have what is called a high "fan-out" capability.

Digital Buffer Fan-out Example



The **Fan-out** parameter of a buffer (or any digital IC) is the output driving capability or output current capability of a logic gate giving greater power amplification of the input signal. It may be necessary to connect more than just one logic gate to the output of another or to switch a high current load such as an LED, then a Buffer will allow us to do just that.

Generally the output of a logic gate is usually connected to the inputs of other gates. Each input requires a certain amount of current from the gate output to change state, so that each additional gate connection adds to the load of the gate. So the fan-out is the number of parallel loads that can be driven simultaneously by one digital buffer of logic gate. Acting as a current source a buffer can have a high fan-out rating of up to 20 gates of the same logic family.

If a digital buffer has a high fan-out rating (current source) it must also have a high "fan-in" rating (current sink) as well. However, the propagation delay of the gate deteriorates rapidly as a function of fan-in so gates with a fan-in greater than 4 should be avoided.

Then there is a limit to the number of inputs and outputs than can be connected together and in applications where we need to decouple gates from each other, we can use a **Tri-state Buffer** or tristate output driver.

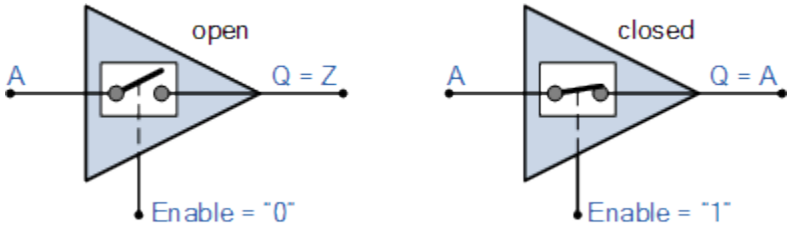
The "Tri-state Buffer"

As well as the standard Digital Buffer seen above, there is another type of digital buffer circuit whose output can be "electronically" disconnected from its output circuitry when required. This type of Buffer is known as a 3-State Buffer or more commonly a Tri-state Buffer.

A Tri-state Buffer can be thought of as an input controlled switch with an output that can be electronically turned "ON" or "OFF" by means of an external "Control" or "Enable" (EN) signal input. This control signal can be either a logic "0" or a logic "1" type signal resulting in the Tri-state Buffer being in one state allowing its output to operate normally producing the required output or in another state were its output is blocked or disconnected.

Then a tri-state buffer requires two inputs. One being the data input and the other being the enable or control input as shown.

Tri-state Buffer Switch Equivalent

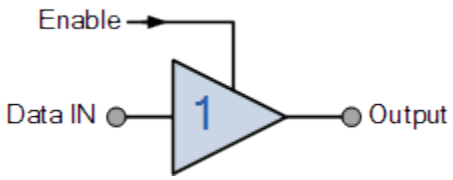


When activated into its third state it disables or turns "OFF" its output producing an open circuit condition that is neither at a logic "HIGH" or "LOW", but instead gives an output state of very high impedance, **High-Z**, or more commonly Hi-Z. Then this type of device has two logic state inputs, "0" or a "1" but can produce three different output states, "0", "1" or " Hi-Z " which is why it is called a "Tri" or "3-state" device.

Note that this third state is NOT equal to a logic level "0" or "1", but is an high impedance state in which the buffers output is electrically disconnected from the rest of the circuit. As a result, no current is drawn from the supply.

There are four different types of Tri-state Buffer, one set whose output is enabled or disabled by an "**Active-HIGH**" control signal producing an inverted or non-inverted output, and another set whose buffer output is controlled by an "**Active-LOW**" control signal producing an inverted or non-inverted output as shown below.

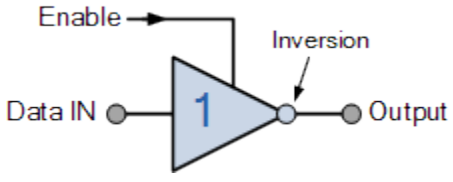
Active "HIGH" Tri-state Buffer

Symbol	Truth Table		
 Tri-state Buffer	Enable	IN	OUT
	0	0	Hi-Z
	0	1	Hi-Z
	1	0	0
	1	1	1
Read as Output = Input if Enable is equal to "1"			

An **Active-high** Tri-state Buffer such as the 74LS241 octal buffer, is activated when a logic level "1" is applied to its "enable" control line and the data passes through from its input to its output. When the enable control line is at logic level "0", the buffer output is disabled and a high impedance condition, Hi-Z is present on the output.

An active-high tri-state buffer can also have an inverting output as well as its high impedance state creating an active-high tri-state inverting buffer as shown.

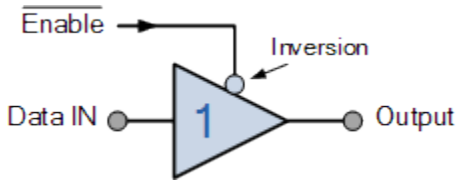
Active "HIGH" Inverting Tri-state Buffer

Symbol	Truth Table		
 Inverting Tri-state Buffer	Enable	IN	OUT
	0	0	Hi-Z
	0	1	Hi-Z
	1	0	1
	1	1	0
Read as Output = Inverted Input if Enable equals "1"			

The output of an active-high inverting tri-state buffer, such as the 74LS240 octal buffer, is activated when a logic level "1" is applied to its "enable" control line. The data at the input is passes through to the output but is inverted producing a complement of the input. When the enable line is LOW at logic level "0", the buffer output is disabled and at a high impedance condition, Hi-Z.

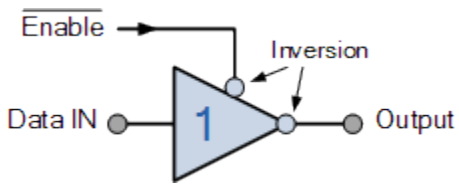
The same two tri-state buffers can also be implemented with an active-low enable input as shown.

Active "LOW" Tri-state Buffer

Symbol	Truth Table		
 Tri-state Buffer	Enable	IN	OUT
	0	0	0
	0	1	1
	1	0	Hi-Z
	1	1	Hi-Z
Read as Output = Input if Enable is **NOT** equal to "1"			

An **Active-low** Tri-state Buffer is the opposite to the above, and is activated when a logic level "0" is applied to its "**enable**" control line. The data passes through from its input to its output. When the enable control line is at logic level "1", the buffer output is disabled and a high impedance condition, Hi-Z is present on the output.

Active "LOW" Inverting Tri-state Buffer

Symbol	Truth Table		
 Inverting Tri-state Buffer	Enable	IN	OUT
	0	0	1
	0	1	0
	1	0	Hi-Z
	1	1	Hi-Z

Read as Output = Inverted Input if Enable is **NOT** equal to "1"

An **Active-low** Inverting Tri-state Buffer is the opposite to the above as its output is enabled or disabled when a logic level "0" is applied to its "**enable**" control line. When a buffer is enabled by a logic "0", the output is the complement of its input. When the enable control line is at logic level "1", the buffer output is disabled and a high impedance condition, Hi-Z is present on the output.

Tri-state Buffer Control

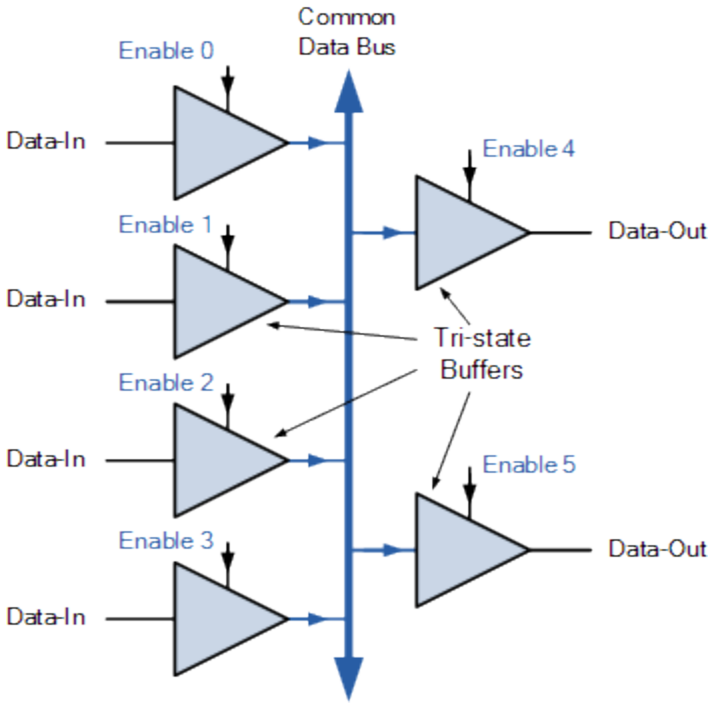
We have seen above that a buffer can provide voltage or current amplification within a digital circuit and it can also be used to invert the input signal. We have also seen that digital buffers are available in the tri-state form that allows the output to be effectively switched-off producing a high impedance state (Hi-Z) equivalent to an open circuit.

The Tri-state Buffer is used in many electronic and microprocessor circuits as they allow multiple logic devices to be connected to the same wire or bus without damage or loss of data. For example, suppose we have a data line or data bus with some memory, peripherals, I/O or a CPU connected to it. Each of these devices is capable of sending or receiving data to each other onto this single data bus at the same time creating what is called a contention.

Contention occurs when multiple devices are connected together because some want to drive their output high and some low. If these devices start to send or receive data at the same time a short circuit may occur when one device outputs to the bus a logic "1", the supply voltage, while another is set at logic level "0" or ground, resulting in a short circuit condition and possibly damage to the devices as well as loss of data.

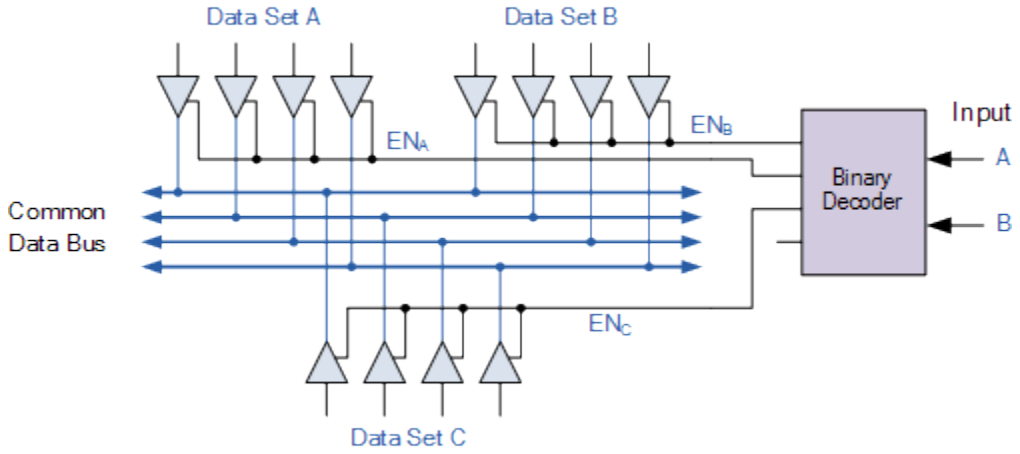
Digital information is sent over these data buses or data highways either serially, one bit at a time, or it may be up to eight (or more) wires together in a parallel form such as in a microprocessor data bus allowing multiple tri-state buffers to be connected to the same data highway without damage or loss of data as shown.

Tri-state Buffer Data Bus Control



Then, the **Tri-state Buffer** can be used to isolate devices and circuits from the data bus and one another. If the outputs of several Tri-state Buffers are electrically connected together Decoders are used to allow only one set of Tri-state Buffers to be active at any one time while the other devices are in their high impedance state. An example of Tri-state Buffers connected to a 4-wire data bus is shown below.

Tri-state Buffer Control



This basic example shows how a binary decoder can be used to control a number of tri-state buffers either individually or together in data sets. The decoder selects the appropriate output that corresponds to its binary input allowing only one set of data to pass either a logic “1” or logic “0” output state onto the bus. At this time all the other tri-state outputs connected to the same bus lines are disabled by being placed in their high impedance Hi-Z state.

Then data from data set “A” can only be transferred to the common bus when an active HIGH signal is applied to the tri-state buffers via the Enable line, EN_A. At all other times it represents a high impedance condition effectively being isolated from the data bus.

Likewise, data set “B” only passes data to the bus when an enable signal is applied via EN_B. A good example of tri-state buffers connected together to control data sets is the TTL 74244 Octal Buffer.

It is also possible to connect Tri-state Buffers “back-to-back” to produce what is called a **Bi-directional Buffer** circuit with one “active-high buffer” connected in parallel but in reverse with one “active-low buffer”.

Here, the “enable” control input acts more like a directional control signal causing the data to be both read “from” and transmitted “to” the same data bus wire. In this type of application a tri-state buffer with bi-directional switching capability such as the TTL 74245 can be used.

We have seen that a **Tri-state buffer** is a non-inverting device which gives an output (which is same as its input) only when the input to the Enable, (EN) pin is HIGH otherwise the output of the buffer goes into its high impedance, (Hi-Z) state. Tri-state outputs are used in many integrated circuits and digital systems and not just in digital tristate buffers.

Both digital buffers and tri-state buffers can be used to provide voltage or current amplification driving much high loads such as relays, lamps or power transistors than with conventional logic gates. But a buffer can also be used to provide electrical isolation between two or more circuits.

We have seen that a data bus can be created if several tristate devices are connected together and as long as only one is selected at any one time, there is no problem. Tri-state buses allow several digital devices to input and output data on the same data bus by using I/O signals and address decoding.

Tri-state Buffers are available in integrated form as quad, hex or octal buffer/drivers in both uni-directional and bi-directional forms, with the more common being the TTL 74240, the TTL 74244 and the TTL 74245 as shown.

Commonly available **Digital Buffer** and **Tri-state Buffer** IC’s include:

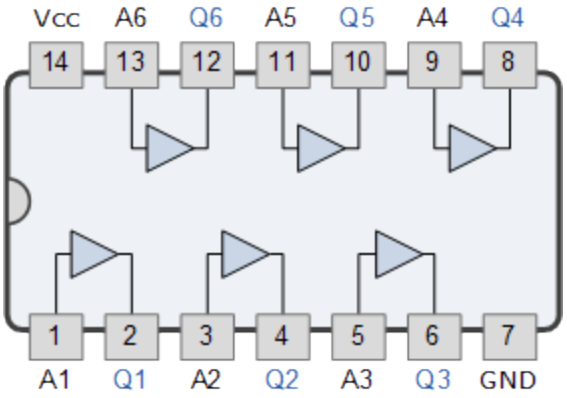
TTL Logic Digital Buffers

- 74LS07 Hex Non-inverting Buffer
- 74LS17 Hex Buffer/Driver
- 74LS244 Octal Buffer/Line Driver
- 74LS245 Octal Bi-directional Buffer

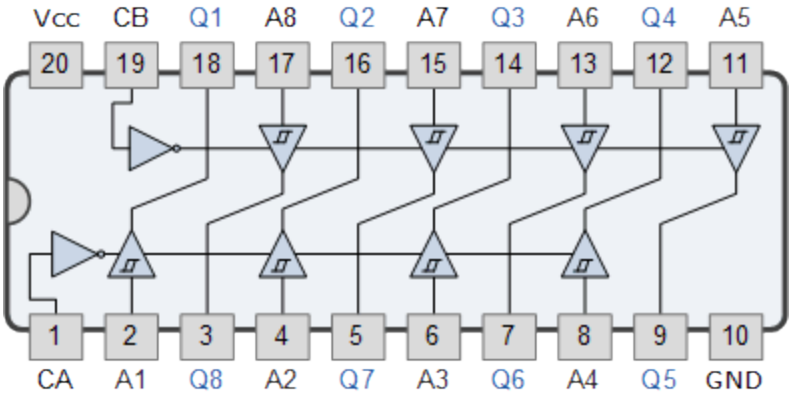
CMOS Logic Digital Buffers

- CD4050 Hex Non-inverting Buffer
- CD4503 Hex Tri-state Buffer
- HEF40244 Tri-state Octal Buffer

74LS07 Digital Buffer



74LS244 Octal Tri-state Buffer



"Memory buffer" redirects here. It is not to be confused with memory buffer register.

