Unit 2: Basic Computer Architecture

Unit 4: Assembly Language Programming

PART-I

8085 Microprocessor & Assembly Language Programming with 8085

Introduction to 8085 Microprocessor

- **Intel 8085A** (simply **8085**) is an 8-bit general-purpose microprocessor introduced by intel in 1977.
- It can address 64K Byte of memory.
- It has **40 pins**.
- An 8085 Microprocessor requires a +5V power supply to operate.
- Its clock speed is about 3 MHz.
- Intel 8085 is an enhanced version of Intel 8080.

Important Features of 8085

1. **BUS:**

   A bus is a collection of lines, which carries data or programs and perform some logical tasks.

   - The size of a bus indicates the number of lines in it, and hence the number of bits the bus can carry – as one line carries one bit of information.
   - There are three types of **buses**:
   - ✓ **Address bus:**
     - This bus carries the address of a particular location, for a transfer.
     - It is unidirectional.
     - 16-bit address bus, which can address up to 64KB.
     - It is very important to know that the size of the address bus decides the maximum number of locations the processor can address.

       i.e. n-bit address bus 2^n memory locations.
     - The memory address range is from 0000H … FFFFH.

Important Features of 8085

1. **BUS:**
   - ✓ **Data Bus:**
     - This bus carries the data to be transferred. It is bidirectional.
     - 8085 has an 8-bit data bus, which means it can transfer 8-bits in one operation and hence it is called as an 8-bit microprocessor.
   - ✓ **Control Bus:**
     - This bus carries the control signals that would cause any kind of an operation. The basic control signals are RD, WR etc.
   - ❖ Together these three buses are called as the **SYSTEM BUS**.
2. This microprocessor is an 8-bit device that receives, operates, or outputs 8-bitinformation in a simultaneous approach.
3. It can generate 8-bit IO Address hence it can access 2^8 I/O Ports i.e. 256 I/O Ports. The I/O Address ranges from 00H … FFH.
4. This is constructed of a single NMOS chip device and has 6200 transistors.

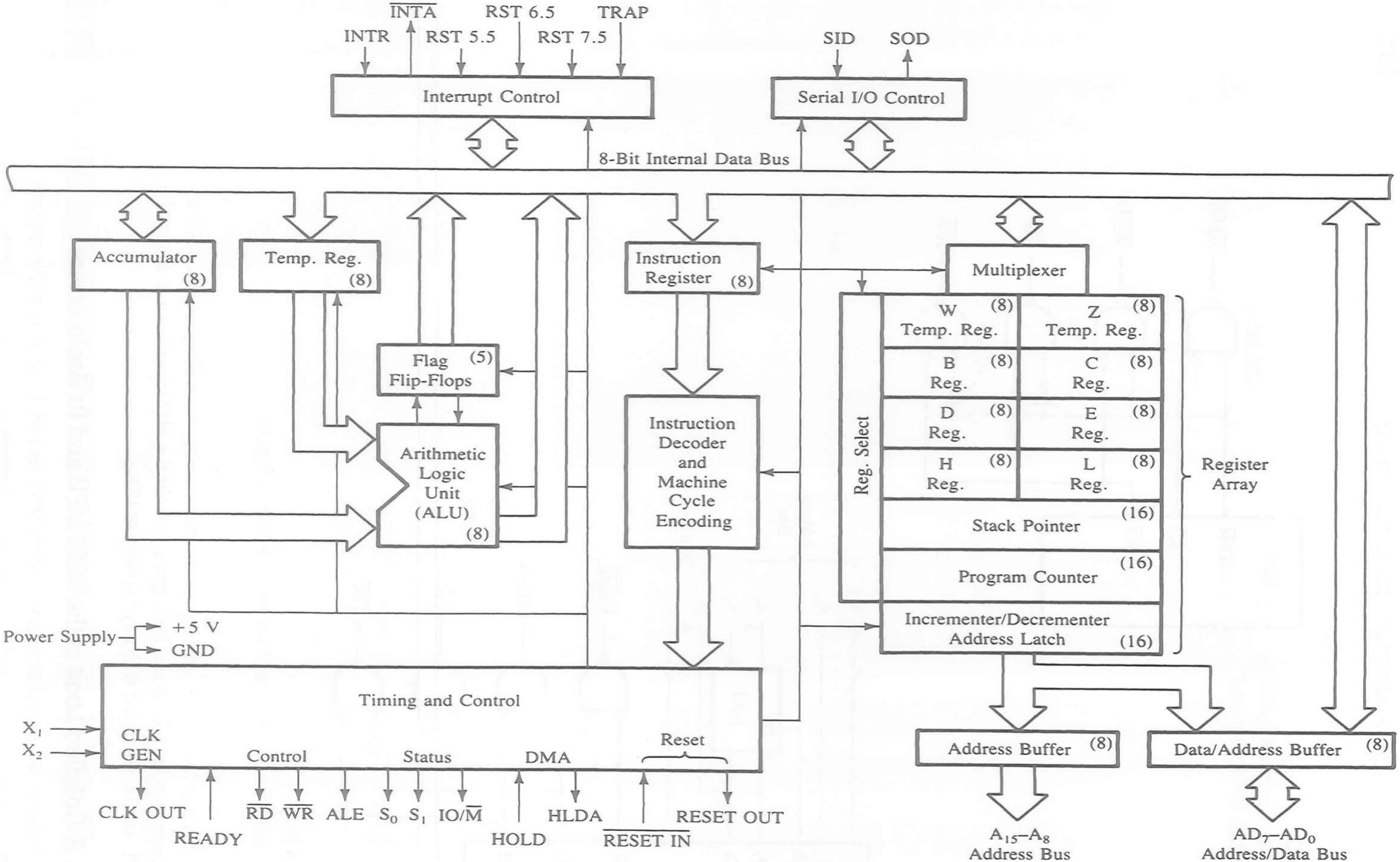
Important Features of 8085

5. This microprocessor is available in a **DIP** package of 40 pins.
6. A 16-bit **program counter**
7. A 16-bit **stack pointer**
8. Six 8-bit **general purpose registers** arranged in pairs: **BC, DE, HL**
9. It works on **+5 V power supply**.
10. Its operating frequency is **3 MHz**
11. A total of **246 operational codes** and **74 different instructions** are present.
12. It accepts 5 external **hardware interrupts** i.e., TRAP**, RST 7.5, RST 6.5, RST 5.5** and INTR.
13. The 8085 microprocessor has eight **software interrupts** from **RST0** through **RST7**.

Internal Architecture of 8-bit Microprocessor and its Registers

- The Intel 8085A is a complete 8-bit parallel **central processing unit**.
- The **main components** of 8085A are **array of registers**, **the arithmetic logic unit**, **the encoder/decoder**, and **timing and control circuits** linked by an **internal data bus**.
- The block diagram is shown below:

8085 Functional Block Diagram (Internal Architecture)



1. Registers

I. **Program Counter (PC, 16-bits):**

- It is a 16-bit Special-Purpose register. It holds **address** of the **next instruction**. PC is incremented by the INR/DCR after every instruction byte is fetched.

II. **Stack Pointer (SP, 16-bits):**

- It is a 16-bit Special-Purpose register. It holds **address** of the **top of the Stack**.
- Stack is a set of memory locations operating in LIFO manner.
- SP is **decremented** on every **PUSH** operation and **incremented** on every **POP**.

III. **B, C, D, E, H, L registers 8-bits each:**

- These are 8-bit General-Purpose registers.
- They can also be used to store 16-bit data in register pairs.
- The possible register **pairs** are **BC** pair, **DE** pair and pair.
- The **HL** pair also holds the **address** for the Memory Pointer "**M**".

IV. **Temporary Registers (WZ, 16-bits):**

- This is a 16-bit register pair.
- It is **used by µP** to hold **temporary** values in some instructions like XCHG,CALL/JMP etc. The **programmer** has **no access** to this register pair.

1. Registers

V. INR/DCR Register (16-bits):

- This is a 16-bit shift register.
- It is used to **increment PC after every instruction byte is fetched** and **increment** or **decrement SP** after a Pop or a Push operation respectively.
- It is not available to the programmer.

VI. A - Accumulator (8-bits):

- It is an 8-bit programmable register.
- The user can read or write this register.
- It has two **special properties**:
  - It **holds one** of the **operands** during most of the arithmetic operations.
  - It **holds** the **result** of most of the arithmetic and logic operations.

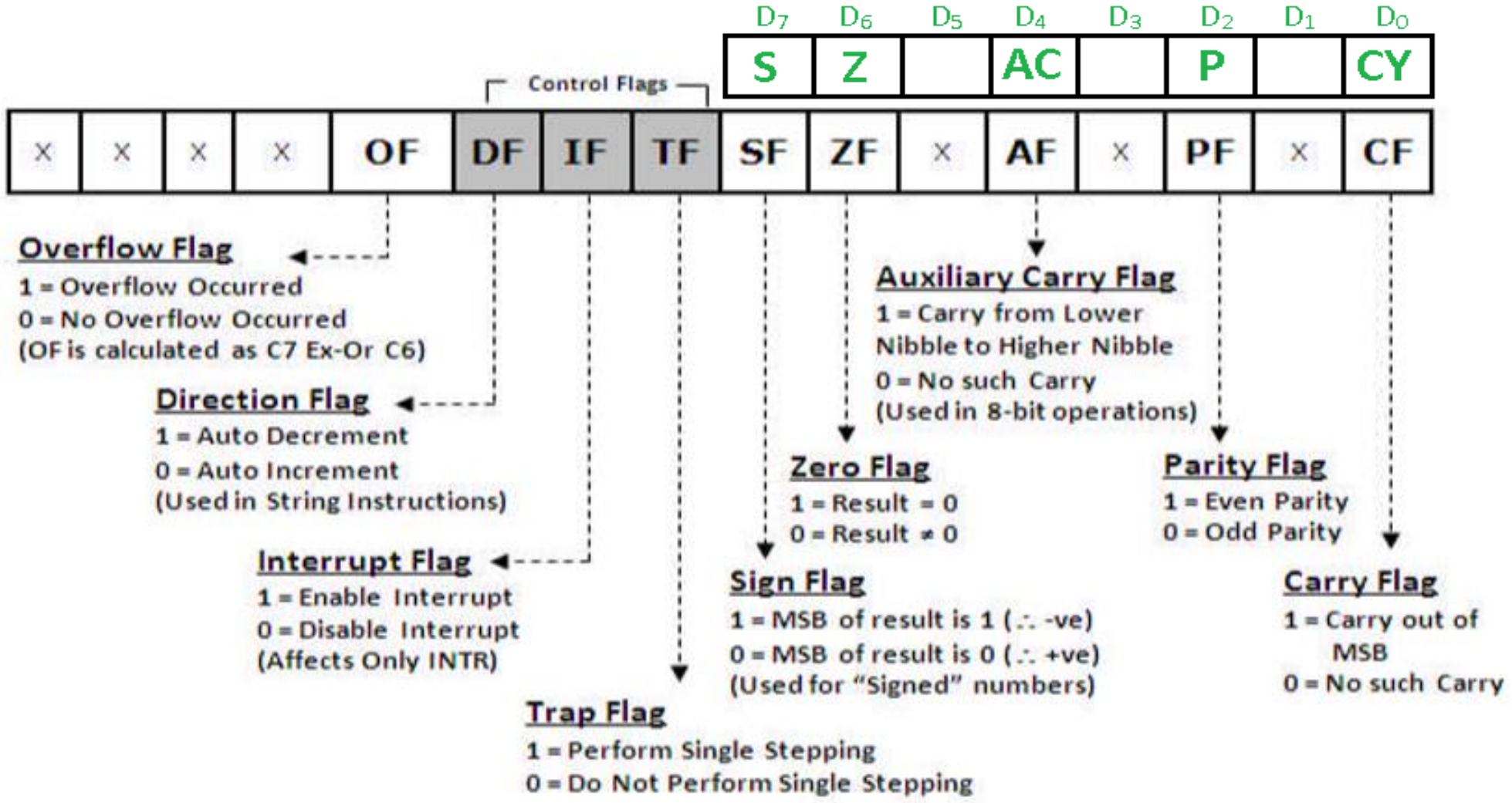
VII. Temp Register (8-bits):

- This is an 8-bit register.
- It is used by **µP** for storing one of the operands during an operation.
- The **programmer** has **NO ACCESS** to this register.

2. FLAG REGISTER

- Register consists of five flip flops, each holding the status of different states separately is known as flag register and each flip flops are called flags.
- 8085A can set or reset one or more of the flags and they are:
  - Sign(S),
  - Zero (Z),
  - Auxiliary Carry (AC)
  - Parity (P) and
  - Carry (CY).
- The state of flags indicates the result of arithmetic and logical operations, which in turn can be used for decision making processes.
- The different flags are described as:

2. FLAG REGISTER



2. FLAG REGISTER

I. S – Sign Flag:

- It is set (1) when MSB of the result is 1 (i.e. result is a -VE number).
- It is reset (0) when MSB of the result is 0 (i.e. result is a +VE number).

II. Z – Zero Flag:

- It is set when the result is = zero.
- It is reset when the result is not = zero.

III. AC – Auxiliary Carry Flag:

- It is set when an Auxiliary Carry / Borrow is generated.
- It is reset when an Auxiliary Carry / Borrow is not generated.
- Auxiliary Carry is the Carry generated between the lower nibble and the higher nibble for an 8bit operation. It is not affected after a 16- bit operation. It is used only in DAA operation.

IV. P – Parity Flag:

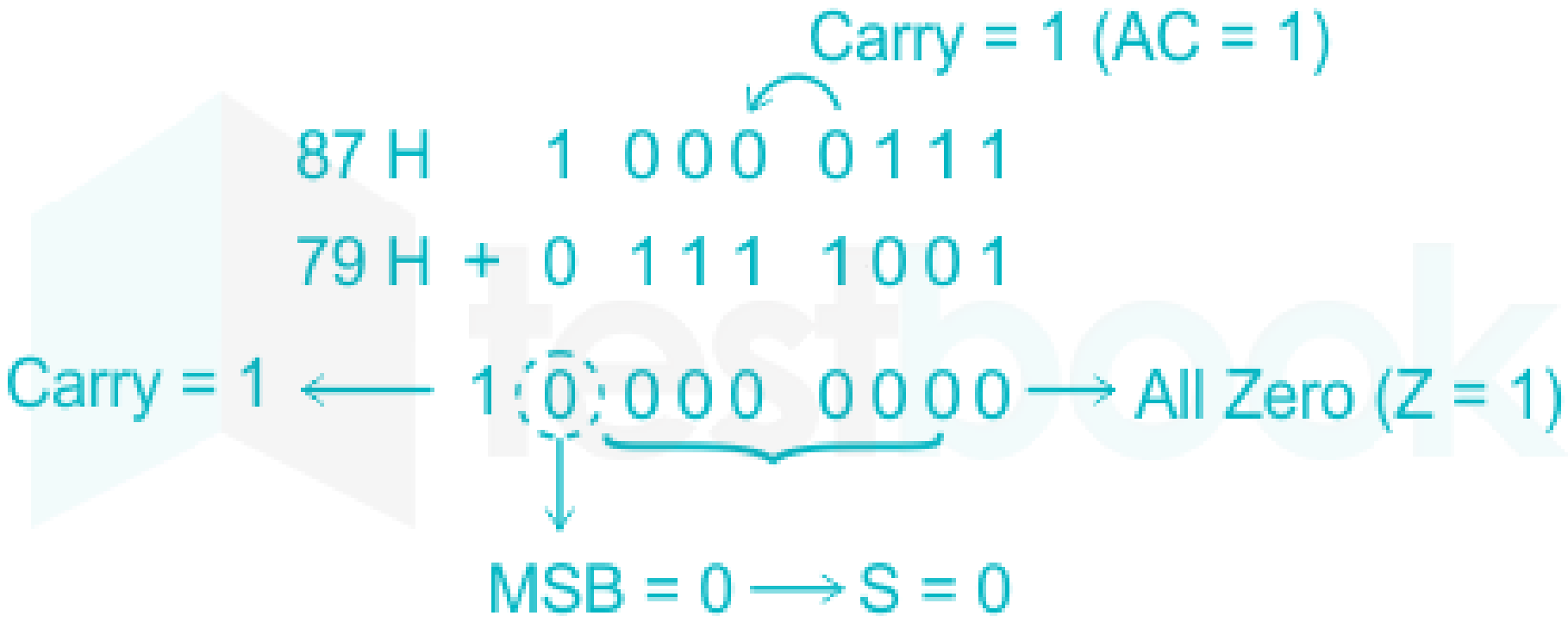
- It is set (1) when result has even parity. It is reset when result has odd parity.

V. C – Carry Flag:

- It is set when a Carry / Borrow is generated from the MSB.
- It is reset when a Carry / Borrow is not generated from the MSB.

2. FLAG REGISTER

❖ **Example of Flag:**



3. TIMING AND CONTROL UNIT

- This unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between the microprocessor and peripherals.
- The timing and control circuit issues the various internal and external control signals for executing and instruction.
- The signals are sync pulses indicating the availability of data on the data bus.

4. INTERRUPT CONTROLS

- The various interrupt control signals (INTR, RST 5.5, RST 6.5, RST 7.5 and TRAP) are used to interrupt a microprocessor.

5. SERIAL CONTROL

- This Block is responsible for transferring data Serially to and from the **µP**.
- **SID – Serial In Data:**
  - **µP** receives data, bit-by-bit through this line.
- **SOD – Serial Out Data:**
  - **µP** sends out data, bit-by-bit through this line.
  - Serial transmission can be done by **RIM (Read Interrupt Mask)** and **SIM (Set Interrupt Mask)** Instructions.

