Name of the Paper: Microprocessors & Microcontrollers
Paper Code: CS602
Contact (Periods/Week): 3L/Week
Credit Point: 3
No. of Lectures: 35

Prerequisite:

1. Familiarity with the number system
2. A solid background in digital logic.

Course Objective(s)

- To learn the basics of a particular microprocessor.
- To learn the basics of a particular microcontroller.
- To learn the interfacing of microprocessor.

Course Outcomes

CS602.1 To acquire the knowledge of hardware details of 8085 and 8086 microprocessor with the related signals and their implications
CS602.2 To develop skill in assembly Language programming of 8085
CS602.3 To understand the concept and techniques of designing and implementing interfacing of microprocessor with memory and peripheral chips involving system design
CS602.4 To acquire the knowledge of the 8051 architecture and its programming
CS602.5 To analyze the performance of computers and its architecture to real-life applications

Module -1: [9L]
Introduction to Microcomputer based system. [1L]
History of evolution of Microprocessor and Microcontrollers and their advantages and disadvantages. [1L]
Architecture of 8085 Microprocessor, Pin description of 8085. [2L] Address/data bus De-multiplexing, Status Signals and the control signals. [1L]
Interrupts of 8085 processor (software and hardware) [2L]
I/O Device Interfacing - I/O Mapped I/O and Memory Mapped I/O,
Memory interfacing with 8085 [2L]

Module -2: [11L]
Instruction set of 8085 microprocessor, Addressing modes. [3L]
Assembly language programming with examples, Counter and Time Delays, Stack and Subroutine. [6L]
Timing diagram of the instructions (a few examples) [2L]
Module 3: [8L]

The 8086 microprocessor- Architecture, Pin Details, Addressing modes, Interrupts [3L]
Instruction set, Examples of Simple Assembly Language [3L]
Memory interfacing with 8086 [2L]

Module -4: [7L]
Introduction to 8051 Microcontroller – Architecture, Pin Details. [3L]
Addressing modes, Instruction set, Examples of Simple Assembly Language. [4L]

Text Books:

1. MICROPROCESSOR architecture, programming and Application with 8085 - R. Gaonkar (Penram international Publishing LTD.) ***[For Module 1 and 2]***
2. Fundamentals of Microprocessor and Microcontrollers - B. Ram (Paperback) ***[For Module 3]***
3. 8051 Microcontroller – K. Ayala (Cengage learning) ***[For Module 4]***

Recommended books:

1. 8086 Microprocessor – K Ayala (Cengage learning)
2. The 8051 microcontroller and Embedded systems - Mazidi, Mazidi and McKinley (PEARSON)
3. Microprocessors – The 8086/8088, 80186/80386/80486 and the Pentium family – N. B. Bahadure (PHI).

CO-PO Mapping

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CS602.1	3		3								2	
CS602.2			2	1								
CS602.3	1		3	2		1	1				1	
CS602.4	2		1									
CS602.5				2		2	2				2	

COMPARISON STUDY

Module	No. of Lectures	Existing Syllabus as per MAKAUT(Module basis)	Addition/Deletion/ Comment	Justification	Syllabus for Autonomy
1	**9**	**Module -1: [8L]** **Introduction to Microcomputer based system. History of evolution of Microprocessor and Microcontrollers and their advantages and disadvantages. [1L]** **Architecture of 8085 Microprocessor, Pin description of 8085. [2L]** **Address/data bus De-multiplexing , Status Signals and the control signals. [1L]** Instruction set of 8085 microprocessor, Addressing modes, [3L] Timing diagram of the instructions (a few examples). [1L]	Instruction set of 8085 microprocessor, Addressing modes and Timing diagram of the instructions (a few examples) has been shifted to Module 2. Interrupts of 8085 processor (software and hardware), I/O Device Interfacing-I/O Mapped I/O and Memory Mapped I/O has been placed in Module 1. Memory interfacing with 8085	8085 Hardware model and software model has been placed in separate modules	**Module -1: [9L]** **Introduction to Microcomputer based system. [1L]** **History of evolution of Microprocessor and Microcontrollers and their advantages and disadvantages. [1L]** **Architecture of 8085 Microprocessor, Pin description of 8085. [2L] Address/data bus De-multiplexing, Status Signals and the control signals. [1L]** Interrupts of 8085 processor (software and hardware) [2L] I/O Device Interfacing - I/O Mapped I/O and Memory Mapped I/O, Memory interfacing with 8085 [2L]
2	11	**Module -2: [9L]** **Assembly language programming with examples, Counter and Time Delays, Stack and Subroutine, [6L]** Interrupts of 8085 processor (software and hardware), I/O Device Interfacing-I/O Mapped I/O and Memory Mapped I/O , Serial (using SID and SOD pins and RIM, SIM Instructions) and Parallel data transfer [3L]	Instruction set of 8085 microprocessor, Addressing modes and Timing diagram of the instructions (a few examples) has been placed in Module 2. Interrupts of 8085 processor (software and hardware), I/O Device Interfacing-I/O Mapped I/O and Memory Mapped I/O has been shifted in Module 1. Serial (using SID and SOD pins and RIM, SIM Instructions) and Parallel data transfer part has been omitted.	8085 Hardware model and software model has been placed in separate modules	**Module -2: [11L]** Instruction set of 8085 microprocessor, Addressing modes. [3L] **Assembly language programming with examples, Counter and Time Delays, Stack and Subroutine. [6L]** Timing diagram of the instructions (a few examples) [2L]
3	8	**Module -3 [10L]** The 8086 microprocessor- Architecture, Addressing modes, Interrupts [3L] Introduction to 8051 Microcontroller – Architecture, Pin Details. [3L] Addressing modes, Instruction set, Examples of Simple Assembly Language. [4L]	Pin Details of 8086 microprocessor has been added. Instruction set, Examples of Simple Assembly Language and Memory interfacing with 8086 has been added. Introduction to 8051 Microcontroller –Architecture, Pin Details. Addressing modes, Instruction set, Examples of Simple Assembly Language have been shifted to Module 4	A dedicated module is for only 8086	**Module 3: [8L]** **The 8086 microprocessor- Architecture,** Pin Details, **Addressing modes, Interrupts [3L]** Instruction set, Examples of Simple Assembly Language [3L] Memory interfacing with 8086 [2L]
4	7	**Module -4: [9L]** Memory interfacing with 8085, 8086 [2L] Support IC chips- 8255 ,8251,8237/8257,8259 [4L] Interfacing of 8255 PPI with 8085 and Microcontroller 8051. [2L] Brief introduction to PIC microcontroller (16F877) [1L]	Memory interfacing with 8085 has been shifted to Module 1 Memory interfacing with 8086 has been shifted to Module 3 Rest are omitted	A dedicated module is for only 8051	**Module -4: [7L]** Introduction to 8051 Microcontroller – Architecture, Pin Details. [3L] Addressing modes, Instruction set, Examples of Simple Assembly Language. [4L]

MODULE 1

LECTURE 1: ARCHITECTURE OF 8085 MICROPROCESSOR (PIN DESCRIPTION)

1. ARCHITECTURE AND PIN DESCRIPTION OF 8085 MICROPROCESSOR

The 8085 is an 8-bit general purpose microprocessor capable of addressing 64K of memory. The device has forty pins, requires a +5V single power supply, and can operate with a 3MHz single phase clock.

Following Figure (Figure 1.1) shows the pin diagram of 8085 Microprocessor.

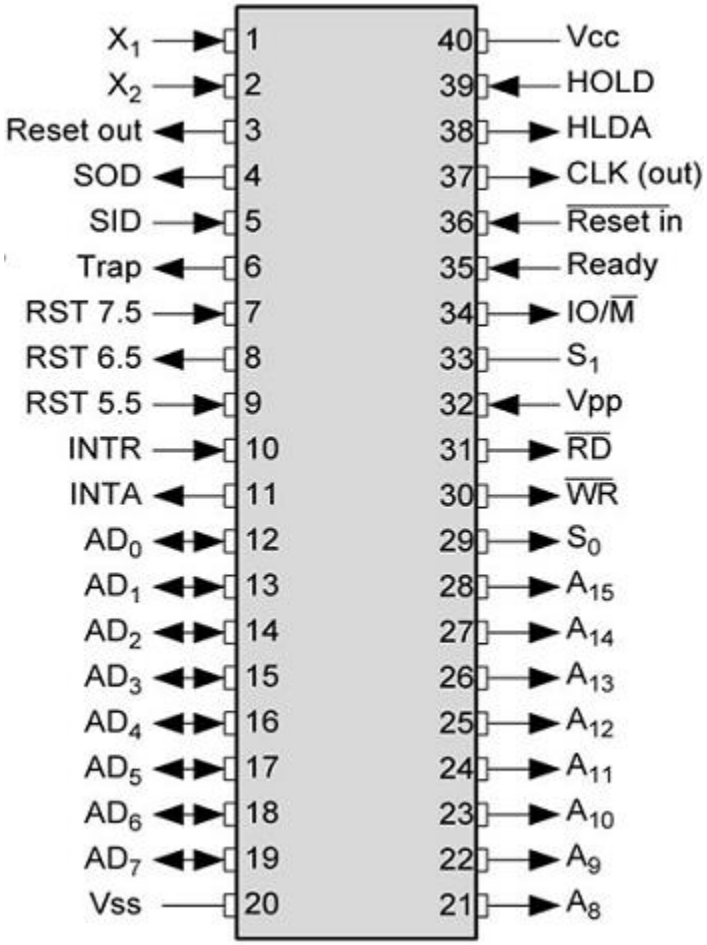


Figure 1.1

All the signals can be classified into six groups.

1. Address bus
2. Data bus
3. Control and status signals
4. Power supply and frequency signals
5. Externally initiated signals
6. Serial I/O ports.

Here details of individual pins are described.

Pin number:

1. X1 (Input)
2. X2 (Input)

An LC tuned circuit or an RC network or a crystal (X-tal) can generate clock pulses. In 8085 MPU, an X-tal is connected at these two pins (pin number 1 and 2). The clock generation circuitry is built inside the unit. The crystal frequency must be at least 1 MHz. The operating frequency of 8085 is 6 MHz. Special variant 8085 is operated with a 10 MHz crystal. The output pulse train got from crystal is not exact square pulse. So certain amount of wave-shaping is required. The wave-shaping circuit may be Smitt-trigger circuit using OP_AMP. Smitt-trigger circuit behaves like a flip-flop (divide by 2) circuit. Thus the frequency at which microprocessor operates is half of the crystal frequency. Hence we can say 8085 operates with 3 MHz clock frequency.

3. RESET OUT (Output)

The “Reset Out” pin gives a signal that indicates that the MPU is being reset. The same signal can be used to reset (or acknowledge) the various other peripheral devices.

4. SOD (Output)
5. SID (Input)

The 8085 has two signals to implement the serial transmission: SOD (Serial Output Data) and SID (Serial Input Data). In serial transmission, data bits are sent over a single line, one bit at a time, such as transmission over telephone lines. The data on SID pin is loaded into Accumulator bit B7 whenever a RIM instruction is executed. The output SOD is set or reset as specified by the SIM instruction.

6. TRAP (Input)
7. RST7.5 (Input)
8. RST6.5 (Input)
9. RST5.5 (Input)
10. INTR (Input)
11. INTA′ (Output)

TRAP, RST7.5, RST6.5, RST5.5 and INTR are interrupt signals and INTA′ is active low interrupt acknowledge signal.

We shall present a discussion about “Interrupt” later.

12. AD0 (Input / Output)
13. AD1 (Input / Output)
14. AD2 (Input / Output)
15. AD3 (Input / Output)
16. AD4 (Input / Output)
17. AD5 (Input / Output)
18. AD6 (Input / Output)
19. AD7 (Input / Output)

The signal lines AD7 - AD0 are bidirectional: they serve a dual purpose. They are used as the low-order address bus as well as the data bus. In executing an instruction, during the earlier part of the cycle, these lines are used as the low-order address bus. During the later part of the cycle, these lines are used as the data bus. (This is also known as multiplexing the bus.)

Low-order address bus and data bus are mutually exclusive, i.e., these are not needed simultaneously.

Pin Number:

20. GND

Pin number 20 is connected with Ground.

Pin Number:

21. A8 (Output)
22. A9 (Output)
23. A10 (Output)
24. A11 (Output)
25. A12 (Output)
26. A13 (Output)
27. A14 (Output)
28. A15 (Output)

The 8085 has these eight signal lines, A15 - A8, which are unidirectional and used as the high-order address bus (the most significant 8 bits of the address bus).

Pin Number:

29. S0 (Output)

S0 is status signal used to identify the nature of the operation.

Pin Number:

30. ALE (Output)

The ALE (Address Latch Enable) signal is used to demultiplex the low-order bus (the least significant 8 bits of the address bus). Logic 1 on this pin indicates that the bits on AD7 - AD0 are address bits. Logic 0 on this pin indicates that AD7-AD0 contain data.

Pin Number:

31. WR′ (Output)
32. RD′ (Output)

WR′ is a Write control signal (active low). This signal indicates the data on the data bus are to be written into a selected memory or I/O location.

RD′ is a Read control signal (active low). This signal indicates that the selected I/O or memory device is to be read and data are available on the data bus.

RD′ and WR′ are mutually exclusive signals. Both of them can be inactive at the same time but both of them cannot be active at the same time.

Pin Number:

33. S1 (Output)

S0 is status signal used to identify the nature of the operation.

Pin Number:

34. IO / M′ (Output)

This is a status signal used to differentiate between I / O and memory operations. When it is HIGH, it indicates an I/O operation; when it is LOW, it indicates a memory operation.

Pin Number:

35. READY (Input)

The MPU checks the status of this pin after every machine cycle within a particular instruction. If READY pin is found HIGH, then the MPU proceeds to the next instruction; otherwise, it waits over there and starts checking the status of this READY pin after every clock cycle till found HIGH.

The functions of READY pin are:

i. We can introduce some sort of wait cycles within instruction.
ii. The READY pin is used for matching and speed synchronization between MPU and peripheral device(s). For example, we can synchronize a slower printer with a MPU. Here, READY signal is used to delay the microprocessor read or write cycles until the printer (a slow-responding peripheral) is ready to send or accept data.

Pin Number:

36. RESET IN′ (Input)

When the signal on pin 36 goes low then

i. The PC is set to zero
ii. The buses are tri-stated
iii. Flip-flops are cleared
iv. Registers are cleared
v. The MPU is reset

Pin Number:

37. CLK (Output)

The Clock Output signal is supplied to all peripheral devices. This will be 3 MHz for 8085.

Pin Number:

38. HLDA (Output)
39. HLD (Input)

When HLD (Hold) signal is HIGH, MPU is reset and buses are not tri-stated. This signal indicates that a peripheral such as a DMA controller is requesting the use of the address and data buses, i.e. DMA transfer can be done when HLD is HIGH.
An acknowledge signal (HLDA) is needed to synchronize the MPU with DMA controller. HLDA signal acknowledges the Hold request.

Pin Number:

40. Vcc

This is +5V power supply.

LECTURE 2: INTERRUPTS IN 8085

2. INTERRUPTS IN 8085

The interrupt I/O is a process of data transfer whereby an external device or a peripheral can inform the processor that is ready for communication and it requests attention. The process is initiated by an external device and is asynchronous which means that it can be initiated at any time without reference to the system clock. However, the response to an interrupt request is directed o controlled by the microprocessor.

Interrupt – Hardware Interrupt and Software Interrupt:

Hardware Interrupts are interrupts which are non-deterministic, i.e., their time of occurrence is unknown to us. Hardware interrupts may be generated due to power-off. [Examples: TRAP, RST7.5, RST6.5, RST5.5and INTR].

Software interrupts are interrupts which are deterministic. RST instructions are commonly used to set up software breakpoints. [Examples: RST0, RST1, RST2, RST3, RST4, RST5, RST6, RST7].

Interrupt – Vectored Interrupt and Non-vectored Interrupt:

In case of vectored interrupt, the MPU knows which device is asking for interrupt, i.e., where to jump. The PC jumps to a specific predefined memory location. [Examples: TRAP, RST7.5, RST6.5, RST5.5, RST0, RST1, RST2, RST3, RST4, RST5, RST6 and RST7].

In case of non-vectored interrupts, the MPU does not know where to jump. If a single interrupt line is connected with a number of devices, then this type of interrupt takes place. In this case, the MPU has to go in a procedure called Daisy Polling Chain, where the device near to MPU gets priority. [Example: INTR].

Vectored Interrupts:

The 8085 instruction set includes eight RST (RESTART) instructions. These are 1-byte call instructions that transfers the program execution to a specific location on page 00H., i.e., that compel the PC to jump to the memory location whose address is predefined, as listed in following table [Table 1.1]:

Instructions	Call Location (in Hex)
RST0	0000
RST1	0008
RST2	0010
RST3	0018
RST4	0020

RST5	0028
RST6	0030
RST7	0038

Table 1.1: Vectored Software Interrupts

Again, the 8085 has four hardware vectored interrupts. These are TRAP, RST7.5, RST6.5 and RST5.5. These four are automatically vectored (transferred) to specific locations on memory page 00H without any external hardware. The necessary hardware is already implemented inside the 8085 MPU. These interrupts and their call locations are listed below [Table 1.2]:

Interrupts	Call Location (in Hex)
TRAP	0024
RST7.5	003C
RST6.5	0034
RST5.5	002C

Table 1.2: Vectored Software Interrupts

Now we can list all the vector interrupts and their corresponding call locations in a single table. The table is listed below [Table 1.3]:

Instructions	Call Location (in Hex)
RST0	0000
RST1	0008
RST2	0010
RST3	0018
RST4	0020
TRAP	0024
RST5	0028
RST5.5	002C
RST6	0030
RST6.5	0034
RST7	0038
RST7.5	003C

Table 1.3: Vectored Software Interrupts

If Interrupt Service Routine (ISR) is written in memory location whose address is between two call locations of interrupts, then 8 bytes / 4 bytes are not sufficient for writing ISR. Hence we can take help of jump instruction ‘JMP’. This jump instruction (3-byte instruction) actually compels the PC to jump to the memory location whose address is given or specified, and at that memory location full length ISR is written. These memory locations are usually ROM locations.

We can take an example. Let the instruction "RST5" is written within a program. When the processor encounters this instruction, the program is transferred to location 0028H. The ISR is written somewhere else in memory (let between 2000 and 2100), and the jump instruction is written at 0028 to specify the address (i.e., between 2000 and 21000 of the ISR.

[Between 2000 and 2100, some ISRs are loaded by the monitor program when the machine gets loaded.]

Non- Vectored Interrupts:

Non-vectored interrupted are processed through a mechanism called Daisy Polling Chain. In 8085, INTR is the only non-vectored interrupt and INTA′ is the interrupt acknowledge (active low).

Interrupt – Maskable Interrupt and Non-maskable Interrupt:

The MPU can ignore or delay a certain interrupt at certain time when it is performing some critical task. This is maskable interrupt. [Examples: RST7.5, RST6.5, RST5.5, RST0, RST1, RST2, RST3, RST4, RST5, RST6, RST7, INTR].

The MPU has to respond a certain type of interrupt request immediately. This is non-maskable interrupt. [Example: TRAP]

Interrupts based on Priority:

i. Vectored interrupt gets higher priority than non-vectored interrupts.
ii. As far as software interrupts are concerned, there is no priority because it is known when the interrupts is going to take place.
iii. Hardware interrupts with higher number gets higher priority. But TRAP is exceptional. TRAP has the highest priority
iv. Vectored interrupts with higher number gets higher priority. But TRAP is exceptional. TRAP has the highest priority. For example, between RST7.5 and RST7, the first one gets higher priority.

LECTURE 3: INSTRUCTIONS ABOUT INTERRUPTS IN 8085

3. THE INSTRUCTIONS ABOUT INTERRUTPS

The 8085 interrupt process is controlled by the Interrupt Enable flip-flop, which is internal to the processor and can be set or reset by using software instructions. If the flip-flop is enabled and the input to the interrupt signal goes high, the microprocessor is interrupted.

The interrupt process should be enabled by writing the instruction EI in the main program. The instruction EI sets the Interrupt Enable Flip-Flop. The instruction DI resets the FF and disables the interrupt process.

Instruction

EI (Enable Interrupt)

- This is 1-byte instruction.
- The instruction sets the Interrupt Enable FF and enables the Interrupt process.
- System reset or an interrupt disables the interrupt disables the interrupt process.

Instruction

DI (Disable Interrupt)

- This is 1-byte instruction.
- The instruction resets the Interrupt Enable FF and disables the interrupt process.
- It should be included in a program segment where an interrupt from an outside source cannot be tolerated.

The 8085 has five interrupt inputs (Figure). One is called INTR, three are called RST5.5, RST6.5 and RST7.5, respectively, and the fifth is called TRAP, a non-maskable interrupt.

The TRAP has the highest priority, followed by RST7.5, RST6.5, RST5.5 and INTR, in that order. [However, the TRAP has a lower priority than the Hold signal used for DMA].

TRAP is a non-maskable interrupt. It has the highest priority among the interrupt signals, it need not be enabled and it cannot be disabled. It is level and edge-sensitive, meaning that the input should go high and stay high to be acknowledged. It cannot be acknowledged. It cannot be acknowledged again until it makes a transition from high to low to high.

Figure 1.2 shows that when this interrupt is triggered, the program control is transferred to location 0024 without any external hardware or the interrupt enable instruction EI. TRAP is generally used for such critical events as power failure and emergency shut-off.