In computer science, a **data buffer** (or just **buffer**) is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another. Typically, the data is stored in a buffer as it is retrieved from an input device (such as a microphone) or just before it is sent to an output device (such as speakers). However, a buffer may be used when moving data between processes within a computer. This is comparable to buffers in telecommunication. Buffers can be implemented in a fixed memory location in hardware—or by using a virtual data buffer in software, pointing at a location in the physical memory. In all cases, the data stored in a data buffer are stored on a physical storage medium. A majority of buffers are implemented in software, which typically use the faster RAM to store temporary data, due to the much faster access time compared with hard disk drives. Buffers are typically used when there is a difference between the rate at which data is received and the rate at which it can be processed, or in the case that these rates are variable, for example in a printer spooler or in online video streaming.

A buffer often adjusts timing by implementing a queue (or FIFO) algorithm in memory, simultaneously writing data into the queue at one rate and reading it at another rate.

Contents

Applications[edit]

Buffers are often used in conjunction with I/O to hardware, such as disk drives, sending or receiving data to or from a network, or playing sound on a speaker. A line to a rollercoasterin an amusement park shares many similarities. People who ride the coaster come in at an unknown and often variable pace, but the roller coaster will be able to load people in bursts (as a coaster arrives and is loaded). The queue area acts as a buffer—a temporary space where those wishing to ride wait until the ride is available. Buffers are usually used in a FIFO (first in, first out) method, outputting data in the order it arrived.

Buffers can increase application performance by allowing synchronous operations such as file reads or writes to complete quickly instead of blocking while waiting for hardware interrupts to access a physical disk subsystem; instead, an operating system can immediately return a successful result from an API call, allowing an application to continue processing while the kernel completes the disk operation in the background. Further benefits can be achieved if the application is reading or writing small blocks of data that do not correspond to the block size of the disk subsystem, allowing a buffer to be used to aggregate many smaller read or write operations into block sizes that are more efficient for the disk subsystem, or in the case of a read, sometimes to completely avoid having to physically access a disk.

Telecommunication buffer[edit]

A buffer routine or storage medium used in telecommunications compensates for a difference in rate of flow of data, or time of occurrence of events, when transferring data from one device to another.

Buffers are used for many purposes, including:

- Interconnecting two digital circuits operating at different rates,
- Holding data for later use,
- Allowing timing corrections to be made on a data stream,
- Collecting binary data bits into groups that can then be operated on as a unit,
- Delaying the transit time of a signal in order to allow other operations to occur.

Examples[edit]

- The BUFFERS command/statement in CONFIG.SYS of DOS.
- The buffer between a serial port (UART) and a MODEM. The COM port speed may be 38400 bit/s while the MODEM may have only a 14400 bit/s carrier.
- The integrated buffer on a Hard Disk Drive, Printer or other piece of hardware.
- The Framebuffer on a video card.

History[edit]

An early mention of a print buffer is the Outscriber devised by image processing pioneer Russel A. Kirsch for the SEAC computer in 1952:[1]

One of the most important problems in the design of automatic digital computers is that of getting the calculated results out of the machine rapidly enough to avoid delaying the further progress of the calculations. In many of the problems to which a general-purpose computer is applied the amount of output data is relatively big —so big that serious inefficiency would result from forcing the computer to wait for these data to be typed on existing printing devices. This difficulty has been solved in the SEAC by providing magnetic recording devices as output units. These devices are able to receive information from the machine at rates up to 100 times as fast as an electric typewriter can be operated. Thus, better efficiency is achieved in recording the output data; transcription can be made later from the magnetic recording device to a printing device without tying up the main computer.

See also[edit]

- Bucket (computing)
- Buffer overflow
- Buffer underrun
- Circular buffer
- Disk buffer
- Streaming media
- Frame buffer for use in graphical display
- Double buffering and Triple buffering for techniques mainly in graphics
- Depth buffer, Stencil buffer, for different parts of image information
- Variable length buffer
- Optical buffer
- MissingNo., the result of buffer data not being cleared properly in Pokémon Red and Blue
- UART buffer
- ENOBUFS, POSIX error caused by lack of memory in buffers
- Write buffer, a type of memory buffer
- 512k day

Interrupts in 8085 microprocessor

When microprocessor receives any interrupt signal from peripheral(s) which are requesting its services, it stops its current execution and program control is transferred to a sub-routine by generating **CALL** signal and after executing sub-routine by generating **RET**signal again program control is transferred to main program from where it had stopped.
When microprocessor receives interrupt signals, it sends an acknowledgement (INTA) to the peripheral which is requesting for its service.

Interrupts can be classified into various categories based on different parameters:

1. **Hardware and Software Interrupts –**
   When microprocessors receive interrupt signals through pins (hardware) of microprocessor, they are known as *Hardware Interrupts*. There are 5 Hardware Interrupts in 8085 microprocessor. They are – *INTR, RST 7.5, RST 6.5, RST 5.5, TRAP*
   Software Interrupts are those which are inserted in between the program which means these are mnemonics of microprocessor. There are 8 software interrupts in 8085 microprocessor. They are – *RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7.*

2. **Vectored and Non-Vectored Interrupts –**
   Vectored Interrupts are those which have fixed vector address (starting address of sub-routine) and after executing these, program control is transferred to that address.
   Vector Addresses are calculated by the formula 8 * TYPE

INTERRUPT	VECTOR ADDRESS
TRAP (RST 4.5)	24 H
RST 5.5	2C H
RST 6.5	34 H
RST 7.5	3C H

3. For Software interrupts vector addresses are given by:

INTERRUPT	VECTOR ADDRESS
RST 0	00 H
RST 1	08 H
RST 2	10 H
RST 3	18 H
RST 4	20 H
RST 5	28 H

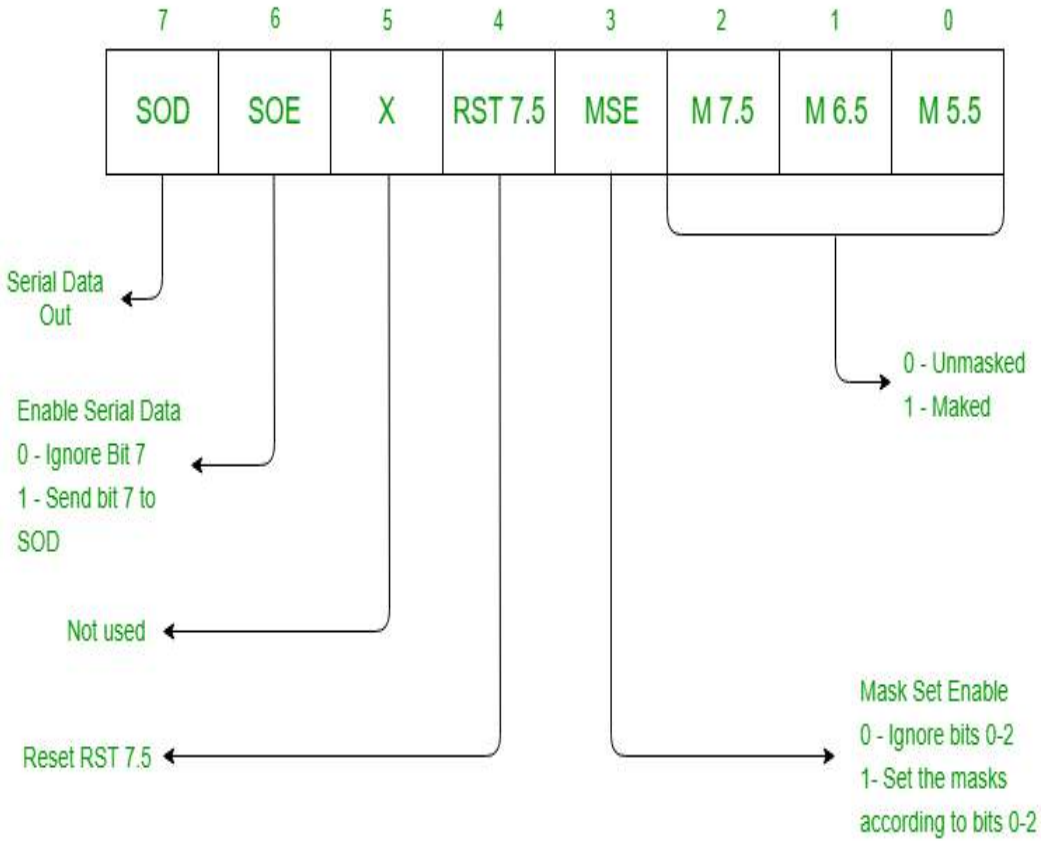
INTERRUPT	VECTOR ADDRESS
RST 6	30 H
RST 7	38 H

4. *Non-Vectored Interrupts* are those in which vector address is not predefined. The interrupting device gives the address of sub-routine for these interrupts. *INTR* is the only non-vectored interrupt in 8085 microprocessor.
5. **Maskable and Non-Maskable Interrupts –**
   Maskable Interrupts are those which can be disabled or ignored by the microprocessor. These interrupts are either edge-triggered or level-triggered, so they can be disabled. *INTR, RST 7.5, RST 6.5, RST 5.5* are maskable interrupts in 8085 microprocessor.
   Non-Maskable Interrupts are those which cannot be disabled or ignored by microprocessor. *TRAP* is a non-maskable interrupt. It consists of both level as well as edge triggering and is used in critical power failure conditions.

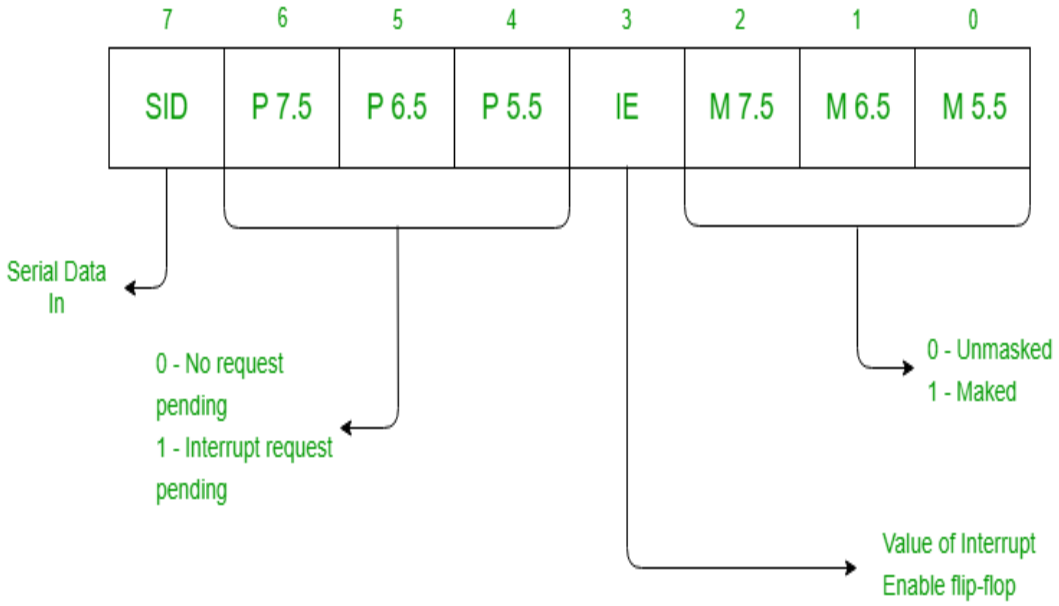
Priority of Interrupts –
When microprocessor receives multiple interrupt requests simultaneously, it will execute the interrupt service request (ISR) according to the priority of the interrupts.

Instruction for Interrupts –

1. **Enable Interrupt (EI) –** The interrupt enable flip-flop is set and all interrupts are enabled following the execution of next instruction followed by EI. No flags are affected. After a system reset, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to enable the interrupts again (except TRAP).
2. **Disable Interrupt (DI) –** This instruction is used to reset the value of enable flip-flop hence disabling all the interrupts. No flags are affected by this instruction.
3. **Set Interrupt Mask (SIM) –** It is used to implement the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by setting various bits to form masks or generate output data via the Serial Output Data (SOD) line. First the required value is loaded in accumulator then SIM will take the bit pattern from it.



4. **Read Interrupt Mask (RIM) –** This instruction is used to read the status of the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by loading into the A register a byte which defines the condition of the mask bits for the interrupts. It also reads the condition of SID (Serial Input Data) bit on the microprocessor.


7
6
5
4
3
2
1
0
SID
P 7.5
P 6.5
P 5.5
IE
M 7.5
M 6.5
M 5.5
Serial Data In
0 - No request pending
1 - Interrupt request pending
0 - Unmasked
1 - Maked
Value of Interrupt Enable flip-flop

Unit 6

Interrupts In 8085

what is Interrupt?

Interrupt is a mechanism by which an I/O or an instruction can suspend the normal execution of processor and get itself serviced. Generally, a particular task is assigned to that interrupt signal. In the microprocessor based system the interrupts are used for data transfer between the peripheral devices and the microprocessor.

Interrupt Service Routine(ISR)

A small program or a routine that when executed services the corresponding interrupting source is called as an ISR.

Maskable/Non-Maskable Interrupt

An interrupt that can be disabled by writing some instruction is known as Maskable Interrupt otherwise it is called Non-Maskable Interrupt.

There are 6 pins available in 8085 for interrupt:

---

1. TRAP
2. RST 7.5
3. RST6.5
4. RST5.5
5. INTR
6. INTA

Execution of Interrupts

When there is an interrupt requests to the Microprocessor then after accepting the interrupts Microprocessor send the INTA (active low) signal to the peripheral. The vectored address of particular interrupt is stored in program counter. The processor executes an interrupt service routine (ISR) addressed in program counter.

There are two types of interrupts used in 8085 Microprocessor:

1. Hardware Interrupts
2. Software Interrupts

Software Interrupts

A software interrupts is a particular instructions that can be inserted into the desired location in the rpogram. There are eight Software interrupts in 8085 Microprocessor. From RST0 to RST7.

1. RST0
2. RST1
3. RST2
4. RST3
5. RST4
6. RST5
7. RST6
8. RST7

They allow the microprocessor to transfer program control from the main program to the subroutine program. After completing the subroutine program, the program control returns back to the main program.
We can calculate the vector address of these interrupts using the formula given below:

Vector Address = Interrupt Number * 8

For Example:

RST2: vector address=2*8 = 16

RST1: vector address=1*8 = 08

RST3: vector address=3*8 = 24

Vector address table for the software interrupts:

Interrupt	**Vector Address**
RST0	0000_H
RST1	0008_H

RST2 RST3	0010_H 0018_H
RST4 RST5	0020_H 0028_H
RST6 RST7	0030_H 0038_H

Hardware Interrupt

As i have already discussed that there are 6 interrupt pins in the microprocessor used as Hardware Interrrupts given below:

1. TRAP
2. RST7.5
3. RST6.5
4. RST5.5
5. INTR

Note:

INTA is not an interrupt. INTA is used by the Microprocessor for sending the acknowledgement.TRAP has highest priority and RST7.5 has second highest priority and so on.