6. ALU – ARITHMETIC LOGIC UNIT

- 8085 has an **8-bit ALU**.
- It performs 8-bit arithmetic operations like Addition and Subtraction.
- It also performs logical operations like AND, OR, EX-OR NOT etc.
- It takes **input** from the **Accumulator** and the **Temp register**.
- The **output** of most of the ALU operations is **stored** back into the **Accumulator**.

7. INSTRUCTION REGISTER AND DECODER

❖Instruction Register:

- ❑ The 8085 places the contents of the PC onto the Address bus and fetches the instruction. This fetched instruction is stored into the Instruction register.

❖Instruction Decoder:

- ❑ The fetched instruction from the Instruction register enters the Instruction Decoder.
- ❑ Here the instruction is decoded and the decoded information is given to the Timing and Control Circuit where the instruction is executed.

Note: ***"Programmer's Model"*** *simply means all registers in the architecture that are available to the programmer. So if they ask Programmers model, draw the internal part of the architecture without the pins, and explain all registers.*

HOW DOES MICROPROCESSOR WORK?

- The instructions are stored sequentially in the memory.
- The microprocessor fetches the first instruction from its memory sheet, decodes it and executes that instruction.
- The sequence of fetch, decode, and execute is continued until the microprocessor comes across an instruction to stop.
- During the entire process, the microprocessor uses the system bus to fetch the binary instructions and data from the memory.
- It uses register from the register section to store data temporarily and it performs computing function in arithmetic logic unit [ALU] section.
- Finally, it sends out the result in binary using the same bus lines.

Memory and Instruction Fetch

- The primary function of memory is to store instructions and data and to provide that information to **MPU** whenever the **MPU** requests it.
- The MPU requests the information by sending the information 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.

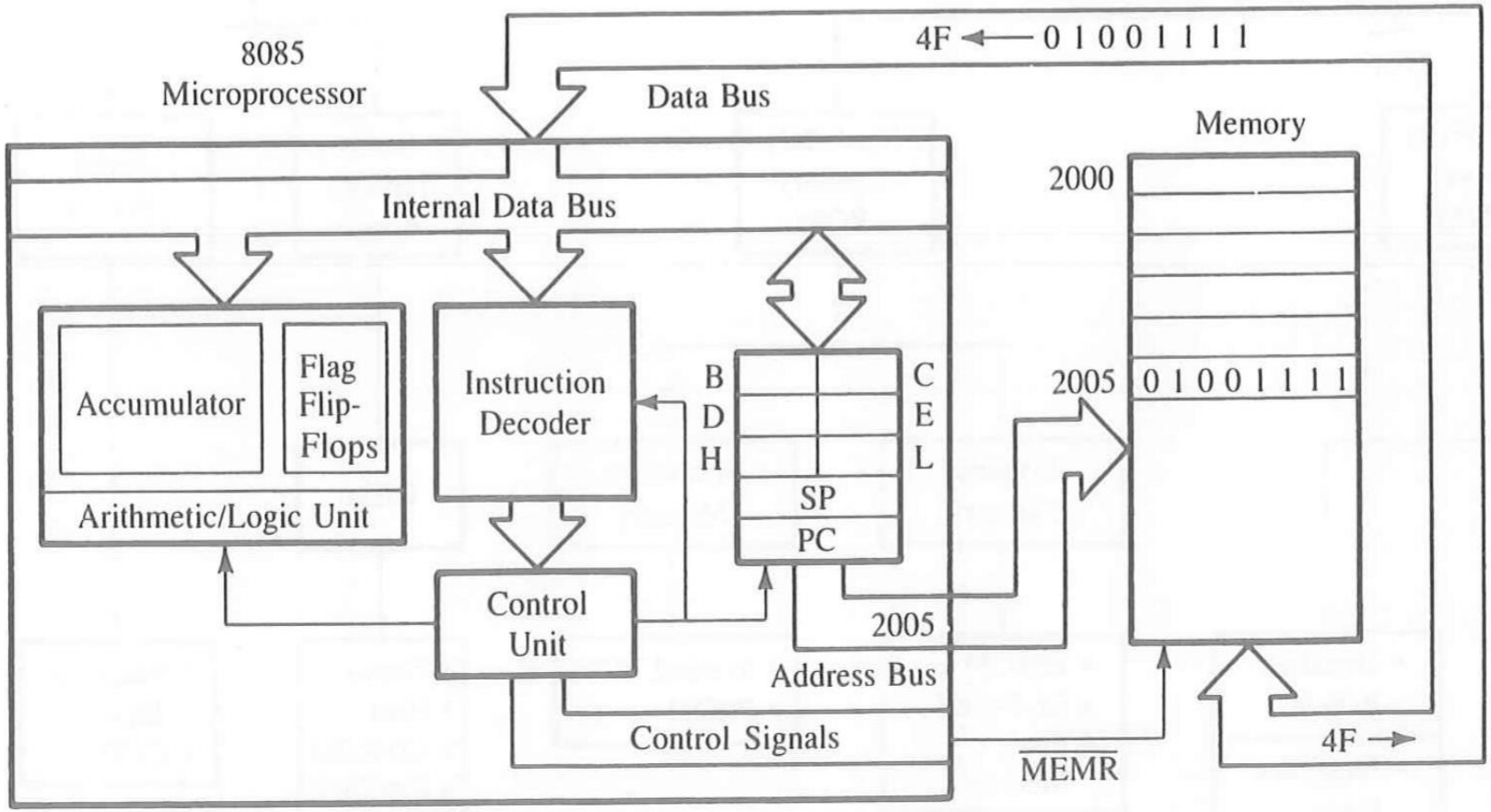


FIGURE 3.12
Instruction Fetch Operation

Memory and Instruction Fetch

- The instruction code 0100 1111 (4FH) is stored in memory location 2005H.
- To fetch the instruction located in memory location 2005H, the following steps are performed:
  i. The program counter places the 16-bit address 2005H of the memory location on the address bus (shown in figure 3.12).
  ii. The control unit sends the Memory Read control signal ($\overline{\mathbf{MEMR}}$, active low) to enable the output buffer of the memory chip.
  iii. The instruction (4FH) stored in the memory location is placed on the data bus and transferred (copied) to the instruction decoder of the µP.
  iv. The instruction is decoded and executed according to the binary pattern of the instruction

THE 8085 HARDWARE/PROGRAMMING MODEL AND FLAG REGISTERS

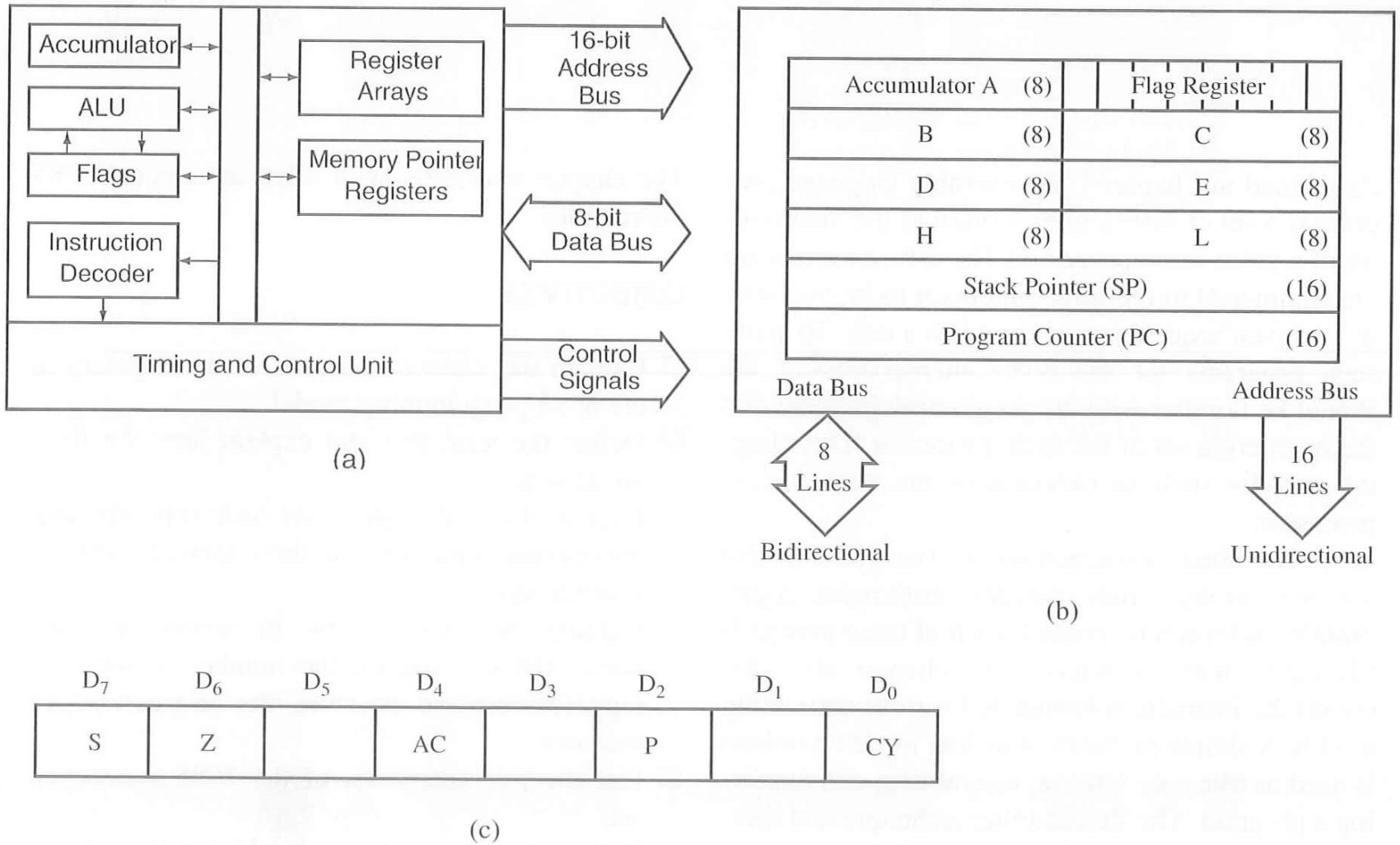


FIGURE 2.1
8085 Hardware Model (a), Programming Model (b), and Flag Register (c)

8085 HARDWARE MODEL

- The hardware model in **figure 2.1(a)** shows **two major segments**.
- **One segment** includes **arithmetic logic unit [ALU]** and an **8-bit register** called an **accumulator**, **instruction decoder**, and **flags**.
- The **second segment** shows **8-bit** and **16-bit registers.**
- Both segments are connected with various internal connections called an **internal bus**.
- The arithmetic and logic operations are performed in the arithmetic logic unit [ALU]. Results are stored in the accumulator, and flip-flops, called flags, are set or reset to reflect the results.
- There are 3 buses: a 16-bit unidirectional **address bus** used to send out **memory address**, an 8-bit bidirectional **data bus** used to **transfer data**, and a **control bus** used for **timing signals**.

8085 PROGRAMMING MODEL

- The programming model consists of some segments of ALU and the registers.
- This model includes **six registers**, **one accumulator**, and **one flag register**, as shown in **figure 2.1(b)**.
- These registers are critically required when programming a 8085 processor.
- In addition, it has two 16-bit registers: the **stack pointer** and the **program counter**.

1. **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 2.1(b)**.
   - 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.

8085 PROGRAMMING MODEL

2. 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**.

3. 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 2.1(c)**.
- The most commonly used flags are **Zero, Carry,** and **Sign**.
- The **µP** uses these flags to test data conditions.

8085 PROGRAMMING MODEL

3. FLAGS:

- For example, after an **addition** of **two numbers**,
- if the **sum** in the **accumulator** is larger than **eight bits**, the flip-flop uses to indicate a **carry** -- called the **Carry flag (CY)** -- is set to **one(High)**.
- 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.

8085 PROGRAMMING MODEL

4. PROGRAM COUNTER (PC):

- This is a **16-bit register** used to hold **memory address**.
- The size of PC register is 16 bits because the memory address are 16 bits.
- The microprocessor uses **PC 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.

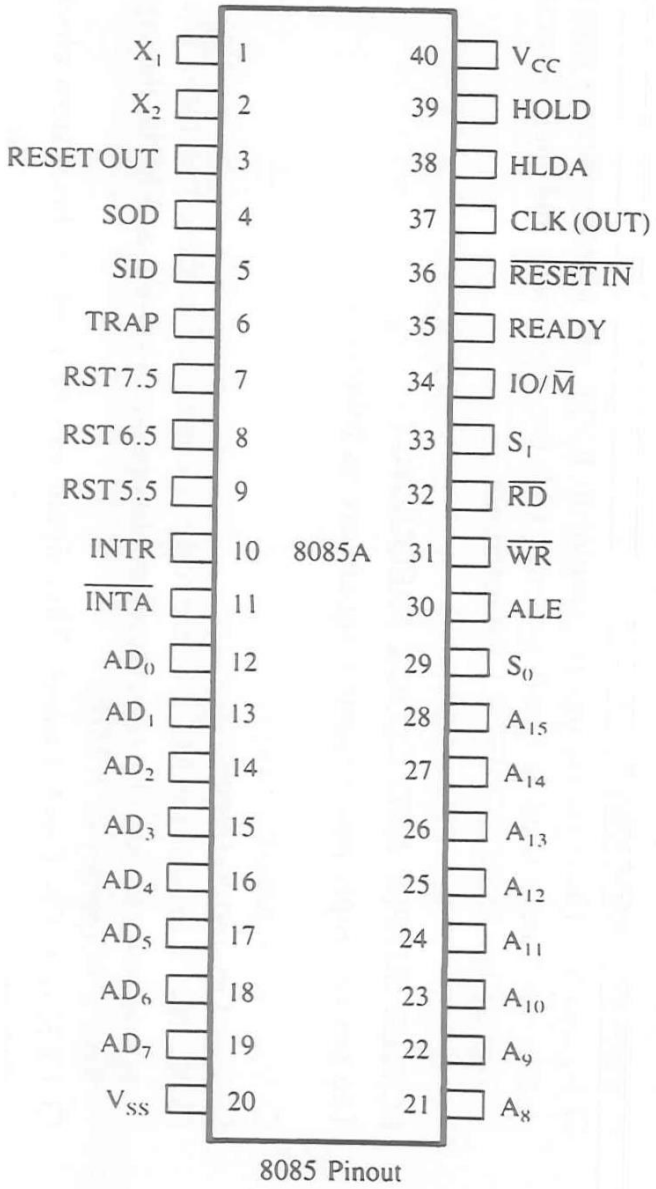
5. STACK POINTER (SP):

- 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 to examine how these registers are affected after the execution of an instruction.

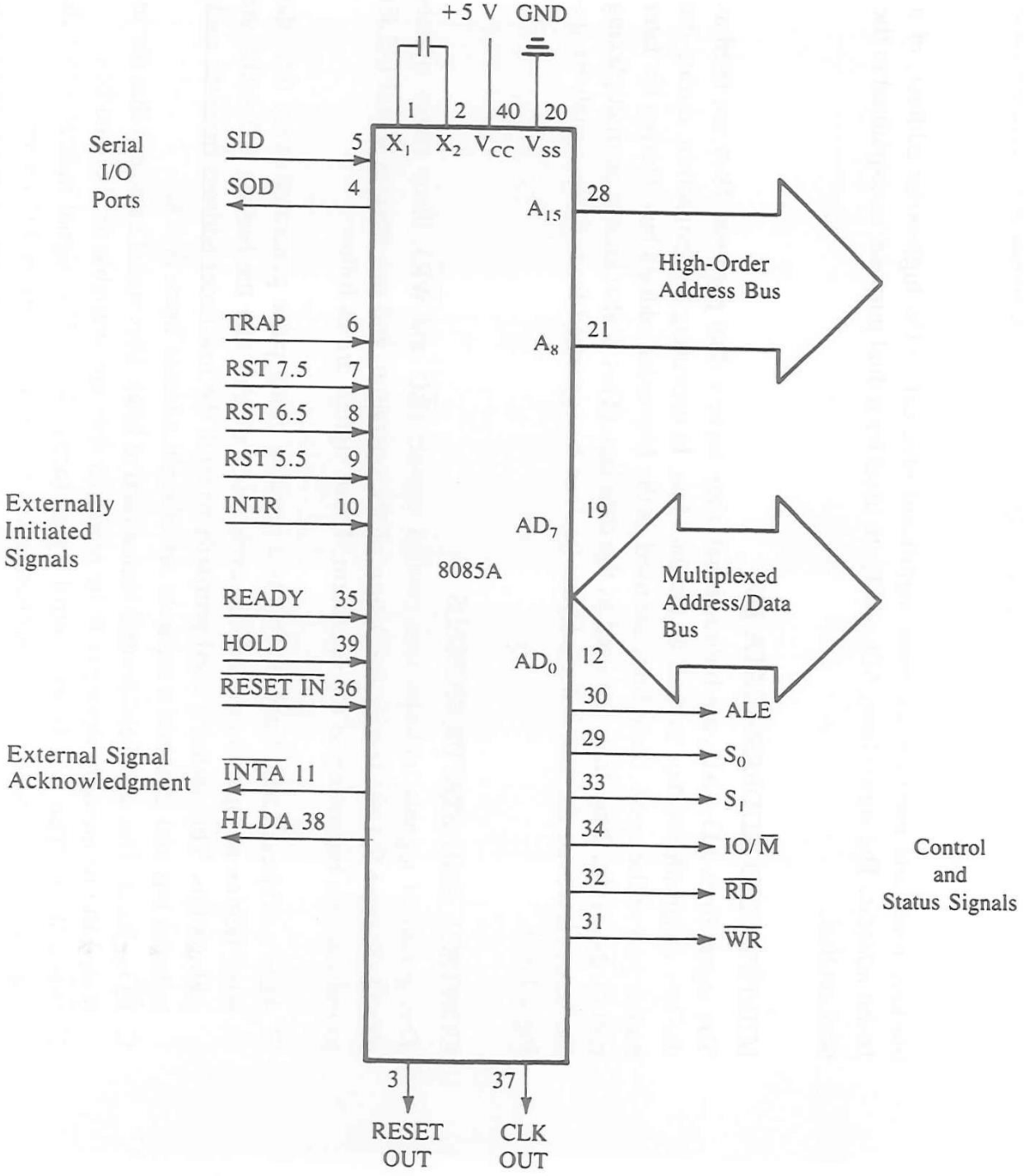
Pin Diagram (Pin Configuration) of 8085

- The **8085 pin diagram** consists of **40 pins** of the microprocessor.
- The all of the signals associated with 8085 can be classified into **6 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

Pin Configuration of 8085



8085 Pinout



Pin Diagram (Pin Configuration) of 8085

1. Address bus:

- The 8085 has 16 signal lines that are used as the address bus; however, these lines are split into two segments A_{15}-A_8 and AD_7- AD_0.
- The eight signals A_{15}-A_8 are unidirectional and used as high order bus.