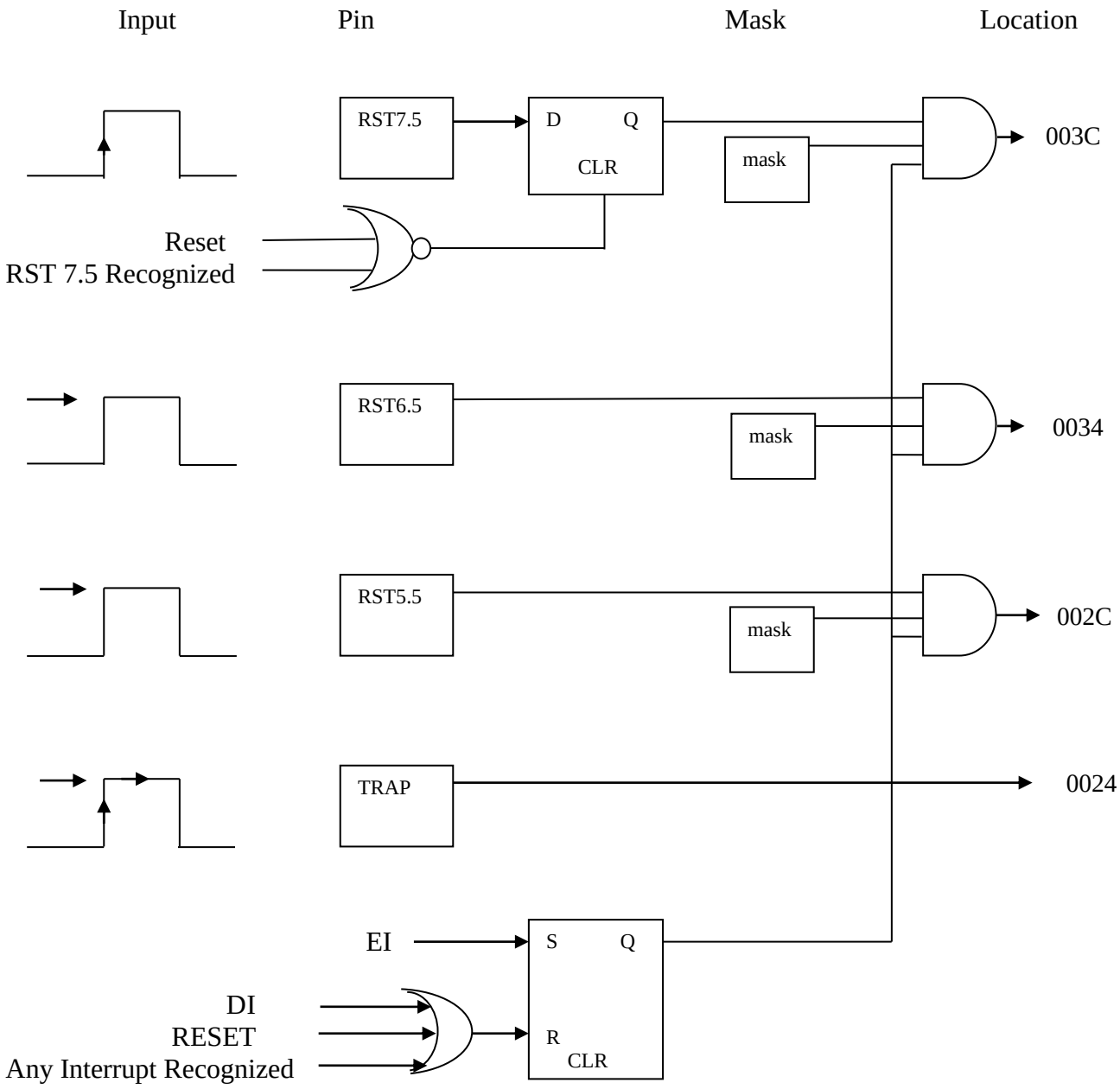


Figure 1.2: Hardware for 8085 Vectored Interrupts

RST 7.5, RST6.5 and RST5.5

The maskable interrupts RST7.5, RST6.5 and RST5.5 (shown in Figure) are enabled under program control with two instructions: EI (Enable Interrupt) described earlier and SIM (Set Interrupt Mask) described below:

Instruction

SIM (Set Interrupt Mask)

- This is 1-byte instruction.
- The instruction can be used for three different functions
  - i) One function is to set mask for RST7.5, rst6.5 and rst5.5 interrupts. This instruction reads the content of the accumulator and enables or disables the interrupts according to the content of the accumulator. Bit B3 is control bit and should be equal to 1 for bits B0, B1 and B2 to be effective. Logic 0 on B0, B1 and B2 will enable the corresponding interrupts and logic 1 will disable the interrupts.
  - ii) The second function is to reset RST7.5 FF. Bit B4 is additional control for RST7.5. If B4 is , RST7.5 is reset. This is used to override (or ignore) RST7.5 without servicing it.
  - iii) The third function is to implement serial I/O. Bits B7 and B6 of the accumulator are used for serial I/O and do not affect the interrupts. Bit B6 = 1 enables the serial I/O and bit B7 is used to transmit (output) bits.

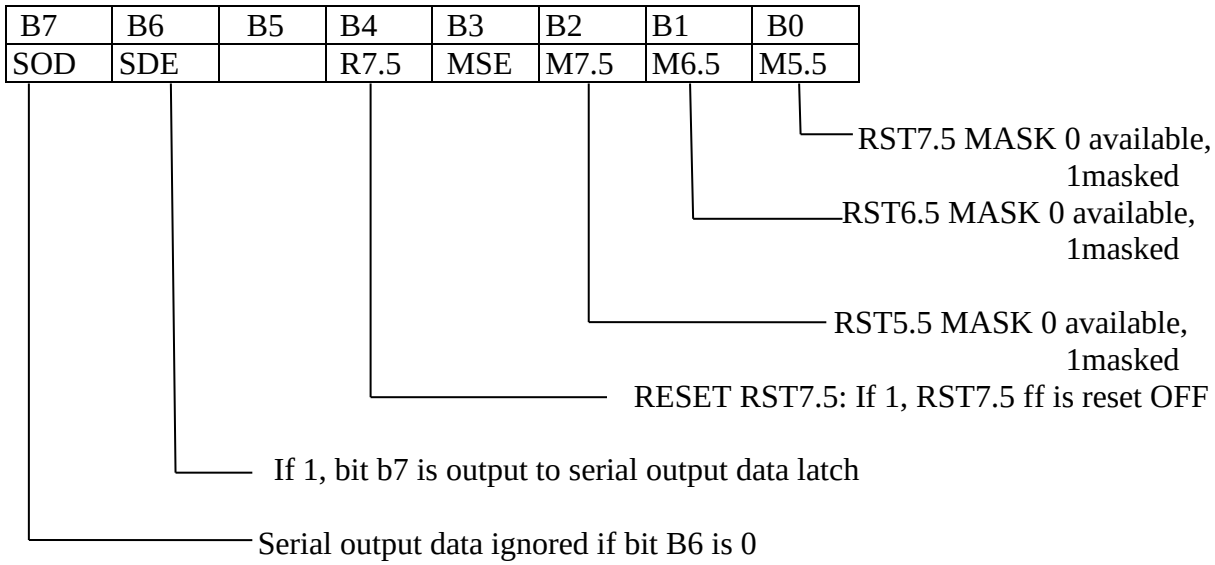


Figure 1.3: SIM INSTRUCTION FORMAT

Here we are concerned with RST7.5, RST6.5 and RST5.5 interrupts and not with serial I/O.

How interrupt process is done in the 8085 is described below:

1. The interrupt process is enabled. The instruction EI sets the Interrupt Enable FF and one of the inputs to the AND gates is set to logic 1. These AND gates activate the program transfer to various vectored locations.

2. An appropriate bit pattern is loaded into the accumulator.
3. If bit B3 is 1, the respective interrupts are enabled according to bits B2- B0.
4. RST7.5, RST6.5 and RST5.5 interrupts are being monitored.
5. If bit B3 is 0, bits B2-B0 have no effect on previous conditions.
6. Bit B4 is 1. This resets RST7.5.

The entire interrupt process (except TRAP) is disabled by resetting the Interrupt Enable FF. The FF can be reset in one of the following three ways:

- Instruction DI
- System Reset
- Recognition of an interrupt request

Figure shows that these three signals are ORed and the output of the OR gate is used to reset the FF.

Triggering Levels:

These interrupts are sensitive to different types of triggering as listed below:

RST7.5: This is positive-edge sensitive and can be triggered with a short pulse. The request is stored internally by the D flip-flop (Figure) until the microprocessor responds to the request or until it is cleared by Reset or by bit B4 in the SIM instruction.

RST6.5 and RST5.5: these interrupts are level-sensitive, meaning that the triggering level should be on until the microprocessor completes the execution of the current instruction. If the microprocessor is unable to respond to these requests immediately, they shpuld be stored or held by external hardware.

Pending Interrupts:

Because there are several interrupt lines, when one interrupt request is being served, other interrupt request may occur and remain pending. The 8085 has an additional instruction called RIM (Read Interrupt Mask) to sense these pending interrupts.

Instruction

RIM: Read Interrupt Mask

- This is 1-byte instruction.
- The instruction can be used for the following functions

i) To read interrupt masks. This instruction loads the accumulator with 8 bits indicating the current status of the interrupt masks (Figure).
ii) To identify pending interrupts. Bit B4, B5 and B6 (Figure) identify the pending interrupts.
iii) To receive serial data. Bit B7 (Figure) is used to receive serial data.

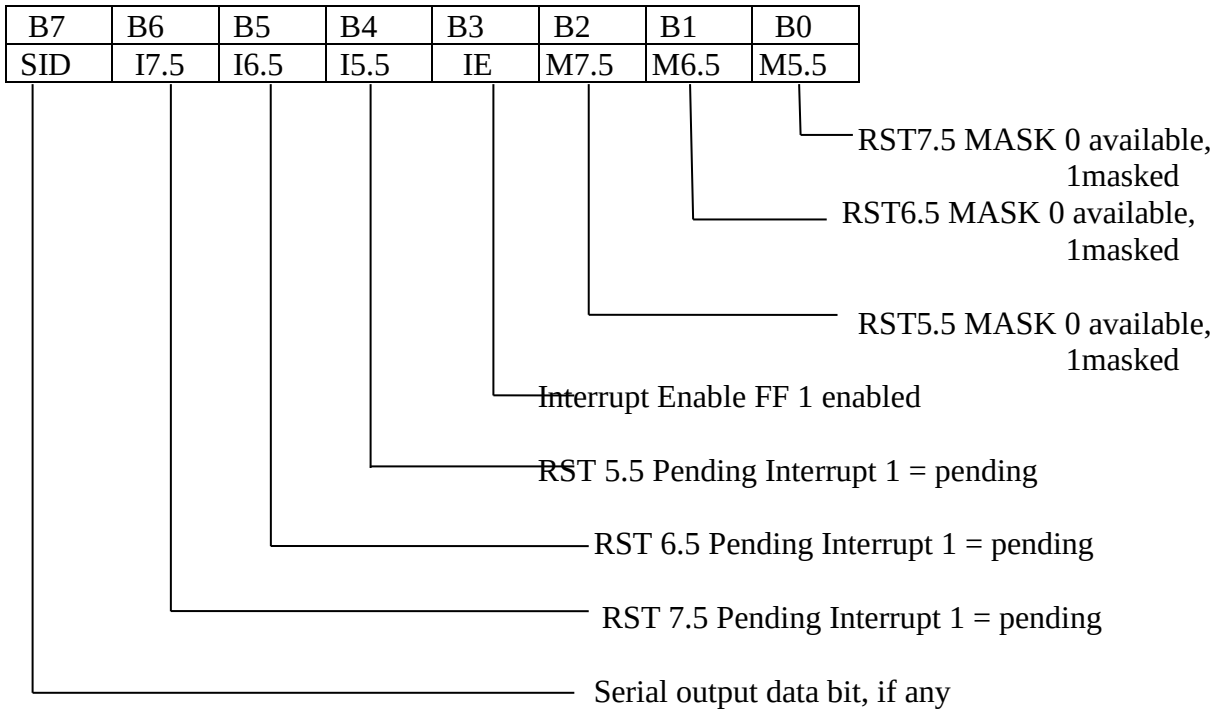


Figure 1.4: RIM INSTRUCTION FORMAT

LECTURE 4: ADDRESS / DATA BUS DEMULTIPLEXING

The 8085 uses a time multiplexed address-data bus. This is due to limited number of pins on the 8085. Low-order 8-bits of the address appear on the AD bus during the first clock cycle i.e., T_1 state of a machine cycle. It then becomes the data bus during the second and third clock cycles i.e., T_2 and T_3states.

ALE stands for address latch enable. It is used to distinguish whether the $AD_7 – AD_0$ bus contains address bits $A_7 – A_0$ or data bits D_7- D_0. It is a single pulse issued during every T_1 state of the microprocessor

Since the lower 8-bits of the address information A_7 to A_0 is available at pin no.19 to pin no.12 only during T_1 period, therefore, ALE pulse can be can be used to latch address A_7 to A_0 in an external latch. ALE output is high during first half of the T_1 period and its falling edge can be used to latch the address bits A_7 to A_0 in an external latch e.g. 74LS373 register latch.

Fig.1.5a shows a schematic that uses a latch and the ALE signal to de multiplex the bus. The bus AD_7-AD_0 is connected as the input to the latch 74LS373. The ALE signal is connected to the enable (G) pin of the latch, and the output control (OC) signal of the latch is grounded. When ALE goes high during the T_1 state of a machine cycle, the latch is transparent and the output of the latch changes according to the input. The CPU is putting lower-order bits of address during this time. When the ALE goes LOW, the address bits get latched on the output and remain so until the next ALE signal.

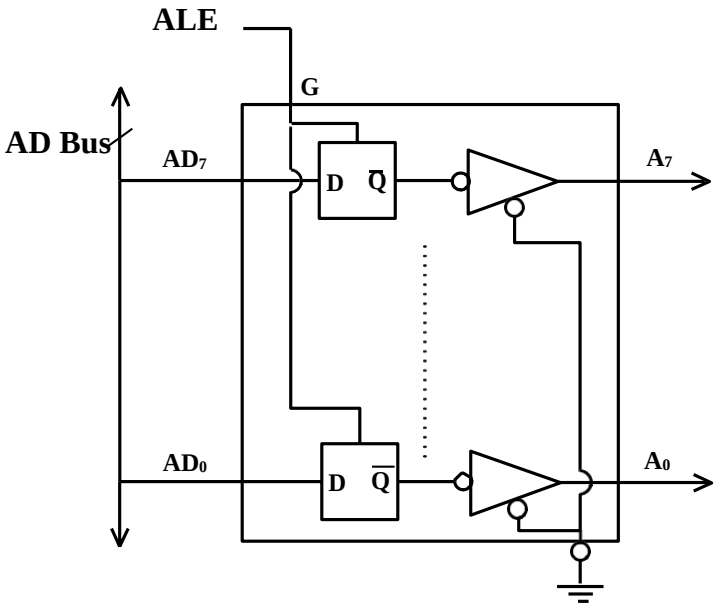


Fig.1.5a: Latching of Lower Order Address in External Latch

Once saved in an external latch the lower order address A_7 to A_0 shall be available at the output of the register latch for the subsequent states T_2, T_3, T_4, T_5 & T_6, while pin no. 19 to pin no.12 can then be utilized by the microprocessor for bi-directional operation. The falling edge of the ALE can also be used to store status information being output by the 8085 during each machine cycle. The ALE output is never tri-stated in the 8085. The manner of utilization of pins 19 to 12 is known as time multiplexed mode of operation.

The de-multiplexing of AD bus by latching lower byte of 16-bit address in external 8-bit latch 74LS373 is shown in Fig.1.5b. Once the lower byte address is latched, the AD bus is available for bi- directional data transfer. The 8-bit higher order address issued by microprocessor in every T1 state along with latched lower byte address constitutes unidirectional 16-bit address bus. The control signals put together constitutes bi-directional control bus, where some of the signals are always input and some are always output. The three buses, address bus, data bus and control bus together constitutes system bus.

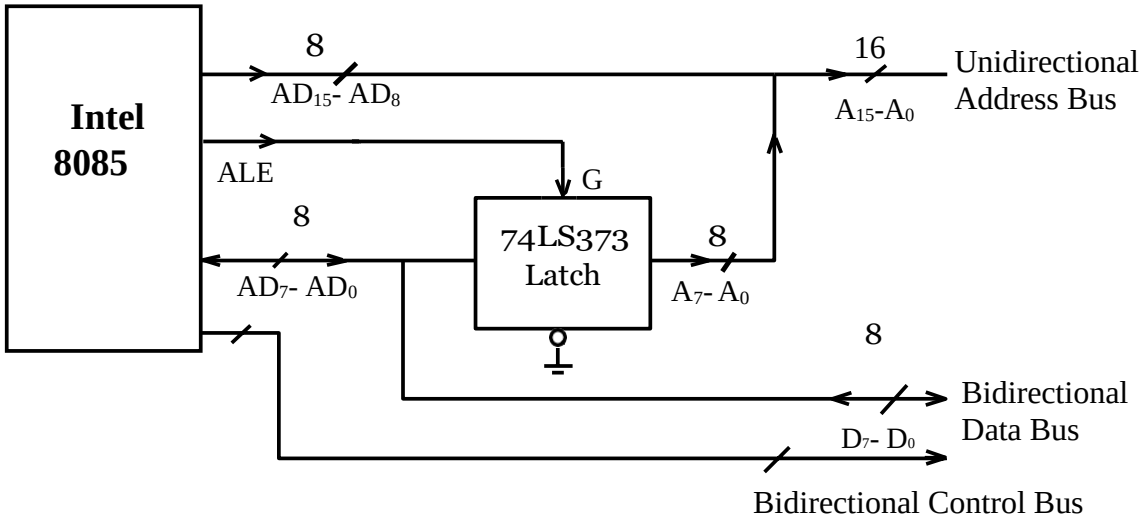


Fig.1.5b: De-multiplexing of AD bus to Generate System Bus

The fact that ALE is required is a direct consequence of having a multiplexed data/address bus. This is unlike the Intel 8080 microprocessor which is similar to the 8085 but where these buses are not multiplexed. Some of the peripheral chips 8155/ 8156/ 8355/ 8755A have internal multiplexing facility, therefore, ALE input pin of these peripheral chips is connected to ALE output pin of the 8085, thus allowing a direct interface with the 8085. Thus IC chips internally de-multiplex the AD bus using the ALE signal. Since a majority of peripheral devices do not have the internal multiplexing facility, there is external hardware necessity for it.

MODULE 2

LECTURE 1: INTRODUCTION TO 8085 PROGRAMMING MODEL

1. THE 8085 PROGRAMMING MODEL

The 8085 programming model includes six registers, one accumulator, and one flag register, Figure. In addition, it has two 16-bit registers: the stack pointer and the program counter. They are described briefly as follows

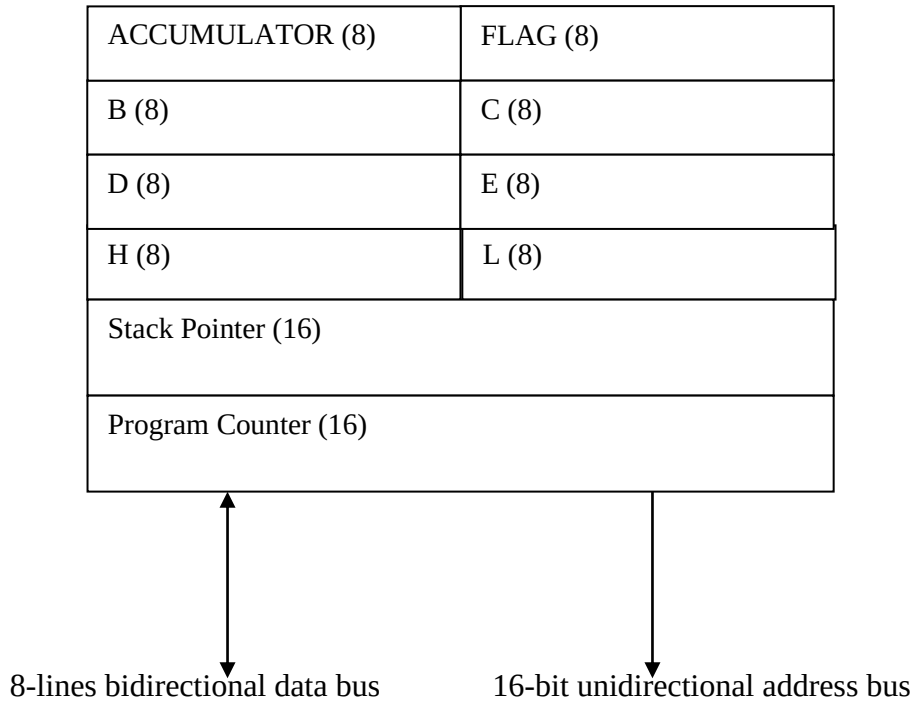


Figure 2.1: 8085 Programming Model

Registers

The 8085 has six general-purpose registers to store 8-bit data; these are identified as B,C,D,E,H, and L as shown in the figure. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions.

Accumulator

The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

Flags

The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags; their bit positions in the flag register are shown in the Figure below. The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.

B7	B6	B5	B4	B3	B2	B1	B0
S	Z		AC		P		CY

Figure 2.1: 8085 Flag Register

For example, after an addition of two numbers, if the sum in the accumulator id larger than eight bits, the flip-flop uses to indicate a carry − called the Carry flag (CY) − is set to one. When an arithmetic operation results in zero, the flip-flop called the Zero (Z) flag is set to one. The first Figure shows an 8-bit register, called the flag register, adjacent to the accumulator. However, it is not used as a register; five bit positions out of eight are used to store the outputs of the five flip-flops. The flags are stored in the 8-bit register so that the programmer can examine these flags (data conditions) by accessing the register through an instruction.

These flags have critical importance in the decision-making process of the microprocessor. The conditions (set or reset) of the flags are tested through the software instructions. For example, the instruction JC (Jump on Carry) is implemented to change the sequence of a program when CY flag is set. The thorough understanding of flag is essential in writing assembly language programs.

Sign Flag (S) – After any operation if the MSB (B(7)) of the result is 1, it in indicates the number is negative and the sign flag becomes set, i.e. 1. If the MSB is 0, it indicates the number is positive and the sign flag becomes reset i.e. 0.

from 00H to 7F, sign flag is 0
from 80H to FF, sign flag is 1

1- MSB is 1 (negative)
0- MSB is 0 (positive)

Example:

MVI A 30 (load 30H in register A)
MVI B 40 (load 40H in register B)
SUB B (A = A – B)
These set of instructions will set the sign flag to 1 as 30 – 40 is a negative number.

1. MVI A 40 (load 40H in register A)
   MVI B 30 (load 30H in register B)
   SUB B (A = A – B)
   These set of instructions will reset the sign flag to 0 as 40 – 30 is a negative number.
2. **Zero Flag (Z) –** After any arithmetical or logical operation if the result is 0 (00)H, the zero flag becomes set i.e. 1, otherwise it becomes reset i.e. 0.

   00H zero flag is 1.

   from 01H to FFH zero flag is 0

   1- zero result
   0- non-zero result

   Example:
   MVI A 10 (load 10H in register A)
   SUB A (A = A – A)
   These set of instructions will set the zero flag to 1 as 10H – 10H is 00H

3. **Auxiliary Cary Flag (AC) –** This flag is used in BCD number system (0-9). If after any arithmetic or logical operation D(3) generates any carry and passes on to B(4) this flag becomes set i.e. 1, otherwise it becomes reset i.e. 0. This is the only flag register which is not accessible by the programmer

   1-carry out from bit 3 on addition or borrow into bit 3 on subtraction
   0-otherwise

   Example:
   MOV A 2B (load 2BH in register A)
   MOV B 39 (load 39H in register B)
   ADD B (A = A + B)

   These set of instructions will set the auxiliary carry flag to 1, as on adding 2B and 39, addition of lower order nibbles B and 9 will generate a carry.