The Vector address of these interrupts are given below:

Interrupt	**Vector Address**
RST7.5	$003C_H$

RST6.5	0034_H
RST5.5	$002C_H$
TRAP	0024_H

TRAP

It is non maskable edge and level triggered interrupt. TRAP has the highest priority and vectores interrupt. Edge and level triggered means that the TRAP must go high and remain high until it is acknowledged. In case of sudden power failure, it executes a ISR and send the data from main memory to backup memory.
As we know that TRAP can not be masked but it can be delayed using HOLD signal. This interrupt transfers the microprocessor's control to location 0024H.
TRAP interrupts can only be masked by reseting the microprocessor. There is no other way to mask it.

RST7.5

It has the second highest priority. It is maskable and edge level triggered interrupt. The vector address of this interrupt is 003CH. Edge sensitive means input goes high and no need to maintain high state until it is recognized.
It can also be reset or masked by reseting microprocessor. It can also be resetted by DI instruction.

RST6.5 and RST5.5

These are level triggered and maskable interrupts. When RST6.5 pin is at logic 1, INTE flip-flop is set. RST 6.5 has third highest priority and RST 5.5 has fourth highest priority.

It can be masked by giving DI and SIM instructions or by reseting microprocessor.

INTR

It is level triggered and maskable interrupt. The following sequence of events occurs when INTR signal goes high:

1. The 8085 checks the status of INTR signal during execution of each instruction.
2. If INTR signal is high, then 8085 complete its current instruction and sends active low interrupt acknowledge signal, if the interrupt is enabled.
3. On receiving the instruction, the 8085 save the address of next instruction on stack and execute received instruction.

It has the lowest priority. It can be disabled by reseting the microprocessor or by DI and SIM instruction.

Programmed I/O Data Transfer scheme of 8085 microprocessor

BY SUBHAM · PUBLISHED FEBRUARY 16, 2018 · UPDATED FEBRUARY 16, 2018

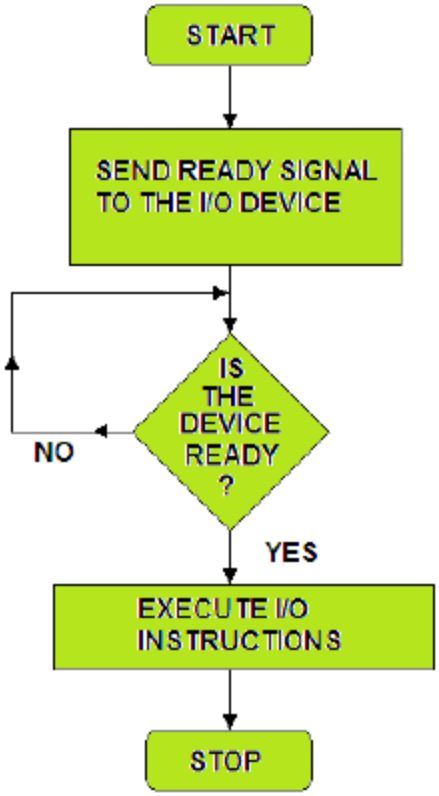
Programmed I/O Data Transfer scheme of 8085 microprocessor is a simple parallel data transfer scheme. This method of data transfer is generally used in the simple microprocessor systems. It is obvious that where speed is unimportant. This method uses instructions to get the data into or out of the microprocessor. Programmed I/O Data Transfer scheme of 8085 microprocessor can be work on synchronous or asynchronous mode. The data transfer can be synchronous or asynchronous it completely depends upon the type and the speed of the I/O devices.

Synchronous type of data transfer

Synchronous type of data transfer can be used when the speed of the I/O devices matches with the speed of the 8085 microprocessor. So for synchronization established between I/O device and microprocessor we need common clock pulse. This common clock pulse synchronizes the microprocessor and the I/O devices. Synchronous type of data transfer scheme because of the matching of the speed, the microprocessor does not have to wait for the availability of the data. The microprocessor immediately sends data for the transfer as soon as the microprocessor issues a signal.

The asynchronous data transfer

The asynchronous data transfer method is used when the speed of the I/O devices is slower than the speed of the microprocessor. Because of the mismatch of the speed, the internal timing of the I/O device is independent from the microprocessor. That is why two units are said to be asynchronous to each other. The asynchronous data transfer is normally implemented using ‘handshaking’ mode. Now question is what is handshaking mode? In the handshaking mode some signals are exchanged between the I/O device and microprocessor before the data transfer takes place.



By this handshaking the microprocessor has to check the status to the input/output device. Now if the device is ready for the data transfer or not.

- First step of microprocessor is initiates the I/O device to get ready.
- Then status of the I/O device is continuously checked by the microprocessor.
- This process remain continues until the I/O device becomes ready.
- After that microprocessor sends instructions to transfer the data.

Direct Memory Access (DMA) Data Transfer

BY SUBHAM · PUBLISHED MARCH 6, 2018 · UPDATED MARCH 6, 2018

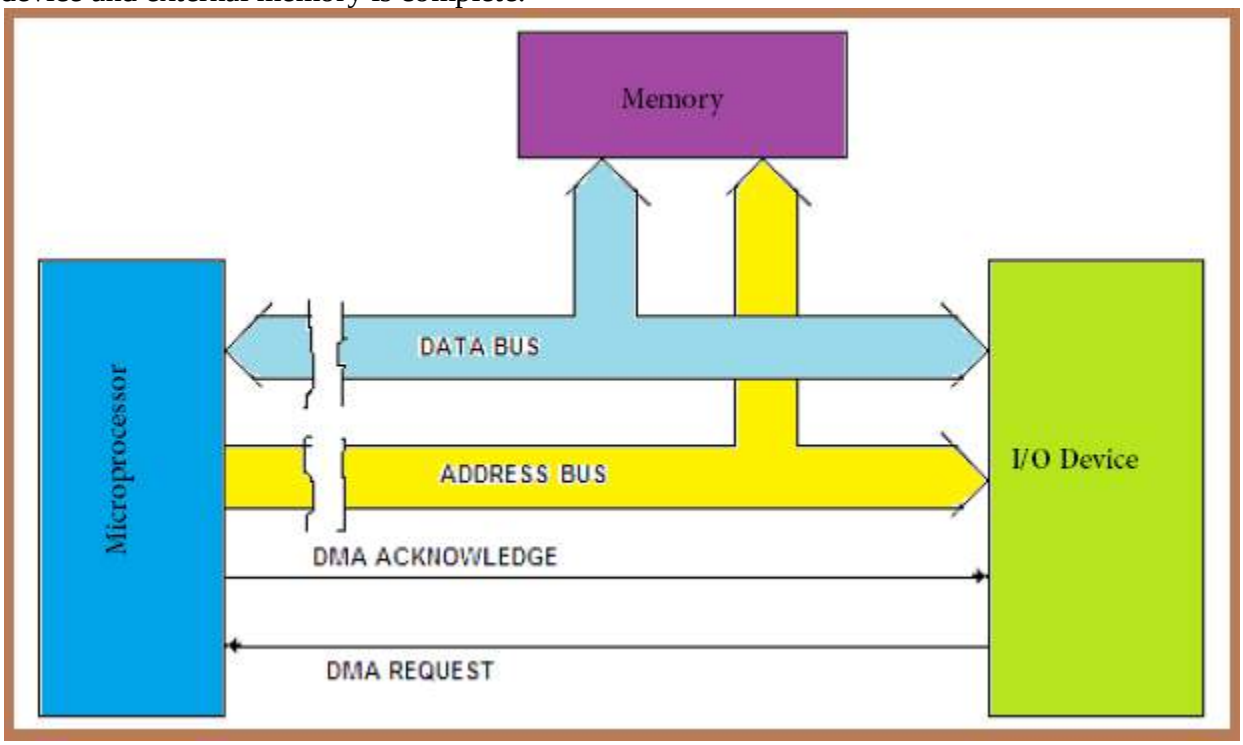
Here we come with the discussion on direct memory access data transfer. In my early post I discussed on the topic "**Programmed I/O Data Transfer scheme of 8085 microprocessor**" and "**Interrupt Driven I/O Data Transfer**". As we discussed earlier that in programmed I/O or interrupt driven I/O methods of data transfer between the I/O devices and external memory is via the accumulator. Now think for bulk data transfer from I/O devices to memory or vice-versa, these two methods discussed above are time consuming and quite uneconomical even though the speed of I/O devices matches with the speed of microprocessor. Because in those methods the data is first transferred to accumulator and then to concerned device.

To overcome those problem direct memory access data transfer method is introduced. The Direct Memory Access (DMA) data transfer method is used for bulk data transfer from I/O devices to microprocessor or vice-versa. In this method I/O devices are allowed to transfer the data directly

to the external memory without being routed through accumulator. For this reason the microprocessor relinquishes the control over the data bus and address bus, so that these can be used for transfer of data between the devices.

Working principle of direct memory access data transfer

So now come to working principle of direct memory access data transfer. For the data transfer using DMA process, a request to the microprocessor in form of HOLD signal, by the I/O device is sent. When microprocessor receipt of such request, the microprocessor relinquishes the address and data buses and informs the I/O devices of the situation by sending Acknowledge signal HLDA. The I/O device withdraws the request when the data transfer between the I/O device and external memory is complete.



If we discuss in brief about working principal of DMA controller. Then we should mention that DMA controller is used with the microprocessor that helps to generate the addresses for the data to be transferred from the I/O devices. The peripheral device sends the request signal (DMARQ) to the DMA controller and the DMA controller in turn passes it to the microprocessor (HOLD signal). On receipt of the DMA request the microprocessor sends an acknowledge signal (HLDA) to the DMA controller. On receipt of this signal (HLDA) the DMA controller sends a DMA acknowledge signal (DMACK) to the I/O device. The DMA controller then takes over the control of the buses of microprocessor and controls the data transfer between RAM and I/O device. When the data transfer is complete, DMA controller returns the control over the buses to the microprocessor by disabling the HOLD and DMACK signals.

Now question is how many way DMA can work? It may be mentioned here that DMA transfer the data of the following types:

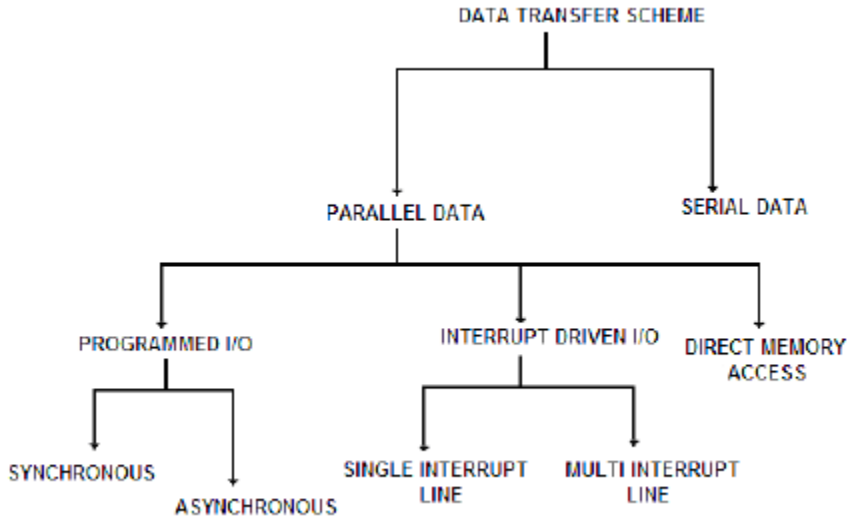
- Memory to I/O device
- I/O device to memory
- Memory to memory
- I/O device to I/O device

Data transfer schemes of 8085 microprocessor

BY SUBHAM · PUBLISHED FEBRUARY 15, 2018 · UPDATED MARCH 6, 2018

Here we can see data transfer schemes of 8085 microprocessor. In 8085 microprocessor based systems several input and output devices are connected. We know that data transfer may take place between microprocessor and memory, microprocessor and I/O devices and memory & I/O devices. As we know not much of the problems arise for the data communication between microprocessor and memory as same technology is used in the manufacturing of memory and microprocessor.

The main reason for that the speed of the memory is almost compatible with the speed of 8085 microprocessor. Now the main concern is for the data transfer between the microprocessor and I/O devices. The main problems arise due to mismatch of the speed of the I/O devices and the speed of microprocessor or memory. To overcome this problem of speed mismatch between the microprocessor and I/O devices we have to do something. For that reason only we introduce data transfer schemes of 8085 microprocessor. So following data transfer schemes may be considered for smooth data transfer process. The data transfer schemes of 8085 microprocessor were categorised depending upon the capabilities of I/O devices for accepting serial or parallel data.



The 8085 microprocessor is a parallel device. That means it transfers eight bits of data simultaneously over eight data lines (parallel I/O mode). However in many situations, the

parallel I/O mode is either impractical or impossible. For example, parallel data communication over a long distance becomes very expensive. Similarly, parallel data communication is not possible with devices such as CRT terminal or Cassette tape etc.

Serial I/O mode transfer

For these devices and for these reasons serial I/O mode is used. In serial I/O mode transfer a single bit of data on a single line at a time. For serial I/O data transmission mode, 8-bit parallel word is converted to a stream of eight serial bit using parallel-to-serial converter. Similarly, in serial reception of data, the microprocessor receives a stream of 8-bit one by one which are then converted to 8- bit parallel word using serial-to-parallel converter. For this purpose data transfer schemes of 8085 microprocessor are introduced.

Parallel data transfer scheme

Parallel data transfer scheme is faster than serial I/O transfer. in parallel data transfer 8-bit data send all together with 8 parallel wire. In 8085 microprocessor mainly three types of parallel data transfer scheme we observed. Those are

- **Programmed I/O Data Transfer**
- **Interrupt Driven I/O Data Transfer**
- **Direct Memory Access (DMA) Data Transfer.**

Hope in above discussion on data transfer schemes of 8085 microprocessor, it is cleared that how data is transfer between microprocessor to memory and microprocessor to I/O devices.

Unit 8

8155/8156: Programmable I/O Ports and Timer

1. In what way 8155 and 8156 differs?

Ans. The Chip Enable (CE) signal is active low for 8155, whereas it is active high for 8156.

2.What are the essential features of 8155.

Ans. The essential features of 8155 are

*8-bit 256 word RAM memory

*Two programmable 8-bit I/O port

* One programmable 6-bit IO port

*One programmable 14-bit binary Timer/Counter

* An internal address latch

*A control/status (C/S) register

* An internal decoder.

3. Functionally, how many sections are there in 8155?

Ans. Functionally, it has two sections—(*a*) a R/W memory and (*b*) programmable I/O and timer section.

4. Is it necessary to demultiplex the lower order bus AD7 –AD0externally for 8155 to be connected to 8055?

Ans. No, it is not. This is because ALE, IO/ M , RD and WR signals of 8085 can be connected directly with 8155.

5. Draw the pin diagram of 8155.

Ans. The pin diagram of 8155 is shown in Fig. 9b.1.