2. Data bus:

- The signal lines AD_7- AD_0 are bidirectional, they serve a dual purpose.
- They are used the low order address bus as well as data bus.

3. Control and status signals:

- This group of signals includes two **control signals** (**$\overline{RD}$** and **$\overline{WR}$**), three **status signals** (**IO/$\overline{M}$**, **S_1** and **S_0**) to identify the nature of the operation, and one **special signals (ALE)** to indicate the beginning of the operation.
  - **ALE- Address Latch Enable:**
    - ✓ This is a positive going pulse generated every time the 8085 begins an operation (machine cycle); it indicates that the bits AD7-AD0 are address bits. This signal is used primarily to latch the low-order address from the multiplexed bus and generate a separate set of eight address lines A7 –A0 .
    - ✓ This signal is **used to latch address** from the multiplexed Address-Data Bus (**AD0-AD7**). When the Bus contains **address**, **ALE** is **high**, else it is **low**.

Pin Diagram (Pin Configuration) of 8085

3. Control and status signals:

- $\overline{\mathbf{RD}}$ – Read: this 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.
- $\overline{\mathbf{WR}}$ – Write: This is a write control signal (**active low**). This signal indicates that the data on the data bus are to be written into a selected memory or I/O location.
- **IO/$\overline{\mathbf{M}}$**: 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** indicates a **memory operation**. This signal is combined with $\overline{\mathbf{RD}}$ (Read) and $\overline{\mathbf{WR}}$ (Write) to generate I/O and memory signals.
- **S_1 and S_0** : These status signals, similar to **IO/$\overline{\mathbf{M}}$**, can identify various operations, but they are rarely used in small systems.

Pin Diagram (Pin Configuration) of 8085

3. Control and status signals:

- These lines denote the status of the µP

S1	S0	Status/ Operation
0	0	Halt/Idle
0	1	Write
1	0	Read
1	1	Opcode fetch

IO/$\overline{M}$	S1	S0	Data Bus Status (Output)
0	1	1	Opcode fetch
0	1	0	Memory read
0	0	1	Memory write
1	1	0	I/O read
1	0	1	I/O write
1	1	1	Interrupt acknowledge
0	0	0	Halt

Pin Diagram (Pin Configuration) of 8085

4. Power Supply and Clock frequency:

❖ V_{CC}: +5V power supply

❖V_{SS}: Ground reference

❖X_1, X_2:

- ❑ A crystal (RC, LC Network) is connected at these two pins and is used to provide the **Clock Input to the µP**.
- ❑ This frequency is internally divided by 2.

∴ To operate a system at 3 MHz, the crystal oscillator should have a frequency of 6 MHz.

❖**CLK OUT:**

- ❑ It can be used as the system clock for other devices.
- ❑ 8085 provides the **Clock input** to all **other peripherals** through the Clock Out pin.
- ❑This takes care of **synchronizing** all peripherals with 8085.
- ❑Its frequency is half the oscillator frequency.

Pin Diagram (Pin Configuration) of 8085

5. Externally Initiated signals:

- **INTR** (input): **interrupt request**,
  - used as a general purpose interrupt.
  - The INTR interrupt is a **maskable interrupt** that is generated by an external device, such as a keyboard or a mouse.
  - It has the **lowest priority** and can be **disabled**..
  - It has an **acknowledgement signal $\overline{\text{INTA}}$** .

- **$\overline{\text{INTA}}$** (Output): **interrupt acknowledgement,**
  - This is used to acknowledge an Interrupt.
  - This is an **acknowledgement signal** for **INTR** (only).

Pin Diagram (Pin Configuration) of 8085

5. Externally Initiated signals:

- **RST 7.5, 6.5, 5.5** (inputs):
  - These are **vectored interrupts** that transfer the program control to specific **memory locations**.
  - maskable interrupt that is generated by a software instruction.
  - They have **higher priorities** than **INTR** interrupt.
  - Among these three, the priority order is **RST7.5 > RST6.5 > RST5.5**.

- **TRAP** (input):
  - This is a **non-maskable** interrupt that is generated by an external device, such as a power failure or a hardware malfunction which has the **highest priority** and cannot be **disabled**.

Pin Diagram (Pin Configuration) of 8085

5. Externally Initiated signals:

- **HOLD(input) and HLDA(output):**
  - The Hold and Hold Acknowledge signals are used for **Direct Memory Access (DMA).**
  - The **DMA Controller** issued the **Hold** signal to the µP.
  - In response the **µP releases** the **System bus**. After releasing the system bus the **µP acknowledges** the Hold signal with **HLDA** signal. The DMA Transfer thus begins.
  - DMA Transfer is terminated by releasing the HOLD signal.

- **$\overline{\text{RESET IN}}$:**
  - This is an active low signal activated when the manual reset signal is applied to the µP.
  - This signal **resets the µP**. On Reset PC contains **0000H**.
  - Hence, the **Reset Vector Address** of 8085 is 0000H.

- **RESET OUT**:
  - This signal is connected to the reset input of all the peripherals.
  - It is used to **reset the peripherals** once the **µP** is **reset**.

Pin Diagram (Pin Configuration) of 8085

5. **Externally Initiated signals:**
   - **READY:**
     - This is an active high input.
     - It is used by the microprocessor to sense whether a peripheral is **ready** or not for data transfer. If not, the processor waits.
     - It is thus used to **synchronize** the µP with "**Slower**" Peripherals.
     - The **µP samples** the **Ready** input in the beginning of every Machine Cycle.
     - **If** it is found to be **LOW**, the **µP executes** one **WAIT CYCLE** after which it re-samples the ready pin till it finds the Ready pin HIGH.

     ∴ The **µP remains** in the **WAIT STATE** until the **READY** pin becomes **high** again. Hence, **if** the **Ready** pin is **not required** it should be **connected** to the **Vcc**, and not, left unconnected, **otherwise** would cause the µP to execute **infinite wait cycles**.

Pin Diagram (Pin Configuration) of 8085

6. Serial I/O Ports:

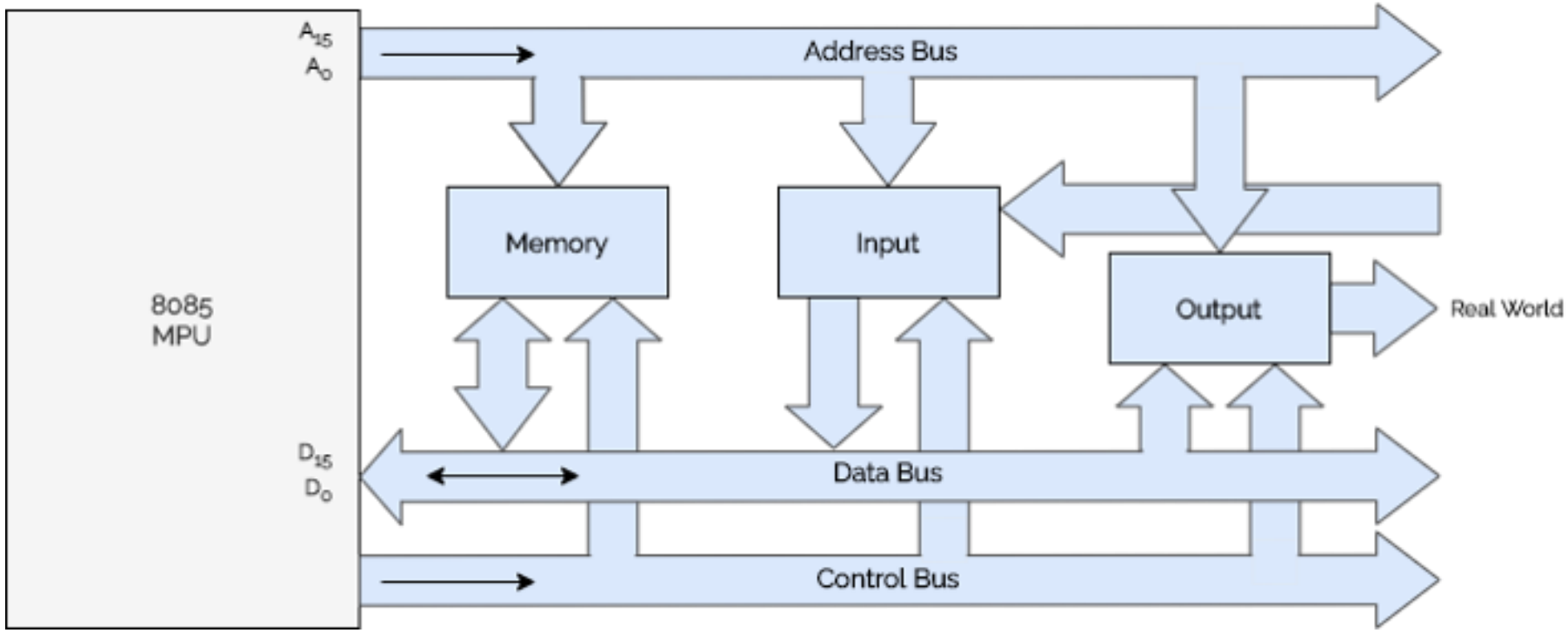
- The 8085 has **two signals** to implement the serial transmission:
  - **SID** (Serial Input Data) which is used to accept serial data bit by bit from the external device.
  - **SOD** (Serial Output Data) which enables the transmission of serial data bit by bit to the external device.
- In serial transmission, data bits are sent over **a single line**, **one bit** at a time, such as the transmission over telephone lines.

8085 Bus Organization

- The operation of a microprocessor unit includes,
  - Memory Read.
  - Memory Write.
  - I/O Read.
  - I/O Write.
- Communication of a Microprocessor unit with peripheral devices involves,
  1. Identify peripheral or memory location.
  2. Transfer data and instruction.
  3. Provide timing and synchronization signals.

8085 Bus Organization

- The microprocessor uses three sets of communication lines called **buses** to perform these functions.
- A **bus** is a collection of lines, which perform the same logical task. The 8085 bus organization is shown in the figure.



8085 Bus Organization

8085 Bus Organization

1. **Address Bus**
   - ❑Group of 16 bits from A0 to A15.
   - ❑Carries the address of a particular location.
   - ❑Unidirectional– Data flows from microprocessor to peripheral devices only.
   - ❑Function– To identify a peripheral or a memory location.
   - ❑Capable of addressing 65536(2^{16}) memory locations. (Generally **64K**)
2. **Data Bus**
   - ❑Group of 8 lines used for data flow.
   - ❑Carries the data to be transfered.
   - ❑Bidirectional– Data flow in both direction between the microprocessor and memory / peripheral devices.
   - ❑Function– To transfer binary data and instruction.
   - ❑Enable the microprocessor to manipulate 8-bit data ranging from 00 to FF. (28=255)

8085 Bus Organization

3. **Control Bus**
   - ❑Group of various single lines that carry control signals.
   - ❑Microprocessor generate specific control signals for every operation.
   - ❑Function– To provide timing and synchronization signals.

Programming with Intel 8085 Microprocessor

8085 ASSEMBLY LANGUAGE

- Even though the instruction can be written in hexadecimal code, it is still difficult to understand a program written in hexadecimal numbers.
- Therefore, each manufacturer of a microprocessor has devised a symbolic code for each instruction called **mnemonic**.
- The **mnemonic** for a particular instruction consists of letters that suggest the operation to be performed by that instruction.
- **Example**- the binary code **0011 1100** of the 8085 microprocessor is represented by the mnemonic **INR A.**

8085 MACHINE LANGUAGE

- The 8085 is a microprocessor with 8 bit word length.
- Its instruction set is designed by using various combinations of these 8 bits.
- The 8085 is an improved version of the earlier processor 8080A.
- An instruction is a binary pattern entered through an input device in memory to command the microprocessor to perform that specific function.
- The 8085 microprocessor has 246 such bit patterns, amounting to 74 different instructions for performing various operations.
- These 74 different instructions are called its **instruction set**.

Instruction description and Format:

- An **Instruction** is a binary pattern which performs a specific task/function.
- The computer can be used to perform a specific task, only by specifying the necessary steps to complete the task.
- The collection of such ordered steps forms a ‘program’ of a computer. These ordered steps are the instructions.
- Computer instructions are stored in central memory locations and are executed sequentially one at a time.
- The control reads an instruction from a specific address in memory and executes it. It then continues by reading the next instruction in sequence and executes it until the completion of the program.

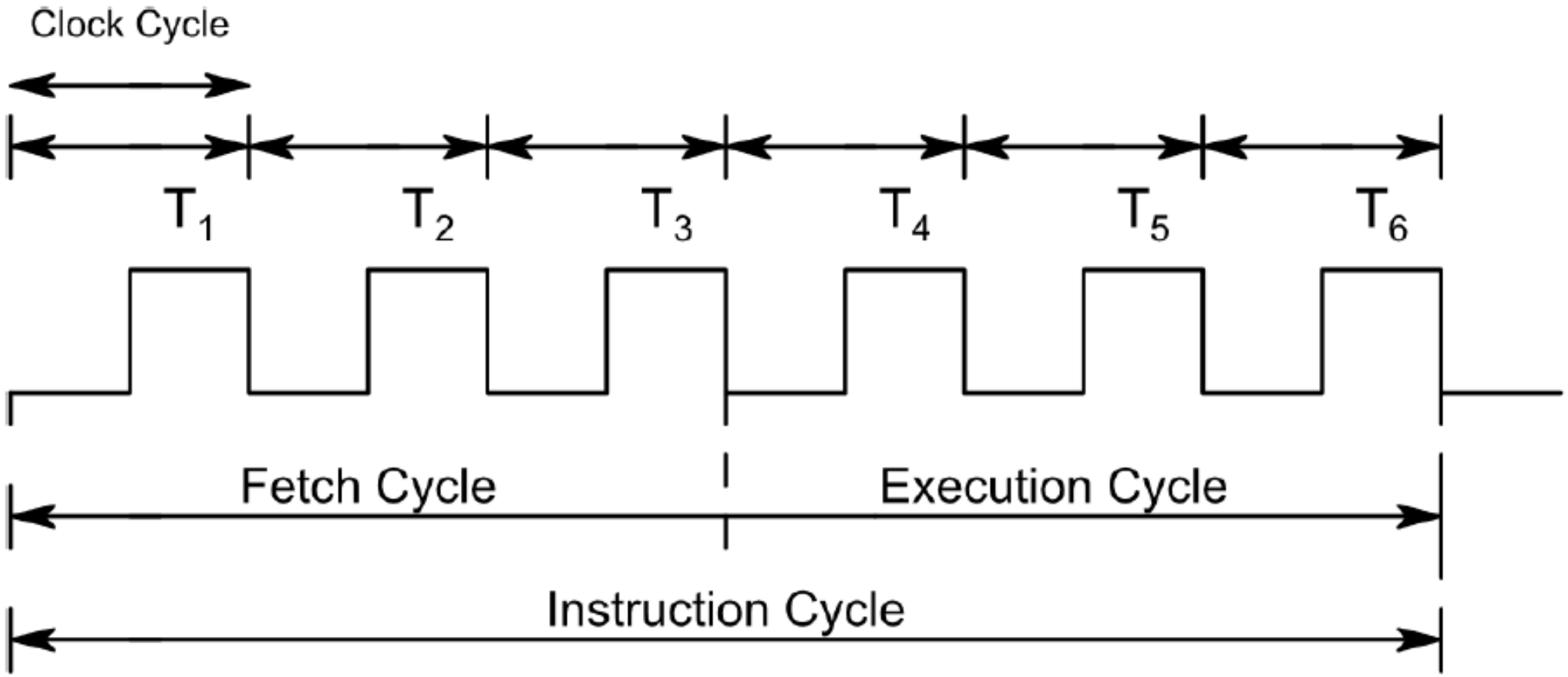
Instruction cycle:

- Instruction contains in the program and is pointed by the program counter.
- It is first moved to the **instruction register** and is decoded in binary form and stored as an instruction in the memory.
- The computer takes a certain period to complete this task i.e., instruction fetching, decoding and executing on the basis of clock speed. Such a time period is called **Instruction Cycle(IC)** and consists two cycles namely fetch and Execute cycle.