4. **Parity Flag (P) –** If after any arithmetic or logical operation the result has even parity, an even number of 1 bits, the parity register becomes set i.e. 1, otherwise it becomes reset i.e. 0.

   1-accumulator has even number of 1 bits
   0-accumulator has odd parity

Example:
MVI A 05 (load 05H in register A)

This instruction will set the parity flag to 1 as the BCD code of 05H is 00000101, which contains even number of ones i.e. 2.

5. **Carry Flag (CY) –** Carry is generated when performing n bit operations and the result is more than n bits, then this flag becomes set i.e. 1, otherwise it becomes reset i.e. 0. During subtraction (A-B), if A>B it becomes reset and if (A<B) it becomes set. Carry flag is also called borrow flag.

   1-carry out from MSB bit on addition or borrow into MSB bit on subtraction
   0-no carry out or borrow into MSB bit

   Example:
   MVI A 30 (load 30H in register A)
   MVI B 40 (load 40H in register B)
   SUB B (A = A – B)

   These set of instructions will set the carry flag to 1 as 30 – 40 generates a carry/borrow.

   MVI A 40 (load 40H in register A)
   MVI B 30 (load 30H in register B)
   SUB B (A = A – B)

   These set of instructions will reset the sign flag to 0 as 40 – 30 does not generate any carry/borrow.

Program Counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

Stack Pointer (SP)

The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

This programming model will be used in subsequent tutorials to examine how these registers are affected after the execution of an instruction.

LECTURE 2: ADDRESSING MODES IN 8085

2. 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 C, 20F: means 20F is copied into register C.

Register addressing mode

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

Direct addressing mode

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

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**:** LDAX B: means data is transferred from the memory address pointed by the register pair BC to the register A.

Implied addressing mode

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

LECTURE 3: INSTRUCTION SET (DATA TRANSFER INSTRUCTION)

3. THE 8085 INSTRUCTIONS:

The 8085 instructions can be classified into the following five functional categories: data transfer (copy) operations, arithmetic operations, logical operations, branching operations and machine-control operations.

3.1.DATA TRANSFER (COPY) INSTRUCTION

This group of instructions copies data for a location (register or I/O or memory) called the source, to another location (register or I/O or memory) called the destination. The contents of the source are not changed. In the8085 processor, data transfer instructions do not affect the flags.
The instructions used to copy are:

1. MOV
2. MVI
3. LXI
4. LDA
5. STA
6. LDAX
7. STAX
8. LHLD
9. SHLD
10. XCHG
11. IN
12. OUT

1. MOV: Move - Copy from Source to Destination

Opcode	Operand	Bytes	Register Transfer Logic
MOV	Rd, Rs	1	(Rd) ← (Rs)
MOV	M, Rs	1	((H)(L)) ← (Rs)
MOV	Rd,M	1	(Rd) ← ((H)(L))

Description: This instruction copies the contents of the source register into the destination register, the content of the source register are not altered. If one of the operands is a memory location, it is specified by the contents of HL registers.
Flags: No flags are affected

Example 1:

Let us assume register B contains 72H and register C contains 9F. We move the contents of

register C to register B.

Instruction: MOV B, C

It can be noted that the first operand B specifies the destination and the second operand C specifies the source.

Register contents before instruction

B	72	9F	C

Register contents before instruction

B	9F	9F	C

Example 2:

Let us assume register B contains 72H and registers H and L are 20 and 50, respectively. Memory location 2050 contains 9F. We transfer the contents of the memory location to register B.

Instruction: MOV B, M

Contents before instruction

B	72	9F	C
D			E
H			L

2050	9F

Contents after instruction

B	72	9F	C
D			E
H			L

2050	9F

2. MVI: Move Immediate 8-bit

Opcode	Operand	Bytes	Register Transfer Logic
MVI	Rd, <data>	2	(Rd) ← <data>
MVI	M, <data>	2	((H)(L)) ← <data>

Description: The 8-bit data are stored in the destination register or memory. If the operand is a memory location, it is specified by the contents of HL registers.

Flags: No flags are affected

Example 1:

Let us load 92 in register B

Instruction: MVI B, 92

This instruction loads 92 in register B.

Example 2:

Let us assume that registers H and L contain 20 and 50, respectively. We want to load 3A in memory location 2050.

Instruction: MVI M, 3A

Contents before instruction

H	20	50	L

2050	XX

Contents after instruction

H	20	50	L

2050	3A

3. LXI: Load Register pair Immediate

Opcode	Operand	Bytes	Register Transfer Logic
LXI	Reg. Pair, <16-bit data>	3	(Rl) ← Bytel (Rh) ← Byteh

Description: The instruction loads 16-bit data in the register pair designated in the operand. This is a 3-byte instruction; the second byte specifies the low-order byte and the third byte specifies the high-order byte.

Flags: No flags are affected

Example :

We want to load the 16-bit data 2050 in register pair BC.

Instruction: LXI B, 2050

This instruction loads 50 in register C and 20 in register B.

[Comments: We can note the reverse order in entering the code of 16-bit data. This is the only instruction that can directly load a 16-bit address in the stack pointer register.]

4. LDA: Load Accumulator Direct

Opcode	Operand	Bytes	Register Transfer Logic
LDA	<16-bit address>	3	(A) ← (16-bit address)

Description: The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. This is a 3-byte instruction; the second byte specifies the low-order address and the third byte specifies the high-order address.

Flags: No flags are affected

Example :

Let us assume memory location 9050 contains byte F8. We want to load the accumulator with the contents of location 9050.

Instruction: LDA 2050

Contents before instruction

A	FF	XX	F

9050	XX

Contents after instruction

A	FF	XX	F

9050	F8

5. STA: Store Accumulator Direct

Opcode	Operand	Bytes	Register Transfer Logic
STA	<16-bit address>	3	(16-bit address) ← (A)

Description: The contents of the accumulator are copied to a memory location specified the operand, are copied to a memory location specified by the operand. This is a 3-byte instruction; the second byte specifies the low-order address and the third byte specifies the high-order address.

Flags: No flags are affected

Example:

Let the accumulator contains byte 9F. We want to store the accumulator content into memory location 9050.

Instruction: STA 9050

Contents before instruction

A	9F	XX	F

9050	XX

Contents after instruction

A	9F	XX	F

9050	9F

6. LDAX: Load Accumulator Indirect

Opcode	Operand	Bytes	Register Transfer Logic
LDAX	B/D Reg. pair	1	(A) ← $((R_h)(R_l))$

Description: The contents of the designated register pair point to a memory location. This instruction copies the content of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered.

Flags: No flags are affected

Example :

Let us assume the contents of register B = 90, C = 50, and memory location 9050 = 9F. We want to transfer the contents of the memory location 9050 to the accumulator.

Instruction: LDAX B

Contents before instruction

A	XX	XX	F
B	90	50	C

9050	9F

Contents after instruction

A	9F	XX	F
B	90	50	C

9050	9F

7. STAX: Store Accumulator Indirect

Opcode	Operand	Bytes	Register Transfer Logic
LDAX	B/D Reg. pair	1	(A) ← ((Rh)(Rl))

Description: The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered.

Flags: No flags are affected

Example :

Let us assume the contents of accumulator are F9 and the contents of B and C are 90 and 50 respectively. We want to store the accumulator contents in memory location 9050

Instruction: STAX B

Contents before instruction

A	F9	XX	F
B	90	50	C

9050	XX

Contents after instruction

A	F9	XX	F
B	90	50	C

9050	F9

[Comments: This instruction performs the same function as MOV M, A except this instruction uses the contents of BC or DE as memory pointers.]

8. LHLD: Load H and L Registers Direct

Opcode	Operand	Bytes	Register Transfer Logic
LHLD	16-bit address	3	(L) ← (address) (H) ← (address+1)

Description: The instruction copies the contents of the memory location pointed out by the 16-bit address in register L and copies the contents of the next memory location.

Flags: No flags are affected

Example :

Let us assume memory location 9050 contains 90 and 9051 contains 01. Transfer of memory contents to register HL is to be performed.

Instruction: LHLD 9050

Contents before instruction

H	XX	XX	L

9050	90
9051	01

Contents after instruction

H	01	90	L

9050	90
9051	01

9. SHLD: Store H and L Registers Direct

Opcode	Operand	Bytes	Register Transfer Logic
SHLD	16-bit address	3	(address) ← (L) (address+1) ← (H)

Description: The contents of register L are stored in the memory location specified by the 16-bit address in the operand, and the contents of register H are stored in the next memory location by incrementing the operand. The contents of register HL are not altered. This is a 3-byte instruction; the second byte specifies the low-order address and the third byte specifies the high-order address.

Flags: No flags are affected

Example :

Let us assume that the H and L registers contain 01 and FF respectively. We want to store the contents at memory locations 9050 and 9051.

Instruction: SHLD 9050

Contents before instruction

H	01	FF	L

9050	XX
9051	XX

Contents after instruction

H			L
	01	FF	

9050	FF
9051	01

10. XCHG: Exchange H and L with D and E

Opcode	Operand	Bytes	Register Transfer Logic
XCHG	-	1	(H) ← (D) (L) ← (E)

Description: The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.

Flags: No flags are affected

Example :

Let us assume that registers H and L contains the data bytes 98 and A4, respectively. Again, let us assume that registers D and E contains the data bytes C7 and 05, respectively.

Instruction: XCHG

Contents before instruction

H			L
	98	A4	

D			E
	C7	05	

Contents after instruction

H	C7	05	L
D	98	A4	E

11. IN: Input Data to Accumulator from a Port with 8-bit Address

Opcode	Operand	Bytes	Register Transfer Logic
IN	<8-bit port address>	2	(A) ← (port address)

Description: The contents of the input port designated in the operand are read and loaded into the accumulator.

Flags: No flags are affected

[Comments: The operand is an 8-bit address; therefore, port addresses can range from 00 to FF. While executing the instruction, a port address is duplicated on low-order (A7-A0) and high-order (A15-A8) address buses. Any one of the sets of address lines can be decoded to enable the input port.]

12. OUT: Output Data from Accumulator to a Port with 8-bit Address

Opcode	Operand	Bytes	Register Transfer Logic
OUT	<8-bit port address>	2	(port address) ← (A)

Description: The contents of the accumulator are copied into the output port specified by the operand.

Flags: No flags are affected

[Comments: The operand is an 8-bit address; therefore, port addresses can range from 00 to FF. While executing the instruction, a port address is duplicated on low-order (A7-A0) and high-order (A15-A8) address buses. Any one of the sets of address lines can be decoded to enable the output port.]

LECTURE 4: INSTRUCTION SET (ARITHMETIC INSTRUCTION)

3.2.ARITHMETIC INSTRUCTION:

The 8085 performs various arithmetic operations, such as addition, subtraction, increment and decrement. Arithmetic instructions perform these operations.
The instructions used to perform arithmetic operations are

1. ADD
2. ADI
3. ACI
4. ADC
5. SUB
6. SUI
7. SBB
8. SBI
9. INR
10. DCR
11. INX
12. DCX
13. DAD
14. DAA

1. ADD: Add Register to Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
ADD	R	1	(A) ⟵ (A) + (R)
ADD	M	1	(A) ⟵ (A) + ((H)(L))

Description: The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, that is indicated by the 16-bit address in the HL register

Flags: All flags are modified to reflect the result of the addition.

2. ADI: Add Immediate to Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
ADI	8-bit data	2	(A) ⟵ (A) + <data>

Description: The 8-bit data are added to the contents of the accumulator and the result is stored in the accumulator.

Flags: All flags are modified to reflect the result of the addition.

3. ACI: Add Immediate to Accumulator with carry

Opcode	Operand	Bytes	Register Transfer Logic
ACI	8-bit data	2	(A) ← (A) + CY + <data>

Description: The 8-bit data (operand) and the carry flag are added to the contents of the accumulator, and the result is stored in the accumulator.

Flags: All flags are modified to reflect the result of the addition.

[Comments: This instruction is commonly used in 16-bit addition]

4. ADC: Add Register to Accumulator with carry

Opcode	Operand	Bytes	Register Transfer Logic
ADC	R	1	(A) ← (A) + (R) + CY
ADC	M	1	(A) ← (A) + ((H)(L) + CY

Description: The contents of the operand (register or memory) and the carry flag are added to the contents of the accumulator and the result is placed in the accumulator. The contents of the operand are not altered, however, the previous carry flag is reset.

Flags: All flags are modified to reflect the result of the addition.

Example :

Let us assume that register pair BC contains 2498 and register pair de contains 54A1. We add these 16-bit numbers and save the result in BC registers.

The steps in adding 16-bit numbers are as follows:

1. We add the contents of registers C and E by placing the contents of one register in the accumulator. This addition generates a carry. We use instruction ADD and save the low-order 8-bits in register C.

```
98 =   10011000
A1 =  10100001
-----------------
1  39 = 1 00111001
```

2. We add the contents of registers B and D by placing the contents of one register in the accumulator. We use instruction ADC.
   The result will be as follows:

24 = 00100100
54 = 01010100
1 1 (Carry from previous addition)

79 = 01111001 (We store in register B)

[Comments: This instruction is commonly used in 16-bit addition]

5. SUB: Subtract Register or Memory from Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
SUB	R	1	(A) ← (A) − (R)
SUB	M	1	(A) ← (A) − ((H)(L))

Description: The contents of the register or the memory location specified by the operand are subtracted from the contents of the accumulator and the results are placed in the accumulator. The contents of the source are not altered.

Flags: All flags are modified to reflect the result of the subtraction.

Example 1:

Let us assume that the contents of the accumulator are 37 and the contents of the register C are 40. We subtract the contents of register C from the accumulator.

Instruction: SUB C

(C) : 40 = 01000000
2's complement of (C) = 11000000

(A) : 37 = 00110111

0/11110111 = F7

Complement carry: 1/11110111

Flags: S=1, Z=0, AC=0, P=0, CY=1

The result, as a negative number, will be in 2's complement and thus the Carry (Borrow) flag

is set.

Example 2:

Let us assume that the contents of the accumulator are 40 and the contents of the register C are 37. We subtract the contents of register C from the accumulator.

Instruction: SUB C

(C) : 37 = 00110111
2's complement of (C) = 11001001

(A): 40 = 01000000

0/00001001

Complement carry: 0/00001001 = 09

Flags: S=0, Z=0, AC=0, P=1, CY=0

6. SUI: Subtract Immediate from Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
SUI	8-bit data	2	(A) ← (A) − <data>

Description: The 8-bit data are subtracted from the contents of the accumulator and the results are placed in the accumulator.

Flags: All flags are modified to reflect the result of the subtraction.

7. SBB: Subtract Source and Borrow from Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
SBB	R	1	(A) ← (A) − (R) − B
SBB	M	1	(A) ← (A) − ((H)(L)) − B

Description: The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the results are placed in the accumulator.

Flags: All flags are modified to reflect the result of the subtraction.

8. SBI: Subtract Immediate with Borrow

Opcode	Operand	Bytes	Register Transfer Logic
SBI	8-bit data	2	(A) ← (A) − data − B

Description: The 8-bit data (operand) and the borrow are subtracted from the contents of the accumulator, and the results are placed in the accumulator.

Flags: All flags are modified to reflect the result of the operation.

9. INR: Increment Contents of Register/Memory by 1

Opcode	Operand	Bytes	Register Transfer Logic
INR	R	1	(R) ← (R) + 1
INR	M	1	((H)(L)) ← ((H)(L)) + 1

Description: The contents of the designated register / memory are incremented by 1 and the results are stored in the same place. If the operand is a memory location, it is specified by the contents of HL register pair.

Flags: All flags except CY are modified to reflect the result of the operation.

Example 1:

Register D contains FF. we shall specify the contents of the register after the increment.

Instruction: INR D

(D) : FF = 11111111
+1 = 00000001
0/00000000 = 00

After the execution of the INR instruction, register D will contain 00; however, carry flag is not modified.

Example 2:

We assume that the HL register contains 9075. We shall increment the contenmts of memory location 9075, which presently holds 7F.

Instruction: INR M

Contents before instruction

H	90	75	L	9075	7F

Contents after instruction

H	90	75	L	9075	80

10. DCR: Decrement Contents of Register/Memory by 1

Opcode	Operand	Bytes	Register Transfer Logic
DCR	R	1	(R) ← (R) − 1
DCR	M	1	((H)(L)) ← ((H)(L)) − 1

Description: The contents of the designated register / memory are decremented by 1 and the results are stored in the same place. If the operand is a memory location, it is specified by the contents of HL register pair.

Flags: All flags except CY are modified to reflect the result of the operation.

Example :

Register B contains 00. we shall specify the contents of the register after the decrement.

Instruction: DCR B

+1 = 00000001
2's complement of 1's = 11111111

(B) : 00 = 00000000

11111111 = FF

After the execution of the DCR instruction, register B will contain FF; however, carry flag is not

modified.

11. INX: Increment Register Pair by 1

Opcode	Operand	Bytes	Register Transfer Logic
INX	Reg. pair	1	(Reg. pair) ⟵ (Reg. pair) + 1

Description: The contents of the designated register pair are incremented by 1. The instruction views the contents of the two registers as a 16-bit number.

Flags: No flags are affected.

Example :

Register pair HL contains 9FFF. We shall specify the contents of the entire register if it is incremented by 1.

Instruction: INX H

After adding 1 to the contents of the HL pair the answer is (H) = A0 , (L) = 00

Contents before instruction

H	9F	FF	L

Contents after instruction

H	A0	00	L

12. DCX: Decrement Register Pair by 1

Opcode	Operand	Bytes	Register Transfer Logic
DCX	Reg. pair	1	(Reg. pair) ⟵ (Reg. pair) −1

Description: The contents of the designated register pair are decremented by 1. The instruction views the contents of the two registers as a 16-bit number.

Flags: No flags are affected.

Example :

Register pair DE contains 2000. We shall specify the contents of the entire register if it is decremented by 1.

Instruction: DCX D

After adding 1 to the contents of the HL pair the answer is (D) = 1F , (E) = FF

Contents before instruction

D	20	00	E

Contents after instruction

D	1F	FF	E

13. DAD: Add Register Pair to H and l register pair

Opcode	Operand	Bytes	Register Transfer Logic
DAD	Reg. pair	1	(HL) ← (Reg. pair) + (HL)

Description: The 16-bit contents of the designated register pair are added to the contents of the HL register and the sum is saved in the HL register. The contents of the source register pair are not altered.

Flags: If the result is larger than 16 bits the CY flag is set. No other flags are affected

Example 1:

We assume that the register pair HL contains 0242. We want to multiply the contents by 2

Instruction: DAD H

Contents before instruction

H	02	42	L

DAD operation

```
  0242
+ 0242
------
  0484
```

Contents after instruction

H	04	84	L

Example 2:

We assume that the register pair HL is cleared. We want to transfer the content of Stack Pointer (SP) register that points to memory location FFFE to the HL register pair

Instruction: DAD SP

Contents before instruction

H	00	00	L	SP	FFFE

DAD operation

```
  0000
+ FFFE
------
  FFFE
```

Contents after instruction

H	FF	FE	L	SP	FFFE

14. DAA: Decimal-Adjust Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
DAA	-	1	

Description: The contents of the Accumulator are changed from a binary value to two binary-coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag (internally) to perform the binary-to-BCD conversion.

Flags: All flags are modified to reflect the result of the operation.

LECTURE 5: INSTRUCTION SET (LOGICAL INSTRUCTION)

3.3.LOGICAL INSTRUCTION:

Logical Instructions:

These instructions perform various logical operations with the contents of the accumulator.

These instructions include:

1. ANA
2. ANI
3. ORA
4. ORI
5. XRA
6. XRI
7. CMA
8. CMP
9. CPI
10. RLC
11. RAL
12. RRC
13. RAR
14. STC
15. CMC

1. ANA: Logical AND with Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
ANA	R	1	(A) ← (A) AND (R)
ANA	M	1	(A) ← (A) AND ((H)(L))

Description: The contents of the Accumulator are logically ANDed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers.

Flags: S, Z, P are modified to reflect the result of the operation. **CY is reset. AC is set**.

2. ANI: AND Immediate with Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
ANI	<8-bit data>	2	(A) ← (A) AND <data>

Description: The contents of the Accumulator are logically ANDed with the 8-bit data (operand)

and the result is placed in the accumulator.

Flags: S, Z, P are modified to reflect the result of the operation. **CY is reset. AC is set**.

3. ORA: Logical OR with Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
ORA	R	1	(A) ⟵ (A) OR (R)
ORA	M	1	(A) ⟵ (A) OR ((H)(L))

Description: The contents of the Accumulator are logically ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers.

Flags: S, Z, P are modified to reflect the result of the operation. **CY and AC are reset.**

4. ORI: OR Immediate with Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
ANI	<8-bit data>	2	(A) ⟵ (A) OR <data>

Description: The contents of the Accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator.

Flags: S, Z, P are modified to reflect the result of the operation. **CY and AC are reset.**

5. XRA: Logical XOR with Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
XRA	R	1	(A) ⟵ (A) XOR (R)
XRA	M	1	(A) ⟵ (A) XOR ((H)(L))

Description: The contents of the Accumulator are logically XORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers.

Flags: S, Z, P are modified to reflect the result of the operation. **CY and AC are reset.**

6. XRI: XOR Immediate with Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
ANI	<8-bit data>	2	(A) ⟵ (A) XOR <data>

Description: The contents of the Accumulator are logically XORed with the 8-bit data (operand) and the result is placed in the accumulator.

Flags: S, Z, P are modified to reflect the result of the operation. **CY and AC are reset.**

7. CMA: Complement Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
CMA	-	1	(A) ⟵ (A)′

Description: The content of the Accumulator is complemented.

Flags: **No flags are affected**.

8. CMP: Compare with Accumulator

Opcode	Operand	Bytes	Register Transfer Logic
CMP	R	1	
CMP	M	1	

Description: The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved and the comparison is shown by setting the flags as follows:

If (A) < (R) / (M) Then CY = 1, Z = 0
If (A) = (R) / (M) Then CY = 0, Z = 1
If (A) > (R) / (M) Then CY = 0, Z = 0

The comparison of two bytes is performed by subtracting the contents of the operand from the contents of the accumulator; however, neither contents are modified.

Flags: S, P, AC are also modified in addition to Z and CY to reflect the results of the operation. (here, S, P, AC are also modified according to the results of the subtraction).