6. Draw the functional block diagram of 8155.

Ans. The functional block diagram of 8155 is shown in Fig. 9b.2 :

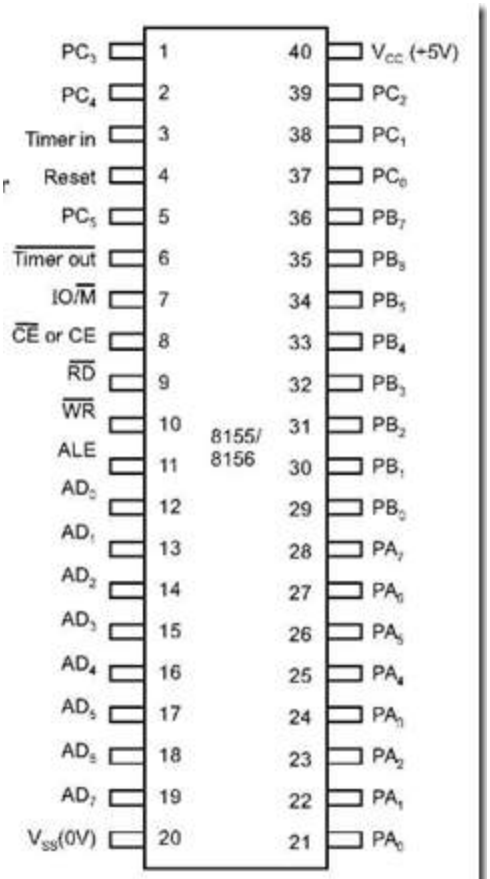


Fig. 9b.1: Pin diagram of 8155 (*Source:* Intel Corporation)

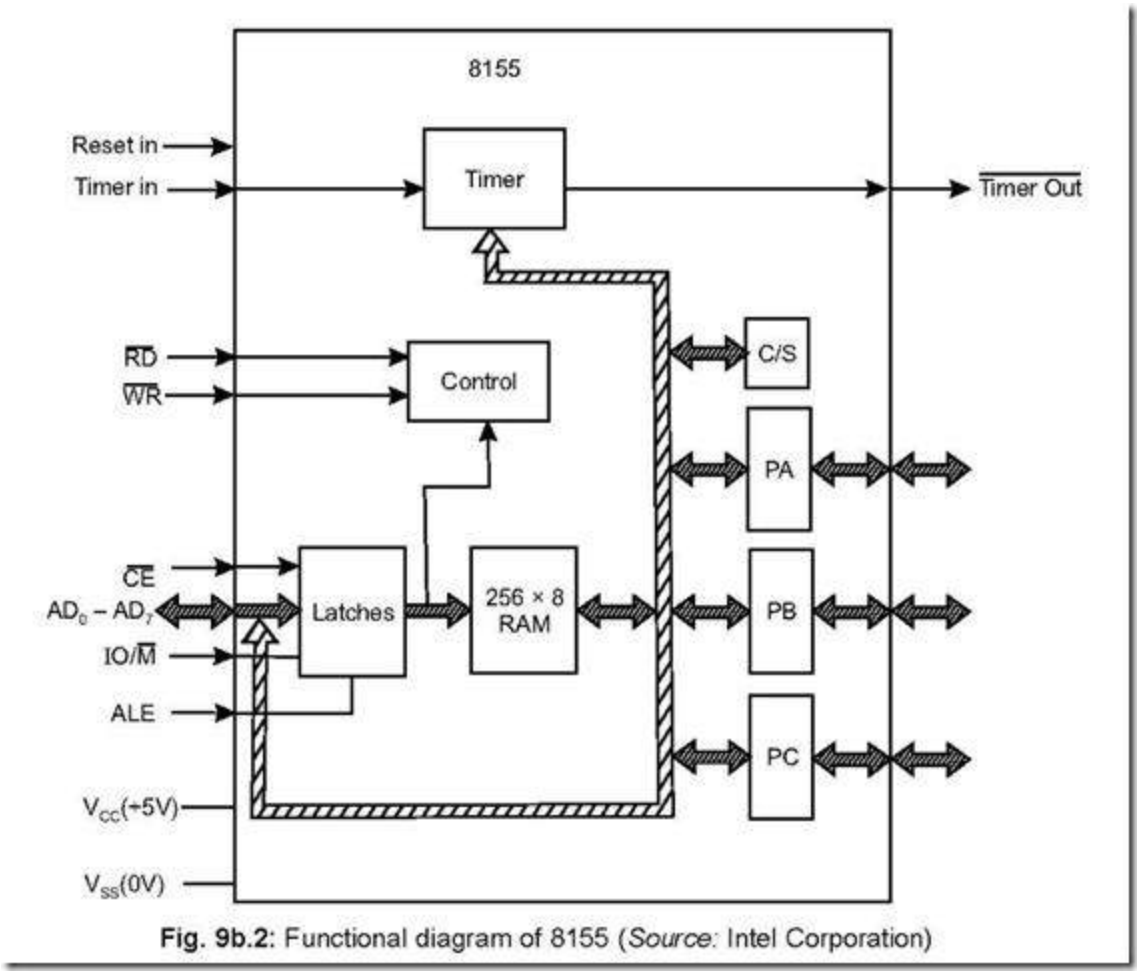


Fig. 9b.2: Functional diagram of 8155 (*Source:* Intel Corporation)

7. How the different ports, control/status register, timers are accessed? Write their addresses also.

Ans. The different combinations on the address lines A2, A1, A0 select one of the above, as shown:

A2	A1	A0	==>>	Control/Status Register
0	0	0		
0	0	1	===>>	Port A
0	1	0	===>>	Port B
0	1	1	===>>	Port C
1	0	0	===>>	LSB Timer
1	0	1	===>>	MSB Timer

The other five (viz., A7 to A3) on the address lines are as: 0 0 1 0 0. Thus

A7	A6	A5	A4	A3	A2	A1	A0	===>>	Address	—	Register/Port/Timer
0	0	1	0	0	0	0	0				
									20H		Control/Status register

0	0	1	0	0	0	0	1	===>>	21H	—	Port A
0	0	1	0	0	0	1	0	===>>	22H	—	Port B
0	0	1	0	0	0	1	1	===>>	23H	—	Port C
0	0	1	0	0	1	0	0	===>>	24H	—	LSB timer
0	0	1	0	0	1	0	1	===>>	25H	—	MSB timer

It is to be noted that the control/status register is having the same address 20H, but the control register is accessed with WR = 0 and RD = 1. For status register access, WR = 1 and RD = 0. The control register can never be read. For any future reference, the control register content is stored in some accessible memory location.

8. Draw the control word format and discuss the same in detail.

Ans. The control word loaded in the control register configures the different ports and the timer of 8155. The control word format is shown below:

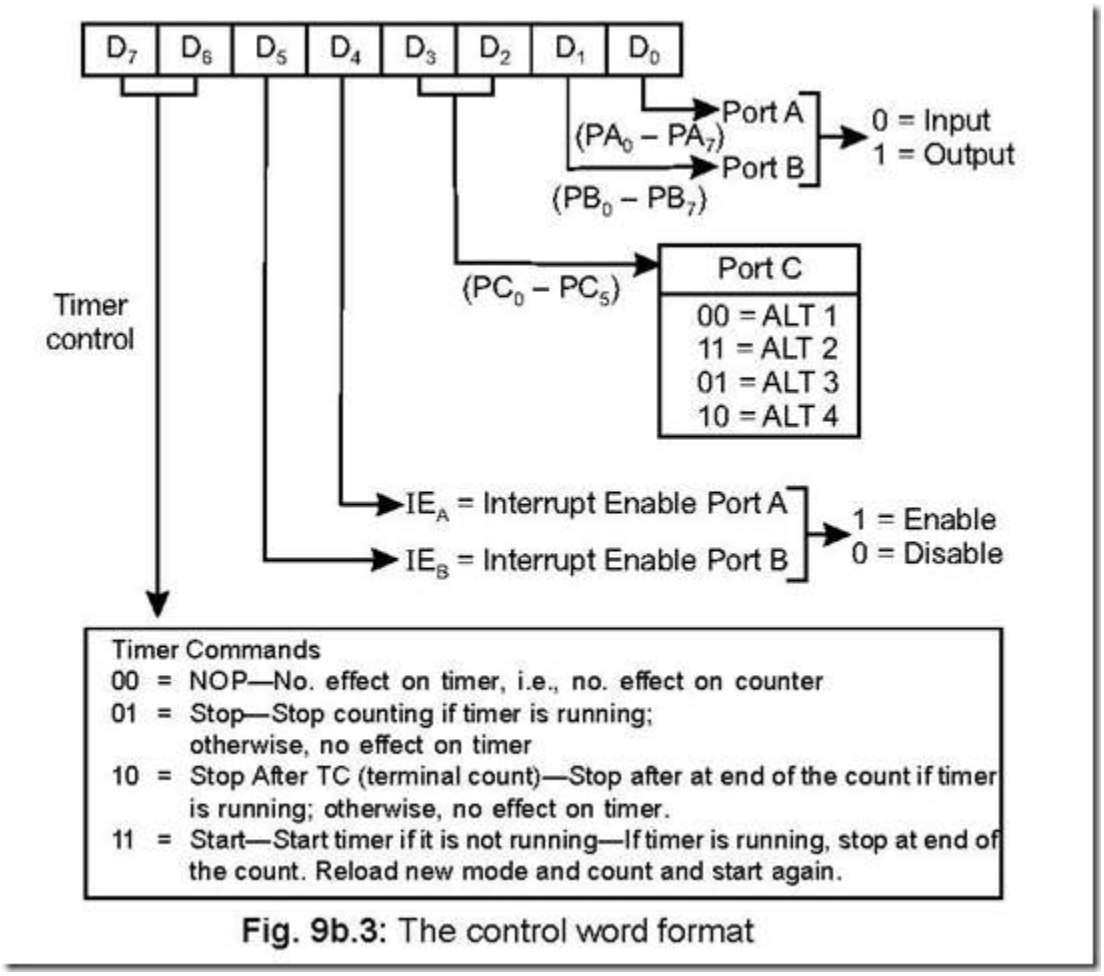


Fig. 9b.3: The control word format

The control register contains eight latches. The content of the lower 2 bits, viz., D1 – D0 configure ports A and B as input/output. Bits D3 and D2 configure bits PC0 – PC5 of port C

(Port C is a 6-bit port while ports A and B both are of 8-bits) and can have four combinations—ALT1, ALT2, ALT3, ALT4 depending on the combinations of D3 and D2. Bits D5 and D4 are enable/disable pins for ports A and B respectively which enable/ disable the internal flip-flop of 8155. Bits D7 and D6 contain the timer commands.

As already mentioned, combinations of D3 – D2 bits give rise to ALT1 to ALT4 modes, which assigns port C bits in different configurations and shown below:

Table 9b.1: Port C pin assignment (*Source:* Intel Corporation)

Pin	**ALT1**	**ALT2**	**ALT3**	**ALT4**
PC0	Input Port	Output Port	A INTR (Port AInterrupt)	A INTR (Port A Interrupt)
PC1	Input Port	Output Port		A BF (Port A Buffer Full)
PC2	Input Port	Output Port	A BF (Port A Buffer Full)	A STB(Port A Strobe)
PC3	Input Port	Output Port	A STB(Port A Strobe)	B INTR (Port B Interrupt)
PC4	Input Port	Output Port	Output Port	B BF (Port Buffer Full)
PC5	Input Port	Output Port	Output Port	B STB(Port B Strobe)
			Output Port	

ALT1 and ALT2 correspond to simple input/output of Port C respectively. In ALT3 mode, PC0 – PC2 bits are used as control signals for port A, while pins PC3 – PC5 act as output pins. In ALT4 mode, PC0 – PC2 bits are used as control signals for port A, while PC3 – PC5 bits are used as control signals for port B.

9. Draw the status word format and discuss the same. Ans. The status word format of 8155 is given below:

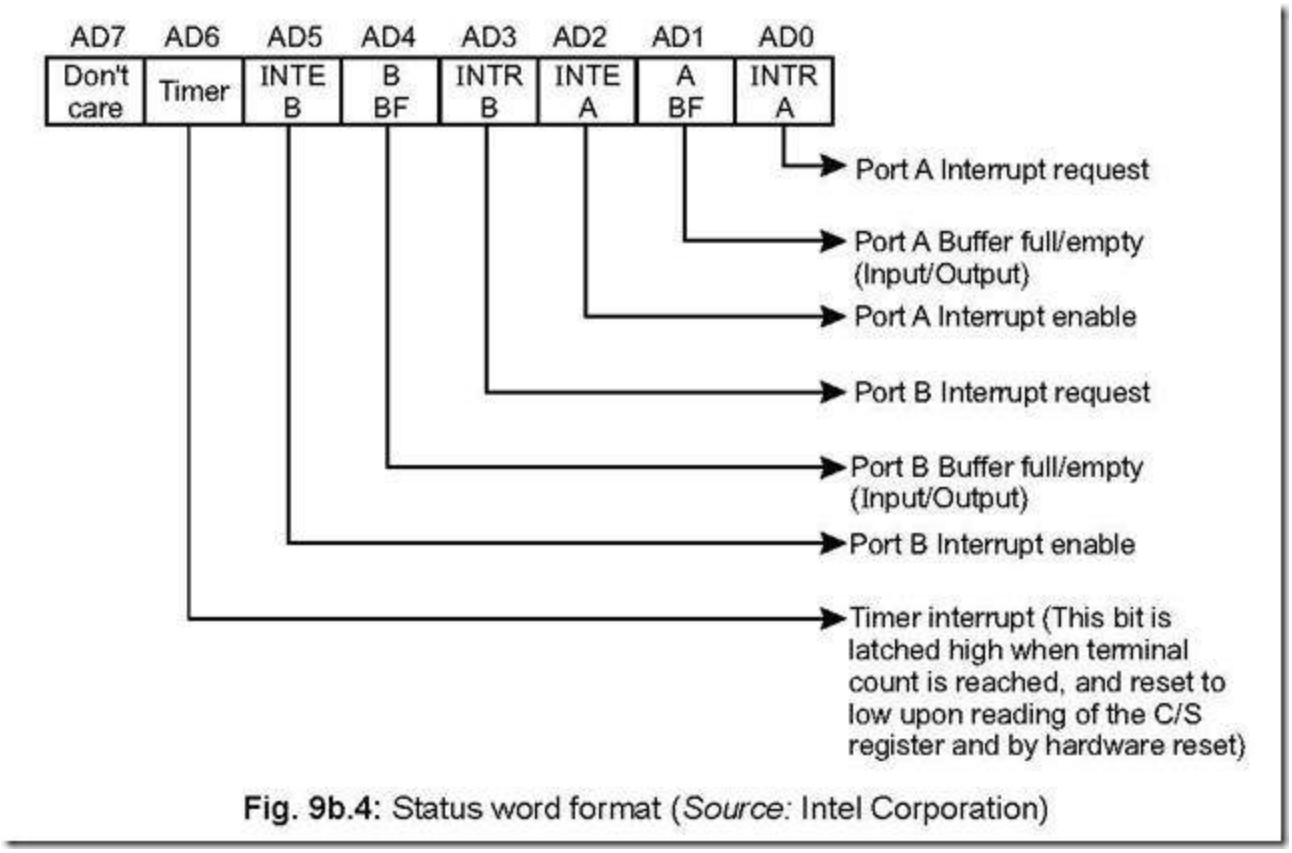


Fig. 9b.4: Status word format (*Source:* Intel Corporation)

It has seven latches. Bit D7 is the 'don't care' bit. Bit D6 contains the status of the timer. Bits D5 – D3 pertain to status of port B while bits D2 – D0 to that of Port A.