  IC=FC+EC

- **In the Fetch Cycle(FC),** the central processing unit **obtains the instruction code from the memory** for its execution. Once the instruction code is fetched from memory, it is then executed.
- The **Execution Cycle(EC)** consists the calculating the address of the operands, fetching them, performing operations on them and finally outputting the result to a specified location.

Instruction cycle:



Instruction description and format:

- An instruction manipulates the data and a sequence of instructions constitutes a program.
- Generally, each *instruction has two parts*: one is the *task to be performed*, called the **operation code** (Op-Code) field, and the second is the *data to be operated on*, called the **operand** or address field.
- The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or an 8-bit (or 16-bit) address.
- The Op-Code field specifies how data is to be manipulated and address field indicates the address of a data item.
- For example:

  ADD R_1, R_0

  "Op-code" "address"

  - Here R0 is the **source register** and R1 is the **destination register**.
  - The instruction adds the contents of R0 with the content of R1 and stores result in R1.

Instruction description and format:

- 8085 A can handle at the maximum of **256 instructions (2^8)**(*246 instructions are used*) . The sheet which contains all these instructions with their *hex code, mnemonics, descriptions and function* is called an **instruction sheet**. Depending on the number of address specified in instruction sheet, ***the instruction format can be classified into the categories.***
  - I. **One address format** (1-byte instruction):
    - ✓ Here 1 byte will be Op-code and operand will be default.
    - ✓ E.g. ADD B, MOV A, B
  - II. **Two address format** (2-byte instruction):
    - ✓ Here first byte will be Op-code and second byte will be the operand/data.
    - ✓ E.g. IN 40H, MVI A, 40H
  - **III. Three address format** (3-byte instruction):
    - ✓ Here first byte will be Op-code, second and third byte will be operands/data i.e. 2nd byte- lower order data. 3rd byte – higher order data
    - ✓ E.g. LXI B, 4050 H
- Micro operation specifies the transfer of data into or out of a register.

Data Format of 8085 Microprocessor

- The operand is an another name for data. It may appear in different forms :
  - **Addresses**
  - **Numbers/Logical data and**
  - **Characters**
- **Addresses :** The address is a 16-bit unsigned integer ,number used to refer a memory location.
- **Numbers/Data :** The 8085 supports following numeric data types.
- **Signed Integer :** A signed integer number is either a positive number or a negative number. In 8085, 8-bits are assigned for signed integer, in which most significant bit is used for sign and remaining seven bits are used for Sign bit 0 indicates positive number whereas sign bit 1 indicates negative number.
- **Unsigned Integer :** The 8085 microprocessor supports 8-bit unsigned integer.
- **BCD :** The term BCD number stands for binary coded decimal number. It uses ten digits from 0 through 9. The 8-bit register of 8085 can store two digit BCD
- **Characters :** The 8085 uses ASCII code to represent characters. It is a 7-bit alphanumeric code that represents decimal numbers, English alphabets, and other special characters.

Instruction and Data Format

- An instruction (instruction format) is a command to the microprocessor to perform a given task on a particular data. Each instruction (instruction format) is of two parts. One is to be performed, called the operation code or opcode and the second one is the data to be operated on, called the operand.
- Operands or data can be specified in different ways. It may include an 8-bit or 16-bit data, an internal register, a memory location, or 16-bit address. In some instructions, the operand is implicit.
- Instruction Word Size - The 8085 instruction set is of three groups according to word size:
  - One-word or one-byte instructions.
  - Two-word or two-byte instructions.
  - Three-word or three-byte instructions.

Instruction and Data Format

- In the 8085 microprocessor, byte and words are synonymous because it is an 8-bit microprocessor. But, instructions are commonly referred to in terms of bytes rather than words.
- **One-byte instructions**
  - A one-byte instruction includes an opcode and an operand in the same byte. Operand(s) are internal registers and are in the instruction in the codes. If there is no numeral present in the instruction then that instruction will be of one-byte.
  - For example; MOV C, A, RAL, and ADD B, etc.
- **Two-byte instructions**
  - In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. The source operand is a data byte and immediately following the opcode. If an 8-bit numeral is present in the instruction then that instruction will be of two-byte. Here, the numeral may be a data or an address.
  - For example, in MVI A, 35H and IN 29H, etc. In a two-byte instruction, the first byte will be the opcode and the second byte will be for the numeral present in the instruction.

Instruction and Data Format

- **Three-byte instructions**
  - In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit operand. The second byte is the low-order operand and the third byte is the high-order operand. If a 16-bit numeral is present in the instruction then that instruction will be of three-byte. Here, the numeral may be a data or an address.
  - For example, in LXI H,3500H and STA 2500H, etc.

Addressing Modes in 8085

❖There are **five addressing** modes in 8085.

1. **Immediate Addressing Mode:** An immediate is transferred directly to the register.
   - E.g.: **MVI A, 30H** (30H is copied into the register A)
   - **MVI B,40H**(40H is copied into the register B).
2. **Register Addressing Mode:** Data is copied from one register to another register.
   - E.g.: **MOV B, A** (the content of A is copied into the register B)
   - **MOV A, C** (the content of C is copied into the register A).
3. **Direct Addressing Mode:** Data is directly copied from the given address to the register.
   - E.g.: **LDA 3000H** (The content at the location 3000H is copied to the register A).
4. **Indirect Addressing Mode:** The data is transferred from the address pointed by the data in a register to other register.
   - E.g.: **MOV A, M** (data is transferred from the memory location pointed by the register to the accumulator).
5. **Implied Addressing Mode:** This mode doesn't require any operand. The data is specified by opcode itself.
   - E.g.: **RAL, CMP**

Addressing Modes in 8085

- Instructions are **command** to perform a certain task in microprocessor.
- The instruction consists of **op-code** and **data** called **operand**. The operand may be the **source only**, **destination only** or **both of them**.
- In these instructions, the **source** can be a register, a memory or an input port.
- Similarly, **destination** can be a register, a memory location, or an output port.
- The various **format (way)** of specifying the **operands** are called **addressing mode**. So addressing mode specifies where the operands are located rather than their nature.
- The 8085 has 5 addressing modes:
  1. Direct addressing mode
  2. Register Direct addressing mode
  3. Register Indirect addressing mode
  4. Immediate addressing mode
  5. Implied or Inherent addressing mode

Direct addressing mode

- The instruction using this mode specifies the effective address as part of instruction.
- The instruction size either 2-bytes or 3-bytes with first byte op-code followed by 1 or 2 bytes of address of data.
- E. g.:
  - LDA 9500H A ← [9500]
  - IN 80H A ← [80]
- This type of addressing is called absolute addressing.

Register Direct addressing mode

- This mode specifies the register or register pair that contains the data.
- E g.:
  - MOV A, B → Here register B contains data rather than address of the data.
- Other examples are: ADD, XCHG etc.

Register Indirect addressing mode

- In this mode, the address part of the instruction specifies the memory whose contents are the address of the operand.
- So in this type of addressing mode, it is the address of the address rather than address itself. (One operand is register)
- E.g.
  - MOV R, M; MOV M, R; STAX, LDAX etc.
  - STAX B; B= 95 C =00
    [9500] ← A

Immediate addressing mode

- In this mode, the operand position is the immediate data.
- For 8-bit data, instruction size is 2 bytes and for 16-bit data, instruction size is 3 bytes.
- E.g.:
  - MVI A, 32H
  - LXI B, 4567H

Implied or Inherent addressing mode

- The instructions of this mode do not have operands.
- E.g.:
  - **NOP**: No operation
  - **HLT**: Halt
  - **EI**: Enable interrupt
  - **DI**: Disable interrupt
  - **CMA**: Complement Accumulator

Addressing Modes in Microprocessor 8085

❖ **Addressing mode: The Various formats of specifying the operands are called addressing modes.**

✓ Immediate Addressing Mode

❑ **In this Addressing mode, data (1byte/2bytes) specified in instruction is directly transferred into register.**

Example:

```
MVI C,15H        ; 15H will gets transferred to C
LXI B,1000H      ; 1000H will gets transferred to BC pair
```

✓ Register Addressing Mode

❑ **In this Addressing mode, data is specified in registers.**

Example:

```
MOV A,C          ; C will gets copied into A
INR D            ; D will get incremented by 1
```

✓ Direct Addressing Mode

❑ **In this Addressing mode, address of operand is specified in instruction.**

Example:

```
LDA 1000H        ; A will get data from 1000H address
STA 1000H        ; A will get stored at address 1000H
```

✓ Indirect Addressing Mode

❑ **In this Addressing mode, the address of operand is stored in registers.**

Example:

```
STAX B           ; Store the content of A into address
                   pointed by BC Pair
INR M            ; increment the content located by
                   memory HL pair
```

✓ Implied/Implicit Addressing Mode

❑ **In this Addressing mode, the operand is implied in instruction.**

Example:

```
STC              ; Set the carry flag
CMC              ; Complements the carry flag
```

Addressing Modes in 8085

Q) What do you mean by addressing modes in microprocessor? Explain all the addressing modes of 8085 up with suitable example for each.

Classification of Instruction

- An instruction is a binary pattern designed inside a microprocessor to perform a specific function (task). The entire group of instructions called the instruction set.
- The 8085 instruction set can be classified in to **5- different groups**.
  - **Data transfer group:** The instructions which are used to transfer data from one register to another register or register to memory and vice-versa.
  - **Arithmetic group:** The instructions which perform arithmetic operations such as addition, subtraction, increment, decrement etc.
  - **Logical group:** The instructions which perform logical operations such as AND, OR, XOR, COMPARE etc.
  - **Branching group:** The instructions which are used for looping and branching are called branching instructions like jump, call etc.
  - **Miscellaneous group:** The instructions relating to stack operation, controlling purposes such as interrupt operations are fall under miscellaneous group including machine control like HLT, NOP.

Data Transfer group Instructions

- It is the longest group of instructions in 8085.
- This group of instruction **copy** data from a **source location** to **destination location** without modifying the contents of the source.
- The transfer of data may be between the registers or between register and memory or between an I/O device and accumulator.
- None of these instructions changes the flag.
- The instructions of this group are:

Data Transfer Instructions in 8085

✓ MOV $R_{Destination}, R_{Source}$ [1 Byte]

- It will transfer Data of Source register into Destination register.

Example:

```
MOV A,B          ; B will get copied into A
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Register	None	1	4

✓ MOV $R_{Destination}, M$ [1 Byte]

- It will transfer Data of Memory (Located by HL pair) to Destination Register.

Example:

```
MOV A,M          ; A = [HL]
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Indirect	None	2	7

✓ MVI $R_{Destination}$,8 bits Data [2 Byte]

- It will transfer Data (Given in instruction) into Destination register.

Example:

```
MVI A,55H        ; A = 55H
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Immediate	None	2	7

✓ MVI M, 8 bits Data [2 Byte]

- It will transfer Data (Given in Instruction) into Memory located by HL pair.

Example:

```
MVI M,55H        ; [HL] = 55H
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Immediate	None	3	10

Data Transfer Instructions in 8085

✓ LXI Rp,16 bits Data [3 Byte]

- It will transfer Data (2 Bytes) into register pair (BC, DE or HL).

Example:

LXI B,2350H ; B=23H & C=50H

Addressing Mode	Flags Effected	Machine Cycles	T States
Immediate	None	3	10

✓ LDA 16 bits Address [3 Byte]

- It will transfer data of given address into Accumulator register.

Example:

LDA 5000H ; A = [5000H]

Addressing Mode	Flags Effected	Machine Cycles	T States
Direct	None	4	13

✓ STA 16 bits Address [3 Byte]

- It will transfer Data of Accumulator into given Address of instruction.

Example:

STA A,5000H ; [5000H]=A

Addressing Mode	Flags Effected	Machine Cycles	T States
Direct	None	4	13

✓ LHLD 16 bits Address [3 Byte]

- It will load HL pair from given Memory Address given by instruction.

Example:

LHLD 5000H ; L=[5000H], H=[5001H]

Addressing Mode	Flags Effected	Machine Cycles	T States
Direct	None	5	16

Data Transfer Instructions in 8085

✓ SHLD 16 bits Address [3 Byte]

- It will store HL pair into memory location specified in the instruction

Example:

SHLD 5000H ; [5000H]=L & [5001H]=H

Addressing Mode	Flags Effected	Machine Cycles	T States
Direct	None	5	16

✓ LDAX Rp [1 Byte]

- Accumulator is loaded with the content located by memory location pointed by Register Pair.

Example:

LDAX B ; A = [BC]

Addressing Mode	Flags Effected	Machine Cycles	T States
Indirect	None	2	7

✓ STAX Rp [1 Byte]

- Accumulator is stored at the location pointed by Register Pair.

Example:

STAX B ; [BC]=A

Addressing Mode	Flags Effected	Machine Cycles	T States
Indirect	None	2	7

✓ PCHL [1 Byte]

- Program counter gets the content of HL Pair. It will create branch in the program.

Example:

PCHL ; PC=HL

Addressing Mode	Flags Effected	Machine Cycles	T States
Register	None	1	6

Data Transfer Instructions in 8085

✓ SPHL [1 Byte]

- Stack Pointer gets the content of HL register pair. It will relocate the stack.

Example:

```
SPHL            ; SP=HL
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Register	None	1	6

✓ XCHG [1 Byte]

- It will exchange the content of HL and DE pair.

Example:

```
XCHG            ; HL ⟷ DE
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Register	None	1	4

✓ XTHL [1 Byte]

- It will exchange content of HL pair with the content of SP and SP+1.

Example:

```
XTHL            ; L⟷[SP] and H⟷[SP+1]
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Indirect	None	5	16

Arithmetic group Instructions

- The 8085 microprocessor performs various arithmetic operations such as addition, subtraction, increment and decrement.
- These arithmetic operations have the following mnemonics.

Arithmetic Instructions in 8085

✓ ADD R [1 Byte]

- It will add Register R with A and answer will get store in Accumulator.

Example:

ADD C ; A=Ai+C

Addressing Mode	Flags Effected	Machine Cycles	T States
Register	ALL	1	4

✓ ADD M [1 Byte]

- It will add A with content pointed by memory (HL pair) and stores in Accumulator

Example:

ADD M ; A = Ai+[HL]

Addressing Mode	Flags Effected	Machine Cycles	T States
Indirect	All	2	7

✓ ADI 8bits Data [2 Byte]

- It will add A with 8 bits data given in instruction and answer will get store in Accumulator.

Example:

ADI 55H ; A=Ai+55H

Addressing Mode	Flags Effected	Machine Cycles	T States
Immediate	All	2	7

✓ ADC R [1 Byte]

- It will add A with given register R along with Carry and answer will store in Accumulator.

Example:

ADC C ; A=Ai+C+Carry

Addressing Mode	Flags Effected	Machine Cycles	T States
Register	All	1	4

Arithmetic Instructions in 8085

✓ ADC M [1 Byte]

❑ It will add A with data pointed by HL pair along with carry and answer will get store in Accumulator.

Example:

ADC M ; A=Ai+[HL]+Carry

Addressing Mode	Flags Effected	Machine Cycles	T States
Indirect	ALL	2	7

✓ ACI 8bits data [2 Byte]

❑ It will add A with data given instruction along with carry and stores in Accumulator.

Example:

ACI 55H ; A = Ai+55H+Carry

Addressing Mode	Flags Effected	Machine Cycles	T States
Immediate	All	2	7

✓ Similarly, Subtraction also done with

- ✓ SUB R [1 Byte]
- ✓ SUB M [1 Byte]
- ✓ SUI 8bits data [2 Byte]
- ✓ SBB R [1 Byte]
- ✓ SBB M [1 Byte]
- ✓ SBI 8bits data [2 Byte]

Arithmetic Instructions in 8085

✓ INR R [1 Byte]

- It will increment the value of given register by 1. Answer will be there in same register.

Example:

```
INR C          ; C=Ci+1
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Register	All except C	1	4

✓ INR M [1 Byte]

- It will increment the value of content pointed by HL pair and load it at same location.

Example:

```
INR M          ; [HL]=[HL]i+1
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Indirect	All except C	3	10

✓ INX Rp [1 Byte]

- It will increment the value of register pair by 1 and stores in same register pair.

Example:

```
INX H          ; HL = HLi + 1
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Register	None	1	6

✓ Similarly, Decrement also done with

- ✓ DCR R [1 Byte]
- ✓ DCR M [1 Byte]
- ✓ DCX Rp [1 Byte]

Arithmetic Instructions in 8085

✓ DAD Rp [1 Byte]

- ❑ It will add HL pair with given register pair and answer will get store with HL pair.

Example:

DAD B ; HL = HLi + BC

Addressing Mode	Flags Effected	Machine Cycles	T States
Register	C	3	10

- ❑ It will execute opcode fetch and two machine cycle will be Bus idle.

✓ DAA [1 Byte] (Decimal Adjust Accumulator)

- ❑ This instruction is used to get answer in BCD form.
- ❑ It will do addition inform of decimal form.
- ❑ It is implied addressing mode and work strictly with register A.
- ❑ If Lower Nibble > 9 or AC = 1 then add 06H.
- ❑ If Higher Nibble > 9 or CY = 1 then add 60H.