9. CPI: Compare Immediate with Accumulator

Opcode	Operand	Bytes
CPI	<8-bit data>	2

Description: The data is compared with the contents of the accumulator. The values being compared remain unchanged and the results of the comparison are indicated by setting the flags as follows:

If (A) < data Then CY = 1, Z = 0
If (A) = data Then CY = 0, Z = 1
If (A) > data Then CY = 0, Z = 0

The comparison of two bytes is performed by subtracting the data byte from the contents of the accumulator; however, neither are modified.

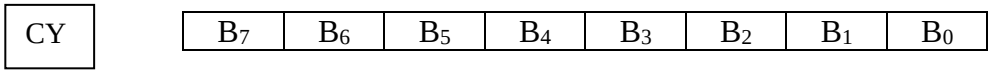
Flags: S, P, AC are also modified in addition to Z and CY to reflect the results of the operation. (here, S, P, AC are also modified according to the results of the subtraction).

10. RLC: Rotate Accumulator Left Without Carry

Opcode	Operand	Bytes
RLC	-	1

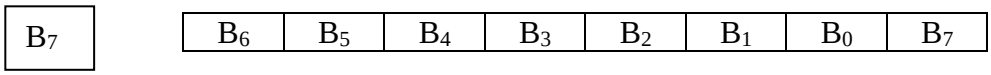
Description: Each binary bit of the accumulator is rotated left by one position. Bit B_7 is placed in the position of B_0 as well as in the CY flag.

Before RLC Instruction:



Accumulator

After RLC Instruction:



Accumulator

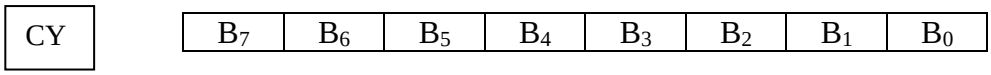
Flags: CY is modified in according to bit B_7. S, Z, P, AC are not affected.

11. RAL: Rotate Accumulator Left with Carry

Opcode	Operand	Bytes
RAL	-	1

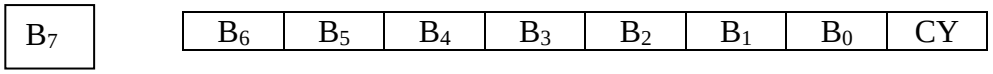
Description: Each binary bit of the accumulator is rotated left by one position through the carry flag. Bit B_7 is placed in the CY flag and the CY flag is placed in the least significant position of Accumulator.

Before RAL Instruction:



Accumulator

After RAL Instruction:



Accumulator

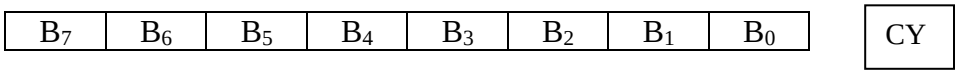
Flags: CY is modified in according to bit B_7. S, Z, P, AC are not affected.

12. RRC: Rotate Accumulator Right Without Carry

Opcode	Operand	Bytes
RRC	-	1

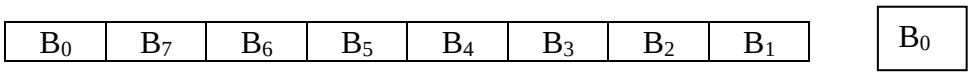
Description: Each binary bit of the accumulator is rotated right by one position. Bit B_0 is placed in the position of B_7 as well as in the CY flag.

Before RRC Instruction:



Accumulator

After RRC Instruction:



Accumulator

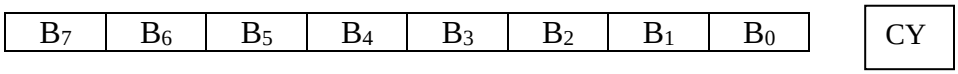
Flags: CY is modified in according to bit B_0. S, Z, P, AC are not affected.

13. RAR: Rotate Accumulator Right with Carry

Opcode	Operand	Bytes
RAR	-	1

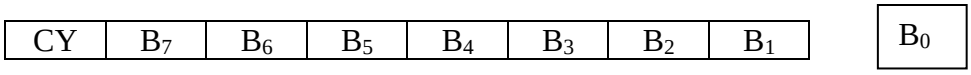
Description: Each binary bit of the accumulator is rotated right by one position through the carry flag. Bit B_0 is placed in the CY flag and the bit in the CY flag is placed in the most significant position of Accumulator.

Before RAR Instruction:



Accumulator

After RAR Instruction:



Accumulator

Flags: CY is modified in according to bit B_0. S, Z, P, AC are not affected.

14. STC: Set Carry

Opcode	Operand	Bytes	Register Transfer Logic
STC	-	1	CY ← 1

Description: The carry flag is set to 1.

Flags: CY is set to 1. S, Z, P, AC are not affected.

15. CMC: Complement Carry

Opcode	Operand	Bytes	Register Transfer Logic
STC	-	1	CY ← CY′

Description: The carry flag is complemented.

Flags: CY is modified. S, Z, P, AC are not affected.

LECTURE 6: INSTRUCTION SET (BRANCH INSTRUCTION)

3.4.BRANCH INSTRUCTION:

BRANCH INSTRUCTIONS:

This group of instructions alters the sequence of program execution either conditionally or unconditionally.

The branch instructions are classified in three categories:

A. Jump instructions
B. Call and Return instructions
C. Restart instructions

A. Jump instructions: The jump instructions specify the memory location explicitly. They are 3-byte instructions: one byte for the operation code, followed by a 16-bit memory address. Jump instructions are classified into two categories: Unconditional Jump and Conditional Jump.

A.1. Unconditional Jump: The 8085 instruction set includes one unconditional Jump instruction. It is

1. JMP:

1. JMP: Jump Unconditionally

Opcode	Operand	Bytes	Register Transfer Logic
JMP	<16-bit address>	3	(PC) ← address

Description: The program sequence is transferred to the memory location specified by the 16-bit address. This is a 3-byte instruction; the second byte specifies the low-order byte and the third byte specifies the high-order byte.

Flags: No flags are affected.

Example: We write the instruction at location 9000 to transfer the program sequence to memory location 9050.

Instruction:

JMP 9050

Memory Address	Hex Code	Instruction
9000	C3	JMP 9050
9001	50	
9002	90	

[Comments: The 16-bit address of the operand is entered in memory in reverse order, the low-order byte first, followed by the high-order byte.]

A.2. Conditional Jump: Conditional Jump instructions allow the microprocessor to make decisions based on certain conditions indicated by the flags. The conditional Jump instructions check the flag conditions and make decisions to change or not to change the sequence of a program. All conditional Jump instructions in the 8085 are 3-byte instructions; the second-byte specifies the low-order memory address, and the third byte specifies the high-order memory address. The following instructions transfer the program sequence to the memory location specified under the given conditions:

1. JC
2. JNC
3. JZ
4. JNZ
5. JP
6. JM
7. JPE
8. JPO

Opcode	Operand	Bytes	Description
JC	<16-bit address>	3	Jump if Carry
JNC	<16-bit address>	3	Jump if Not Carry
JZ	<16-bit address>	3	Jump if Zero
JNZ	<16-bit address>	3	Jump if Not Zero
JP	<16-bit address>	3	Jump if Plus / Positive
JM	<16-bit address>	3	Jump if Minus
JPE	<16-bit address>	3	Jump if Parity Even
JPO	<16-bit address>	3	Jump if Parity Odd

Table 2.1: Conditional Jump Instructions

Flags: No flags are affected.

B. Call and Return instructions:

B.a. Call Instructions:

The Call instructions specify the memory location explicitly. They are 3-byte instructions: one

byte for the operation code, followed by a 16-bit memory address. Call instructions are classified into two categories: Unconditional Call and Conditional Call.

B.a.1. Unconditional Call: The 8085 instruction set includes one unconditional Call instruction. It is

1. CALL:

1. CALL: Call Unconditionally

Opcode	Operand	Bytes	Register Transfer Logic
CALL	<16-bit address>	3	SP ← SP − 1 M(SP) ← $(PC)_h$ SP ← SP − 1 M(SP) ← $(PC)_l$ $(PC)_l$ ← byte 2 $(PC)_h$ ← byte 3

Description: The program sequence is transferred to the memory location specified by the 16-bit address. Before the transfer, the address of the next instruction to CALL (the contents of the program counter) is pushed on the stack.

Flags: No flags are affected.

Example: We write CALL instruction at location 9010 to call a subroutine at 9150.

Instruction:

CALL 9150

Memory Address	Hex Code	Instruction
9010	CD	CALL 9150
9011	50	
9012	91	

Execution of CALL:

The address of the program counter (9013) is placed on the stack

FFFD	13
FFFE	90

SP → FFFF

Call address (9150) is stored in the PC.

PC → 9150

[Comments: The CALL instruction should be accompanied by one of the return (RET or conditional return) instructions in the subroutine]

B.a.2. Conditional Call: The Conditional Call instructions are based on four data conditions (flags): CY, Z, S and P. The conditions are tested by checking the respective flags. In case of a conditional Call instruction, the program is transferred to the subroutine if the condition is met; otherwise, the main program is continued. The conditional Call instructions are listed below:

1. CC
2. CNC
3. CZ
4. CNZ
5. CP
6. CM
7. CPE
8. CPO

Opcode	Operand	Bytes	Description
CC	<16-bit address>	3	Call if Carry
CNC	<16-bit address>	3	Call if Not Carry
CZ	<16-bit address>	3	Call if Zero
CNZ	<16-bit address>	3	Call if Not Zero
CP	<16-bit address>	3	Call if Plus / Positive
CM	<16-bit address>	3	Call if Minus
CPE	<16-bit address>	3	Call if Parity Even
CPO	<16-bit address>	3	Call if Parity Odd

Table 2.2: Conditional Call Instructions

Flags: No flags are affected.

B.b. Return Instructions:

The Return instructions are used at the end of the subroutine to return to the calling program. Return instructions can be classified into two categories: Unconditional return and Conditional return

B.b.1. Unconditional Return: The 8085 instruction set includes one unconditional Return instruction. It is

1. RET:

1. RET: Return Unconditionally

Opcode	Operand	Bytes	Register Transfer Logic
RET	–	1	$(PC)_l \leftarrow M(SP)$ $SP \leftarrow SP + 1$ $(PC)_h \leftarrow M(SP)$ $SP \leftarrow SP + 1$

Description: The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter and the program execution begins at the new address.

Flags: No flags are affected.

Example: Let us assume that the stack pointer is pointing to location FFFD. If we write the RET instruction, then after execution, the effect will be as follows:

Instruction:

RET

Memory Address	Hex Code	Instruction
9010	CD	CALL 9150
9011	50	
9012	91	

Execution of RET:

The address of the program counter (9013) is placed on the stack

	FFFD	13
	FFFE	90
SP →	FFFF	

Call address (9150) is stored in the PC.

PC → | 90 13 |

[Comments: This instruction is used in conjunction with CALL or conditional call instructions.]

B.b.2. Conditional Return: The Conditional Return instructions are based on four data conditions (flags): CY, Z, S and P. The conditions are tested by checking the respective flags. In case of a

conditional Return instruction, the program sequence returns to the calling program if the condition is met; otherwise, the sequence in the subroutine is continued.. The conditional Call instructions are listed below:

1. RC
2. RNC
3. RZ
4. RNZ
5. RP
6. RM
7. RPE
8. RPO

Opcode	Operand	Bytes	Description
RC	-	1	Return if Carry
RNC	-	1	Return if Not Carry
RZ	-	1	Return if Zero
RNZ	-	1	Return if Not Zero
RP	-	1	Return if Plus / Positive
RM	-	1	Return if Minus
RPE	-	1	Return if Parity Even
RPO	-	1	Return if Parity Odd

Table 2.3: Conditional Return Instructions

Flags: No flags are affected.

3. Restart (RST) instructions:

RST instructions are 1-byte call instructions that transfer the program execution to a specific location on page 00. They are executed the same way as Call instructions. When an RST instruction is executed, the 8085 stores the contents of the program counter (the address of the next instruction) on the top of the stack and transfer the program to the Restart location.

There are 8 RST instructions as follows:

1. RST 0
2. RST 1
3. RST 2
4. RST 3
5. RST 4
6. RST 5
7. RST 6
8. RST 7

Opcode	Operand	Bytes	Restart Address (in Hex)

RST0	-	1	0000
RST1	-	1	0008
RST2	-	1	0010
RST3	-	1	0018
RST4	-	1	0020
RST5	-	1	0028
RST6	-	1	0030
RST7	-	1	0038

Table 2.4: Restart Instructions

Flags: No flags are affected.

There is an additional branch instruction namely

PCHL

Opcode	**Operand**	**Bytes**	**Register Transfer Logic**
PCHL	-	1	(PC)h ← (H) (PC)l ← (L)

Description: The contents of registers H and L are copied into the program counter. The contents of H are placed as a high-order byte and of L as low-order byte.

Flags: No flags are affected.

LECTURE 7: INSTRUCTION SET (MACHINE CONTROL INSTRUCTION)

3.5.MACHINE CONTROL INSTRUCTIONS:

These instructions control machine functions such as Stack, Interrupt, Halt or do nothing. These instructions include:

1. PUSH
2. POP
3. XTHL
4. SPHL

5. DI
6. EI
7. RIM
8. SIM

9. HLT
10. NOP

1. PUSH: Push Register Pair Onto Stack

Opcode	Operand	Bytes	Register Transfer Logic
PUSH	Reg. Pair	1	SP ← SP − 1 M(SP) ← $(R)_h$ SP ← SP − 1 M(SP) ← $(R)_l$
PUSH	PSW	1	SP ← SP − 1 M(SP) ← (A) SP ← SP − 1 M(SP) ← (F)

Description: The contents of the register pair designated in the operand are copied into the stack in the following sequence. The stack pointer register (SP register) is decremented and the contents of high-order register are copied into the location. The SP register is decremented again and the contents of the low-order register are copied to that location.

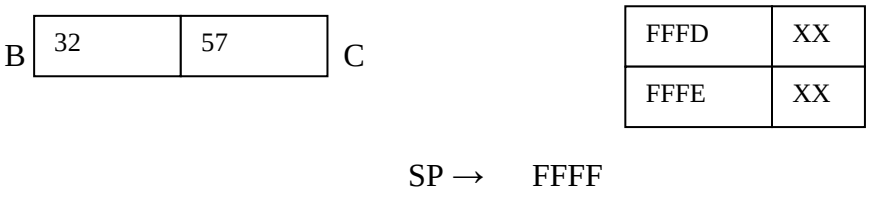
Example:

We assume that the SP registers points to location FFFF. Let register B contains 32 and register C contains 57. We want to save the contents of the BC register pair on the stack.

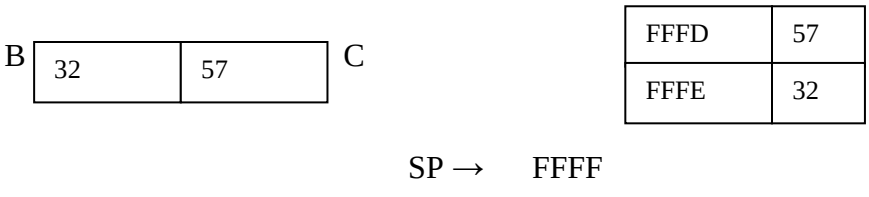
Instruction:

PUSH B

Contents before instruction



Contents after instruction



[Comments: Operand PSW (Program Status Word) represents the contents of the accumulator and the flag register; the accumulator is the high-order register and the flags are the low-order register].

2. POP: Pop off Stack to Register Pair

Opcode	Operand	Bytes	Register Transfer Logic
POP	Reg. Pair	1	$(R)_l \leftarrow M(SP)$ $SP \leftarrow SP + 1$ $(R)_h \leftarrow M(SP)$ $SP \leftarrow SP + 1$
POP	PSW	1	$(A) \leftarrow M(SP)$ $SP \leftarrow SP + 1$ $(F) \leftarrow M(SP)$ $SP \leftarrow SP + 1$

Description: The contents of the memory location pointed out by the stack pointer register are copied to the low-order register of the operand. The stack pointer is incremented by 1 and contents of that memory location are copied to the high-order register of the operand. The stack pointer register is incremented by 1.

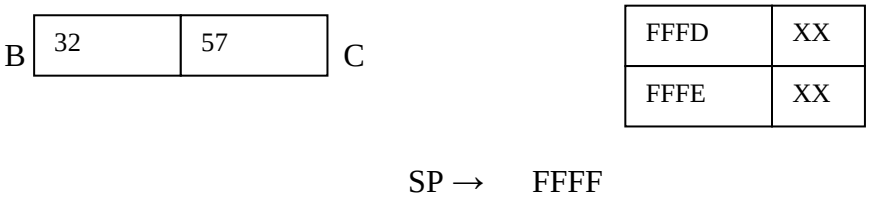
Example:

We assume that the SP registers points to location FFFF. Let register B contains 32 and register C contains 57. We want to save the contents of the BC register pair on the stack.

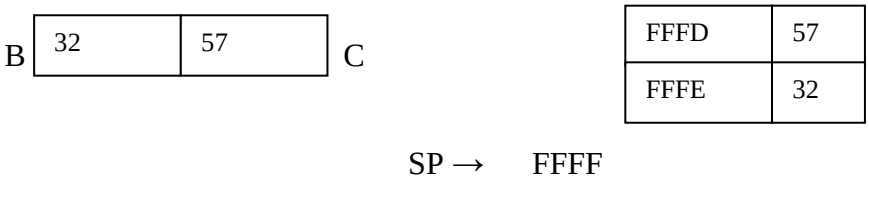
Instruction:

PUSH B

Contents before instruction



Contents after instruction



[Comments: Operand PSW (Program Status Word) represents the contents of the accumulator and the flag register; the accumulator is the high-order register and the flags are the low-order register].

LECTURE 8: COUNTER AND TIME DELAYS

4. COUNTER AND TIME DELAYS:

Steps:

- A counter is designed simply by loading appropriate number into one of the registers and using INR or DNR instructions
- Loop is established to update the count.
- Each count is checked to determine whether it has reached final number; if not, the loop is repeated.

Time Delay procedure is used to design a specific delay. A register is loaded with a number, depending on the time delay required and then the register is decremented until it reaches zero by setting up a loop with conditional jump instruction

4.1. Time Delay using one Register:

LABEL	OPCODE	OPERAND	COMMENTS
START	MVI	C, FF	Load Register C
LOOP	DCR	C	Decrement C
	JNZ	LOOP	Halt the program

Clock frequency of the system = 2 MHz
Clock period= 1/T= 0.5 μs
Time to execute MVI = 7 T states * 0.5= 3.5 μs

Time Delay in Loop TL= T*Loop T states * N10
= 0.5 * 14 * 255
= 1785 μs = 1.8 ms

N10 = Equivalent decimal number of hexadecimal count loaded in delay register

TLA=Time to execute loop instructions
= TL – (3T states* clock period)=1785-1.5=1783.5 μs

4.2.Time Delay using a Register Pair:

LABEL	OPCODE	OPERAND	COMMENTS
START	LXI	B, 2384	Load BC with 16-bit count
LOOP	DCX	B	Decrement BC by 1
	MOV	A, C	Place contents of C in A
	ORA	B	OR B with C to set Zero flag
	JNZ	LOOP	If result is not equal to 0, jump back to loop

Time Delay in Loop TL= T*Loop T states * N10
= 0.5 * 24 * 9092
= 109 ms

4.3.Time Delay using a Loop within a Loop:

LABEL	OPCODE	OPERAND	COMMENTS
START	MVI	B, 38	Load B
LOOP2	MVI	C, FF	Load C
LOOP1	DCR	C	Decrement C by 1
	JNZ	LOOP1	If result is not equal to 0, jump back to inner loop
	DCR	B	Decrement B by 1
	JNZ	LOOP2	If result is not equal to 0, jump back to outer loop

Time Delay in Loop TL= T*Loop T states * N10
= 0.5 * 24 * 9092
= 109 ms

Time Delay in Loop TL1= 1783.5 μs
Time Delay in Loop TL1 = (0.5 * 21 + TL1) * 56
= 100.46 ms

LECTURE 9: ASSEMBLY LANGUAGE PROGRAMMING

5. SAMPLE ASSEMBLY LANGUAGE PROGRAMS

1. **Write an assembly language program to transfer a data from one memory location to another memory location using direct addressing mode.**

LABEL	OPCODE	OPERAND	COMMENTS
START	LDA	9140	(A)← (9140)
	STA	9141	(9141)← (A)
END	RST1		Halt the program

2. **Write an assembly language program to transfer a data from one memory location to another memory location using indirect addressing mode.**

LABEL	OPCODE	OPERAND	COMMENTS
START	LXI	H 9140	Set (H)(L) as memory pointer
	MOV	A M	(A)← (9140)
	INR	L	Point to next memory location
	MOV	M A	(9141)← (A)
END	RST1		Halt the program

3. **Write an assembly language program to add two numbers stored in memory locations 9140 and 9141. Store the sum at 9142 and carry at 9143.**

LABEL	OPCODE	OPERAND	COMMENTS
START	LXI	H, 9140	Set (H)(L) as memory pointer
	MOV	A, M	Get first number in Accumulator
	INR	L	Increment memory pointer
	ADD	M	Add the numbers together
	INR	L	Increment memory pointer
	MOV	A, M	Store the sum at 9142
	MVI	A,00	Clear the Accumulator
	ADC	A	Transfer the Carry into Accumulator
	INR	L	Increment memory pointer
	MOV	A, M	Store the carry at 9143
END	RST1		Halt the program

4. **Write an assembly language program to add a series of numbers. The length is given in the location 913F and the series itself starts from 9140. Store the result at 9160. Assume no final carry is generated.**

LABEL	OPCODE	OPERAND	COMMENTS
START	LXI	H, 913F	Set (H)(L) as memory pointer
	MOV	C, M	Set C as a counter
	MVI	A, 00	sum = 0
LOOP	INX	H	Increment pointer
	MOV	A, M	Get the data in Accumulator
	ADD	M	sum = sum + ((H)(L))
	DCR	C	Decrement counter
	JNZ	LOOP	Continue if counter ≠ 0
	STA	9160	Store result
END	RST1		Halt the program

5. **Write an assembly language program to add two numbers using subroutine.**

LABEL	OPCODE	OPERAND	COMMENTS
START	LXI	SP, FFFF	Initialize stack pointer to FFFF
	LXI	H 9140	Set (H)(L) as memory pointer
	MOV	A, M	Get first data into A
	INR	L	Increment memory pointer
	MOV	B, M	Get second data into B
	CALL	SBR	Call the subroutine
	INR	L	Increment memory pointer
	MOV	M, A	Store the sum
END	RST1		Halt the program
SBR	ADD	B	Add the numbers
	RET		Return

LECTURE 10: INSTRUCTION CYCLE

6. INSTRUCTION FETCH

To communicate with a memory for example, to read an instruction from a memory location- the MPU places the 16 bit address on the address bus. The address on the bus is decoded by an external logic circuit, which will be explained later, and the memory location identified. The MPU sends a pulse called Memory Read as the control signal. The pulse activates the memory chip, and the contents of the memory location (8-bit data) are placed on the data bus and brought inside the MPU.