10. Discuss the timer section of 8155 and discuss its operating modes.

Ans. The timer section consists of two 8-bit registers. 14-bits of the two registers comprise to specify the count of the timer, which counts in a count-down manner. Contents of bits 6 and 7 of the most significant byte of the register decide the mode of operation of the counter. The following shows the timer register format. The timer section needs a 'TIMER IN' pulse, which is fed via pin 3 of 8155. A square wave or a pulse

is obtained via pin 6 (TIMER OUT) when the terminal count (TC) is reached. The maximum and minimum values of the count down timer are 3FFH and 002H respectively. A single square wave or a continuous square wave or a single pulse

on TC or a pulse on each TC (i.e., continuous pulses) are obtained, depending on the mode setting bits M2 and M1.

The following figure shows the nature of the outputs for the different modes.

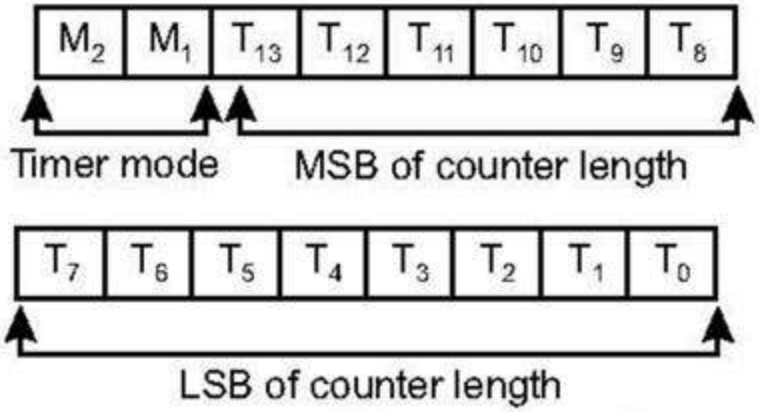


Fig. 9b.5: Timer registers format (*Source:* Intel Corporation)

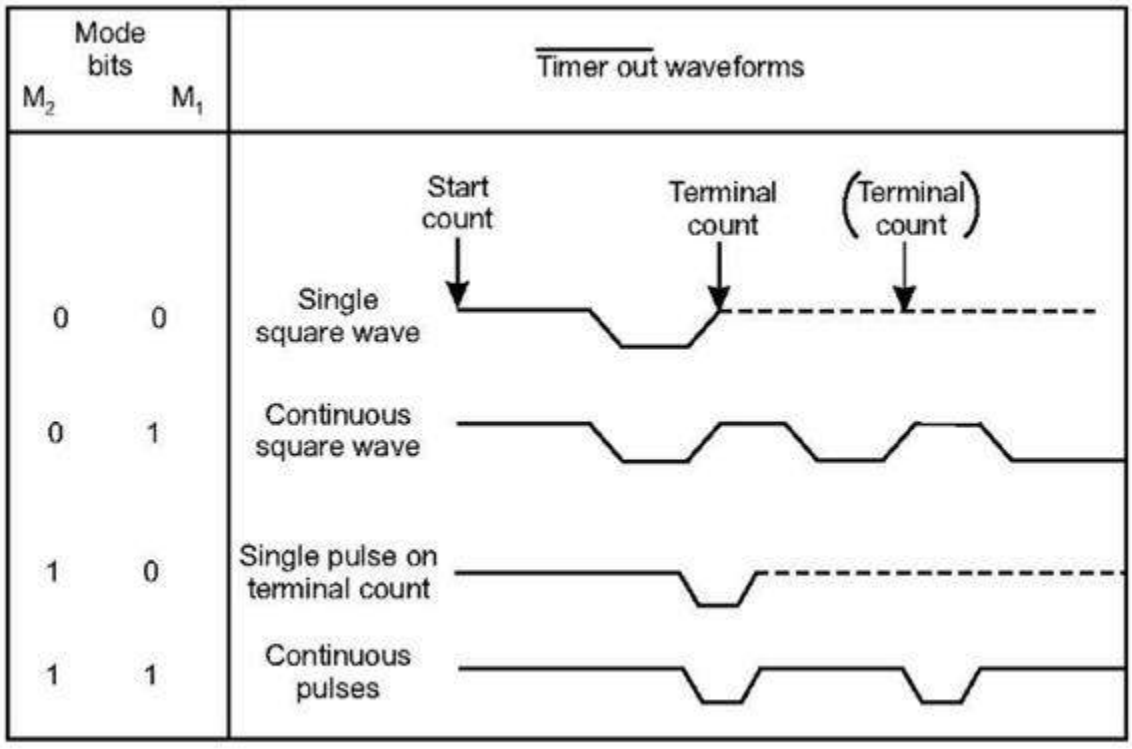


Fig. 9b.6: Timer modes and outputs (*Source:* Intel Corporation)

11. What happens when a high is applied on RESET ?

Ans. A high reset input resets the counter. To restart counting after resetting, a START command is required through the control register.

8355/8755: Programmable I/O Ports with ROM/EPROM

1. Draw the pin connection diagram of 8355.

Ans. The pin connection diagram of 8355 is shown in Fig. 9c.1.

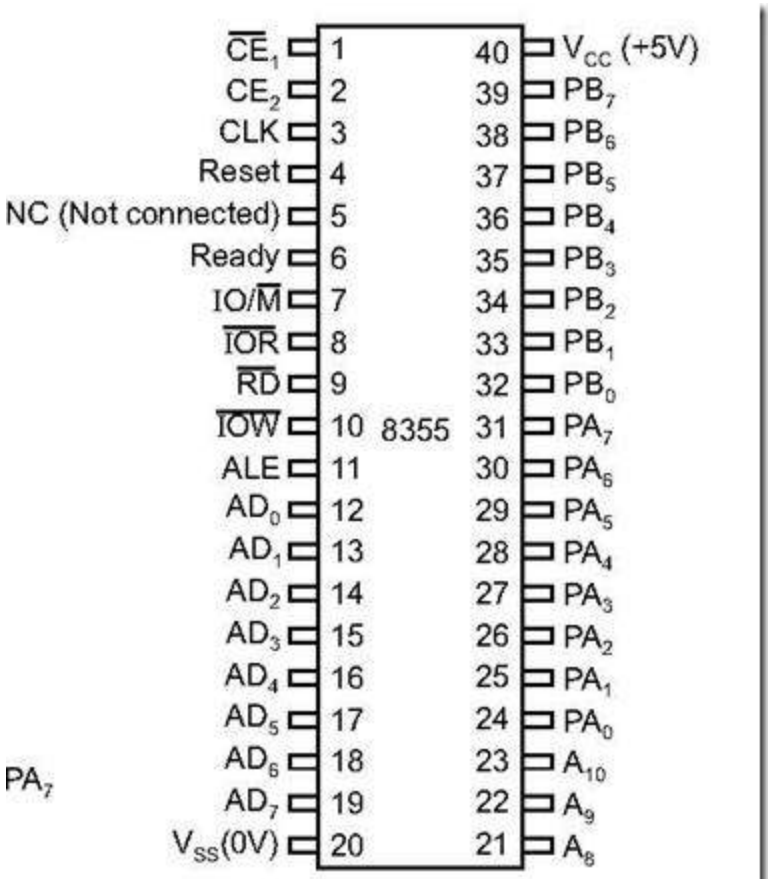


Fig. 9c.1: Pin diagram of 8355
(Source: Intel Corporation)

2. Draw the functional block diagram of 8355 and discuss.

Ans. The functional block diagram of 8355 is shown in Fig. 9c.2.

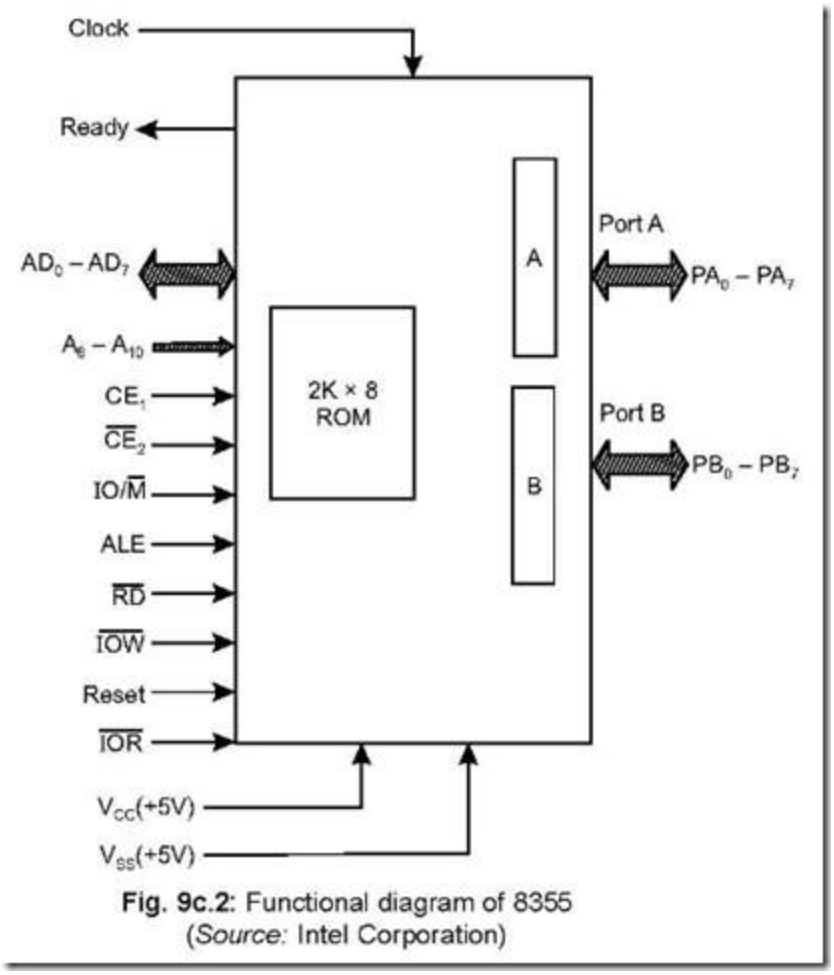


Fig. 9c.2: Functional diagram of 8355
(*Source:* Intel Corporation)

3. What is the difference between 8355 and 8755?

Ans. The 2 KB memory for 8355 is a ROM, while that for 8755, it is EPROM.

Again for 8355, there are two chip enable signals—CE1 and CE 2, while for 8755 this signal is designated as CE2.

Both have two I/O Ports—each I/O line of either port can be programmed either as input or output.

4. What the DDR's do?

Ans. There are two internal control registers, called Data Direction Registers (DDRs)—both the registers are 1-byte in length and designated as DDRA an DDRB. Each bit in the two DDR registers control the corresponding bit in the I/O ports. For example bit D0 of DDRA controls D0 bit of Port A and bit D5 of DDRB controls D5 bit of Port B.

5. How the two ports and the two DDRs are selected?

Ans. The bits AD1 and AD0 controls/selects one out of the four of the above. This is like this:

AD_1	AD_0		Selected Port or DDR
0	0	⇒	Port A
0	1	⇒	Port B
1	0	⇒	DDR_A
1	1	⇒	DDR_B

The IO/M signal is to remain high during the above.

6. How the 2 KB ROM of 8355 is accessed?

Ans. The 2 KB ROM of 8355 is accessed by the A10 – A0 latched address in conjunction with a low on IO/ M signal.

8255A - Programmable Peripheral Interface

Previous Page

Next Page

The 8255A is a general purpose programmable I/O device designed to transfer the data from I/O to interrupt I/O under certain conditions as required. It can be used with almost any microprocessor.

It consists of three 8-bit bidirectional I/O ports (24I/O lines) which can be configured as per the requirement.

Ports of 8255A

8255A has three ports, i.e., PORT A, PORT B, and PORT C.

- **Port A** contains one 8-bit output latch/buffer and one 8-bit input buffer.
- **Port B** is similar to PORT A.
- **Port C** can be split into two parts, i.e. PORT C lower (PC0-PC3) and PORT C upper (PC7-PC4) by the control word.

These three ports are further divided into two groups, i.e. Group A includes PORT A and upper PORT C. Group B includes PORT B and lower PORT C. These two groups can be programmed in three different modes, i.e. the first mode is named as mode 0, the second mode is named as Mode 1 and the third mode is named as Mode 2.

Operating Modes

8255A has three different operating modes −

- **Mode 0** − In this mode, Port A and B is used as two 8-bit ports and Port C as two 4-bit ports. Each port can be programmed in either input mode or output mode where outputs are latched and inputs are not latched. Ports do not have interrupt capability.
- **Mode 1** − In this mode, Port A and B is used as 8-bit I/O ports. They can be configured as either input or output ports. Each port uses three lines from port C as handshake signals. Inputs and outputs are latched.
- **Mode 2** − In this mode, Port A can be configured as the bidirectional port and Port B either in Mode 0 or Mode 1. Port A uses five signals from Port C as handshake signals for data transfer. The remaining three signals from Port C can be used either as simple I/O or as handshake for port B.

Features of 8255A

The prominent features of 8255A are as follows −

- It consists of 3 8-bit IO ports i.e. PA, PB, and PC.
- Address/data bus must be externally demux'd.
- It is TTL compatible.
- It has improved DC driving capability.

8255 Architecture

The following figure shows the architecture of 8255A −

Previous Page

Next Page

The Intel 8253 and 8254 are Programmable Interval Timers (PTIs) designed for microprocessors to perform timing and counting functions using three 16-bit registers. Each counter has 2 input pins, i.e. Clock & Gate, and 1 pin for “OUT” output. To operate a counter, a 16-bit count is loaded in its register. On command, it begins to decrement the count until it reaches 0, then it generates a pulse that can be used to interrupt the CPU.

Difference between 8253 and 8254

The following table differentiates the features of 8253 and 8254 −

8253	8254
Its operating frequency is 0 - 2.6 MHz	Its operating frequency is 0 - 10 MHz
It uses N-MOS technology	It uses H-MOS technology
Read-Back command is not available	Read-Back command is available
Reads and writes of the same counter cannot be interleaved.	Reads and writes of the same counter can be interleaved.

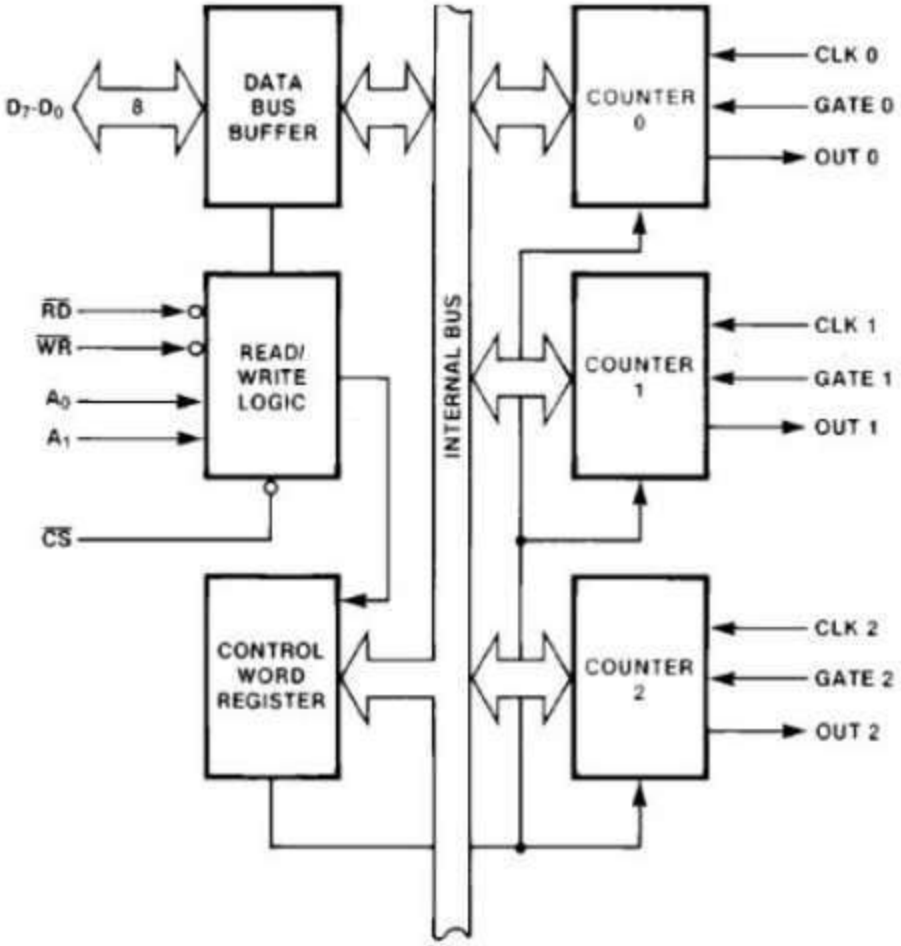
Features of 8253 / 54

The most prominent features of 8253/54 are as follows −

- It has three independent 16-bit down counters.
- It can handle inputs from DC to 10 MHz.
- These three counters can be programmed for either binary or BCD count.
- It is compatible with almost all microprocessors.
- 8254 has a powerful command called READ BACK command, which allows the user to check the count value, the programmed mode, the current mode, and the current status of the counter.