Addressing Mode	Flags Effected	Machine Cycles	T States
Implied	All	1	4

- ❑ Let say we want to add (55)bcd and (55)bcd.
- ❑ To do that in 8085 execute this instructions
- ❑ MVI A,55H
- ❑ MVI B,55H
- ❑ ADD B
- ❑ DAA

55H + 55H = AAH

AAH + 06H + 60H = 110H

Addition operation in 8085

- 8085 performs addition with 8-bit binary numbers and stores the result in accumulator. If the sum is greater than 8-bits (FFH), it sets the carry flag.

E.g.

MVI A, 93H
MVI C, B7H
ADD C

```
  1 0 1 1 0 1 1 1        B7
+ 1 0 0 1 0 0 1 1      + 93
-----------------      ----
1 0 1 0 0 1 0 1 0      1 4A
↑                      ↑
CY                     CY
```

Subtraction operation in 8085

- 8085 performs subtraction operation by using 2's complement and the steps used are:
  1) Converts the subtrahend (the number to be subtracted) into its 1's complement.
  2) Adds 1 to 1's complement to obtain 2's complement of the subtrahend.
  3) Adds 2's complement to the minuend (the contents of the accumulator).
  4) Complements the carry flag.

E.g.

```
MVI A, 97H
MVI B, 65H
SUB B
```

```
65H: 0 1 1 0 0 1 0 1
1's complement of 65H : 1 0 0 1 1 0 1 0
                                    +1
2's Complement of 65H:   1 0 0 1 1 0 1 1
97H:                  + 1 0 0 1 0 1 1 1
                    1 0 0 1 1 0 0 1 0
                      |_____32_____|
                      ^
                      CY
```

Complement carry =0 ∴ **A=32 CY=0**

Subtraction operation in 8085

B=97H, A=65H

97H: 1 0 0 1 0 1 1 1

MVI B, 65H

SUB B

1's complement of 97H : 0 1 1 0 1 0 0 0

+1

2's Complement of 97H: 0 1 1 0 1 0 0 1

65H: +0 1 1 0 0 1 0 1

0 1 1 0 0 1 1 1 0

(Result in 2's complement form)

CY

CY=1, A= CE: 1 1 0 0 1 1 1 1 0

1's complement: 0 0 1 1 0 0 0 0 1

2's complement: 0 0 1 1 0 0 0 1 0

32

Logical group Instructions

- A microprocessor is basically a programmable logic chip.
- It can perform all the logic functions of the hardwired logic through its instruction set.
- The 8085 instruction set includes such logic functions as AND, OR, XOR and NOT (Complement).
- The following features hold true for all logic instructions:
  - The instructions implicitly assume that the accumulator is one of the operands.
  - All instructions reset (clear) carry flag except for complement where flag remain unchanged.
  - They modify Z, P & S flags according to the data conditions of the result.
  - Place the result in the accumulator.
  - They do not affect the contents of the operand register.

Logical Instructions in 8085

✓ ANA R [1 Byte]

- It will perform Logic AND operation of A with register R and answer will gets store in Accumulator.

Example:

```
ANA C          ; A = Ai AND C
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Register	ALL	1	4

✓ ANA M [1 Byte]

- It will perform Logic AND operation of A with Memory M (data pointed by HL) and answer will gets store in A.

Example:

```
ANA M          ; A = Ai AND [HL]
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Indirect	All	2	7

✓ ANI 8bits Data [2 Byte]

- It will perform Logic AND operation of A with data and answer will gets store in Accumulator.
- Example:

```
ANI 55H        ; A = Ai AND 55H
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Immediate	All	2	7

✓ Similarly Logic OR and XOR can be done by:

- ✓ ORA R [1 Byte]
- ✓ ORA M [1 Byte]
- ✓ ORI 8bits data [2 Byte]
- ✓ XRA R [1 Byte]
- ✓ XRA M [1 Byte]
- ✓ XRI 8bits data [2 Byte]

Logical Instructions in 8085

✓ CMP R [1 Byte]

❑ It will compare Accumulator with Register R.

Example:

CMP C ; It will compare A and C.

Conclusion	Zero Flag	Carry Flag
A>C	0	0
A<C	0	1
A=C	1	0

Addressing Mode	Flags Effected	Machine Cycles	T States
Register	ALL	1	4

✓ Similarly, Comparison can done by:

✓ CMP M [1 Byte]

✓ CPI 8bits data [2 Byte]

Logical Instructions in 8085

✓ STC [1 Byte]

- ❑ It will Set carry flag to 1.

Example:

STC ; Carry Flag = 1

Addressing Mode	Flags Effected	Machine Cycles	T States
Implied	C	1	4

✓ CMC [1 Byte]

- ❑ Complement Carry Flag

Example:

CMC ; $CY = \overline{CY_i}$

Addressing Mode	Flags Effected	Machine Cycles	T States
Implied	C	1	4

✓ CMA [1 Byte]

- ❑ It will compliment Accumulator.
- ❑ It will do 1's complement of Accumulator.

Example:

CMA ; A = 1's Complement of Ai

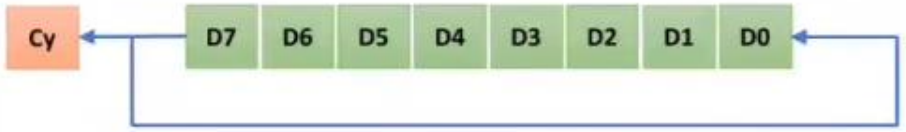
Addressing Mode	Flags Effected	Machine Cycles	T States
Implied	None	1	4

- ✓ CMA is working as logic NOT operation.
- ✓ So, AND followed by NOT will make NAND.
- ✓ So, OR followed by NOT will make NOR.
- ✓ So, XOR followed by NOT will make XNOR.

Logical Instructions in 8085

✓ RLC [1 Byte] (Rotate Accumulator Left without Carry)

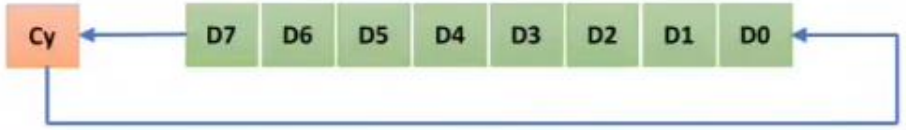
- ❑ The content of Accumulator rotated Left by 1.
- ❑ MSB goes to Carry and LSB after Instruction.



Addressing Mode	Flags Effected	Machine Cycles	T States
Implied	Carry	1	4

✓ RAL [1 Byte] (Rotate Accumulator Left Through Carry)

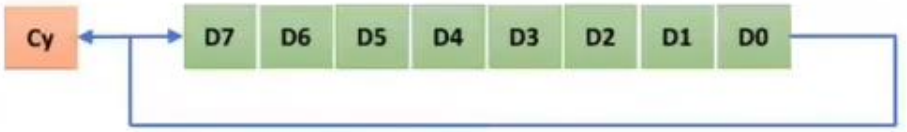
- ❑ The content of Accumulator rotated Left by 1.
- ❑ MSB goes to Carry and Carry goes to LSB.



Addressing Mode	Flags Effected	Machine Cycles	T States
Implied	Carry	1	4

✓ RRC [1 Byte] (Rotate Accumulator Right without Carry)

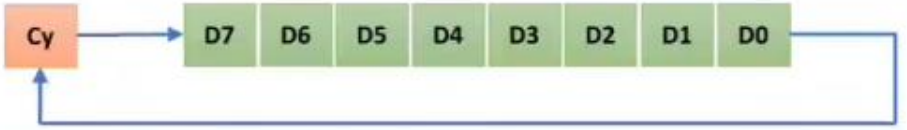
- ❑ The content of Accumulator rotated Right by 1.
- ❑ MSB goes to Carry and LSB after Instruction.



Addressing Mode	Flags Effected	Machine Cycles	T States
Implied	Carry	1	4

✓ RAR [1 Byte] (Rotate Accumulator Right Through Carry)

- ❑ The content of Accumulator rotated Right by 1.
- ❑ MSB goes to Carry and LSB after Instruction.



Addressing Mode	Flags Effected	Machine Cycles	T States
Implied	Carry	1	4

Binary to Gray Code Conversion: 8085 ALP

LDA C050H

MOV B, A

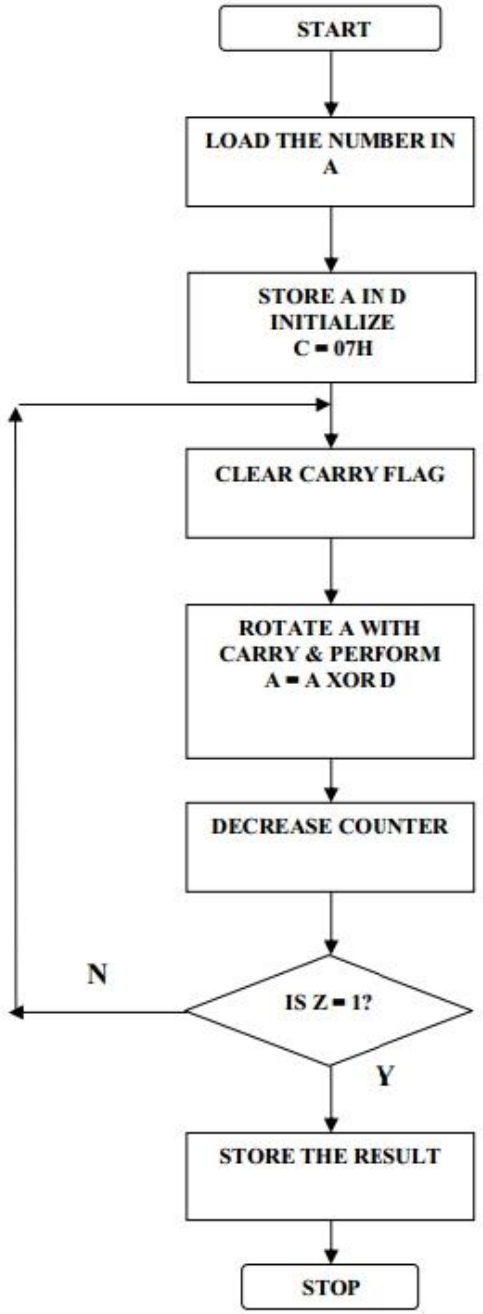
RAR

XRA B

STA C051H

HLT

Gray to Binary Code Conversion: 8085 ALP



Gray to Binary Code Conversion: 8085 ALP

```
LDA C050H
MOV D, A
MVI C, 07H
Up: STC
CMC
RAR
XRA D
DCR C
JNZ Up
STA C051H
HLT
```

```
LDA C050
MOV D,A
MVI C,07

UP:  RAR
XRA D
DCR C
JNZ UP
STA C051
HLT
```

Branching group Instructions

- The branching instructions instruct the microprocessor to go to a **different memory location** and the microprocessor continues executing machine codes **from that new location**.
- The branching instructions are the most powerful instructions because they allow the microprocessor to change the sequence of a program, either unconditionally or under certain test conditions.
- The branching instruction code categorized in following three groups:
  - **Jump instructions**
  - **Call and return instruction**
  - **Restart instruction**

Branch Control Instructions in 8085

✓ JMP 16 bits Address [3 Byte]

- ❑ It will load PC to given 16bits address in Instruction.
- ❑ It is unconditional Jump instruction.

Example:

```
JMP 5000H          ; PC = 5000H
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Immediate	None	3	10

✓ JZ 16 bits Address [3 Byte]

- ❑ It will load PC to given 16bits address in Instruction If Zero flag is set to '1'.

Example:

```
JZ 5000H           ; If Zero Flag = 1, PC = 5000H
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Immediate	None	2/3	7/10

✓ Similarly, There are other conditional Jump Instructions.

- ❑ JZ 16 bits Address [3 Bytes]
- ❑ JNZ 16 bits Address [3 Bytes]
- ❑ JC 16 bits Address [3 Bytes]
- ❑ JNC 16 bits Address [3 Bytes]
- ❑ JPO 16 bits Address [3 Bytes]
- ❑ JPE 16 bits Address [3 Bytes]
- ❑ JP 16 bits Address [3 Bytes]
- ❑ JM 16 bits Address [3 Bytes]

Branch Control Instructions in 8085

✓ CALL 16 bits Address [3 Byte]

- ❑ It will load PC to given 16bits address in Instruction.
- ❑ It is also loading current PC on stack.

Example:

CALL 5000H ; PC = 5000H, Old PC on Stack

Addressing Mode	Flags Effected	Machine Cycles	T States
Immediate	None	5	18

✓ CZ 16 bits Address [3 Byte]

- ❑ It will CALL if Zero flag is set to '1'.
- ❑ It is conditional CALL instruction.

Example:

CZ 5000H ; If Zero Flag = 1, PC = 5000H, Old PC on Stack

Addressing Mode	Flags Effected	Machine Cycles	T States
Immediate	None	2/5	9/18

✓ Similarly, There are other conditional CALL Instructions.

- ❑ CZ 16 bits Address [3 Bytes]
- ❑ CNZ 16 bits Address [3 Bytes]
- ❑ CC 16 bits Address [3 Bytes]
- ❑ CNC 16 bits Address [3 Bytes]
- ❑ CPO 16 bits Address [3 Bytes]
- ❑ CPE 16 bits Address [3 Bytes]
- ❑ CP 16 bits Address [3 Bytes]
- ❑ CM 16 bits Address [3 Bytes]

✓ Similarly, RST instructions are there for Interrupt service.

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

Branch Control Instructions in 8085

✓ RET [1 Byte]

- ❑ It will retrieve PC from Stack, It is used to get back from subroutine. It is used with RST and CALL.

Example:

```
RET        ; PCL = [SP], PCH = [SP+1]
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Indirect	None	3	10

✓ RZ [1 Byte]

- ❑ It will retrieve PC from Stack if Z = 1, It is used to get back from subroutine. It is used with RST and CALL.

Example:

```
RZ         ; PCL = [SP], PCH = [SP+1]
```

Addressing Mode	Flags Effected	Machine Cycles	T States
Indirect	None	1/3	6/12

✓ Similarly, There are other conditional RET Instructions.

- ❑ RZ [1 Byte]
- ❑ RNZ [1 Byte]
- ❑ RC [1 Byte]
- ❑ RNC [1 Byte]
- ❑ RPO [1 Byte]
- ❑ RPE [1 Byte]
- ❑ RP [1 Byte]
- ❑ RM [1 Byte]

8085 program to add two 8 bit numbers with Carry

```
MVI C, 00H
LDA 4300H
MOV B, A
LDA 4301H
ADD B
JNC loop
INR C
Loop: STA 4302
MOV A, C
STA 4302
MOV A, C
STA 4303
HLT
```

```
LXI H, 2000H
MOV A,  M
INX H
MOV B, M
MVI C, 00H
ADD B
JNC  down
INR C
INX H
down: MOV M, A
INX H
MOV M, C
HLT
```

Packing of BCD Numbers

Input Data

Memory Address(H)	Data(H)
C200	04
C201	09

RESULT

Memory Address(H)	Data(H)
C300	94

PROGRAM:

```
LDA C201H
RLC
RLC
RLC
RLC
ANI F0H
MOV C, A
LDA C200H
ADD C
STA C300H
HLT
```

Unpacking of BCD Numbers

Input Data

Memory Address(H)	Data(H)
C200	58

RESULT

Memory Address(H)	Data(H)
C300	08
C301	05

PROGRAM:

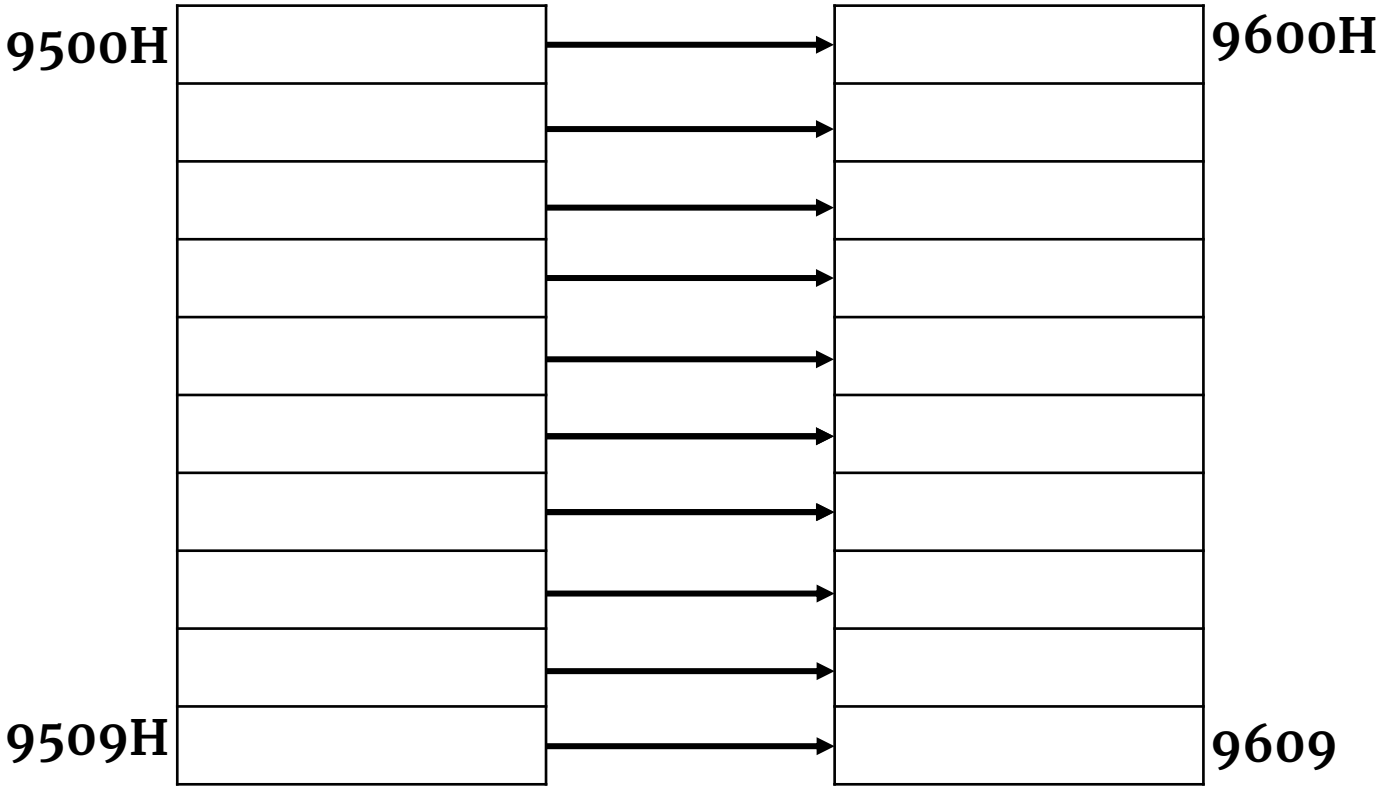
```
LDA C200H
ANI F0H
RLC
RLC
RLC
RLC
STA C300H
LDA C200H
ANI 0FH
STA C301H
HLT
```

Assignment

Q) Explain the instructions that fall in data transfer, arithmetic and logical groups with example. Show how the flags are affected by each instruction. [10]

ALP with 8085

Q1) WAP to move 10 bytes of data from starting address 9500 H to 9600H.