The primary function of memory is to store instructions and data and to provide that information to the MPU whenever the MPU requests it. The MPU requests the information by sending the address of a specific memory register on the address bus and enables the data flow by sending the control signal, as illustrated in the following example.

The instruction code 01001111 ($4F_H$) is stored in memory location 2005.

To fetch the instruction located in memory location 2005, the following steps are performed.

1. The PC places the 16-bit address 2005 of the memory location on the address bus (Figure).
2. The control unit sends the Memory Read control signals (MEMR, active low) to enable the output buffer of the memory chip.
3. The instruction ($4F_H$) stored in the memory location is placed on the data bus and transferred (copied) to the instruction decoder of the microprocessor.
4. The instruction is decoded and executed according to the binary pattern of the instruction.

Figure shows how the 8085 MPU fetches the instruction using the address, the data and the control buses.

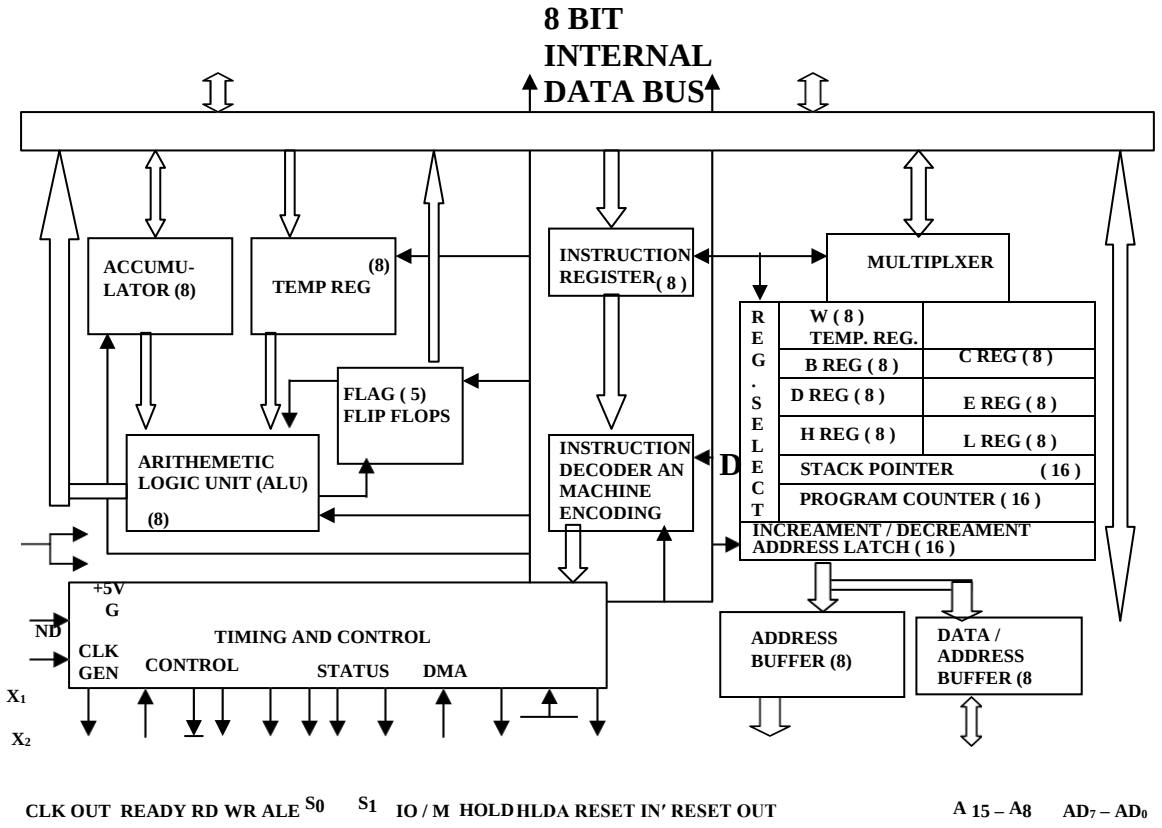


Figure 2.1: Communication between 8085 and memory

7. INSTRUCTION CYCLE:

1. Instruction cycle: this term is defined as the number of steps required by the CPU to complete the entire process i.e., fetching and execution of one instruction. The fetch and execute cycles are carried out in synchronization with the clock.

2. Machine cycle: It is the time required by the microprocessor to complete the operation of accessing the memory devices or I/O devices. In machine cycle various operations like opcode fetch, memory read, memory write, I/O read, I/O write are performed.

3. T-state: Each clock cycle is called as T-states.

The seven 8085 Machine Cycles are:

1. Opcode Fetch
2. Memory Read
3. Memory Write
4. I/O Read
5. I/O Write
6. Interrupt Acknowledge

7. Bus Idle

LECTURE 10: TIMING DIAGRAM

7.1.OPCODE FETCH MACHINE CYCLE

Figure shows the 8085 instruction fetch timing diagram. The instruction fetch cycle requires either four or six clock periods (T- states). The other machine cycles that follow OFMC will need three clock cycles.

The purpose of an OFMC is to read the contents of a memory location containing the opcode addressed by the program counter and to place it in the instruction register (IR).

In the beginning of state T_1, the 8085 puts a low on the IO/M line of the system bus indicating a memory operation. The 8085 sets S_1=1 and S_0=1 on the system bus, indicating the memory fetch operation. This status information remains available for the duration of the machine cycle. During T_1 state, the 16-bit address A_{15}-A_0 of the memory location containing the opcode is obtained from the program counter (PC) and placed on the address and address/data latches. The higher order 8-bits of the address appear on the address bus A8- A_{15} remains constants until the end of the state T_3. During T_4 state the data on the address bus is unspecified. The low order 8-bits of the address are placed on the address/data bus, AD_7-AD_0 at the beginning of T_1. This data however remains valid only until the beginning of state T_2 at which time the address/data bus is floated (tri-stated) because this is time multiplexed bus and used as the data bus during T_2 and T_3 states. Therefore address latch enable (ALE) signal issued by the p p during T_1 is used to latch this lower order address in some external latch 74LS373 on its falling edge. The 16- bit address selects a particular memory location.

During state T_2, at the beginning, the RD signal goes low indicating read operation and the opcode to be fetched is placed on the data bus, AD_7-AD_0 by the addressed memory location. The contents of (PC) is incremented be 1 during this state as during T_1 state the (PC) has sent the address to address bus. The accessed memory should be fast enough to output its data before RD goes high. Slower memories can gain more time by pulling the READY signal of 8085 LOW. This will introduce an integral number of T_{wait} states between T_2 and T_3 as long as READY is low. On the rising edge of the R D control signal in T3 state, the opcode obtained from the memory is transferred to the microprocessor instruction register.

During state T_4, the 8085 decodes the instruction and determines whether to enter state T5 or to enter T1 state of the next machine cycle. From the operation code, the microprocessor determines what other machine cycles, if any, must be executed to complete the instruction cycle. State T_5 and T_6 when entered, are used for internal p p operations necessitated by the instruction.

The micro RTL flow for 4-states OFMC is shown below.

OFMC: Status signals IO/M=0, S_1=1, S_0=1

T1: A15−A8 ← (PCH), AD7−AD0 ← (PCL), ALE =

T2: RD′ = 0, (PC) ← (PC) +1, AD7−AD0 ← M(AB)

T3: RD′ = 1, ↑ , (IR) ← BDB

T4: microprocessor decodes the opcode and decides whether T_5 and T_6 states are required or next machine cycle executed is T_1

During T_2 state, after the R D signal is made LOW, the external decoding circuit decodes the address put on the address bus during T_1 state. One of the memory locations is selected and it puts 8-bit information on the data bus during T_2 and T_3 states. Processor has no control on it. Processor has already issued the signals and now it is the job of the external decoding circuit to make use of the signals IO/M and RD and address lines to allow the external memory to put the data on the data bus. Therefore, this action is shown by shaded area. Whatever information is available on BDB at LOW to HIGH transition of RD, that will be read and processed. The timing waveform during 4-state OFMC is shown in fig.2.2.

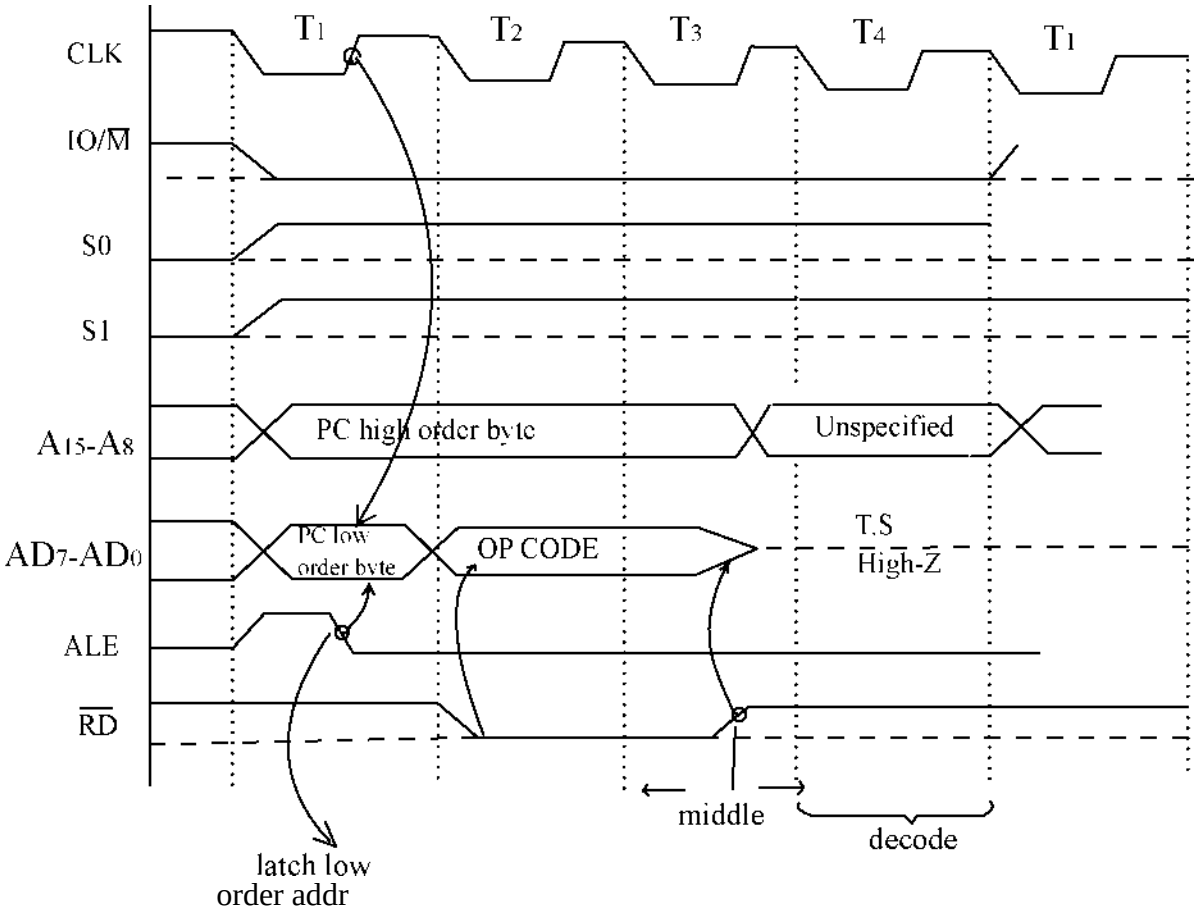


Fig.2.2 Timing Diagram During 4-state OpCode Fetch Machine Cycle

7.2.MEMORY READ MACHINE CYCLE:

It requires 3 states T_1 to T_3. The purpose of the memory READ operation is to read the contents of a memory location addressed by a register pair and place the data in one of internal registers of the microprocessor. The source of address issued during T_1 is not always the program counter but may be any one of the several other register pairs in the microprocessor depending on the particular instruction of which the machine cycle is a part.

The 8085 uses machine cycle MC−1 to fetch and decode the instruction. It then performs the memory read operation in MC−2. e.g., in LXI H, Addr.

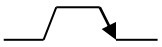
The IO / M′ signal is made LOW to indicate the external world that a memory reference is required. Then microprocessor made S_0=0 and S_1=1 indicating that memory READ operation is to be performed. During T_1, the microprocessor places the contents of higher byte of the memory address register, such as that contents of the (PCH) or (H) register on A_{15}-A_8 and the contents of the lower byte of the memory address register such as contents of the (PCL) or (L) register on AD_7-AD_0. The microprocessor sets ALE signal HIGH indicating the beginning of MC-2. As soon as ALE goes to LOW in the middle of T_1, the lower byte of the address is latched in an external latch. The same bus is now going to be used as data bus.

During T_2 state, the RD′ signal goes LOW indicating a READ operation. If the address sent out during T_1 state is from (PC), then (PC) is incremented by 1 otherwise not. The external logic gets the data from the memory location addressed by the memory address register such as (H,L) pair and places the data on to bi-directional data bus AD_7-AD_0.

During T_3 state, signal RD′ goes HIGH. This LOW to HIGH transition of signal transfers the data from the data bus to internal register such as the accumulator.

MRMC:

Status signals IO/M̄=0, S_1=1, S_0=0

T1: A15−A8 ← (PCH), AD7−AD0 ← (PCL), ALE = 

T2: RD′ = 0, (PC) ← (PC) +1, AD7−AD0 ← M(AB)

T3: RD′ = 1, ↑, (IR) ←AD7-AD0 or BDB

The timing diagram during memory ready machine cycle is shown in fig.2.3.

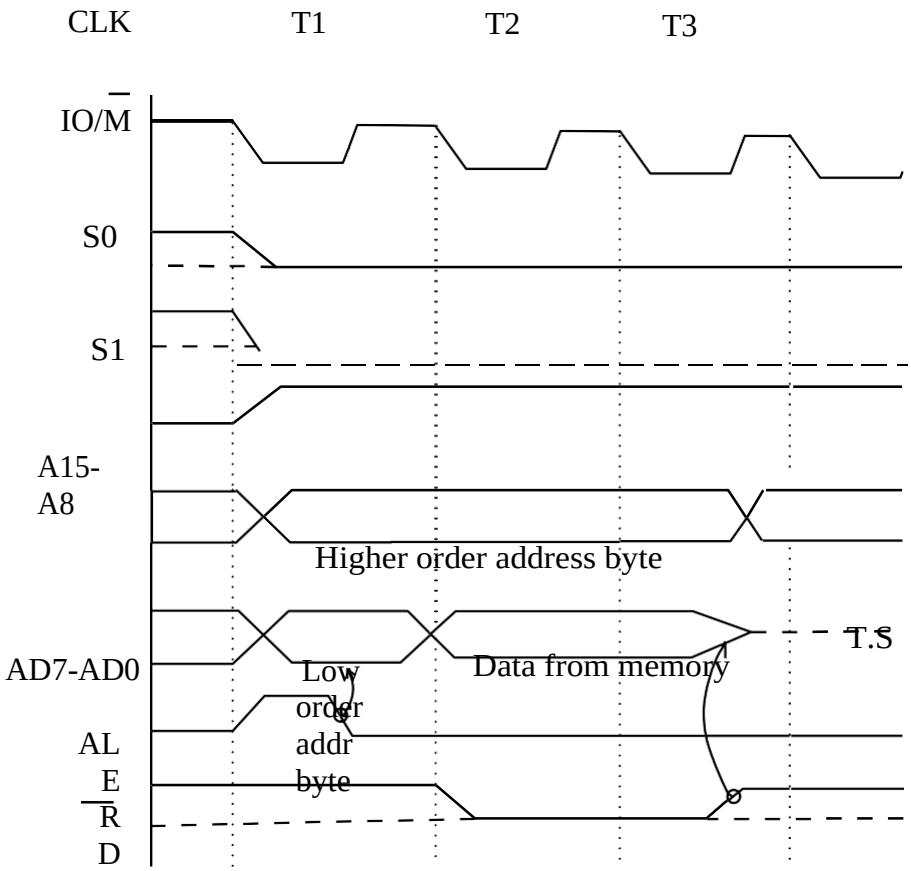


Fig. 2.3 Timing Diagram During Memory Read Machine Cycle

7.3.MEMORY WRITE MACHINE CYCLE:

It also requires only T to T_3 states. The purpose of memory write machine cycle is to store the contents of any of the 8085 register such as the accumulator into a memory location addressed by a register pair such as (H,L).

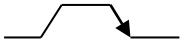
The 8085 microprocessor made IO/M = 0 in the beginning of T state to indicate memory reference operation. Then it puts S_0 = 1 and Si = 0 indicating a memory write operation.

During T_1 state 8085 places the memory address register (MAR) higher byte such as the contents of the (H) register on lines A_{15}-A_8 and also places the MAR lower byte such as the contents of the (L) register on lines AD_7-AD_0. The microprocessor sets ALE signal HIGH indicating the beginning of MWRMC. As soon as ALE goes to low, the lower byte of the address is latched in an external latch. During T_2 state, WR goes LOW indicating memory write operation. It also places the contents of the internal register, say accumulator, on data lines AD_7-AD_0.

During T_3 state, WR goes HIGH. This LOW to HIGH transition is used to transfer the data from the data lines to the memory location address by MAR such as (H,L) register pair.

MWRMC:

Status signals IO/M=0, S_1=0, S_0=1

T1: A15−A8 ← (H), AD7−AD0 ← (L), ALE = 

T_2: WR′ = 0, AD_7-AD_0 ← (microprocessor Internal Reg.)

T3: WR′ = 1, ↑ , M(AB) ← AD7-AD0 or BDB

Similar to MRMC, the processor simply puts the data on the data bus and makes required signals LOW or HIGH. It is the job of the external decoding circuit to make use of these signals to enable the external memory to accept the data from the data bus. Processor has no control over it. Therefore, this action during T_3 state is shown shaded. The timing diagram during MWRMC is shown in fig.4.14:

The timing diagram during MWRMC is shown in fig.2.4:

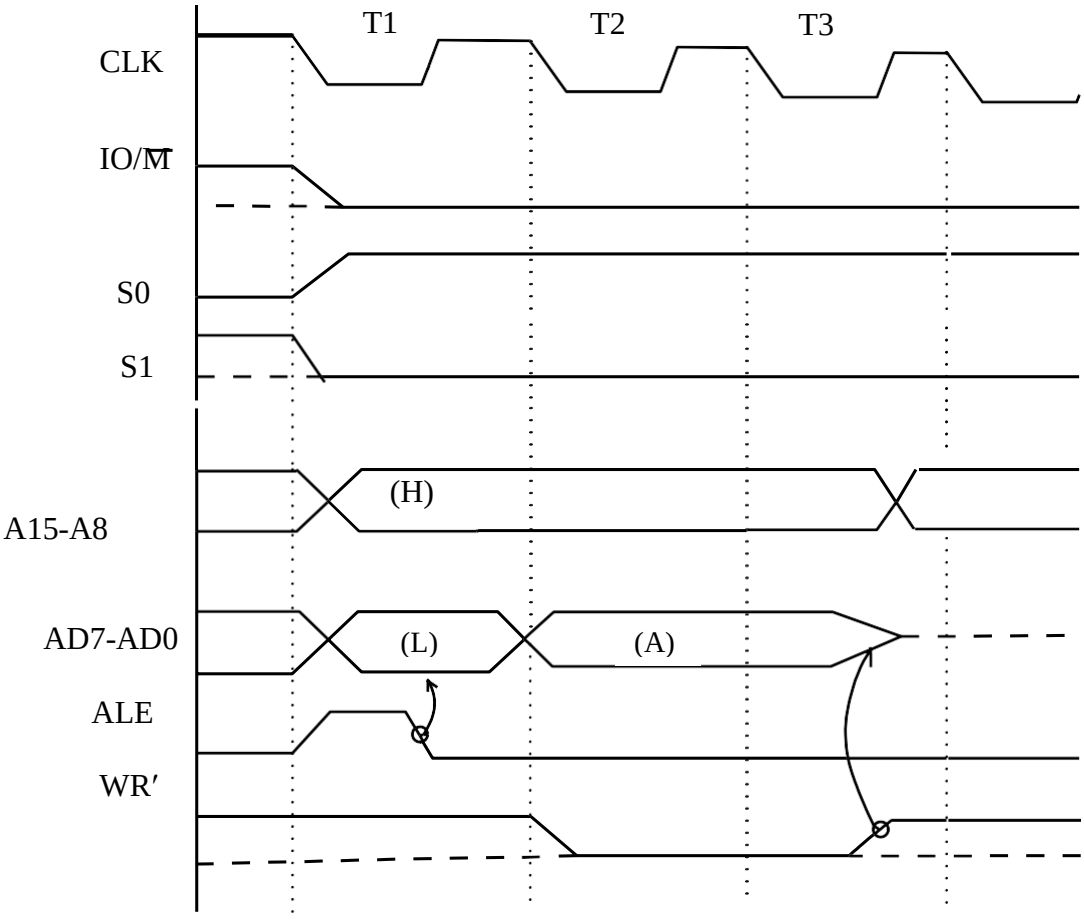


Fig. 2.4 Timing Diagram during Memory Write Machine Cycle

MODULE 3

LECTURE 1: ARCHITECTURE OF 8086

1. ARCHITECTURE OF 8086

Unlike microcontrollers, microprocessors do not have inbuilt memory. Mostly Princeton architecture is used for microprocessors where data and program memory are combined in a single memory interface. Since a microprocessor does not have any inbuilt peripheral, the circuit is purely digital and the clock speed can be anywhere from a few MHZ to a few hundred MHZ or even GHZ. This increased clock speed facilitates intensive computation that a microprocessor is supposed to do.
We will discuss the basic architecture of Intel 8086 before discussing more advanced microprocessor architectures.

Internal architecture of Intel 8086:

Intel 8086 is a 16 bit integer processor. It has 16-bit data bus and 20-bit address bus. The lower 16- bit address lines and 16-bit data lines are multiplexed (AD0-AD15). Since 20-bit address lines are available, 8086 can access up to 2 20 or 1 Giga byte of physical memory.