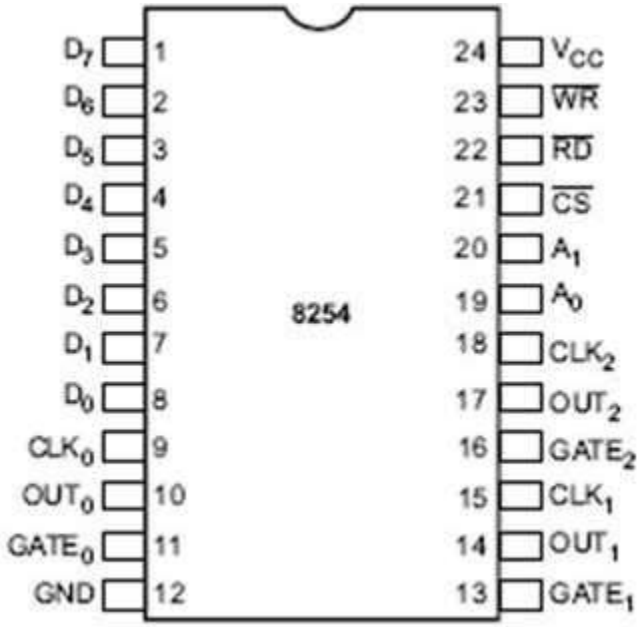
8254 Architecture

The architecture of 8254 looks as follows −



8254 Pin Description

Here is the pin diagram of 8254 −



In the above figure, there are three counters, a data bus buffer, Read/Write control logic, and a control register. Each counter has two input signals - CLOCK & GATE, and one output signal - OUT.

Data Bus Buffer

It is a tri-state, bi-directional, 8-bit buffer, which is used to interface the 8253/54 to the system data bus. It has three basic functions −

- Programming the modes of 8253/54.
- Loading the count registers.
- Reading the count values.

Read/Write Logic

It includes 5 signals, i.e. RD, WR, CS, and the address lines A_0 & A_1. In the peripheral I/O mode, the RD and WR signals are connected to IOR and IOW, respectively. In the memorymapped I/O mode, these are connected to MEMR and MEMW.

Address lines A_0 & A_1 of the CPU are connected to lines A_0 and A_1 of the 8253/54, and CS is tied to a decoded address. The control word register and counters are selected according to the signals on lines A_0 & A_1.

A_1	**A_0**	**Result**
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control Word Register
X	X	No Selection

Control Word Register

This register is accessed when lines A_0 & A_1 are at logic 1. It is used to write a command word, which specifies the counter to be used, its mode, and either a read or write operation. Following table shows the result for various control inputs.

A_1	A_0	RD	WR	CS	Result
0	0	1	0	0	Write Counter 0
0	1	1	0	0	Write Counter 1
1	0	1	0	0	Write Counter 2
1	1	1	0	0	Write Control Word
0	0	0	1	0	Read Counter 0
0	1	0	1	0	Read Counter 1
1	0	0	1	0	Read Counter 2
1	1	0	1	0	No operation
X	X	1	1	0	No operation
X	X	X	X	1	No operation

Counters

Each counter consists of a single, 16 bit-down counter, which can be operated in either binary or BCD. Its input and output is configured by the selection of modes stored in the control word register. The programmer can read the contents of any of the three counters without disturbing the actual count in process.

DMA stands for Direct Memory Access. It is designed by Intel to transfer data at the fastest rate. It allows the device to transfer the data directly to/from memory without any interference of the CPU.

Using a DMA controller, the device requests the CPU to hold its data, address and control bus, so the device is free to transfer data directly to/from the memory. The DMA data transfer is initiated only after receiving HLDA signal from the CPU.

How DMA Operations are Performed?

Following is the sequence of operations performed by a DMA −

- Initially, when any device has to send data between the device and the memory, the device has to send DMA request (DRQ) to DMA controller.
- The DMA controller sends Hold request (HRQ) to the CPU and waits for the CPU to assert the HLDA.
- Then the microprocessor tri-states all the data bus, address bus, and control bus. The CPU leaves the control over bus and acknowledges the HOLD request through HLDA signal.
- Now the CPU is in HOLD state and the DMA controller has to manage the operations over buses between the CPU, memory, and I/O devices.

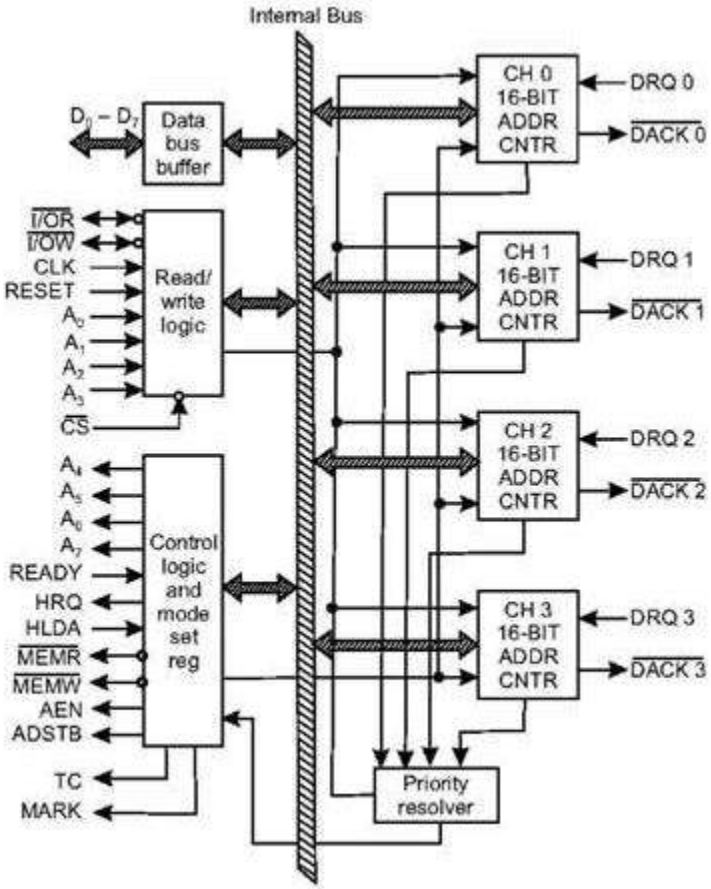
Features of 8257

Here is a list of some of the prominent features of 8257 −

- It has four channels which can be used over four I/O devices.
- Each channel has 16-bit address and 14-bit counter.
- Each channel can transfer data up to 64kb.
- Each channel can be programmed independently.
- Each channel can perform read transfer, write transfer and verify transfer operations.
- It generates MARK signal to the peripheral device that 128 bytes have been transferred.
- It requires a single phase clock.
- Its frequency ranges from 250Hz to 3MHz.
- It operates in 2 modes, i.e., **Master mode** and **Slave mode**.

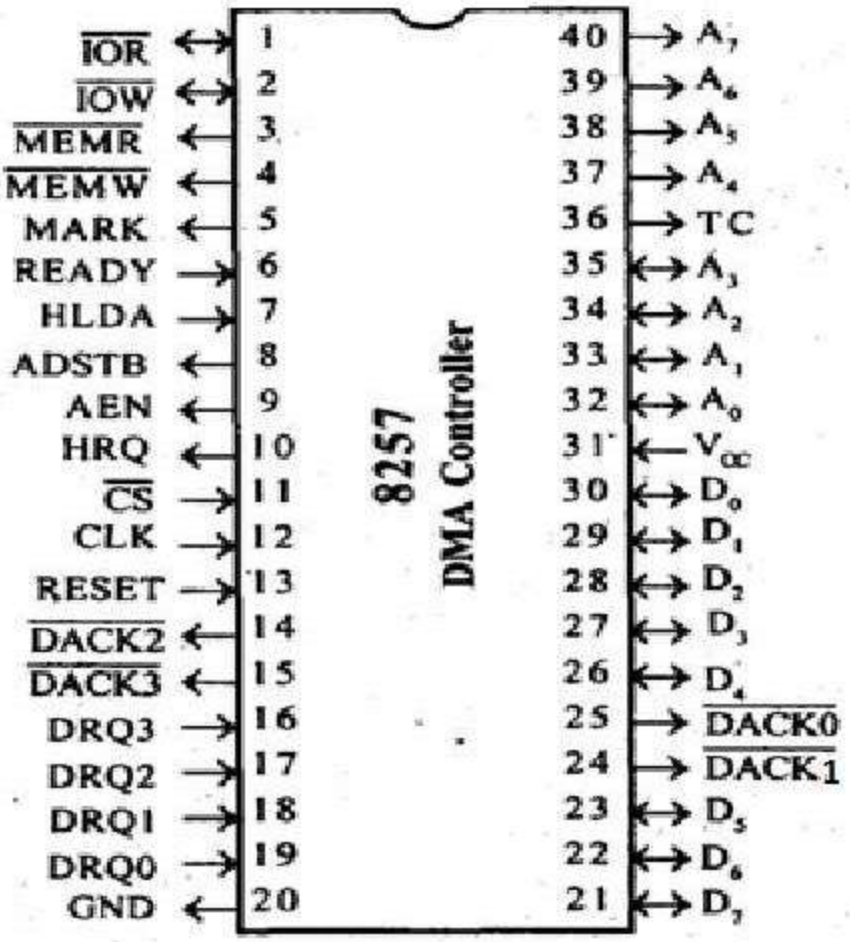
8257 Architecture

The following image shows the architecture of 8257 −



8257 Pin Description

The following image shows the pin diagram of a 8257 DMA controller −



DRQ_0–DRQ**3**

These are the four individual channel DMA request inputs, which are used by the peripheral devices for using DMA services. When the fixed priority mode is selected, then DRQ_0 has the highest priority and DRQ_3 has the lowest priority among them.

$DACK_o - DACK_3$

These are the active-low DMA acknowledge lines, which updates the requesting peripheral about the status of their request by the CPU. These lines can also act as strobe lines for the requesting devices.

$D_o - D_7$

These are bidirectional, data lines which are used to interface the system bus with the internal data bus of DMA controller. In the Slave mode, it carries command words to 8257 and status word from 8257. In the master mode, these lines are used to send higher byte of the generated address to the latch. This address is further latched using ADSTB signal.

IOR

It is an active-low bidirectional tri-state input line, which is used by the CPU to read internal registers of 8257 in the Slave mode. In the master mode, it is used to read data from the peripheral devices during a memory write cycle.

IOW

It is an active low bi-direction tri-state line, which is used to load the contents of the data bus to the 8-bit mode register or upper/lower byte of a 16-bit DMA address register or terminal count register. In the master mode, it is used to load the data to the peripheral devices during DMA memory read cycle.

CLK

It is a clock frequency signal which is required for the internal operation of 8257.

RESET

This signal is used to RESET the DMA controller by disabling all the DMA channels.

A_o - A_3

These are the four least significant address lines. In the slave mode, they act as an input, which selects one of the registers to be read or written. In the master mode, they are the four least significant memory address output lines generated by 8257.

CS

It is an active-low chip select line. In the Slave mode, it enables the read/write operations to/from 8257. In the master mode, it disables the read/write operations to/from 8257.

A_4 - A_7

These are the higher nibble of the lower byte address generated by DMA in the master mode.

READY

It is an active-high asynchronous input signal, which makes DMA ready by inserting wait states.

HRQ

This signal is used to receive the hold request signal from the output device. In the slave mode, it is connected with a DRQ input line 8257. In Master mode, it is connected with HOLD input of the CPU.

HLDA

It is the hold acknowledgement signal which indicates the DMA controller that the bus has been granted to the requesting peripheral by the CPU when it is set to 1.

MEMR

It is the low memory read signal, which is used to read the data from the addressed memory locations during DMA read cycles.

MEMW

It is the active-low three state signal which is used to write the data to the addressed memory location during DMA write operation.

ADST

This signal is used to convert the higher byte of the memory address generated by the DMA controller into the latches.

AEN

This signal is used to disable the address bus/data bus.

TC

It stands for 'Terminal Count', which indicates the present DMA cycle to the present peripheral devices.

MARK

The mark will be activated after each 128 cycles or integral multiples of it from the beginning. It indicates the current DMA cycle is the 128th cycle since the previous MARK output to the selected peripheral device.

V_{cc}

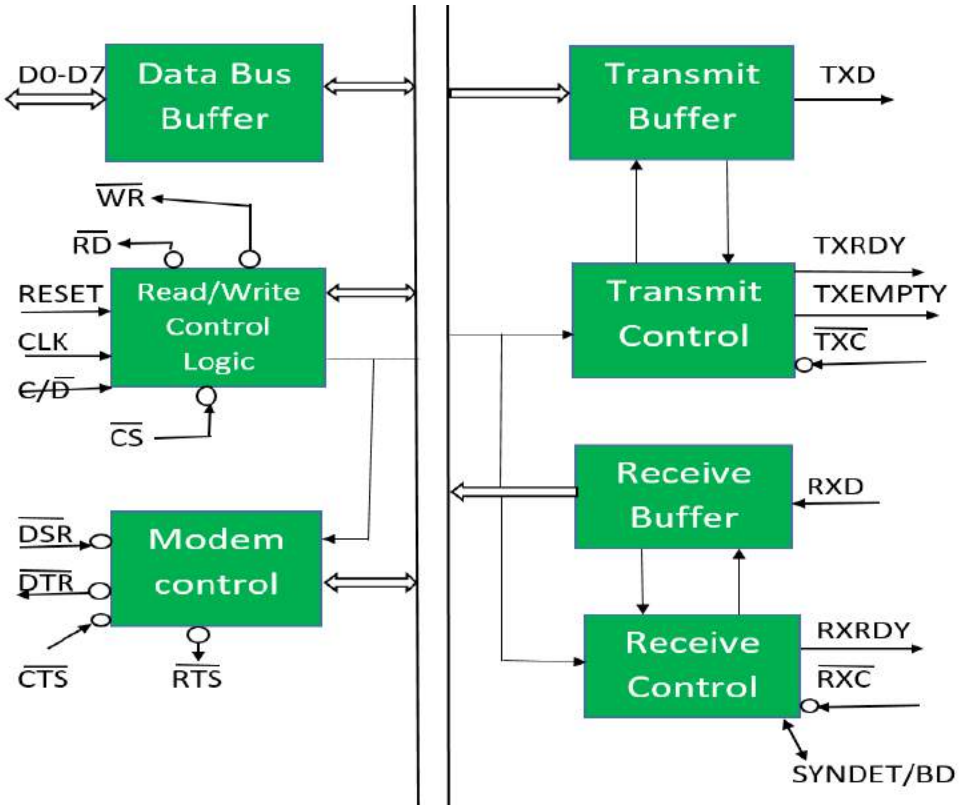
Microprocessor | 8251 USART

Prerequisite – 8259 PIC Microprocessor

8251 universal synchronous asynchronous receiver transmitter (USART) acts as a mediator between microprocessor and peripheral to transmit serial data into parallel form and vice versa.