ALP with 8085

Q1) WAP to move 10 bytes of data from starting address 9500 H to 9600H.

Solution:

```
MVI C, 0AH
LXI H, 9500H
LXI D, 9600H
up: MOV A, M
STAX D ; Store the contents of accumulator to register pair.
INX H ; Increment the register pair HL by 1
INX D ; Increment the register pair DE by 1
DCR C
JNZ up
HLT
```

ALP with 8085

Q2) Write to transfer 30 data starting from 8500 to 9500H if data is odd else store 00H.

Solution:

```
      MVI B, 1EH
      LXI H, 8500H
      LXI D, 9500H
L2:   MOV A, M
      ANI 01H
      JNZ L1 ; If data is odd then go to L1.
      MVI A, 00H
      JMP L3
L1:   MOV A, M
L3:   STAX D
      INX D
      INX H
      DCR B
      JNZ L2
      HLT
```

```
      MVI C,1E
      LXI H,2500
      LXI D,2600

UP:   MOV A,M
      MOV B,A
      RRC
      JC DOWN
      MVI B,00

DOWN: MOV A,B
      STAX D
      INX H
      INX D
      DCR C
      JNZ UP
      HLT
```

Miscellaneous group Instructions

- This group of instructions are also called **Stack Operation, I/O and Machine Control Instructions group.**

Stack in 8085

- The **stack** in an 8085 can be described as a reserved area of the memory in the R/W memory where we can store **temporary information**.
- It is a **shared resource** as it can be shared by the **microprocessor** and the **programmer**.
- **Programmers** use the stack to **store data** and the **microprocessors** use the stack to **execute subroutines**.
- The 8085 has a **16-bit** register known as the **Stack Pointer**.
- The function of the **stack pointer** is to hold the **starting address** of the stack. This address can be decided by the **programmer**.

Stack in 8085

- The stack operates on the **Last In, First Out (LIFO)** principle.
- The location of the most recent data on the stack is known as the **TOP of the stack**.
- The stack pointer always points to the top of the stack.
- Contents can be stored in the stack using the **PUSH** instruction and can restore the contents by using the instruction **POP.**

Stack in 8085

MNEMONIC	DESCRIPTION
LXI SP, 16-bit	Load the stack pointer register with a 16-bit address.
PUSH R_p	Copies the contents of the specified register pair on the stack
POP R_p	Copies the contents of the top two memory locations of the stack into the specified register pair.

PUSH and POP Operation in 8085

1. **PUSH Rp/PSW**
   This is a 1-byte instruction. This instruction copies the contents of the specified register pair on the stack as described below:
   - The stack pointer is decremented and the contents of the higher-order register are copied to the location shown by the stack pointer register.
   - The stack pointer is again decremented and the contents of the low-order register are copied to that location.

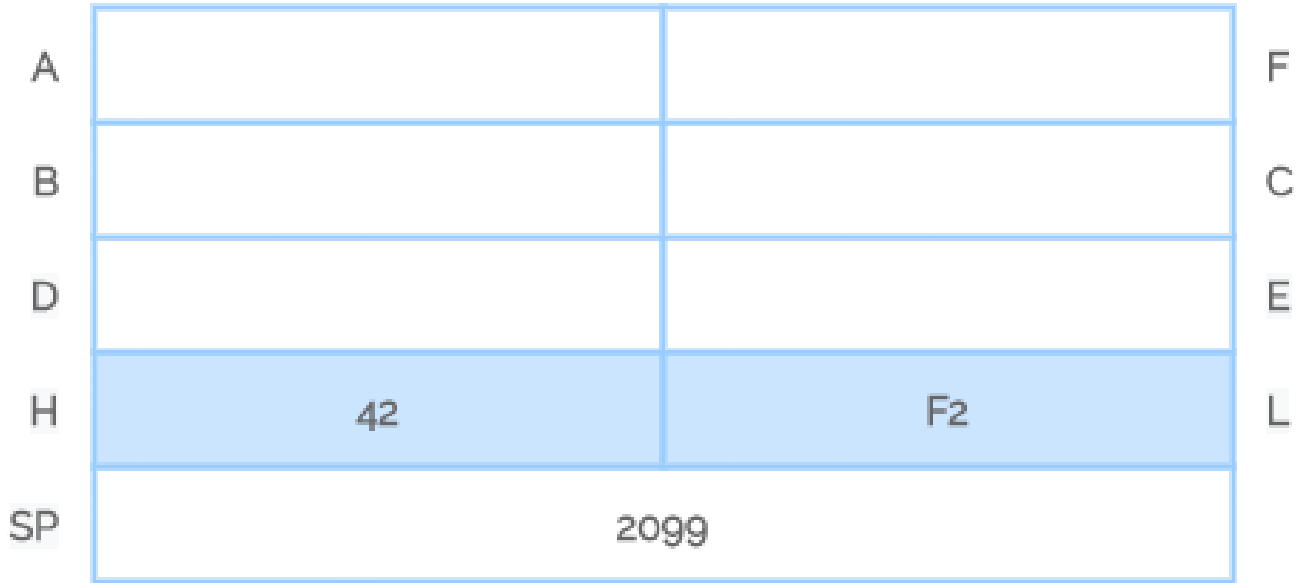
2. **POP Rp/PSW**
   This is a 1-byte instruction. This instruction copies the contents of the top two memory locations of the stack into the specified register pair.
   - First, the contents of the memory location indicated by the stack pointer register are copied into the low-order register and then the stack pointer register is incremented by 1.
   - The contents of the next memory location are copied into the high-order register and the stack pointer register is again incremented by 1.

PUSH and POP Operation in 8085

- **Example:**

  LXI SP, *2099H*
  LXI H, *42F2H*
  PUSH H
  Delay Counter
  POP H
  HLT

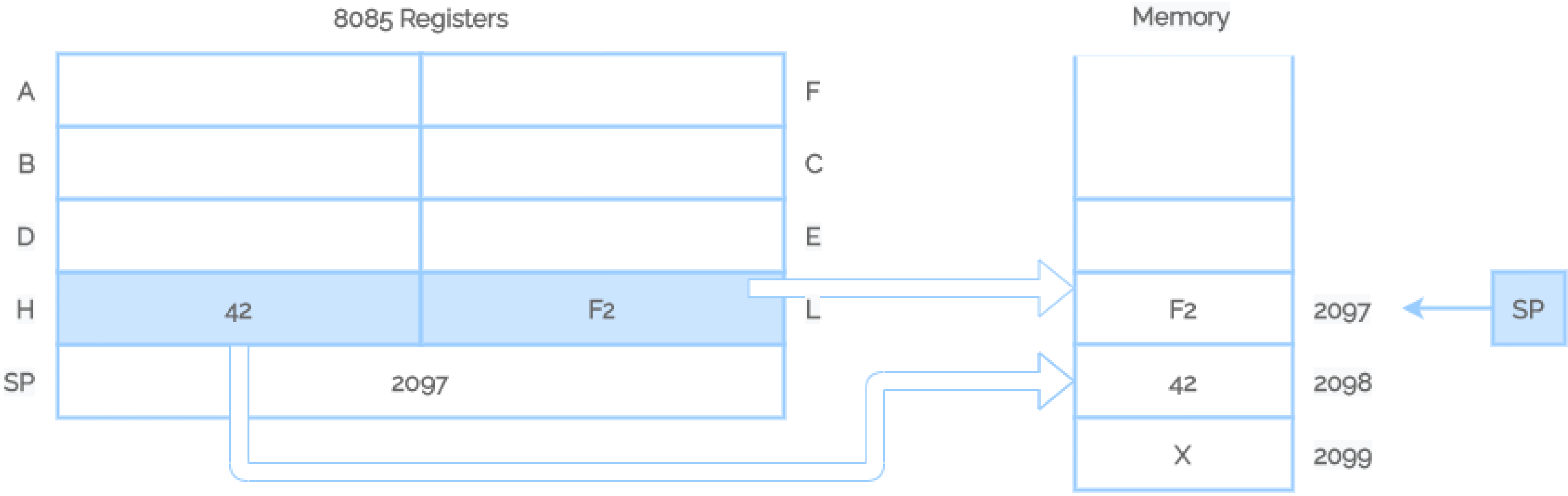
- The instruction **LXI SP, 2099H** will initialize the stack pointer with the address of **2099H**.
- **LXI H, 42F2H** will initialize or load **HL register pair** with **42F2H** data so **H = 42** and **L = F2**.



PUSH and POP Operation in 8085

- After the execution of **PUSH H** instruction the stack pointer is decreased by one to **2098H** and the contents of the H register are copied to memory location 2098H.
- The stack pointer is again decreased by one to 2097H and the contents of the L register are copied to memory location 2097H.

PUSH and POP Operation in 8085

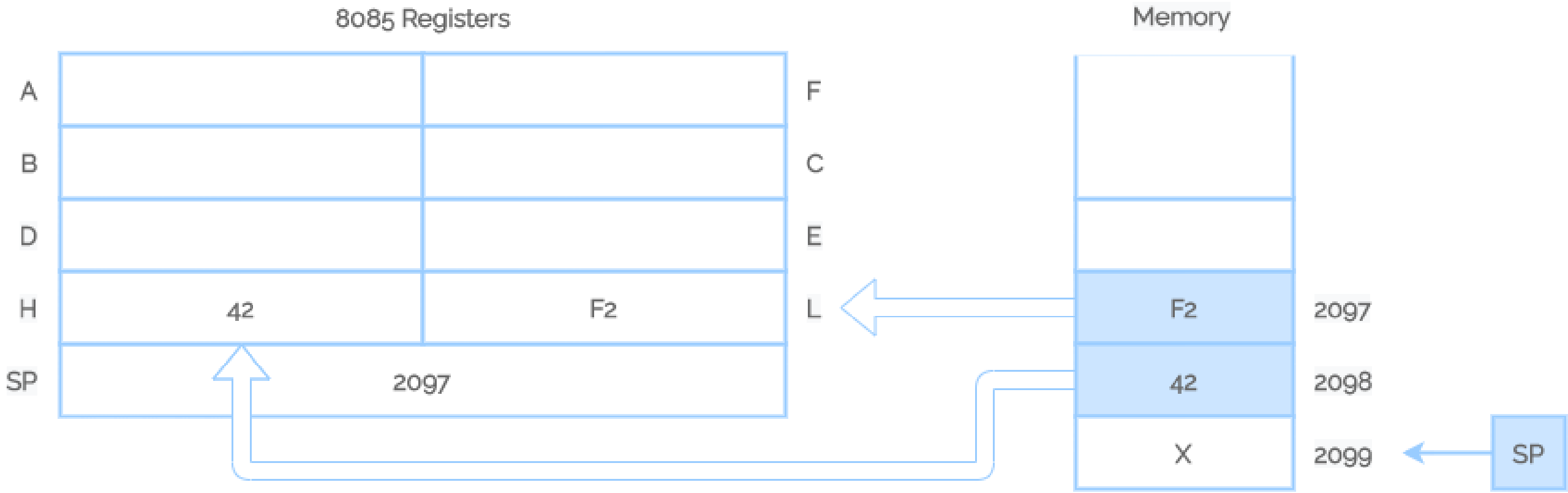


Contents of Stack and Registers After **PUSH Operation**

PUSH and POP Operation in 8085

- After the execution of **POP H** instruction, the contents of the top of the stack location shown by the stack pointer are copied in the **L** register and the stack pointer is increased by one to **2098H**.
- The contents of the top of the stack are copied in the **H** register and the stack pointer is increased by one.
- The contents of the memory locations **2097H** and **2098H** are not destroyed until some other data bytes are stored in these locations.

PUSH and POP Operation in 8085



Contents of Stack and Registers After **POP Operation**

Stack in 8085

Basics of Stack in 8085

- Stack is R/W memory interfaced with 8085.
- To operate with stack memory, 8085 has stack pointer.
- Stack pointer hold address of top of Stack.
- Using PUSH and POP instructions, we can use stack.
- It works as per LAST IN FIRST OUT.

PUSH B [Opcode C5H] Instruction execution in 8085

PC = 8000H
8085
SP = 1000H

Add	Data
8000H	C5H

Add	Data
0FFEH	57H
0FFFH	32H
1000H	
1001H	
1002H	

B	32H	57H	C

SP = 0FFEH

POP H [Opcode E1H] Instruction execution in 8085

PC = 8000H
8085
SP = 1000H

Add	Data
8000H	E1H

Add	Data
0FFEH	
0FFFH	
1000H	57H
1001H	32H
1002H	

H	32H	57H	L

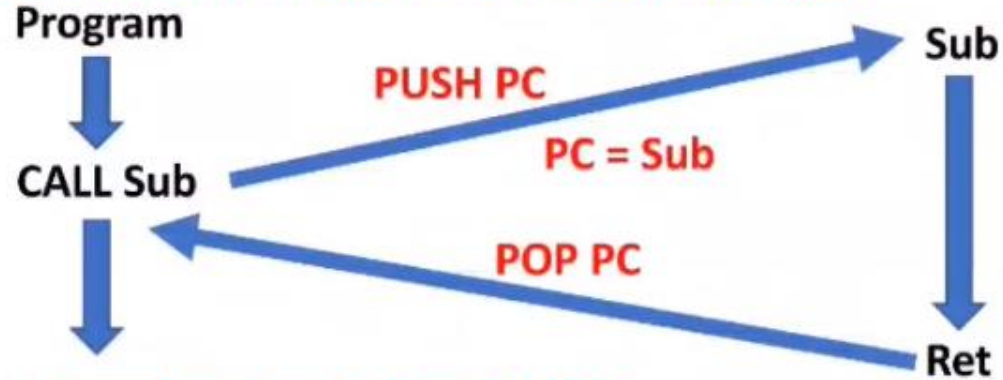
SP = 1002H

Uses of Stack in 8085

1. To store data in 8085

- Microprocessor has limited numbers of registers available for programming. [A, B, C, D, E, H & L]
- So, we push can PUSH data on stack to perform complicated task and again we can take it back by POP instruction.