The basic architecture of 8086 is shown below.

The internal architecture of Intel 8086 is divided into two units, viz., Bus Interface Unit (BIU) and Execution Unit (EU).

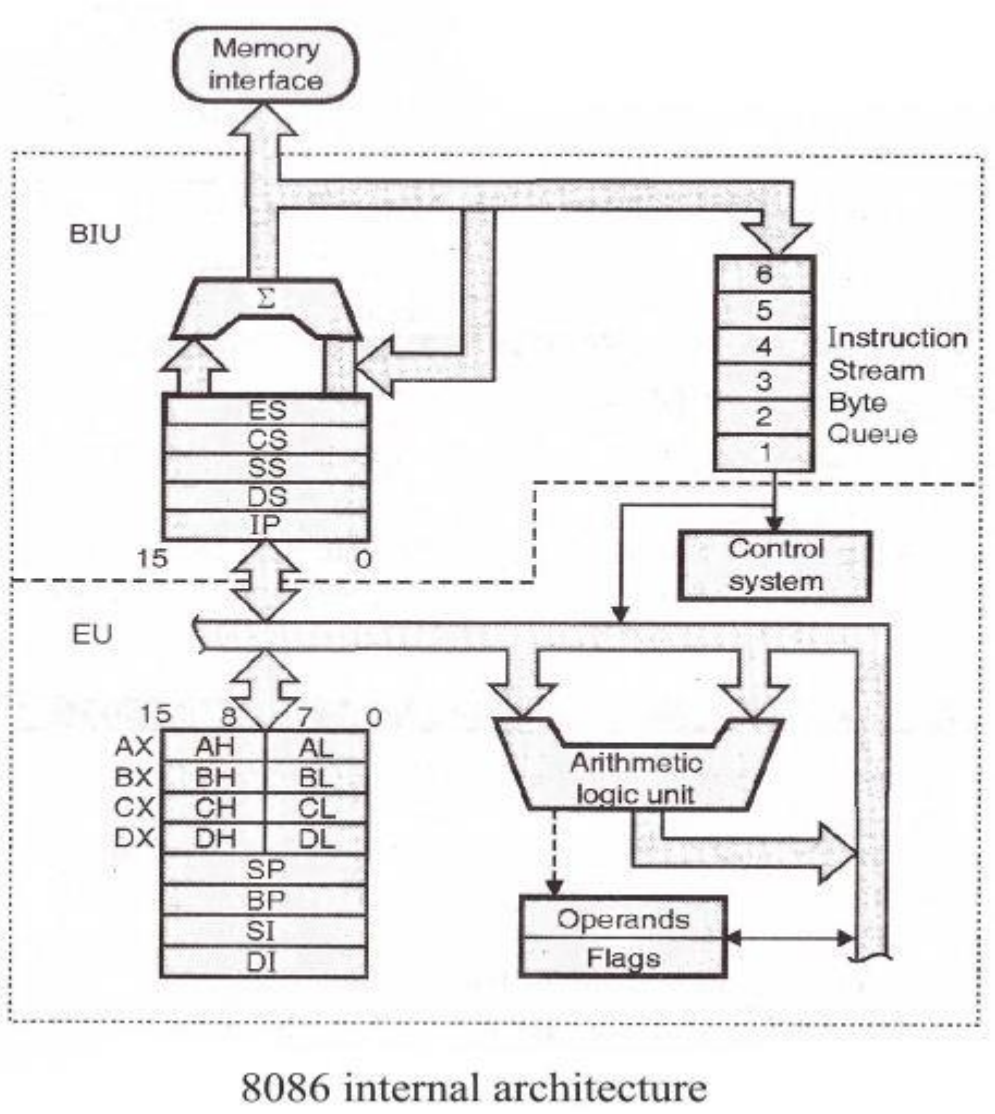
Bus Interface Unit (BIU)

The Bus Interface Unit (BIU) generates the 20-bit physical memory address and provides the interface with external memory (ROM/RAM). As mentioned earlier, 8086 has a single memory interface. To speed up the execution, 6-bytes of instruction are fetched in advance and kept in a 6- byte Instruction Queue while other instructions are being executed in the Execution Unit (EU). Hence after the execution of an instruction, the next instruction is directly fetched from the instruction queue without having to wait for the external memory to send the instruction. This is called pipe-lining and is helpful for speeding up the overall execution process.

8086's BIU produces the 20-bit physical memory address by combining a 16-bit segment address with a 16-bit offset address. There are four 16-bit segment registers, viz., the code segment (CS), the stack segment (SS), the extra segment (ES), and the data segment (DS). These segment registers hold the corresponding 16-bit segment addresses. A segment address is the upper 16-bits of the starting address of that segment. The lower 4-bits of the starting address of a segment is always zero. The offset address is held by another 16-bit register. The physical 20-bit

address is calculated by shifting the segment address 4-bit left and then adding that to the offset address.

For Example:



8086 internal architecture

Figure 3.1: 8086 Architecture

Code segment Register CS holds the segment address which is 4569 H Instruction pointer IP holds the offset address which is 10A0 H The physical 20-bit address is calculated as follows

Segment address: 45690 H
Offset address:+ 10A0 H
Physical address: 46730 H

Most of the registers contain data/instruction offsets within 64 KB memory segment. There are

four different 64 KB segments for instructions, stack, data and extra data. To specify where in 1 MB of processor memory these 4 segments are located the processor uses four segment registers:

Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.

Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions.

Extra segment (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions.

It is possible to change default segments used by general and index registers by prefixing instructions with a CS, SS, DS or ES prefix.

Execution Unit:

All general registers of the 8086 microprocessor can be used for arithmetic and logic operations. The general registers are:

Accumulator register consists of 2 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation. **Base** register consists of 2 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

Count register consists of 2 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low-order byte of the word, and CH contains the high-order byte. Count register can be used as a counter in string manipulation and shift/rotate instructions.

Data register consists of 2 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low-order byte of the word, and DH contains the high-order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-

order word of the initial or resulting number.

The following registers are both general and index registers:

Stack Pointer (SP) is a 16-bit register pointing to program stack.

Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.

Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Other registers:

Instruction Pointer (IP) is a 16-bit register.

Flags is a 16-bit register containing 9 1-bit flags:

- Overflow Flag (OF) - set if the result is too large positive number, or is too small negative number to fit into destination operand.
- Direction Flag (DF) - if set then string manipulation instructions will auto-decrement index registers. If cleared then the index registers will be auto-incremented.
- Interrupt-enable Flag (IF) - setting this bit enables maskable interrupts.
- Single-step Flag (TF) - if set then single-step interrupt will occur after the next instruction.
- Sign Flag (SF) - set if the most significant bit of the result is set.
    - Zero Flag (ZF) - set if the result is zero.
    - Auxiliary carry Flag (AF) - set if there was a carry from or borrow to bits 0-3 in the AL register.
    - Parity Flag (PF) - set if parity (the number of "1" bits) in the low-order byte of the result is even.
    - Carry Flag (CF) - set if there was a carry from or borrow to the most significant bit during last result calculation.

LECTURE 2: PIN DETAILS OF 8086

2. PIN DETAILS OF 8086

The following pin function descriptions are for 8086 systems in either minimum or maximum mode. The 'Local
Bus" in these descriptions is the direct multiplexed bus interface connection to the 8086 (without regard to additional bus buffers).

AD15±AD0 2±16, 39 -ADDRESS DATA BUS: These lines constitute the time multiplexed memory/IO address (T1), and data (T2, T3, TW, T4) bus. A0 is analogous to BHE for the lower byte of the data bus, pins D7±D0. It is LOW during T1 when a byte is to be transferred on the lower portion of the bus in memory or I/O operations. Eight-bit oriented devices tied to the lower half would normally use A0 to condition chip select functions. (See BHE.) These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge"

A19/S6,, A18/S5 , A17/S4, A16/S3 35±38 - ADDRESS/STATUS: During T1 these are the four most significant, address lines for memory operations. During I/O operations these, lines are LOW. During memory and I/O operations, status information is available on these lines during T2, T3, TW, T4. The status of the interrupt enable FLAG bit (S5) is updated at the beginning of each CLK cycle. A17/S4 and A16/S3 are encoded as shown. This information indicates which relocation register is presently being used for data accessing. These lines float to 3-state OFF during local bus "hold acknowledge."

BHE/S7 34 - BUS HIGH ENABLE/STATUS: During T1 the bus high enable signal (BHE) should be used to enable data onto the most significant half of the data bus, pins D15±D8. Eight-bit oriented devices tied to the upper half of the bus would normally use BHE to condition chip select functions. BHE is LOW during T1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. The S7 status information is available during T2, T3, and T4. The signal is active LOW, and floats to 3-state OFF in hold". It is LOW during T1 for the first interrupt acknowledge cycle.

RD 32 READ: Read strobe indicates that the processor is performing a memory or I/O read cycle, depending on the state of the S2 pin. This signal is used to read devices which reside on the 8086 local bus. RD is active LOW during T2, T3 and TW of any read cycle, and is guaranteed to remain HIGH in T2 until the 8086 local bus has floated. This signal floats to 3-state OFF in 'hold acknowledge".

READY 22 - READY: is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory/IO is synchronized by the 8284A Clock Generator to form READY. This signal is active HIGH. The 8086 READY input is not synchronized. Correct operation is not guaranteed if the setup and hold times are not

met.

INTR 18 - INTERRUPT REQUEST: is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.

TEST 23 TEST: input is examined by the "Wait" instruction. If the TEST input is LOW execution continues, otherwise the processor waits in an "Idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.

NMI 17 NON-MASKABLE INTERRUPT an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.

RESET 21 - RESET: causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the Instruction Set description, when RESET returns LOW. RESET is internally synchronized.

CLK 19 CLOCK: provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.

VCC 40 VCC: a5V power supply pin**. ,**GND 1, 20 GROUND

MN/MX 33 I MINIMUM/MAXIMUM: indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

Operating Modes of 8086

There are two modes of operation for Intel 8086, namely the minimum mode and the maximum mode. When only one 8086 CPU is to be used in a microcomputer system the 8086 is used in the minimum mode of operation. In this mode the CPU issues the control signals required by memory and I/O devices. In a multiprocessor system it operates in the maximum mode. In case of maximum mode of operation control signals are issued by Intel 8288 bus controller which is used with 8086 for this very purpose. The level of the pin MN/MX′ decides the operating mode of 8086. When MN/MX′ is high the CPU operates in the minimum mode. When it is low the CPU operates in the maximum mode. From pin 24 to 31 issue two different sets of signals. One set of signals is issued when the CPU operates in the maximum mode. Thus the pins from 24 to 31 have alternate functions.

Pin description for Minimum Mode:

For the minimum mode of operation the pin MN / MX′ is connected to 5 V D.C. supply, i.e., MN / MX′ = Vcc. The description of the pins from 24 to 31 for the minimum mode is as follows:

INTA′ (Output): Pin No. 24 Interrupt acknowledge. On receiving interrupt signal the processor issues an interrupt acknowledge signal. It is active LOW.

ALE (Output) Pin No. 25 Address latch enable. It goes High during T1. The microprocessor sends this signal to latch the address into the Intel 8282 / 8283 latch.

DEN′ (Output) Pin No. 26. Data enable. When Intel 8286 / 8287 octal bus transceiver is used this signal acts as an output enable signal. It is active low.

DT / R′ (Output) Pin No. 27. Data Transmit / Receive. When Intel 8286 / 8287 octal bus transceiver is used this signal controls the direction of data flow through the transceiver. When it is HIGH data are sent out. When it is LOW data are received.

M / IO′ (Output) Pin No. 28. Memory or I/O access. When it is HIGH the CPU wants to access memory. When it is LOW the CPU wants to access I/O device.

WR′ (Output): Pin No. 29 Write. When it is LOW the CPU performs memory or I/O write operation.
HLDA(Output): Pin No. 30. HOLD acknowledge. It is issued by the processor when it receives HOLD signal. It is active HIGH signal. When HOLD request is removed HLDA goes LOW.

HOLD (Input) Pin No. 31. Hold. When another device in microcomputer system system wants to use the address and data bus, it sends a HOLD request to CPU through this pin. It is an active HIGH signal.

Pin description for Maximum Mode:

For the maximum mode of operation the pin MN / MX′ is connected to Ground, i.e., MN / MX′ = Vcc. The description of the pins from 24 to 31 for the maximum mode is as follows:

QS1, QS0 (Output) : Pin No. 24, 25. Instruction Queue Status. Logics are given below:

QS1	QS2	Function
0	0	No Operation
0	1	1st byte of opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

Table 4.1: Functions of QS1 and QS2

S2, S1, S0 (Output) 26±28 STATUS: active during T4, T1, and T2 and is returned to the passive state (1, 1, 1) during T3 or during TW when READY is HIGH. This status is used by the 8288 Bus Controller to generate all memory and I/O access control signals. Any change by S2, S1, or S0 during T4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T3 or TW is used to indicate the end of a bus cycle

S2	S1	S0	Function
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1	0	0	Code access
1	0	1	Write memory
1	1	0	Passive state
1	1	1	

Table 4.1: Functions of S2, S1 and S0

LOCK (O)

It indicates to another system bus master, not to gain control of the system bus while LOCK is active Low. The LOCK signal is activated by the "LOCK" prefix instruction and remains active until the completion of the instruction. This signal is active Low and floats to tri-state OFF during 'hold acknowledge'.

RQ/GT0 and RQ/GT1 (I/O): Request/Grant

These pins are used by other processors in a multi processor organization. Local bus masters of other processors force the processor to release the local bus at the end of the processors current bus cycle. Each pin is bi-directional and has an internal pull up resistors. Hence they may be left un-connected.

LECTURE 3: ADDRESSING MODES OF 8086

3. ADDRESSING MODES

Addressing mode indicates a way of locating data or operands. Depending upon the data types used in the instruction and the memory addressing modes, any instruction may belong to one or more addressing modes, or some instruction may not belong to any of the addressing modes. Thus the addressing modes describe the types of operands and the way they are accessed for executing an instruction. Here, we will present the addressing modes of the instructions depending upon their types. According to the flow of instruction execution, the instructions may be categorized as

i. Sequential control flow instructions and
ii. Control transfer instructions

Sequential control flow instructions are the instructions, which after execution, transfer control to the next instruction appearing immediately after it (in the sequence) in the program. For example, the arithmetic, logical, data transfer and processor control instructions are sequential control flow instructions. The control transfer instructions, on the other hand, transfer control to some predefined address somehow specified in the instruction after their execution. For example, INT, CALL, RET and JUMP instructions fall under this category.
The addressing modes for sequential control transfer instructions are explained as follows:

1. Immediate: In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes.

   Example: MOV AX, 0005H

   In the above example, 0005H is the immediate data. The immediate data may be 8-bit or 16-bit in size.

2. Direct: In the direct addressing mode, a 16-bit memory address (offset) is directly specified in the instruction as a part of it.

   Example: MOV AX, [5000H]

   Here, data resides in a memory location in the data segment, whose effective address may be computed using 5000H as the offset address and content of DS as segment address. The effective address, here, is 10H*DS+5000H.

3. Register: In the direct addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP, may be used in this mode.

   Example: MOV BX, AX

4. Register Indirect: Sometimes, the address of the memory location, which contains data or operand, is determined in an indirect way, using the offset registers. This mode of addressing is known as register indirect mode. In this addressing mode, the offset address of data is in either BX or SI or DI registers. The default segment is either DS or ES. The data is supposed to be available at the address pointed to by the content of any of the above registers in the default data segment.

   Example: MOV AX, [BX]

   Here, data is present in a memory location in DS whose offset address is in BX. The effective address of the data is given as 10H*DS+ [BX].

5. Indexed: In this addressing mode, offset of the operand is stored in one of the index registers. DS and ES are the default segments for index registers SI and DI respectively. This mode is a special case of the above discussed register indirect addressing mode.

   Example: MOV AX,[SI]

   Here, data is available at an offset address stored in SI in DS. The effective address, in this case, is computed as 10H*DS+ [SI].

6. Register Relative: In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI and DI in the default (either DS or ES) segment. The example given before explains this mode.

   Example: MOV AX, 50H [BX]

   Here, effective address is given as 10H*DS+50H+ [BX].

7. Based Indexed: The effective address of data is formed, in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

   Example: MOV AX, [BX] [SI]

   Here, BX is the base register and SI is the index register. The effective address is computed as 10H*DS+ [BX] + [SI].

8. Relative Based Indexed: The effective address is formed by adding an 8-bit or 16-bit displacement with the sum of contents of any one of the bases registers (BX or BP) and any one of the index registers, in a default segment.
   Example: MOV AX, 50H [BX] [SI]

   Here, 50H is an immediate displacement, BX is a base register and SI is an index register. The effective address of data is computed as 160H*DS+ [BX]+ [SI] + 50H.

LECTURE 4: 8086 INSTRUCTION SET

4. 8086 INSTRUCTION SET

The 8086 microprocessor supports 8 types of instructions −

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- String Instructions
- Program Execution Transfer Instructions (Branch & Loop Instructions)
- Processor Control Instructions
- Iteration Control Instructions
- Interrupt Instructions

Let us now discuss these instruction sets in detail.

Data Transfer Instructions

These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group −

Instruction to transfer a word

- **MOV** − Used to copy the byte or word from the provided source to the provided destination.
- **PPUSH** − Used to put a word at the top of the stack.
- **POP** − Used to get a word from the top of the stack to the provided location.
- **PUSHA** − Used to put all the registers into the stack.
- **POPA** − Used to get words from the stack to all registers.
- **XCHG** − Used to exchange the data from two locations.
- **XLAT** − Used to translate a byte in AL using a table in the memory.

Instructions for input and output port transfer

- **IN** − Used to read a byte or word from the provided port to the accumulator.
- **OUT** − Used to send out a byte or word from the accumulator to the provided port.

Instructions to transfer the address

- **LEA** − Used to load the address of operand into the provided register.
- **LDS** − Used to load DS register and other provided register from the memory
- **LES** − Used to load ES register and other provided register from the memory.

Instructions to transfer flag registers

- **LAHF** − Used to load AH with the low byte of the flag register.
- **SAHF** − Used to store AH register to low byte of the flag register.
- **PUSHF** − Used to copy the flag register at the top of the stack.
- **POPF** − Used to copy a word at the top of the stack to the flag register.

Arithmetic Instructions

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.
Following is the list of instructions under this group −
Instructions to perform addition

- **ADD** − Used to add the provided byte to byte/word to word.
- **ADC** − Used to add with carry.
- **INC** − Used to increment the provided byte/word by 1.
- **AAA** − Used to adjust ASCII after addition.
- **DAA** − Used to adjust the decimal after the addition/subtraction operation.

Instructions to perform subtraction

- **SUB** − Used to subtract the byte from byte/word from word.
- **SBB** − Used to perform subtraction with borrow.
- **DEC** − Used to decrement the provided byte/word by 1.
- **NPG** − Used to negate each bit of the provided byte/word and add 1/2's complement.
- **CMP** − Used to compare 2 provided byte/word.
- **AAS** − Used to adjust ASCII codes after subtraction.
- **DAS** − Used to adjust decimal after subtraction.

Instruction to perform multiplication

- **MUL** − Used to multiply unsigned byte by byte/word by word.
- **IMUL** − Used to multiply signed byte by byte/word by word.
- **AAM** − Used to adjust ASCII codes after multiplication.

Instructions to perform division

- **DIV** − Used to divide the unsigned word by byte or unsigned double word by word.
- **IDIV** − Used to divide the signed word by byte or signed double word by word.
- **AAD** − Used to adjust ASCII codes after division.
- **CBW** − Used to fill the upper byte of the word with the copies of sign bit of the lower byte.
- **CWD** − Used to fill the upper word of the double word with the sign bit of the lower word.

Bit Manipulation Instructions

These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc.
Following is the list of instructions under this group −
Instructions to perform logical operation

- **NOT** − Used to invert each bit of a byte or word.
- **AND** − Used for adding each bit in a byte/word with the corresponding bit in another byte/word.
- **OR** − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.
- **XOR** − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.

- **TEST** − Used to add operands to update flags, without affecting operands.

Instructions to perform shift operations

- **SHL/SAL** − Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
- **SHR** − Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.
- **SAR** − Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

Instructions to perform rotate operations

- **ROL** − Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].
- **ROR** − Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
- **RCR** − Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.
- **RCL** − Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

String Instructions

String is a group of bytes/words and their memory is always allocated in a sequential order. Following is the list of instructions under this group −

- **REP** − Used to repeat the given instruction till CX ≠ 0.
- **REPE/REPZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
- **REPNE/REPNZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
- **MOVS/MOVSB/MOVSW** − Used to move the byte/word from one string to another.
- **COMS/COMPSB/COMPSW** − Used to compare two string bytes/words.
- **INS/INSB/INSW** − Used as an input string/byte/word from the I/O port to the provided memory location.
- **OUTS/OUTSB/OUTSW** − Used as an output string/byte/word from the provided memory location to the I/O port.
- **SCAS/SCASB/SCASW** − Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
- **LODS/LODSB/LODSW** − Used to store the string byte into AL or string word into AX.

Program Execution Transfer Instructions (Branch and Loop Instructions)

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions −

Instructions to transfer the instruction during an execution without any condition −

- **CALL** − Used to call a procedure and save their return address to the stack.
- **RET** − Used to return from the procedure to the main program.
- **JMP** − Used to jump to the provided address to proceed to the next instruction.

Instructions to transfer the instruction during an execution with some conditions −

- **JA/JNBE** − Used to jump if above/not below/equal instruction satisfies.
- **JAE/JNB** − Used to jump if above/not below instruction satisfies.
- **JBE/JNA** − Used to jump if below/equal/ not above instruction satisfies.
- **JC** − Used to jump if carry flag CF = 1
- **JE/JZ** − Used to jump if equal/zero flag ZF = 1

- **JG/JNLE** − Used to jump if greater/not less than/equal instruction satisfies.
- **JGE/JNL** − Used to jump if greater than/equal/not less than instruction satisfies.
- **JL/JNGE** − Used to jump if less than/not greater than/equal instruction satisfies.
- **JLE/JNG** − Used to jump if less than/equal/if not greater than instruction satisfies.
- **JNC** − Used to jump if no carry flag (CF = 0)
- **JNE/JNZ** − Used to jump if not equal/zero flag ZF = 0
- **JNO** − Used to jump if no overflow flag OF = 0
- **JNP/JPO** − Used to jump if not parity/parity odd PF = 0
- **JNS** − Used to jump if not sign SF = 0
- **JO** − Used to jump if overflow flag OF = 1
- **JP/JPE** − Used to jump if parity/parity even PF = 1
- **JS** − Used to jump if sign flag SF = 1

Processor Control Instructions

These instructions are used to control the processor action by setting/resetting the flag values. Following are the instructions under this group −

- **STC** − Used to set carry flag CF to 1
- **CLC** − Used to clear/reset carry flag CF to 0
- **CMC** − Used to put complement at the state of carry flag CF.
- **STD** − Used to set the direction flag DF to 1
- **CLD** − Used to clear/reset the direction flag DF to 0
- **STI** − Used to set the interrupt enable flag to 1, i.e., enable INTR input.
- **CLI** − Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

Iteration Control Instructions

These instructions are used to execute the given instructions for number of times. Following is the list of instructions under this group −

- **LOOP** − Used to loop a group of instructions until the condition satisfies, i.e., CX = 0
- **LOOPE/LOOPZ** − Used to loop a group of instructions till it satisfies ZF = 1 & CX = 0
- **LOOPNE/LOOPNZ** − Used to loop a group of instructions till it satisfies ZF = 0 & CX = 0
- **JCXZ** − Used to jump to the provided address if CX = 0

Interrupt Instructions

These instructions are used to call the interrupt during program execution.