1. It takes data serially from peripheral (outside devices) and converts into parallel data.
2. After converting the data into parallel form, it transmits it to the CPU.
3. Similarly, it receives parallel data from microprocessor and converts it into serial form.
4. After converting data into serial form, it transmits it to outside device (peripheral).

Block Diagram of 8251 USART –



It contains the following blocks:

1. **Data bus buffer –**
   This block helps in interfacing the internal data bus of 8251 to the system data bus. The data transmission is possible between 8251 and CPU by the data bus buffer block.

2. **Read/Write control logic –**
   It is a control block for overall device. It controls the overall working by selecting the operation to be done. The operation selection depends upon input signals as:

$\overline{CS}$	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	Operation
1	X	X	X	Invalid
0	0	0	1	data CPU< ----- 8251
0	0	1	0	data CPU ----- > 8251
0	1	0	1	Status word CPU < ----------8251
0	1	1	0	Control word CPU---------- > 8251

In this way, this unit selects one of the three registers- data buffer register, control register, status register.

3. **Modem control (modulator/demodulator) –**
   A device converts analog signals to digital signals and vice-versa and helps the computers to communicate over telephone lines or cable wires. The following are active-low pins of Modem.
   - **DSR:** Data Set Ready signal is an input signal.
   - **DTR:** Data terminal Ready is an output signal.
   - **CTS:** It is an input signal which controls the data transmit circuit.
     RTS: It is an output signal which is used to set the status RTS.
4. **Transmit buffer –**
   This block is used for parallel to serial converter that receives a parallel byte for conversion into serial signal and further transmission onto the common channel.
   - **TXD:** It is an output signal, if its value is one, means transmitter will transmit the data.
5. **Transmit control –**
   This block is used to control the data transmission with the help of following pins:
   - **TXRDY:** It means transmitter is ready to transmit data character.
   - **TXEMPTY:** An output signal which indicates that TXEMPTY pin has transmitted all the data characters and transmitter is empty now.
   - **TXC:** An active-low input pin which controls the data transmission rate of transmitted data.
6. **Receive buffer –**
   This block acts as a buffer for the received data.
   - **RXD:** An input signal which receives the data.
7. **Receive control –**
   This block controls the receiving data.
   - **RXRDY:** An input signal indicates that it is ready to receive the data.
   - **RXC:** An active-low output signal which controls the data transmission rate of received data.
   - **SYNDET/BD:** An input or output terminal. External synchronous mode-input terminal and asynchronous mode-output terminal.

Previous Page

Next Page

8279 programmable keyboard/display controller is designed by Intel that interfaces a keyboard with the CPU. The keyboard first scans the keyboard and identifies if any key has been pressed. It then sends their relative response of the pressed key to the CPU and vice-a-versa.

How Many Ways the Keyboard is Interfaced with the CPU?

The Keyboard can be interfaced either in the interrupt or the polled mode. In the **Interrupt mode**, the processor is requested service only if any key is pressed, otherwise the CPU will continue with its main task.

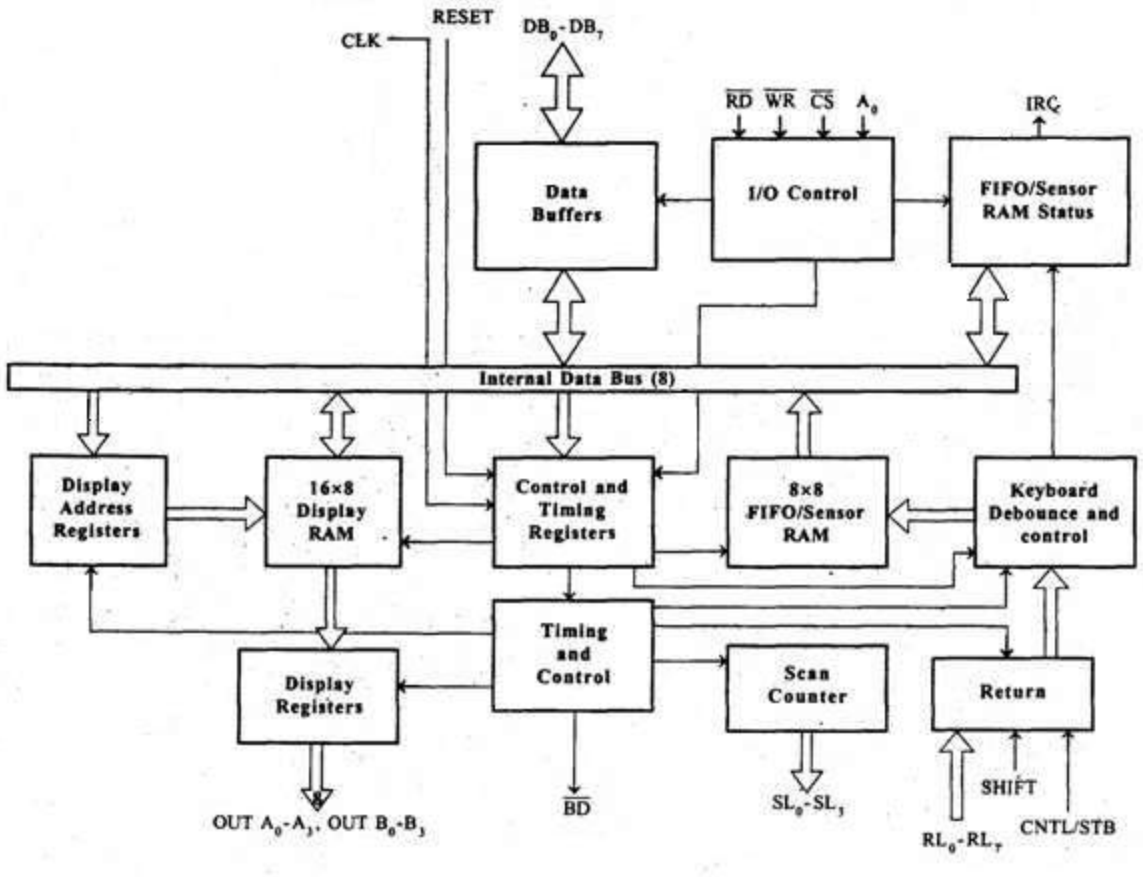
In the **Polled mode**, the CPU periodically reads an internal flag of 8279 to check whether any key is pressed or not with key pressure.

How Does 8279 Keyboard Work?

The keyboard consists of maximum 64 keys, which are interfaced with the CPU by using the key-codes. These key-codes are de-bounced and stored in an 8-byte FIFORAM, which can be accessed by the CPU. If more than 8 characters are entered in the FIFO, then it means more than eight keys are pressed at a time. This is when the overrun status is set.

If a FIFO contains a valid key entry, then the CPU is interrupted in an interrupt mode else the CPU checks the status in polling to read the entry. Once the CPU reads a key entry, then FIFO is updated, and the key entry is pushed out of the FIFO to generate space for new entries.

Architecture and Description



I/O Control and Data Buffer

This unit controls the flow of data through the microprocessor. It is enabled only when D is low. Its data buffer interfaces the external bus of the system with the internal bus of the microprocessor. The pins A0, RD, and WR are used for command, status or data read/write operations.

Control and Timing Register and Timing Control

This unit contains registers to store the keyboard, display modes, and other operations as programmed by the CPU. The timing and control unit handles the timings for the operation of the circuit.

Scan Counter

It has two modes i.e. **Encoded mode** and Decoded mode. In the encoded mode, the counter provides the binary count that is to be externally decoded to provide the scan lines for the keyboard and display.

In the **decoded scan mode**, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL_0-SL_3.

Return Buffers, Keyboard Debounce, and Control

This unit first scans the key closure row-wise, if found then the keyboard debounce unit debounces the key entry. In case, the same key is detected, then the code of that key is directly transferred to the sensor RAM along with SHIFT & CONTROL key status.

FIFO/Sensor RAM and Status Logic

This unit acts as 8-byte first-in-first-out (FIFO) RAM where the key code of every pressed key is entered into the RAM as per their sequence. The status logic generates an interrupt request after each FIFO read operation till the FIFO gets empty.

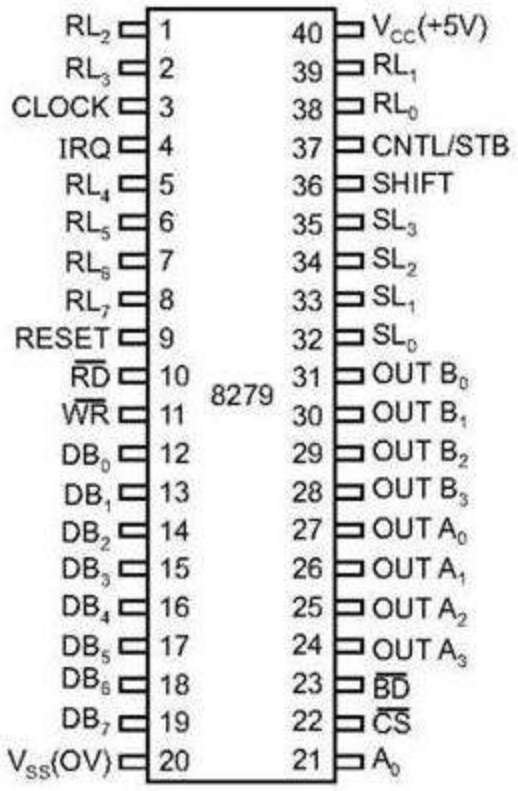
In the scanned sensor matrix mode, this unit acts as sensor RAM where its each row is loaded with the status of their corresponding row of sensors into the matrix. When the sensor changes its state, the IRQ line changes to high and interrupts the CPU.

Display Address Registers and Display RAM

This unit consists of display address registers which holds the addresses of the word currently read/written by the CPU to/from the display RAM.

8279 − Pin Description

The following figure shows the pin diagram of 8279 −



Data Bus Lines, DB_0 - DB_7

These are 8 bidirectional data bus lines used to transfer the data to/from the CPU.

CLK

The clock input is used to generate internal timings required by the microprocessor.

RESET

As the name suggests this pin is used to reset the microprocessor.

CS Chip Select

When this pin is set to low, it allows read/write operations, else this pin should be set to high.

A_0

This pin indicates the transfer of command/status information. When it is low, it indicates the transfer of data.

RD, WR

This Read/Write pin enables the data buffer to send/receive data over the data bus.

IRQ

This interrupt output line goes high when there is data in the FIFO sensor RAM. The interrupt line goes low with each FIFO RAM read operation. However, if the FIFO RAM further contains any key-code entry to be read by the CPU, this pin again goes high to generate an interrupt to the CPU.

V_{ss}, V_{cc}

These are the ground and power supply lines of the microprocessor.

$SL_0 - SL_3$

These are the scan lines used to scan the keyboard matrix and display the digits. These lines can be programmed as encoded or decoded, using the mode control register.

$RL_0 - RL_7$

These are the Return Lines which are connected to one terminal of keys, while the other terminal of the keys is connected to the decoded scan lines. These lines are set to 0 when any key is pressed.

SHIFT

The Shift input line status is stored along with every key code in FIFO in the scanned keyboard mode. Till it is pulled low with a key closure, it is pulled up internally to keep it high

CNTL/STB - CONTROL/STROBED I/P Mode

In the keyboard mode, this line is used as a control input and stored in FIFO on a key closure. The line is a strobe line that enters the data into FIFO RAM, in the strobed input mode. It has an internal pull up. The line is pulled down with a key closure.

BD

It stands for blank display. It is used to blank the display during digit switching.

$OUTA_0 - OUTA_3$ and $OUTB_0 - OUTB_3$

These are the output ports for two 16x4 or one 16x8 internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display and the keyboard.

Operational Modes of 8279

There are two modes of operation on 8279 − **Input Mode** and **Output Mode**.

Input Mode

This mode deals with the input given by the keyboard and this mode is further classified into 3 modes.

- **Scanned Keyboard Mode** − In this mode, the key matrix can be interfaced using either encoded or decoded scans. In the encoded scan, an 8×8 keyboard or in the decoded scan, a 4×8 keyboard can be interfaced. The code of key pressed with SHIFT and CONTROL status is stored into the FIFO RAM.
- **Scanned Sensor Matrix** − In this mode, a sensor array can be interfaced with the processor using either encoder or decoder scans. In the encoder scan, 8×8 sensor matrix or with decoder scan 4×8 sensor matrix can be interfaced.

- **Strobed Input** − In this mode, when the control line is set to 0, the data on the return lines is stored in the FIFO byte by byte.

Output Mode

This mode deals with display-related operations. This mode is further classified into two output modes.

- **Display Scan** − This mode allows 8/16 character multiplexed displays to be organized as dual 4-bit/single 8-bit display units.
- **Display Entry** − This mode allows the data to be entered for display either from the right side/left side.

Unit 9

The Z80 Microprocessor

History

In 1969 Intel were approached by a Japanese company called Busicom to produce chips for Busicom's electronic desktop calculator. Intel suggested that the calculator should be built around a single-chip generalized computing engine and thus was born the first microprocessor - the 4004. Although it was based on ideas from much larger mainframe and mini-computers the 4004 was cut down to fit onto a 16-pin chip, the largest that was available at the time, so that its data bus and address bus were each only 4-bits wide.

Intel went on to improve the design and produced the 4040 (an improved 4-bit design) the 8008 (the first 8-bit microprocessor) and then in 1974 the 8080. This last one turned out to be a very useful and popular design and was used in the first home computer, the Altair 8800, and CP/M.

In 1975 Federico Faggin who had had worked at Intel on the 4004 and its successors left the company and joined forces with Masatoshi Shima to form Zilog. At their new company Faggin and Shima designed a microprocessor that was compatible with Intel's 8080 (it ran all 78 instructions of the 8080 in exactly the same way that Intel's chip did) but had many more abilities (an extra 120 instructions, many more registers, simplified connection to hardware). Thus was born the mighty Z80!

The original Z80 was first released in July 1976. Since then newer versions have appeared with exactly the same architecture but running at higher speeds. The original Z80 ran with a clock rate of 2.5 MHz, the Z80A runs at 4MHz, the Z80B at 6MHz, and the Z80H at 8mhz.

Many companies produced machines based around Zilog's improved chip during the 1970's and 80's and because the chip could run 8080 code without needing any changes to the code the perfect choice of operating system was CP/M.

So, let's dive into the Z80 and see what's inside...

8-bit data, 16-bit addresses

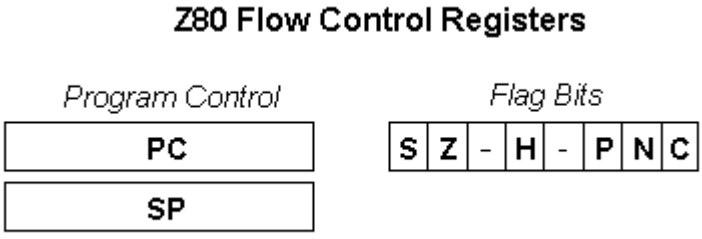
The Z80 uses 8-bit bytes which are stored in memory. These bytes contain both the program that the processor is executing and the data items that the program is working on. The processor uses 16-bit addresses to access these bytes, so there can be anything up to 64k (65536) bytes of memory. How much memory is actually available, and how much of it is read only (ROM) or random access (RAM) varies from one machine to another: in a simple controller there might be just 4k of ROM holding the program and 1k of RAM for the data items; in a large CP/M system you would find a full 64k of RAM.

The processor also uses a separate 64k of addresses for input and output devices. This I/O address space is only used by a few instructions (called, not surprisingly, IN and OUT) to send or receive bytes from I/O devices. The addresses for I/O instructions use the same wiring as the addresses for memory and the data bytes travel through the same wiring that is used for bytes in memory so the processor uses special control signals to differentiate between memory accesses and I/O operations. These control signals have to be decoded by I/O devices together with the addresses sent by the processor so that devices do not mistakenly respond to requests that were intended to read or write bytes in memory.

Programmer's Model

All programming of the Z80 revolves around its internal registers. These registers are used to control the flow of the program and to operate on data items. There are 16-bit registers inside the processor that address the program bytes in memory and there are 8-bit registers into which data bytes are loaded and then operated on. These 8-bit registers can also be combined in pairs to form 16-bit addresses and these addresses can be used to index data structures in memory that span several bytes.

Flow of Control Registers



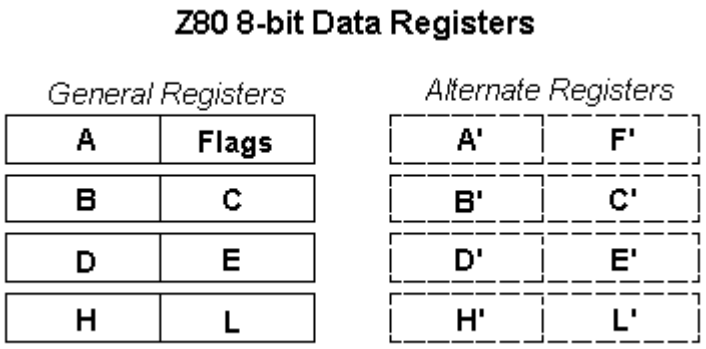
The Z80 uses a 16-bit program counter (PC) to hold the memory address of the next instruction to execute. This register gradually steps through memory as instructions are executed, but some

instructions can alter it directly to cause the flow of the program to branch to a new location.

The Z80 also supports subroutines calls with a stack pointer (SP). When a subroutine is called the contents of PC are pushed into the memory location pointed to by SP and the stack pointer is decremented. When the subroutine has finished executing the stack pointer is incremented and the contents of PC popped back from the memory location. In this way calls to subroutines can be nested to any depth, as long as there is space to push the program counter into memory for each call.

The Z80 also contains flags to control the flow of a program. The flags are changed whenever an arithmetic operation is done and the flags can be tested one at a time by a jump instructions to change the flow of the program, depending on the state of the flag. The flags record the sign (S), zero (Z), half-carry (H), parity (P), and carry (C) of the result of the last arithmetic instruction and all of these can be tested by jump instructions. There is also a flag that records whether the last arithmetic operation was an addition or subtraction (N) which is only used to decimal arithmetic. These six flags are stored in a single byte (F) with bits 3 and 5 left unused.

8-bit Data Registers



The Z80 processor contains 14 general purpose registers which can be used for holding and manipulating data bytes. Each of these registers is 8-bits wide and they are named A, B, C, D, E, H, L, A', B', C', D', E', H', L'.

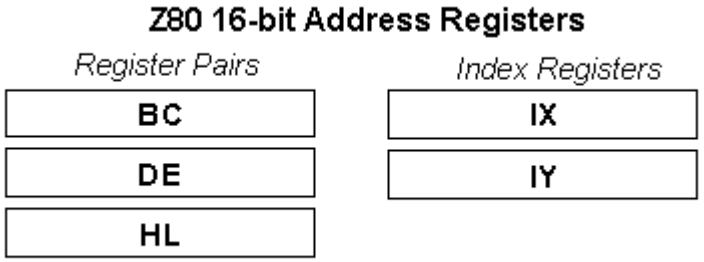
As their names suggest the registers are divided into two sets. The first set are the registers A..L and the second set, called the alternate registers, are A'..L'. Most operations in the processor only work on the first set and the alternates just provide temporary space to save the contents of A..L. The Z80 has short instructions that exchange the contents of the working registers with the contents of the alternates. Of course, this temporary

space could easily be allocated in memory but in applications where timing is critical it is much quicker to exchange the working registers and the alternates rather than having to save the working registers and then load new values from memory.

Once data has been loaded into the working registers the Z80 has a full range of arithmetic, logical, shifting, and rotating instructions to handle 8-bit bytes. The results of these operations generally end up in register A (the accumulator) from where they can be moved to other registers or saved back to memory locations.

In a few instructions the flags are treated as a single byte that is called the F register.

16-bit Address Registers



The Z80 contains two types of 16-bit addressing registers: register pairs and index registers.

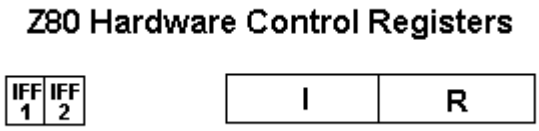
Register pairs are formed from combinations of the 8-bit general purpose registers. The Z80 allows three combinations: registers B and C can be combined to form a 16-bit register that is called BC, registers D and E combined to form DE, and H and L combined to form HL.

These register pairs can be used to do some limited 16-bit arithmetic (addition, subtraction, increment, decrement) but generally they are used as pointers to data in memory. The instruction set is heavily biased towards using HL as a data pointer. Many arithmetic instructions can combine the accumulator A with the byte in memory that is pointed to by HL. In this way the byte in memory does not have to be loaded into a internal register before it is combined with the accumulator, saving instructions and register space.

The processor also contains two index registers (IX and IY) that can be used to address data in memory. These registers work very like the HL register pair in that many instructions can use a byte that is addressed by IX or IY as an operand. However, unlike HL,

an offset is added to the value in the index register before addressing memory. This offset is contained in the instruction that is using IX or IY and it is an 8-bit value. This helps the programmer create fancy data structures in memory using an index register to point to the start of the structure and the 8-bit offset to select fields within the structure.

Hardware Registers



The Z80 also contains a few registers that let the user control its hardware in some simple ways.

Interrupts can be controlled with two flags (IFF.1 and IFF.2) and an 8-bit page pointer (I). The programmer can enable and disable some interrupts by setting and clearing IFF.1. When an interrupt does happen the current contents of IFF.1 are saved in IFF.2 and then IFF.1 is set so that further interrupts are disabled. When the program has finished handling the interrupt IFF.2 is copied back to IFF.1, restoring the flag to its original value.

Instruction Set

These links will take you to tables that describe the Z80 instructions in detail. They are broken down into groups of similar instructions:

8-bit Load and Store - Using the general purpose registers
16-bit Load and Store - Using register pairs, index registers, and the stack pointer

1-bit Arithmetic - Set a bit, Clear a bit, Test a bit

8-bit Arithmetic - Add, Subtract, And, Or, etc.
8-bit Shift and Rotate - Shift left, Shift right, Rotate, Insert carry bit, etc.

16-bit Arithmetic - Add, Subtract, Increment, Decrement

Flag Control - Set carry flag, Clear carry, Enable and Disable interrupts
Program Flow - Jump, Conditional jump, Call subroutine

I/O - Input a byte, Output a byte
Block Instructions - Move block of memory, Search a block of

memory, Input or Output a block

3-Aug-98 22:38:42

Motorola 6800 Microprocessoe

Memory

Program, data and stack memories occupy the same memory space. The total addressable memory size is 64 KB.

Program memory - program can be located anywhere in memory. Jump and subroutine call instructions can be used to jump anywhere in memory. Conditional and unconditional branches are limited to memory addresses positioned no farther than -125 - +129 bytes from the branch instruction.

Data memory - data can be anywhere in memory space.

Stack memory - stack can be placed anywhere in memory space.

Reserved memory locations:

- FFF8h - FFF9h: Pointer to IRQ interrupt-processing routine.
- FFFAh - FFFBh: Pointer to software interrupt-processing routine.
- FFFCh - FFFDh: Pointer to NMI interrupt-processing routine.
- FFFEh - FFFFh: Pointer to RESET handling code.
- Some memory addresses may be reserved for memory mapped I/O as the processor doesn't have hardware I/O capability.

Interrupts

IRQ - maskable interrupt. When the interrupt occurs the program counter, index register, accumulators and condition code registers are stored in the stack, the further interrupts are disabled and the processor jumps to memory location address of which is stored in memory FFF8h - FFF9h. To return from the interrupt the processing routine should use RTI instruction. This interrupt can be enabled/disabled using CLI/SEI instructions.

NMI - non-maskable interrupt. When the interrupt occurs the program counter, index register, accumulators and condition code registers are stored in the stack, the further interrupts are disabled and the processor jumps to memory location address of which is stored in memory FFFCh - FFFDh. To return from the interrupt the processing routine should use RTI instruction. This interrupt can not be disabled.

SWI - software interrupt. This interrupt can be only invoked from the program. When the interrupt occurs the processor stores the program counter, index register, accumulators and condition code registers in the stack, disables the further interrupts and jumps to memory location address of which is stored in memory FFFAh - FFFBh. To return from the interrupt the processing routine should use RTI instruction. This interrupt can not be disabled.

I/O ports

None.

Registers

Accumulator A (ACCA) is an 8-bit register used for arithmetic and logic operations.

Accumulator B (ACCB) is an 8-bit register used for arithmetic and logic operations.

Index (IX) is a 16-bit register usually used for temporary storage or as an index when indexed addressing is used.

Program counter (PC) is a 16-bit register.

Stack pointer (SP) is a 16-bit register.

Condition code register contains the following flags:

- Half carry (H) - set if there was a carry from bit 3 to bit 4 of the result when the result was calculated.
- Interrupt mask (I) - set if the IRQ interrupt is disabled.
- Negative (N) - set if the most significant bit of the result is set.
- Zero (Z) - set if the result is zero.
- Overflow (V) - set if there was an overflow during last result calculation.
- Carry (C) - set if there was a carry from the bit 7 during last result calculation.

Instruction Set

6800 instruction set consists of 72 instructions:

- Data moving instructions.
- Arithmetic - add, subtract, negate, increment, decrement and compare.
- Logic - AND, OR, exclusive OR, complement and shift/rotate.
- Control transfer - conditional, unconditional, call subroutine and return from subroutine.
- Other - clear/set condition flags, bit test, stack operations, software interrupt, etc.

Addressing modes

Implied - the data value/data address is implicitly associated with the instruction.

Accumulator - the instruction implies that the data is one of the accumulator registers.

Immediate - 8-bit or 16-bit data is provided in the instruction.

Direct - one-byte operand provided in the instruction specifies the memory address in page zero (0000h - 00FFh) where data is located.

Extended - two-byte operand provided in the instruction specifies the memory address where data is located.

Relative - one byte offset is added to the address of the next instruction (the contents of the program counter register + 2). The offset is a signed number in the range -127 - +127.

Indexed - one byte operand is added to the contents of the IX register, the resulting 16-bit value is a pointer to memory where data is located.

Comparison between 8085 and Z80 Microprocessors

This tutorial gives a brief comparison among different classic microprocessor families like 8085, 8086, 80186, Zilog 80 and Motorola 6800 processor. This comparison we are giving because of demand from our students of different countries.

Compare between 8085 and 8086, Compare between 8051 and MC6800, Compare between 8086 and 80386, Compare between 8086 and 8088

S.No.	8085 Microprocessor	MC6800 Microprocessor
1	It operates on Clock frequency of 3 to 5 MHz.	It operates at 1 MHz frequency.
2	8085 has no Index register.	It has one index register.

3	8085 has on board clock logic circuit.	No clock logic circuit.
4	8085 has one Accumulator Register.	MC6800 has two Accumulator Registers.
5	8085 has five interrupts.	MC 6800 have two interrupts.
6	It has total 674 Instructions.	MC6800 has total 72 instructions.

Comparison between 8085 and MC6800 Microprocessors

S.No.	8085 Microprocessor	MC6800 Microprocessor
1	It is a 16 bit microprocessor and it is first 16 bit microprocessor after 8085(8-bit).	It is a 32 bit microprocessor and it is logical ex of the 80236.
2	It has pipelined architecture (not highly) and high speed bus interface on single chip.	It is highly pipelined architecture and much fas bus than 8086.
3	It is upward compatible with 80386.It means all 8086 instructions are followed by 80386.	However, 80386 can support 8086 programmir & can also directly run the programs written f in virtual mode if VM=1(in protected mode)
4	It is housed on a 40 pin DIP package.	The chip of 80836 contains 132 pins.
5	It is a built on a HMOS technology.	The 80386 using High-speed CHMOS III techr
6	No special hardware is equipped for task Switching.	It has a special hardware for task switching.
7	The 8086 operates on a 5MHz. Clock.	The 80386 operate 33MHz clock frequency ma
8	The address bus and data bus are multiplexed.	It has separate address and data bus for time sa
9	It has a transistor package density of 29,500 transistors.	Transistor density and complexity further incre 2,75,000.
10	It has a total of 117 instructions.	It has total 129 instructions
11	It has no mechanism protection, paging.	The 80386 contains protection mechanism pag which has instruction two support them
12	It is operated in one mode only.	It operate in three modes a)Real b)Virtual c)Protected
13	It has only instruction Queue.	It has instruction Queue as well as pre fetch qu
14	In 8086, It is not necessity that all operation are in parallel mode.	80386 all functional units are not parallel
15	8086 has nine flags.	It contains all nine flags of 8086 but other flags IOP,NT,RF,VM.

Comparison between 8086 and 80386 Microprocessors

S.No.	8086 Microprocessor	8088 Microprocessor
1	The instruction Queue is 6 byte long.	The instruction Queue is 4 byte long.

2	In 8086 memory divides into two banks, up to 1,048,576 bytes.	The memory in 8088 does not divide in to t as 8086.
3	The data bus of 8086 is 16-bit wide	The data bus of 8088 is 8-bit wide.
4	It has BHE(bar) signal on pin no. 34 & there is no SSO(bar) signal.	It does not has BHE(bar) signal on pin no. only SSO(bar) signal. It has no S7 pin.
5	The output signal is used to select memory or I/O at M/IO(bar) but if IO(bar)/M low or logic '0' it selects I/O devices and if IO(bar)/M is high or logic '1'it selects memory.	The output signal is used to select memory M(bar)/IO but if IO/M(bar) is low or at logi selects Memory devices and if IO/M(bar) is logic '1'it selects I/O.
6	It needs one machine cycle to R/W signal if it is at even location otherwise it needs two.	It needs one machine cycle to R/W signal if even location otherwise it needs two.
	In 8086, all address & data Buses are multiplexed.	In 8088, address bus, AD_7- AD_0 buses are multiplexed.
	It needs two IC 74343 for de-multiplexing AD_0-AD_{19}.	It needs one IC 74343 for de-multiplexing A

References:

1. www.google.com
2. www.tutorialspoint.com
3. www.w3schools.com
4. www.studytonight.com