2. To store return address in CALL instruction



3. To read and write flags in 8085

- PUSH PSW
- POP B

A	32H	95H	F

B	32H	95H	C

- PUSH B
- POP PSW

B	32H	95H	C

A	32H	95H	F

ALP Example of Stack Operation

E.g.:

```
LXI SP, 1FFFH
LXI H, 9320H
LXI B, 4732H
LXI D, ABCDH
MVI A, 34H
PUSH H
PUSH B
PUSH D
PUSH PSW
POP H
POP B
POP D
POP PSW
HLT
```

BEORE EXECUTION

H = 93 L = 20
B = 47 C = 32
D = AB E = CD
A = 34 F = 10

AFTER EXECUTION

H = 34 L = 10
B =AB C = CD
D = 47 E = 32
A = 93 F = 20

Note: **STACK Works in LIFO (Last In First Out) manner.**

Miscellaneous group Instructions

Interrupt related Instructions;

- **SIM** – Set Interrupt Mask
- **RIM** – Read Interrupt Mask
- **DI** – Disable Interrupt
- **EI** – Enable Interrupt

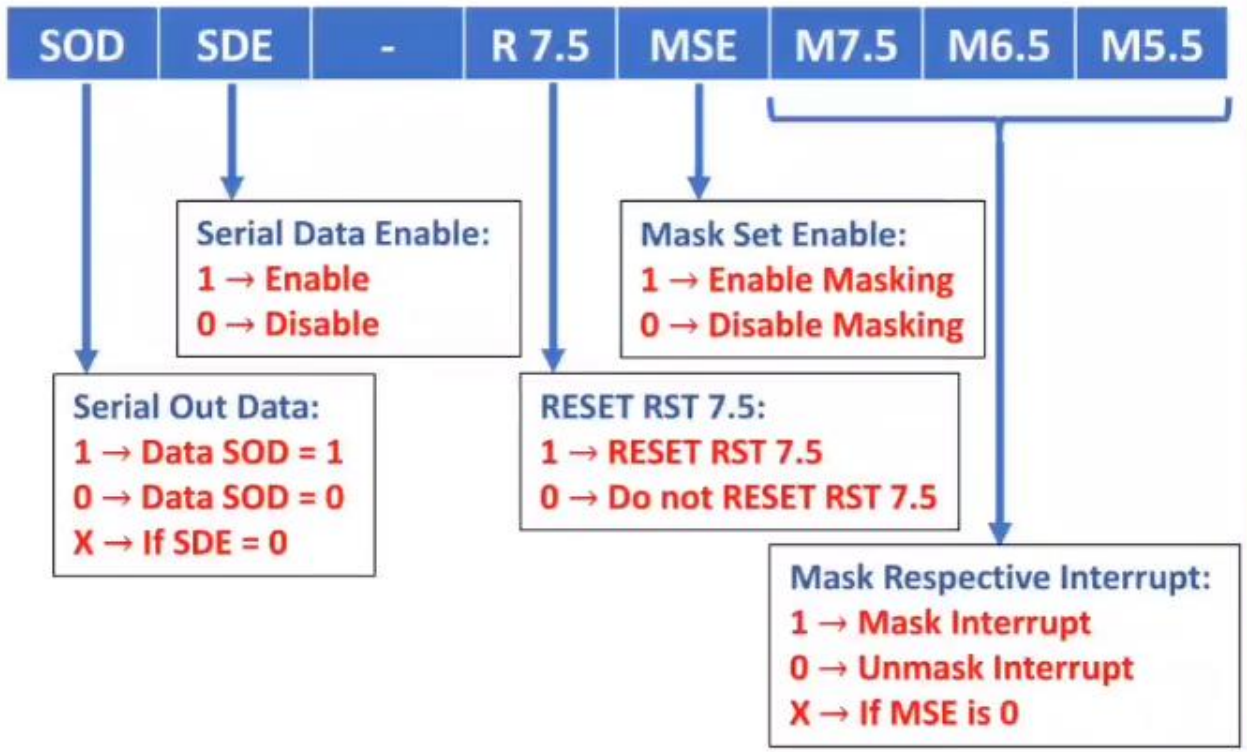
SIM and RIM Instruction in 8085

✓ SIM Instruction in 8085

- ❑ SIM is Set Interrupt Mask.
- ❑ It is operated with Accumulator. We load A here.
- ❑ Example :

```
MVI A, C1H ; SIM command is 1100 0001
SIM
```

❖ SIM Command

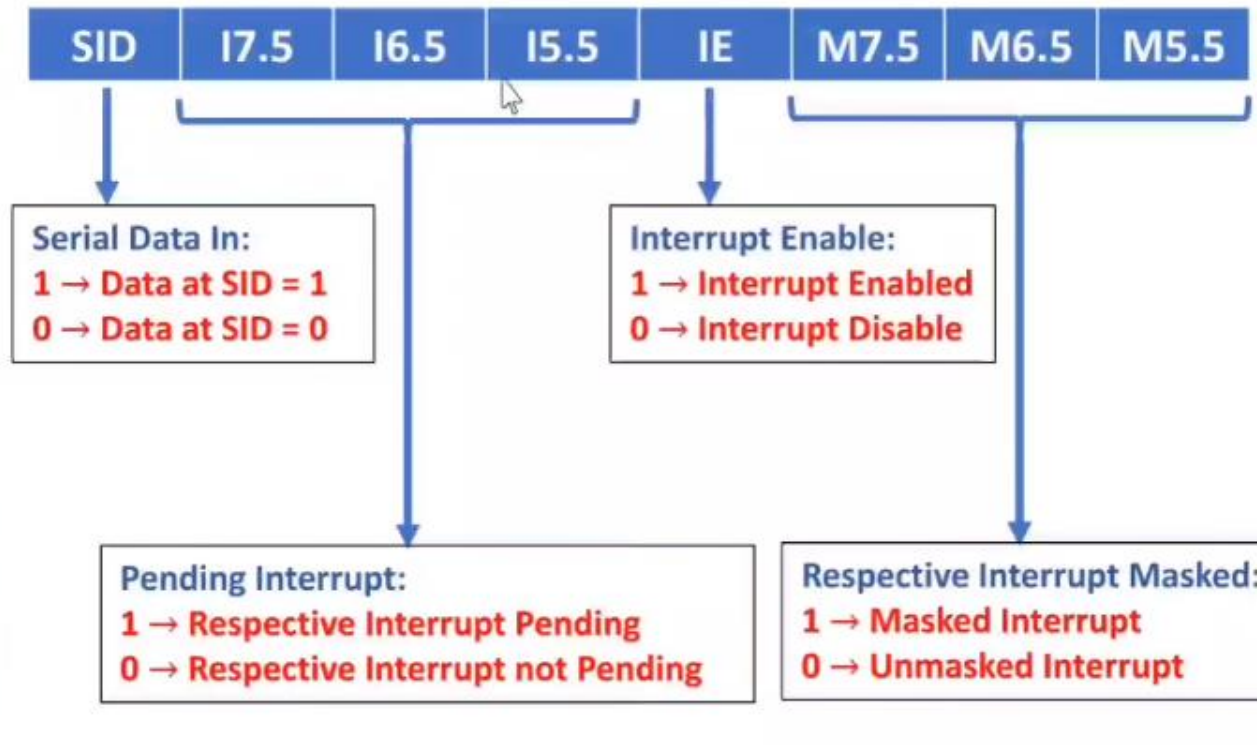


✓ RIM Instruction in 8085

- ❑ RIM is Read Interrupt Mask.
- ❑ It is operated with Accumulator. We read A here.
- ❑ Example :

```
RIM
If A = 88H, RIM command is 1000 1000
```

❖ RIM Command



Miscellaneous group Instructions

Interrupt related Instructions;

1. **DI:** Disable Interrupts

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
DI	None	1	1	4	F3

- After execution of this instruction the **Interrupt Enable** flipflop is reset and all the interrupts except the **TRAP** are disabled.
- No flags are affected.
- This instruction is commonly used when the execution of a code sequence cannot be interrupted.
- For example, in critical time delays, this instruction is used at the beginning of the program and the interrupts are enabled at the end of the program.
- The 8085 **TRAP** cannot be **disabled**.

Miscellaneous group Instructions

Interrupt related Instructions;

2. **EI:** Enable Interrupts

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
EI	None	1	1	4	FB

- After execution of this instruction the **Interrupt Enable** flipflop is set and all the interrupts are **enabled**.
- No flags are affected.
- After a system reset or the acknowledgement of an interrupt, the Interrupt Enable flip-flop is reset, thus disabling the interrupts.
- This instruction is necessary to re-enable the interrupts (except TRAP).

Miscellaneous group Instructions

Machine Control Instructions;

- **NOP –** No Operation
- **HLT –** Halt the program

Miscellaneous group Instructions

❖HLT: Halt and Enter Wait State

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
HLT	None	1	2 or more	5 or more	76

- ❑ The MPU finishes the current instruction and halts(stops) any further execution
- ❑ The **contents** of the **registers** are **unaffected** during HLT state.
- ❑ An interrupt or reset is necessary to exit from the Halt state.
- ❑ **No flags** are affected.

❖NOP: No Operation

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
NOP	None	1	1	4	00

- ❑ No operation is performed
- ❑ Instruction is fetched and decoded; however, no operation is executed.
- ❑ The instruction is used to fill in time delays or to delete and insert instructions while troubleshooting.
- ❑ It does not affect the registers and flags

Miscellaneous group Instructions

Peripheral Input/Output Instructions;

✓ The 8085 microprocessor has two instructions for data transfer between the processor and I/O device: **IN** and **OUT**.

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

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
IN	8-bit port address	2	3	10	DB

- ❑ The contents of the input port(00H-FFH) designated in the operand are read and loaded in to the accumulator.
- ❑ E.g.: **IN 40H;** A ← [40H]

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

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
OUT	8-bit port address	2	3	10	D3

- ❑ The contents of the accumulator are copied into the output port specified by the operand.
- ❑ E.g.: **OUT 01H;** [01H] ← A

ALP with 8085

Q) WAP to sort in ascending order for 10 bytes from 1120H.

Solution:

```
START:  LXI H, 1120H
        MVI D, 00H
        MVI C, 09H

L2:     MOV A, M
        INX H
        CMP M
        JC L1 ; if A<M
        MOV B, M
        MOV M, A
        DCX H
        MOV M, B
        INX H
        MVI D, 01H

L1:     DCR C
        JNZ L2
        MOV A, D
        RRC
        JC START
        HLT
```

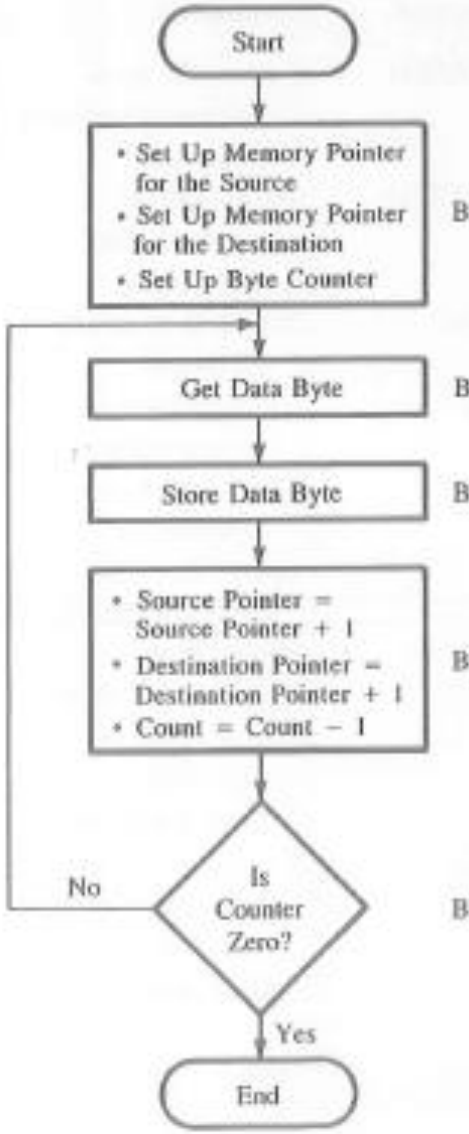
Note: *For sorting in Descending order use **JNC** instead **JC**.*

ALP with 8085

PROBLEM STATEMENT

Sixteen bytes of data are stored in memory locations at XX50H to XX5FH. Transfer the entire block of data to new memory locations starting at XX70H.

ALP with 8085



	Mnemonics	Steps
Block 1	LXI H,XX50H	1. Set up HL as a pointer for the source memory
	LXI D,XX70H	Set up DE as a pointer for the destination memory
	MVI B,10H	Set up B as byte counter
Block 2	NEXT: MOV A,M	2. Get data byte from the source memory
Block 7	STAX D	3. Store the data byte in destination memory
Block 5	INX H INX D DCR B	4. Get ready to transfer next byte
Block 6	JNZ NEXT	5. Go back to get next byte if byte counter ≠0
	HLT	

FIGURE 7.8
Flowchart for Block Transfer of Data Bytes

ALP with 8085

PROGRAM

Memory Address HI-LO	Hex Code	Label	Instructions Opcode	Operand	Comments
XX00	21	START:	LXI	H,XX50H	;Set up HL as a
01	50				; pointer for source
02	XX				; memory
03	11		LXI	D,XX70H	;Set up DE as
04	70				; a pointer for
05	XX				; destination
06	06		MVI	B,10H	;Set up B to count
07	10				; 16 bytes
08	7E	NEXT:	MOV	A,M	;Get data byte from
					; source memory
09	12		STAX	D	;Store data byte at
					; destination
0A	23		INX	H	;Point HL to next
					; source location
0B	13		INX	D	;Point DE to
					; next destination
0C	05		DCR	B	;One transfer is
					; complete,
					; decrement count
0D	C2		JNZ	NEXT	;If counter is not 0,
0E	08				; go back to transfer
0F	XX				; next byte
10	76		HLT		;End of program
XX50	37				;Data
↓	↓				
XX5F	98				

ALP with 8085

Memory locations 2050H and 2051H contain 3FH and 42H, respectively, and register pair DE contains 856FH. Write instructions to exchange the contents of DE with the contents of the memory locations.

Example 10.3

Memory

Before Instructions: D 85 E 6F

3F	2050
42	2051

Instructions

Machine Code	Mnemonics					
2A	LHLD 2050H				3F	2050
50					42	2051
20		H	42	3F L		
EB	XCHG	D	42	3F E	3F	2050
		H	85	6F L	42	2051
22	SHLD 2050H				6F	2050
50					85	2051
20		H	85	6F L		

ALP with 8085

Example 10.4 Registers BC contain 2793H, and registers DE contain 3182H. Write instructions to add these two 16-bit numbers, and place the sum in memory locations 2050H and 2051H.

Before instructions:

B	27	93	C	
D	31	82	E	

Instructions

MOV A,C	A	93	F		93H
ADD E	A	15 CY = 1	F	+	82H
MOV L,A	H	15	L		1/15H
MOV A,B					27H
ADC D				+	31H
MOV H,A	H	59 15	L		59H
SHLD 2050H					

ALP with 8085

Example 10.5

Registers BC contain 8538H and registers DE contain 62A5H. Write instructions to subtract the contents of DE from the contents of BC, and place the result in BC.

Instructions

```
MOV A,C      (B)  85 | 38  (C)
SUB E               |  -
MOV C,A      (D)  62 | A5  (E)
MOV A,B           -1 | 1/93
SBB D        (B)  22 | 93  (C)
MOV B,A
```

ALP with 8085

Write instructions to display the contents of the stack pointer register at output ports. Example 10.6

Instructions

```
LXI H,0000H     ;Clear HL
DAD SP          ;Place the stack pointer contents in HL
MOV A,H         ;Place high-order address of the stack pointer in the accumulator
OUT PORT1
MOV A,L         ;Place low-order address of the stack pointer in the accumulator
OUT PORT2
```

The instruction DAD SP adds the contents of the stack pointer register to the HL register pair, which is already cleared. This is the only instruction in the 8085 that enables the programmer to examine the contents of the stack pointer register.

BCD-TO-BINARY(HEX) CONVERSION

- The conversion of a BCD number into its binary equivalent employs the principle of *positional weighting* in a given number
- For example: 72_{10} = 7*10 + 2
- The digit 7 represents 70, based on its second position from the right . Therefore, converting 72_{BCD} into its binary equivalent requires multiplying the second digit by 10 and adding the first digit.
- Converting a 2-digit BCD number into its binary equivalent requires the following steps:
  - Step-1: Separate an 8-bit packed BCD number into two 4-bit unpacked BCD digits: BCD_1 and BCD_2.
  - Step-2: Convert each digit into its binary value according to its position
  - Add both binary numbers to obtain the binary equivalent of the BCD number.

BCD–TO–BINARY(HEX) CONVERSION

❖**EXAMPLE:** Convert 72_{BCD} into its binary equivalent.

Solution:

Step 1: 0111 0010 → 0000 0010 Unpacked BCD_1.

→ 0000 0111 Unpacked BCD_2.

Step 2: Multiply BCD_2 by 10 (7 x 10)

Step 3: Add BCD_1 to the answer in Step 2

BCD-TO-BINARY(HEX) CONVERSION

❖**PROGRAM**

```
LDA 3000H
MOV B, A
ANI 0FH
MOV C, A
MOV A, B
ANI F0H
RRC
RRC
RRC
RRC
MOV D, A
XRA A
MVI E, 0AH
UP: ADD D
DCR E
JNZ UP
ADD C
STA 3001H
HLT
```

BINARY (HEX)–TO–BCD CONVERSION

- The conversion of binary to BCD is performed by dividing the number by the powers of ten; the division is performed by the subtraction method.

  For example, assume the binary number is

  $1111\ \ 1111_2$ (FFH) = 255_{10}

- To represent this number in BCD requires twelve bits or three BCD digits, labeled here as BCD_3 (MSB), BCD_2, and BCD_3 (LSB).

  = 0 0 1 0 0 1 0 1 0 1 01

  BCD_3 BCD_2 BCD_1

BINARY-TO-BCD CONVERSION

❖***PROGRAM:***

```
        LDA 5000H
        MVI B,00H
        MOV C,B

HUND:   CPI 64H
        JC TENS
        SUI 64H
        INR B
        JMP HUND

TENS:   CPI 0AH
        JC UNITS
        SUI 0AH
        INR C
        JMP TENS
UNITS:  STA 5001H ;storing units place BCD value at 5001H
        MOV A,C
        STA 5002H ;storing tens place BCD value at 5002H
        MOV A,B
        STA 5003H ;storing hundreds place BCD value at
5003H

        HLT
```

BINARY-TO-ASCII HEX CODE CONVERSION

- Concept:
  - If the ASCII Character value is **<0AH; add 30H** to that value
  - If the ASCII Character value is **≥ 0AH**; **add 37H** to that value

- We Know; ASCII characters(0-9,A-F) equivalent HEX value is

ALP:

```
LDA 5000H
CPI OAH
JC skip
ADI 07H
skip: ADI 30H
STA 5001H
HLT
```

ASCII Character	HEX Value
0	30H
. . .	. . .
9	39H
A	41H
. . .	. . .
F	46H

ASCII HEX–TO– BINARY CODE CONVERSION

❖Concept:

- ❑ Subtract **30H** from given HEX value
- ❑ If the HEX value is **<0AH;** store the result
- ❑ If the HEX value is **≥ 0AH**; **subtract 07H** to the previous result

ALP:

```
LDA 5000H
SUI 30H
CPI OAH
JC skip
SUI 07H
skip: STA 5001H
HLT
```

Counters and Time Delays in 8085

- Microprocessors perform different operations in sequence and one operation at a time. To complete an operation, some time is required.
- When some time delay is required between two operations, a Time Delay Loop in Microprocessor is used to provide it.
- Time delay can be generated using a **register** or a **register pair** or a **nested loop** or **NOP** instruction.
- Initially, a register is loaded with an **operand** or **number**, depending on the time delay required, and then the number is decremented until it reaches **zero** setting up a loop with a conditional jump instruction. So a conditional jump instruction is used in a delay loop to come out from the loop.
- Figure 4.9 shows the flowchart of time delay loop using **One register**.

Counters and Time Delays in 8085

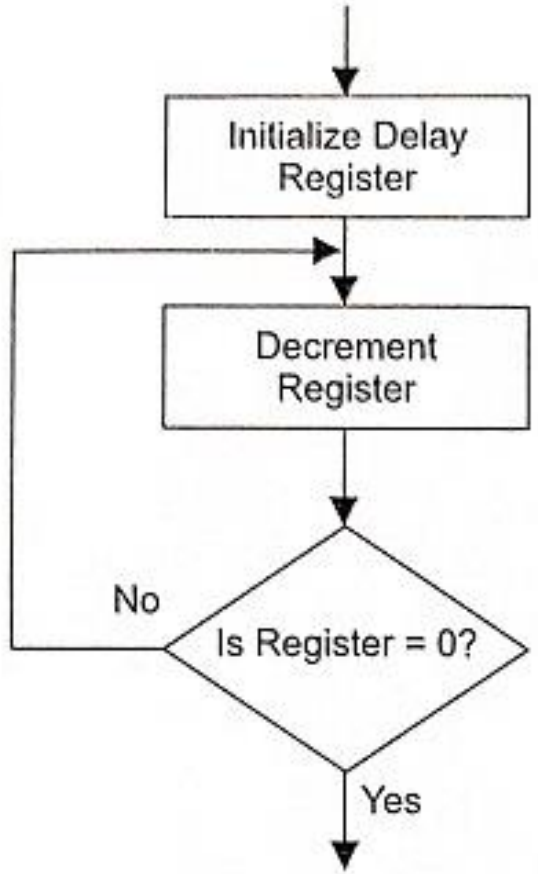


Fig. 4.9 *Flowchart for time delay using a register*

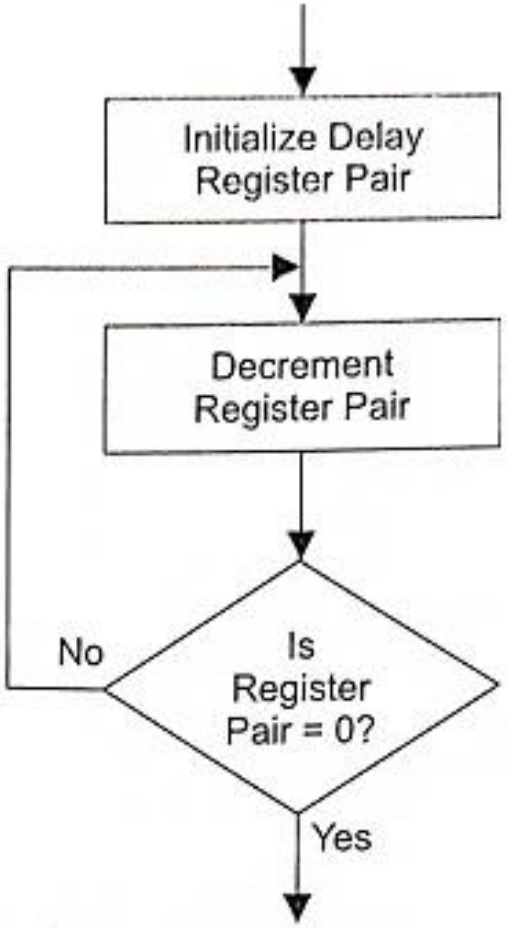


Fig. 4.10 *Flowchart for time delay using register pair*

Calculation of Time Delay Using NOP

Program	Time(T-State)
`Delay: NOP`	**4T**

$\therefore$ **Max. Time Delay(T_D)** = 4T
= 1.333 μ sec; if operating frequency(f) = **3 MHz**

Calculation of Time Delay Using a Register

Program	Time(T-state)
Delay: MVI B, Count LOOP: DCR B JNZ LOOP RET	**7T** **4T** **10T(if condⁿ true)/7T(false)** **10T**

$\therefore$ **Max. Time Delay(T_D)** = 7T + (4T +10T)*Count − 3T + 10T ; here Count is a 8-bit data
= 3584T; if Count = $FF_H = 255_{10}$
= 1.19 msec; if operating frequency(f) = **3 MHz**

Calculation of Time Delay Using Register Pair

Program	Time (T-States)
Delay: B, Count LOOP: DCX B MOV A,B ORA C JNZ LOOP RET	**10T** **6T** **4T** **4T** **10T (if true)/ 7T(if false)** **10T**

$\therefore$ **Max Time Delay(T_D)** = 10T + (6T+4T+4T+10T)*Count-3T + 10T ; here Count is a 16-bit data
= 1572857T; if Count = $FFFF_H$ = 65535_{10}
= 0.524 sec; if operating frequency(f) = **3 MHz**

Time Delay Using Two LOOPs(Nested Loop)

- The time delay can also be generated by using two loops as depicted in Fig. 4.11.
- The C register is used in inner loop and the B register is used in external loop.
- Here, both B and C registers are loaded with numbers. Then Register C is decremented until it becomes zero.
- When the content of Register C is zero, decrement Register B. If the content of Register B is not zero, load the Register C with initial value and repeat the process.

Time Delay Using Two LOOPs(Nested Loop)

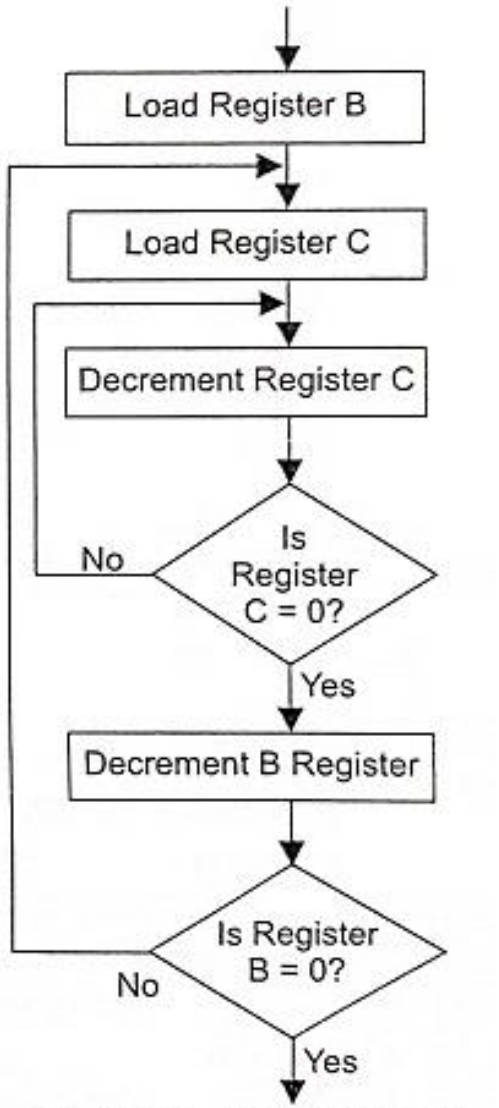


Fig. 4.11 *Flowchart for time delay using two loops*

Time Delay Using Two LOOPs(Nested Loop)

Program	Time (T-States)
Delay :MVI B,Count1 L1: MVI C,Count2 L2: DCR C JNZ L2 DCR B JNZ L1 RET	**7t** **7T** **4T** **10T (if true)/ 7T(if false)** **4T** **10T (if true)/ 7T(if false)** **10T**

$\therefore$ Time Delay(T_d) = 7T +[{7T + (4T + 10T)*Count2 – 3T} + 4T + 10T]*Count1 -3T + 10T
= 914954T ;if Count1 = Count2 = $FF_H = 255_{10}$
= 0.305sec; if operating frequency(f) = 3 MHz

8085 Practice Programs

1. Write a program in 8-bit Microprocessor to store 68h, B3h, C Oh, and 11h in the memory location starting from 3000h. Move these data and store in the memory location starting from 3200h.
2. Write a program in 8-bit Microprocessor to multiply two 16 bits numbers (ABCDh and 1234h) and store in the memory location starting from 3000h.
3. Write a program in 8-bit Microprocessor to store 60h, BAh, 7Ch and 10h in the memory location starting from 2000h. add these data and store the result in 3000h and carry flag in 5001h. explain all the steps.
4. Write a program in 8-bit Microprocessor to store 45h, A0h, B5h and 15h in the memory location starting from 4000h. Add these data and store the result in 5000h and carry flag in 5001h.
5. Ten number of 8-bit data stored at memory location 6000H. Write a program for 8085 microprocessor to calculate the sum of odd numbers and store the sum at 6010H. (The sum may exceed 8-bits).

8085 Practice Programs

6. Write an assembly language program to find the smallest number in an array using 8 bit microprocessor. (Assume appropriate array data and address where minimum array size of 15 should be considered.) **[2075]**
7. Write an assembly language program to find the greatest number in an array using 8-bit microprocessor. (Assume appropriate array data and address where minimum array size of 20 should be considered.) **[2076]**
8. Explain LXI and CMP instruction. Write an assembly language program for 8-bit microprocessor to divide 8 bit data stored in memory location 8050 by 8 bit data stored in 8051 and store the quotient in 8052 and remainder in 8053. **[2078]**
9. Write an assembly language program to find the largest number of an array using 8 bit microprocessor. (Assume appropriate array data and address where minimum array size of 15 should be considered). **[2079]**

Solutions

1. ***Solution:***

```
        LXI H,3000H     UP:     MOV A,M
        MVI M,68H               STAX D
        INX H                   INX H
        MVI M,B3H               INX D
        INX H                   DCR C
        MVI M,C0H               JNZ UP
        INX H                   HLT
        MVI M,11H
        MVI C,04H
        LXI H,3000H
        LXI D,3200H
```

Solutions

2. ***Solution:***

```
      LHLD 5000H
      SPHL
      LHLD 5002H
      XCHG
      LXI H, 0000H
      LXI B, 0000H
up:   DAD SP
      JNC skip
      INX B
skip: DCX D
      MOV A, E
      ORA D
      JNZ up
      SHLD 3000H
      MOV L, C
      MOV H, B
      SHLD 3002H
      HLT
```

Solutions

3. ***Solution:***

```
        LXI H,2000H
        MVI M,60H
        INX H
        MVI M,BAH
        INX H
        MVI M,7CH
        INX H
        MVI M,10H
        MVI C,04H
        MVI D,00H
        LXI H,2000H
        MOV A,M
        INX H
        ADD M
        JNC SKIP
        INR D
SKIP:   STA 3000H
        MOV A,D
        STA 5001H
        HLT
```

Solutions

4. ***Solution:***

```
        LXI H,4000H
        MVI M,45H
        INX H
        MVI M,A0H
        INX H
        MVI M,B5H
        INX H
        MVI M,15H
        MVI C,04H
        MVI D,00H
        LXI H,4000H
        XRA A
UP:     ADD M
        JNC SKIP
        INR D
SKIP:   INX H
        DCR C
        JNZ UP
        STA 5000H
        MOV A,D
        STA 5001H
        HLT
```

Solutions

5. ***Solution:***

```
        MVI E,00   ; carry
        MVI B,00   ; first operand
        MVI C,0A   ; size of array
        LXI H,6000

UP:     MOV A,M
        RRC
        JNC SKIP
        MOV A,B    ; no. is odd
        ADD M
        JNC SKIP1
        INR E

SKIP1:  MOV B,A

SKIP:   INX H      ;no. is even
        DCR C
        JNZ UP
        MOV A,B
        STA 6010   // store sum
        MOV A,E
        STA 6011   // store carry
        HLT
```

Note: To find the sum of even numbers from an array put 'JC SKIP' instead 'JNC SKIP' in line no. 6 of above ALP.

Solutions

6. ***Solution:***

```
        MVI D,0FH  ; size of an array/counter=15₁₀
        MVI A,FFH  ; initializing A with FFH
        LXI H,5000H      ; memory pointer at 5000H

UP:     CMP M      ; compare M with A
        JC SKIP    ; If carry sets, go to SKIP label
        MOV A,M    ; else update A with the content of M

SKIP:   INX H      ; Increase Memory address
        DCR D      ; Decrease counter
        JNZ UP     ; If Z != 0, go to UP label
        STA 5100H  ; else store A to 5100H
        HLT
```

Note: To find the greatest number from an array, initialize Acc. with 00 H and put 'JNC SKIP' instead 'JC SKIP' in line no. 6 of above ALP.

Solutions

7. ***Solution:***

```
        MVI D,14H  ; size of an array/counter=20₁₀
        MVI A,00H  ; initializing A with 00H
        LXI H,5000H      ; memory pointer at 5000H

UP:     CMP M      ; compare M with A
        JNC SKIP   ; If carry resets, go to SKIP label
        MOV A,M    ; else update A with the content of M

SKIP:   INX H      ; Increase Memory address
        DCR D      ; Decrease counter
        JNZ UP     ; If Z != 0, go to UP label
        STA 5100H  ; else store A to 5100H
        HLT
```

Solutions

8. ***Solution:***

```
; division of two 8-bit numbers
; dividend = [8050H] = reg. D
; deviser = [8051H] = reg. E
; [8052] <= Quotient = reg. C
; [8053] <= Reminder = reg. A
```

ALP:

```
        MVI C,00H  ; quotient
        LXI H,8050H
        MOV D,M    ; dividend
        INX H
        MOV E,M    ; divisor
        MOV A,D

UP:     CMP E
        JC SKIP    ; skip if dividend is
                   smaller than deviser
        SUB E
        INR C
        JMP UP

SKIP:   INX H
        MOV M,C
        INX H
        MOV M,A
        HLT
```

Solutions

9. ***Solution:***

```
        MVI D,0FH  ; size of an array/counter=15₁₀
        MVI A,00H  ; initializing A with 00H
        LXI H,5000H      ; memory pointer at 5000H

UP:     CMP M      ; compare M with A
        JNC SKIP   ; If carry resets, go to SKIP label
        MOV A,M    ; else update A with the content of M

SKIP:   INX H      ; Increase Memory address
        DCR D      ; Decrease counter
        JNZ UP     ; If Z != 0, go to UP label
        STA 5100H  ; else store A to 5100H
        HLT
```

8085 Opcode Sheet

	B	D	H	SP
LXI	1	11	21	31
INX	3	13	23	33
DCX	0B	1B	2B	3B
DAD	9	19	29	39
LDAX	0A	1A		
STAX	2	12		PSW
PUSH	C5	D5	E5	F5
POP	C1	D1	E1	F1

		C	NC	Z	NZ
JMP	C3	DA	D2	CA	C2
CALL	CD	DC	D4	CC	C4
RET	C9	D8	D0	C8	C0

	P	M	PE	PO
JMP	F2	FA	EA	E2
CALL	F4	FC	EC	E4
RET	F0	F8	E8	E0

MOV	B	C	D	E	H	L	M	A
B	40	41	42	43	44	45	46	47
C	48	49	4A	4B	4C	4D	4E	4F
D	50	51	52	53	54	55	56	57
E	58	59	5A	5B	5C	5D	5E	5F
H	60	61	62	63	64	65	66	67
L	68	69	6A	6B	6C	6D	6E	6F
M	70	71	72	73	74	75	76	77
A	78	79	7A	7B	7C	7D	7E	7F

SHLD	22
LHLD	2A
XCHG	EB
SPHL	F9
XTHL	E3
LDA	3A
STA	32
PCHL	E9

EI	FB	RRC	0F	CMA	2F		
DI	F3	RLC	07	STC	37	RIM	20
NOP	00	RAR	1F	CMC	3F	SIM	30
HLT	76	RAL	17	DAA	27		

R			S			T	
0	1	2	3	4	5	6	7
C7	CF	D7	DF	E7	EF	F7	FF

IN	DB
OUT	D3

WITH ACCUMULATOR							
ADI	C6	SUI	D6	ANI	E6	ORI	F6
ACI	CF	SBI	DF	XRI	EE	CPI	FE

ADD	80	81	82	83	84	85	86	87
ADC	88	89	8A	8B	8C	8D	8E	8F
SUB	90	91	92	93	94	95	96	97
SBB	98	99	9A	9B	9C	9D	9E	9F
ANA	A0	A1	A2	A3	A4	A5	A6	A7
XRA	A8	A9	AA	AB	AC	AD	AE	AF
ORA	B0	B1	B2	B3	B4	B5	B6	B7
CMP	B8	B9	BA	BB	BC	BD	BE	BF
INR	04	0C	14	1C	24	2C	34	3C
DCR	05	0D	15	1D	25	2D	35	3D
MVI	06	0E	16	1E	26	2E	36	3E
	B	C	D	E	H	L	M	A

For your reference of 8085 Microprocessor:

Visit YouTube playlist: Microprocessor and Microcontroller(lect 1 – lect 20) by Prof. Ajit Pal on YouTube channel Satish Kashyap

Link: https://www.youtube.com/playlist?list=PL0E131A78ABFBFDD0