- **INT** − Used to interrupt the program during execution and calling service specified.
- **INTO** − Used to interrupt the program during execution if OF = 1
- **IRET** − Used to return from interrupt service to the main program

Sample Assembly Language Program:

1. **Write an assembly language program in 8086 to add two 16-bit numbers.**

PROGRAM:

LABEL	**OPCODE**	**OPERAND**	**COMMENTS**
START	MOV	CX, 9273	Get 16-bit data in AX
	MOV	DX, 2464	Get another 16-bit data in DX
	ADD	CX, DX	(CX) ← (CX) + (DX)
END	INT3		Halt the program

MODULE 4

LECTURE 1: ARCHITECTURE OF 8051

1. 8051 MICROCONTROLLER

The 8051 Microcontroller was designed in 1980's by Intel. Its foundation was on Harvard Architecture and was developed principally for bringing into play in Embedded Systems. At first it was created by means of NMOS technology but as NMOS technology needs more power to function therefore Intel re-intended Microcontroller 8051 employing CMOS technology and a new edition came into existence with a letter 'C' in the title name, for illustration: 80C51. These most modern Microcontrollers need fewer amount of power to function in comparison to their forerunners.

There are two buses in 8051 Microcontroller one for program and other for data. As a result, it has two storage rooms for both program and data of 64K by 8 size. The microcontroller comprise of 8 bit accumulator & 8 bit processing unit. It also consists of 8 bit B register as majorly functioning blocks and 8051 microcontroller programming is done with embedded C Language using Keil software. It also has a number of other 8 bit and 16 bit registers.

For internal functioning & processing Microcontroller 8051 comes with integrated built-in RAM. This is prime memory and is employed for storing temporary data. It is unpredictable memory i.e. its data can get be lost when the power supply to the Microcontroller switched OFF.

1.1.8051 MICROCONTROLLER ARCHITECTURE

Microcontroller 8051 block diagram is shown below. Let's have a closer look on features of 8051 microcontroller design:

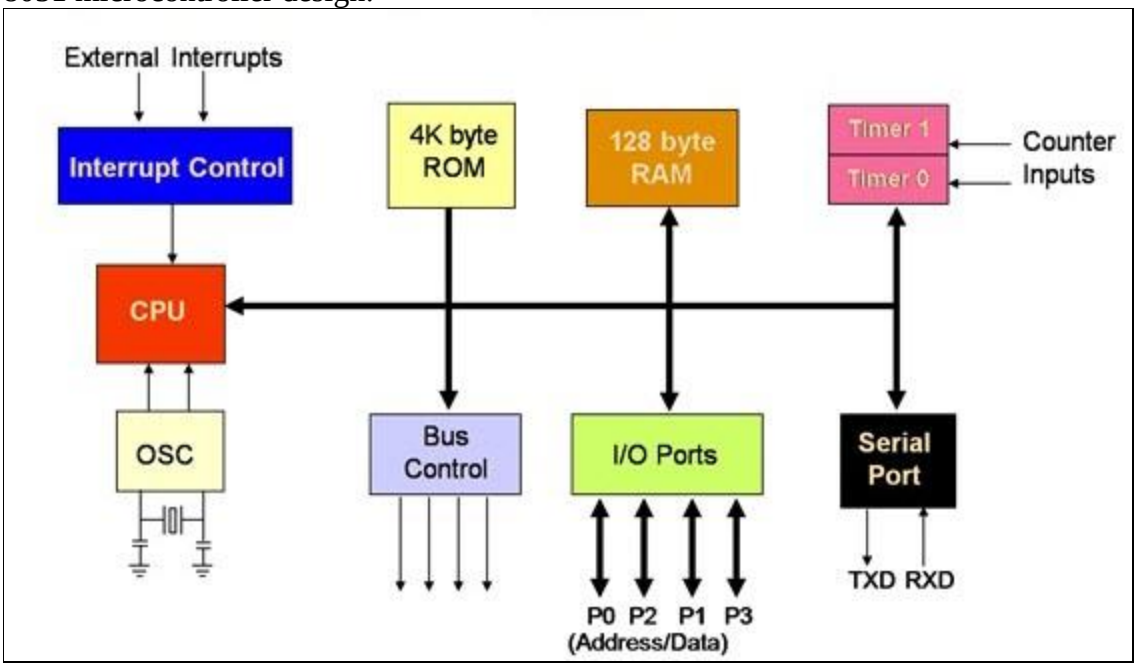


Figure 4.1: Block Diagram of 8051 Microcontroller

CPU (Central Processor Unit):
As we may be familiar that Central Processor Unit or CPU is the mind of any processing machine. It scrutinizes and manages all processes that are carried out in the Microcontroller. User has no power over the functioning of CPU. It interprets program printed in storage space (ROM) and carries out all of them and do the projected duty. CPU manages different types of registers in 8051 microcontroller.

Interrupts:
As the heading put forward, Interrupt is a sub-routine call that reads the Microcontroller's key function or job and helps it to perform some other program which is extra important at that point of time. The characteristic of 8051 Interrupt is extremely constructive as it aids in emergency cases. Interrupts provides us a method to postpone or delay the current process, carry out a sub-routine task and then all over again restart standard program implementation.
The Micro-controller 8051 can be assembled in such a manner that it momentarily stops or break the core program at the happening of interrupt. When sub-routine task is finished then the implementation of core program initiates automatically as usual. There are 5 interrupt supplies in 8051 Microcontroller, two out of five are peripheral interrupts, two are timer interrupts and one is serial port interrupt.

Memory:
Micro-controller needs a program which is a set of commands. This program enlightens Microcontroller to perform precise tasks. These programs need a storage space on which they can be accumulated and interpret by Microcontroller to act upon any specific process. The memory which is brought into play to accumulate the program of Microcontroller is recognized as Program memory or code memory. In common language it's also known as Read Only Memory or ROM.
Microcontroller also needs a memory to amass data or operands for the short term. The storage space which is employed to momentarily data storage for functioning is acknowledged as Data Memory and we employ Random Access Memory or RAM for this principle reason. Microcontroller 8051 contains code memory or program memory 4K so that is has 4KB Rom and it also comprise of data memory (RAM) of 128 bytes.

Bus:
Fundamentally Bus is a group of wires which functions as a communication canal or mean for the transfer Data. These buses comprise of 8, 16 or more cables. As a result, a bus can bear 8 bits, 16 bits all together. There are two types of buses:

1. **Address Bus:** Microcontroller 8051 consists of 16 bit address bus. It is brought into play to address memory positions. It is also utilized to transmit the address from Central Processing Unit to Memory.
2. **Data Bus:** Microcontroller 8051 comprise of 8 bits data bus. It is employed to cart data.

Oscillator:
As we all make out that Microcontroller is a digital circuit piece of equipment, thus it needs timer for its function. For this function, Microcontroller 8051 consists of an on-chip oscillator which toils as a time source for CPU (Central Processing Unit). As the productivity thumps of

oscillator are steady as a result, it facilitates harmonized employment of all pieces of 8051 Microcontroller. Input/output Port: As we are acquainted with that Microcontroller is employed in embedded systems to manage the functions of devices.
Thus to gather it to other machinery, gadgets or peripherals we need I/O (input/output) interfacing ports in Micro-controller. For this function Micro-controller 8051 consists of 4 input/output ports to unite it to other peripherals. Timers / Counters: Micro-controller 8051 is incorporated with two 16 bit counters & timers. The counters are separated into 8 bit registers. The timers are utilized for measuring the intervals, to find out pulse width etc.

1.2.8051 MICROCONTROLLER MEMORY ORGANIZATION

The 8051 Microcontroller Memory is separated in Program Memory (ROM) and Data Memory (RAM). The Program Memory of the 8051 Microcontroller is used for storing the program to be executed i.e. instructions. The Data Memory on the other hand, is used for storing temporary variable data and intermediate results.

Program Memory (ROM) of 8051 Microcontroller

In 8051 Microcontroller, the code or instructions to be executed are stored in the Program Memory, which is also called as the ROM of the Microcontroller. The original 8051 Microcontroller by Intel has 4KB of internal ROM.
Some variants of 8051 like the 8031 and 8032 series doesn't have any internal ROM (Program Memory) and must be interfaced with external Program Memory with instructions loaded in it.
Almost all modern 8051 Microcontrollers, like 8052 Series, have 8KB of Internal Program Memory (ROM) in the form of Flash Memory (ROM) and provide the option of reprogramming the memory.

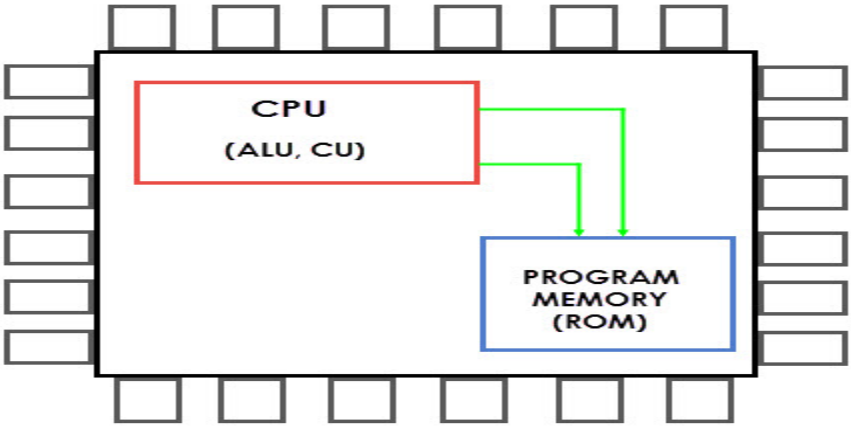


Figure 4.2a

In case of 4KB of Internal ROM, the address space is 0000H to 0FFFH. If the address space i.e. the program addresses exceed this value, then the CPU will automatically fetch the code from the external Program Memory.

For this, the External Access Pin (EA Pin) must be pulled HIGH i.e. when the EA Pin is high, the CPU first fetches instructions from the Internal Program Memory in the address range of

0000H to 0FFFFH and if the memory addresses exceed the limit, then the instructions are fetched from the external ROM in the address range of 1000H to FFFFH.

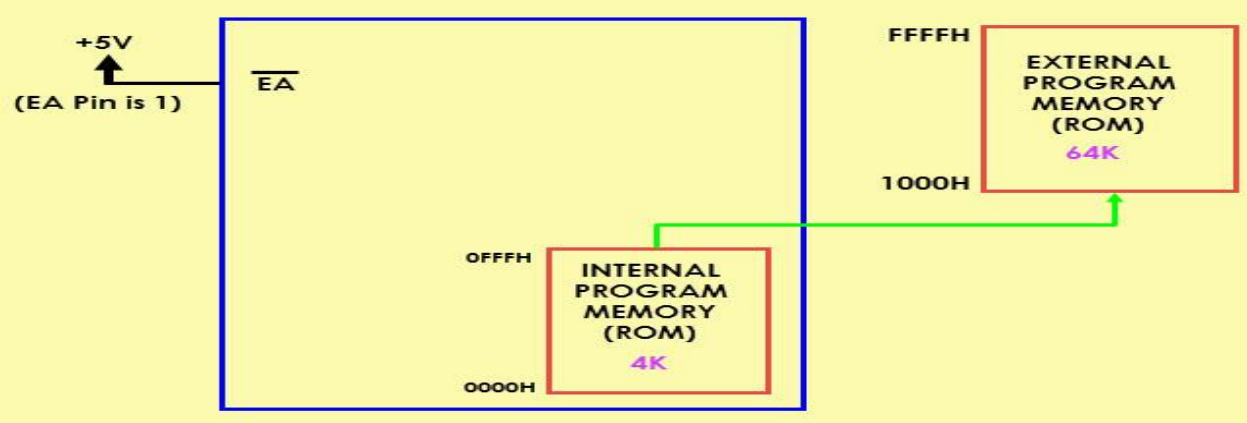


Figure 4.2b

There is another way to fetch the instructions: ignore the Internal ROM and fetch all the instructions only from the External Program Memory (External ROM). For this scenario, the EA Pin must be connected to GND. In this case, the memory addresses of the external ROM will be from 0000H to FFFFH.

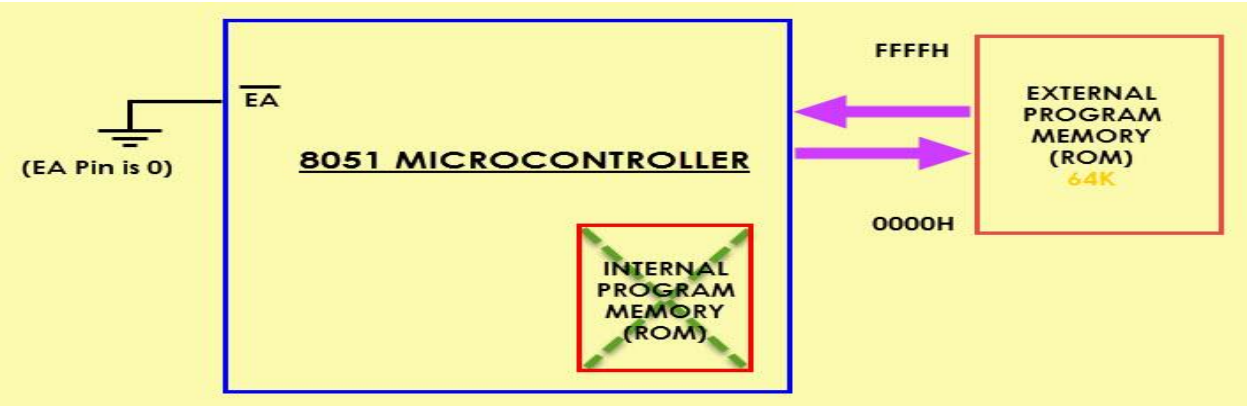


Figure 4.2c

Data Memory (RAM) of 8051 Microcontroller

The Data Memory or RAM of the 8051 Microcontroller stores temporary data and intermediate results that are generated and used during the normal operation of the microcontroller. Original Intel's 8051 Microcontroller had 128B of internal RAM.
But almost all modern variants of 8051 Microcontroller have 256B of RAM. In this 256B, the first 128B i.e. memory addresses from 00H to 7FH is divided in to Working Registers (organized as Register Banks), Bit – Addressable Area and General Purpose RAM (also known as Scratchpad area).
In the first 128B of RAM (from 00H to 7FH), the first 32B i.e. memory from addresses 00H to

1FH consists of 32 Working Registers that are organized as four banks with 8 Registers in each Bank.

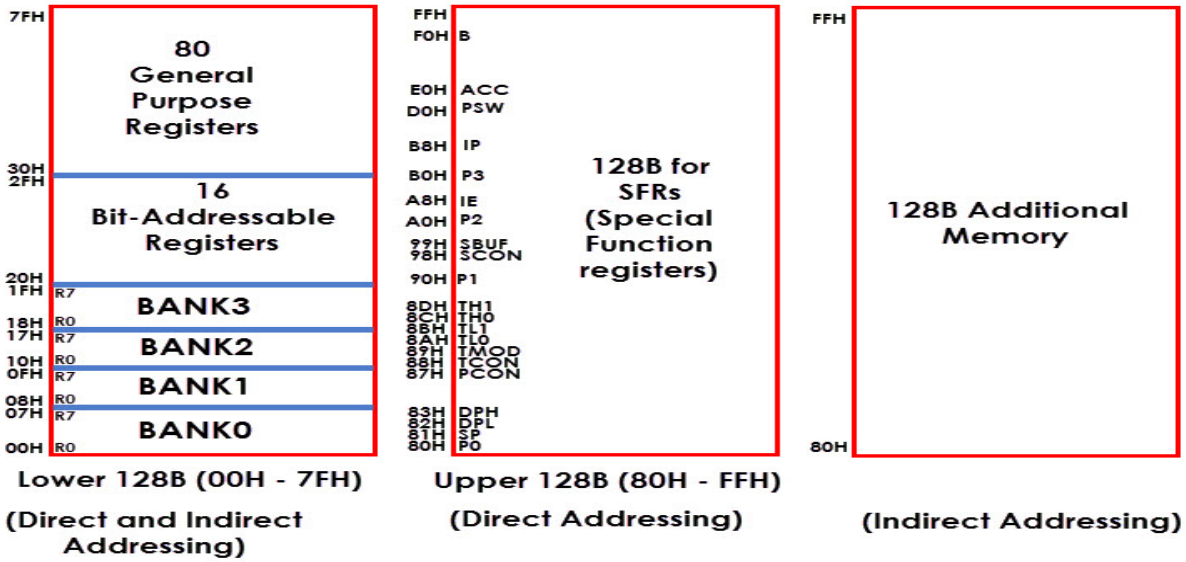


Figure 4.3: Data Memory of 8051

The 4 banks are named as Bank0, Bank1, Bank2 and Bank3. Each Bank consists of 8 registers named as R0 – R7. Each Register can be addressed in two ways: either by name or by address.
To address the register by name, first the corresponding Bank must be selected. In order to select the bank, we have to use the RS0 and RS1 bits of the Program Status Word (PSW) Register (RS0 and RS1 are 3rd and 4th bits in the PSW Register).
When addressing the Register using its address i.e. 12H for example, the corresponding Bank may or may not be selected. (12H corresponds to R2 in Bank2).
The next 16B of the RAM i.e. from 20H to 2FH are Bit – Addressable memory locations. There are totally 128 bits that can be addressed individually using 00H to 7FH or the entire byte can be addressed as 20H to 2FH.
For example 32H is the bit 2 of the internal RAM location 26H.
The final 80B of the internal RAM i.e. addresses from 30H to 7FH, is the general purpose RAM area which are byte addressable.
These lower 128B of RAM can be addressed directly or indirectly.
The upper 128B of the RAM i.e. memory addresses from 80H to FFH is allocated for Special Function Registers (SFRs). SFRs control specific functions of the 8051 Microcontroller. Some of the SFRs are I/O Port Registers (P0, P1, P2 and P3), PSW (Program Status Word), A (Accumulator), IE (Interrupt Enable), PCON (Power Control), etc.

Name of the Register	*Function*	*Internal RAM Address (HEX)*
ACC	Accumulator	E0H
B	B Register (for Arithmetic)	F0H
DPH	Addressing External Memory	83H
DPL	Addressing External Memory	82H
IE	Interrupt Enable Control	A8H
IP	Interrupt Priority	B8H
P0	PORT 0 Latch	80H
P1	PORT 1 Latch	90H
P2	PORT 2 Latch	A0H
P3	PORT 3 Latch	B0H
PCON	Power Control	87H
PSW	Program Status Word	D0H
SCON	Serial Port Control	98H
SBUF	Serial Port Data Buffer	99H
SP	Stack Pointer	81H
TMOD	Timer / Counter Mode Control	89H
TCON	Timer / Counter Control	88H
TL0	Timer 0 LOW Byte	8AH
TH0	Timer 0 HIGH Byte	8CH
TL1	Timer 1 LOW Byte	8BH
TH1	Timer 1 HIGH Byte	8DH

Table 4.1

SRFs Memory addresses are only direct addressable. Even though some of the addresses between 80H and FFH are not assigned to any SFR, they cannot be used as additional RAM area. In some microcontrollers, there is an additional 128B of RAM, which share the memory address with SFRs i.e. 80H to FFH. But, this additional RAM block is only accessed by indirect addressing.

LECTURE 2: PIN DESCRIPTION OF 8051

2. 8051 MICROCONTROLLER PIN DIAGRAM

As mentioned in the previous tutorial, 8051 Microcontroller is available in a variety of packages like 40 – pin DIP or 44 – lead PLCC and TQFP. The pin orientation of an 8051 Microcontroller may change with the package but the Pin Configuration is same.
The following image shows the 8051 Microcontroller Pin Diagram with respect to a 40 – pin Dual In-line Package (DIP).

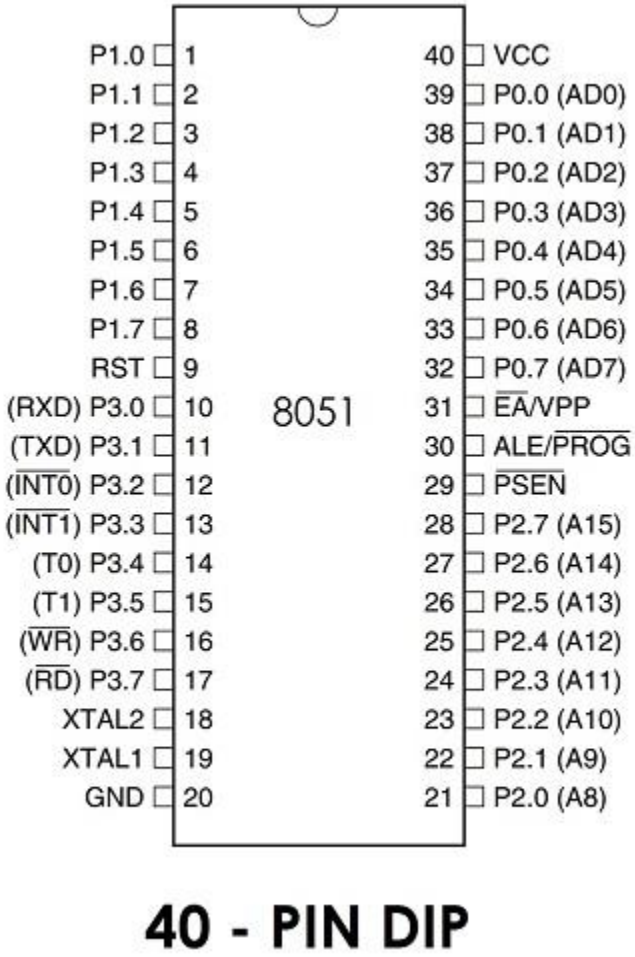


Figure 4.4: Pin Diagram of 8051 Microcontroller

Since it is a 40 – pin DIP IC, each side contains 20 Pins. We have also seen that there other packages of 8051 like the 44 – Lead PLCC and the 44 – Lead TQFP. The following image shows the 8051 Microcontroller Pin Diagram for these packages specifically.

The Pin Description or Pin Configuration of the 8051 Microcontroller will describe the functions of each pins of the 8051 Microcontroller. Let us now see the pin description.

Pins 1 – 8 (PORT 1): Pins 1 to 8 are the PORT 1 Pins of 8051. PORT 1 Pins consists of 8 – bit bidirectional Input / Output Port with internal pull – up resistors. In older 8051 Microcontrollers, PORT 1 doesn't serve any additional purpose but just 8 – bit I/O PORT.

In some of the newer 8051 Microcontrollers, few PORT 1 Pins have dual functions. P1.0 and P1.1 act as Timer 2 and Timer 2 Trigger Input respectively.

P1.5, P1.6 and P1.7 act as In-System Programming Pins i.e. MOSI, MISO and SCK respectively.

Pin 9 (RST): Pin 9 is the Reset Input Pin. It is an active HIGH Pin i.e. if the RST Pin is HIGH for a minimum of two machine cycles, the microcontroller will be reset. During this time, the oscillator must be running.

Pins 10 – 17 (PORT 3): Pins 10 to 17 form the PORT 3 pins of the 8051 Microcontroller. PORT 3 also acts as a bidirectional Input / Output PORT with internal pull-ups. Additionally, all the PORT 3 Pins have special functions. The following table gives the details of the additional functions of PORT 3 Pins.

PORT 3 Pin	Function	Description
P3.0	RXD	Serial Input
P3.1	TXD	Serial Output
P3.2	INT0	External Interrupt 0
P3.3	INT1	External Interrupt 1
P3.4	T0	Timer 0
P3.5	T1	Timer 1
P3.6	WR	External Memory Write
P3.7	RD	External Memory Read

Table 4.2

Pins 18 & 19: Pins 18 and 19 i.e. XTAL 2 and XTAL 1 are the pins for connecting external oscillator. Generally, a Quartz Crystal Oscillator is connected here.

Pin 20 (GND): Pin 20 is the Ground Pin of the 8051 Microcontroller. It represents 0V and is connected to the negative terminal (0V) of the Power Supply.

Pins 21 – 28 (PORT 2): These are the PORT 2 Pins of the 8051 Microcontroller. PORT 2 is also a Bidirectional Port i.e. all the PORT 2 pins act as Input or Output. Additionally, when external memory is interfaced, PORT 2 pins act as the higher order address byte. PORT 2 Pins have internal pull-ups.

Pin 29 (PSEN): Pin 29 is the Program Store Enable Pin (PSEN). Using this pins, external Program Memory can be read.

Pin 30 (ALE/PROG): Pin 30 is the Address Latch Enable Pin. Using this Pins, external address can be separated from data (as they are multiplexed by 8051).

During Flash Programming, this pin acts as program pulse input (PROG).

Pin 31 (EA/VPP): Pin 31 is the External Access Enable Pin i.e. allows external Program Memory. Code from external program memory can be fetched only if this pin is LOW. For normal operations, this pins is pulled HIGH.
During Flash Programming, this Pin receives 12V Programming Enable Voltage (VPP).
Pins 32 – 39 (PORT 0): Pins 32 to 39 are PORT 0 Pins. They are also bidirectional Input / Output Pins but without any internal pull-ups. Hence, we need external pull-ups in order to use PORT 0 pins as I/O PORT.
In addition to acting as I/O PORT, PORT 0 also acts as lower order address/data bus when external memory is accessed.
Pin 40 (VCC): Pin 40 is the power supply pin to which the supply voltage is given (+5V).

8051 Microcontroller Basic Circuit

Now that we have seen the 8051 Microcontroller Pin Diagram and corresponding Pin Description, we will proceed to the basic circuit or schematic of the 8051 Microcontroller. The following image shows the basic circuit of the 8051 Microcontroller.

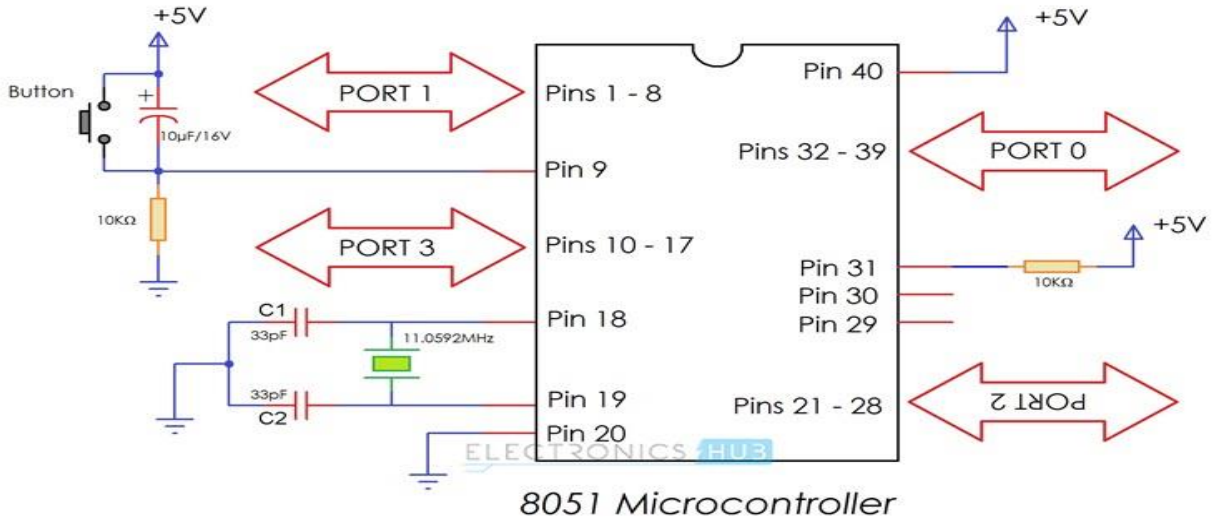


Figure 4.5: 8051 Microcontroller Basic Circuit

his basic circuit of 8051 microcontroller is the minimal interface required for it to work. The basic circuit includes a Reset Circuit, the oscillator circuit and power supply. Let us discuss a little bit deeper about this basic circuit of 8051 Microcontroller.
First is the power supply. Pins 40 and 20 (VCC and GND) of the 8051 Microcontroller are connected to +5V and GND respectively.
Next is the Reset Circuit. A logic HIGH (+5V) on Reset Pin for a minimum of two machine cycles (24 clock cycles) will reset the 8051 Microcontroller. The reset circuit of the 8051 Microcontroller consists of a capacitor, a resistor and a push button and this type of reset circuit provides a Manual Reset Option. If you remove the push button, then the reset circuit becomes a Power-On Reset Circuit.

The next part of the basic circuit of the 8051 Microcontroller is the Oscillator Circuit or the Clock Circuit. A Quartz Crystal Oscillator is connected across XTAL1 and XTAL2 pins i.e. Pins 19 and 18. The capacitors C1 and C2 can be selected in the range of 20pF to 40pF.

As mentioned in the 8051 Microcontroller Pin Description, PORTS 1, 2 and 3, all have internal pull – ups and hence can be directly used as Bidirectional I/O Ports. But, we need to add external Pull – ups for PORT 0 Pins in order to use it as an I/O Port.

Generally, a 1KΩ Resistor Pack of 8 Resistors is used as a Pull – up for the PORT 0 of the 8051 Microcontroller.

LECTURE 3: INSTRUCTION SET OF 8051

3. 8051 MICROCONTROLLER INSTRUCTION SET

Writing a Program for any Microcontroller consists of giving commands to the Microcontroller in a particular order in which they must be executed in order to perform a specific task. The commands to the Microcontroller are known as a Microcontroller's Instruction Set.
Just as our sentences are made of words, a Microcontroller's (for that matter, any computer) program is made of Instructions. Instructions written in a program tell the Microcontroller which operation to carry out.
An Instruction Set is unique to a family of computers. This tutorial introduces the 8051 Microcontroller Instruction Set also called as the MCS-51 Instruction Set.
As the 8051 family of Microcontrollers are 8-bit processors, the 8051 Microcontroller Instruction Set is optimized for 8-bit control applications. As a typical 8-bit processor, the 8051 Microcontroller instructions have 8-bit Opcodes. As a result, the 8051 Microcontroller instruction set can have up to $2^8 = 256$ Instructions.
.

A Brief Look at 8051 Microcontroller Instructions and Groups

Before going into the details of the 8051 Microcontroller Instruction Set, Types of Instructions and the Addressing Mode, let us take a brief look at the instructions and the instruction groups of the 8051 Microcontroller Instruction Set (the MCS-51 Instruction Set).
The following table shows the 8051 Instruction Groups and Instructions in each group. There are 49 Instruction Mnemonics in the 8051 Microcontroller Instruction Set and these 49 Mnemonics are divided into five groups.

DATA TRANSFER	ARITHMETIC	LOGICAL	BOOLEAN	PROGRAM BRANCHING
MOV	ADD	ANL	CLR	LJMP
MOVC	ADDC	ORL	SETB	AJMP
MOVX	SUBB	XRL	MOV	SJMP
PUSH	INC	CLR	JC	JZ
POP	DEC	CPL	JNC	JNZ
XCH	MUL	RL	JB	CJNE
XCHD	DIV	RLC	JNB	DJNZ
	DA A	RR	JBC	NOP
		RRC	ANL	LCALL
		SWAP	ORL	ACALL
			CPL	RET
				RETI
				JMP

Table 4.3

A simple instruction consists of just the opcode. Other instructions may include one or more operands. Instruction can be one-byte instruction, which contains only opcode, or two-byte instructions, where the second byte is the operand or three byte instructions, where the operand makes up the second and third byte.
Based on the operation they perform, all the instructions in the 8051 Microcontroller Instruction Set are divided into five groups. They are:

- Data Transfer Instructions
- Arithmetic Instructions
- Logical Instructions
- Boolean or Bit Manipulation Instructions
- Program Branching Instructions

We will now see about these instructions briefly.

Data Transfer Instructions

The Data Transfer Instructions are associated with transfer with data between registers or external program memory or external data memory. The Mnemonics associated with Data Transfer are given below.

- MOV
- MOVC
- MOVX
- PUSH
- POP
- XCH
- XCHD

The following table lists out all the possible data transfer instruction along with other details like addressing mode, size occupied and number machine cycles it takes.

Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
MOV	A, #Data	A ← Data	Immediate	2	1
	A, Rn	A ← Rn	Register	1	1
	A, Direct	A ← (Direct)	Direct	2	1
	A, @Ri	A ← @Ri	Indirect	1	1
	Rn, #Data	Rn ← data	Immediate	2	1
	Rn, A	Rn ← A	Register	1	1
	Rn, Direct	Rn ← (Direct)	Direct	2	2
	Direct, A	(Direct) ← A	Direct	2	1
	Direct, Rn	(Direct) ← Rn	Direct	2	2
	Direct1, Direct2	(Direct1) ← (Direct2)	Direct	3	2
	Direct, @Ri	(Direct) ← @Ri	Indirect	2	2
	Direct, #Data	(Direct) ← #Data	Direct	3	2
	@Ri, A	@Ri ← A	Indirect	1	1
	@Ri, Direct	@Ri ← Direct	Indirect	2	2
	@Ri, #Data	@Ri ← #Data	Indirect	2	1
	DPTR, #Data16	DPTR ← #Data16	Immediate	3	2
MOVC	A, @A+DPTR	A ← Code Pointed by A+DPTR	Indexed	1	2
	A, @A+PC	A ← Code Pointed by A+PC	Indexed	1	2
	A, @Ri	A ← Code Pointed by Ri (8-bit Address)	Indirect	1	2
MOVX	A, @DPTR	A ← External Data Pointed by DPTR	Indirect	1	2
	@Ri, A	@Ri ← A (External Data 8-bit Addr)	Indirect	1	2
	@DPTR, A	@DPTR ← A (External Data 16-bit Addr)	Indirect	1	2
PUSH	Direct	Stack Pointer SP ← (Direct)	Direct	2	2
POP	Direct	(Direct) ← Stack Pointer SP	Direct	2	2
XCH	Rn	Exchange ACC with Rn	Register	1	1
	Direct	Exchange ACC with Direct Byte	Direct	2	1
	@Ri	Exchange ACC with Indirect RAM	Indirect	1	1
XCHD	A, @Ri	Exchange ACC with Lower Order Indirect RAM	Indirect	1	1

Table 4.4

Arithmetic Instructions

Using Arithmetic Instructions, you can perform addition, subtraction, multiplication and division. The arithmetic instructions also include increment by one, decrement by one and a special instruction called Decimal Adjust Accumulator.

The Mnemonics associated with the Arithmetic Instructions of the 8051 Microcontroller Instruction Set are:

- ADD
- ADDC
- SUBB
- INC

- DEC
- MUL
- DIV
- DAA

The arithmetic instructions has no knowledge about the data format i.e. signed, unsigned, ASCII, BCD, etc. Also, the operations performed by the arithmetic instructions affect flags like carry, overflow, zero, etc. in the PSW Register.
All the possible Mnemonics associated with Arithmetic Instructions are mentioned in the following table.

Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
ADD	A, #Data	A ← A + Data	Immediate	2	1
	A, Rn	A ← A + Rn	Register	1	1
	A, Direct	A ← A + (Direct)	Direct	2	1
	A, @Ri	A ← A + @Ri	Indirect	1	1
ADDC	A, #Data	A ← A + Data + C	Immediate	2	1
	A, Rn	A ← A + Rn + C	Register	1	1
	A, Direct	A ← A + (Direct) + C	Direct	2	1
	A, @Ri	A ← A + @Ri + C	Indirect	1	1
SUBB	A, #Data	A ← A – Data – C	Immediate	2	1
	A, Rn	A ← A – Rn – C	Register	1	1
	A, Direct	A ← A – (Direct) – C	Direct	2	1
	A, @Ri	A ← A – @Ri – C	Indirect	1	1
MUL	AB	Multiply A with B (A ← Lower Byte of A*B and B ← Higher Byte of A*B)	--	1	4
DIV	AB	Divide A by B (A ← Quotient and B ← Remainder)	--	1	4
DEC	A	A ← A – 1	Register	1	1
	Rn	Rn ← Rn – 1	Register	1	1
	Direct	(Direct) ← (Direct) – 1	Direct	2	1
	@Ri	@Ri ← @Ri – 1	Indirect	1	1
INC	A	A ← A + 1	Register	1	1
	Rn	Rn ← Rn + 1	Register	1	1
	Direct	(Direct) ← (Direct) + 1	Direct	2	1
	@Ri	@Ri ← @Ri + 1	Indirect	1	1
	DPTR	DPTR ← DPTR + 1	Register	1	2
DA	A	Decimal Adjust Accumulator	--	1	1

Table 4.5

Logical Instructions

The next group of instructions are the Logical Instructions, which perform logical operations like AND, OR, XOR, NOT, Rotate, Clear and Swap. Logical Instruction are performed on Bytes of data on a bit-by-bit basis.
Mnemonics associated with Logical Instructions are as follows:

- ANL
- ORL
- XRL
- CLR
- CPL
- RL
- RLC
- RR
- RRC
- SWAP

The following table shows all the possible Mnemonics of the Logical Instructions.

Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
ANL	A, #Data	A ← A AND Data	Immediate	2	1
	A, Rn	A ← A AND Rn	Register	1	1
	A, Direct	A ← A AND (Direct)	Direct	2	1
	A, @Ri	A ← A AND @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) AND A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) AND #Data	Direct	3	2
ORL	A, #Data	A ← A OR Data	Immediate	2	1
	A, Rn	A ← A OR Rn	Register	1	1
	A, Direct	A ← A OR (Direct)	Direct	2	1
	A, @Ri	A ← A OR @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) OR A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) OR #Data	Direct	3	2
XRL	A, #Data	A ← A XRL Data	Immediate	2	1
	A, Rn	A ← A XRL Rn	Register	1	1
	A, Direct	A ← A XRL (Direct)	Direct	2	1
	A, @Ri	A ← A XRL @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) XRL A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) XRL #Data	Direct	3	2
CLR	A	A← 00H	--	1	1
CPL	A	A ← A	--	1	1
RL	A	Rotate ACC Left	--	1	1
RLC	A	Rotate ACC Left through Carry	--	1	1
RR	A	Rotate ACC Right	--	1	1
RRC	A	Rotate ACC Right through Carry	--	1	1
SWAP	A	Swap Nibbles within ACC	--	1	1

Table 4.6

Boolean or Bit Manipulation Instructions

As the name suggests, Boolean or Bit Manipulation Instructions will deal with bit variables. We know that there is a special bit-addressable area in the RAM and some of the Special Function Registers (SFRs) are also bit addressable.
The Mnemonics corresponding to the Boolean or Bit Manipulation instructions are:

- CLR
- SETB
- MOV
- JC
- JNC
- JB
- JNB
- JBC
- ANL
- ORL
- CPL

These instructions can perform set, clear, and, or, complement etc. at bit level. All the possible mnemoics of the Boolean Instructions are specified in the following table.

Mnemonic	Instruction	Description	# of Bytes	# of Cycles
CLR	C	C ← 0 (C = Carry Bit)	1	1
	Bit	Bit ← 0 (Bit = Direct Bit)	2	1
SET	C	C ← 1	1	1
	Bit	Bit ← 1	2	1
CPL	C	C ← $\overline{C}$	1	1
	Bit	Bit ← $\overline{Bit}$	2	1
ANL	C, /Bit	C ← C. $\overline{Bit}$ (AND)	2	1
	C, Bit	C ← C . Bit (AND)	2	1
ORL	C, /Bit	C ← C + $\overline{Bit}$ (OR)	2	1
	C, Bit	C ← C + Bit (OR)	2	1
MOV	C, Bit	C ← Bit	2	1
	Bit, C	Bit ← C	2	2
JC	rel	Jump is Carry (C) is Set	2	2
JNC	rel	Jump is Carry (C) is Not Set	2	2
JB	Bit, rel	Jump is Direct Bit is Set	3	2
JNB	Bit, rel	Jump is Direct Bit is Not Set	3	2
JBC	Bit, rel	Jump is Direct Bit is Set and Clear Bit	3	2

Table 4.7

Program Branching Instructions

The last group of instructions in the 8051 Microcontroller Instruction Set are the Program Branching Instructions. These instructions control the flow of program logic. The mnemonics of the Program Branching Instructions are as follows.

- LJMP
- AJMP
- SJMP
- JZ
- JNZ
- CJNE
- DJNZ
- NOP
- LCALL
- ACALL
- RET
- RETI
- JMP

All these instructions, except the NOP (No Operation) affect the Program Counter (PC) in one way or other. Some of these instructions has decision making capability before transferring control to other part of the program.
The following table shows all the mnemonics with respect to the program branching instructions.

LECTURE 4: ADDRESSING MODES OF 8051

4.8051 ADDRESSING MODES

What is an Addressing Mode?

An Addressing Mode is a way to locate a target Data, which is also called as Operand. The 8051 Family of Microcontrollers allows five types of Addressing Modes for addressing the Operands. They are:

- Immediate Addressing
- Register Addressing
- Direct Addressing
- Register – Indirect Addressing
- Indexed Addressing

Immediate Addressing

In Immediate Addressing mode, the operand, which follows the Opcode, is a constant data of either 8 or 16 bits. The name Immediate Addressing came from the fact that the constant data to be stored in the memory immediately follows the Opcode.

The constant value to be stored is specified in the instruction itself rather than taking from a register. The destination register to which the constant data must be copied should be the same size as the operand mentioned in the instruction.

Example: MOV A, #030H

Here, the Accumulator is loaded with 30 (hexadecimal). The # in the operand indicates that it is a data and not the address of a Register.

Immediate Addressing is very fast as the data to be loaded is given in the instruction itself.

Register Addressing

In the 8051 Microcontroller Memory Organization Tutorial, we have seen the organization of RAM and four banks of Working Registers with eight Registers in each bank.

In Register Addressing mode, one of the eight registers (R0 – R7) is specified as Operand in the Instruction.

It is important to select the appropriate Bank with the help of PSW Register. Let us see a example of Register Addressing assuming that Bank0 is selected.

Example: MOV A, R5

Here, the 8-bit content of the Register R5 of Bank0 is moved to the Accumulator.

Direct Addressing

In Direct Addressing Mode, the address of the data is specified as the Operand in the instruction. Using Direct Addressing Mode, we can access any register or on-chip variable. This includes general purpose RAM, SFRs, I/O Ports, Control registers.

Example: MOV A, 47H

Here, the data in the RAM location 47H is moved to the Accumulator.

Register Indirect Addressing

In the Indirect Addressing Mode or Register Indirect Addressing Mode, the address of the Operand is specified as the content of a Register. This will be clearer with an example.

Example: MOV A, @R1

The @ symbol indicates that the addressing mode is indirect. If the contents of R1 is 56H, for example, then the operand is in the internal RAM location 56H. If the contents of the RAM location 56H is 24H, then 24H is moved into accumulator.

Only R0 and R1 are allowed in Indirect Addressing Mode. These register in the indirect addressing mode are called as Pointer registers.

Indexed Addressing Mode

With Indexed Addressing Mode, the effective address of the Operand is the sum of a base register and an offset register. The Base Register can be either Data Pointer (DPTR) or Program Counter (PC) while the Offset register is the Accumulator (A).

In Indexed Addressing Mode, only MOVC and JMP instructions can be used. Indexed Addressing Mode is useful when retrieving data from look-up tables.

Example: MOVC A, @A+DPTR

Here, the address for the operand is the sum of contents of DPTR and Accumulator.

Types of Instructions in 8051 Microcontroller Instruction Set

Before seeing the types of instructions, let us see the structure of the 8051 Microcontroller Instruction. An 8051 Instruction consists of an Opcode (short of Operation – Code) followed by Operand(s) of size Zero Byte, One Byte or Two Bytes.

The Op-Code part of the instruction contains the Mnemonic, which specifies the type of operation to be performed. All Mnemonics or the Opcode part of the instruction are of One Byte size.

Coming to the Operand part of the instruction, it defines the data being processed by the instructions. The operand can be any of the following:

- No Operand
- Data value
- I/O Port
- Memory Location
- CPU register

There can multiple operands and the format of instruction is as follows: