Microprocessors & its Applications
CS-3305

Books and References

- Microprocessors and Programmed Logic

  By Kenneth .L . Short.
- Microprocessor Architecture, Programming and Applications with the 8085

  by Ramesh Gaonkar
- Microprocessors by B. Ram
- Microprocessors by Tokoheim
- Digital Computer Fundamentals

  by Paul Albert Malvino

Microprocessor basics

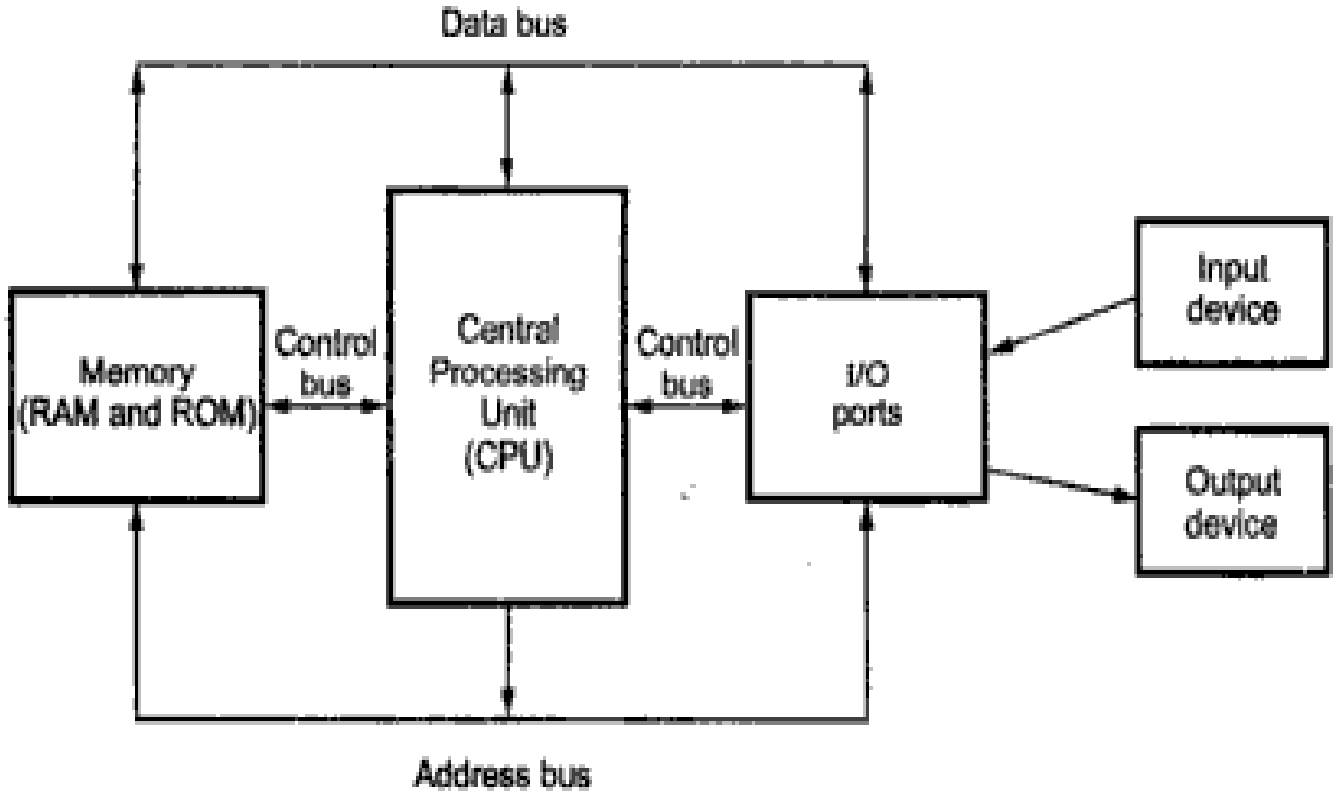
A **microprocessor** is a multipurpose, programmable, clock driven, register-based electronic device that reads binary instructions from a storage device called memory accepts binary data as input and processes data according to instructions, and provides result as output.

OR

When the CPU of a computer system is fabricated on a single chip it is termed as microprocessor.

- A **micro-computer** is a device which uses microprocessor as its central processing unit along with other associated peripherals like I/O and Memory devices.
- A **microcontroller** is a device which contains CPU, few MB of memory and few I/O devices all fabricated on a single chip and is used for specific tasks like washing machine control etc.

Block Diagram of a simple microcomputer system



Microprocessor Generations

- Microprocessors are categorized into five generations: **first, second, third, fourth, and fifth generations**. Their characteristics are described below:-

First Generation:-

- The microprocessors that were introduced in 1971 to 1972 were referred to as the first generation systems. First-generation microprocessors processed their instructions serially—they fetched the instruction, decoded it, then executed it. When an

- **Second Generation:-**
- By the late 1970s, enough transistors were available on the IC to give birth to the second generation of microprocessor sophistication: 16-bit arithmetic and pipelined instruction processing.
- Motorola's MC68000 microprocessor, introduced in 1979, is an example. Another example is Intel's 8080. This generation is defined by overlapped fetch, decode, and execute steps .As the first instruction is processed in the execution unit, the second instruction is decoded and the third

Third Generation:-

- The third generation, introduced in 1978, was represented by Intel's 8086 and the Zilog Z8000,which were 16-bit processors with minicomputer-like performance. The third generation came about as IC transistor counts approached 250,000.
- Motorola's MC68020, for example, incorporated an on-chip cache for the first time and the depth of the pipeline increased to five or more stages.

Fourth Generation:-

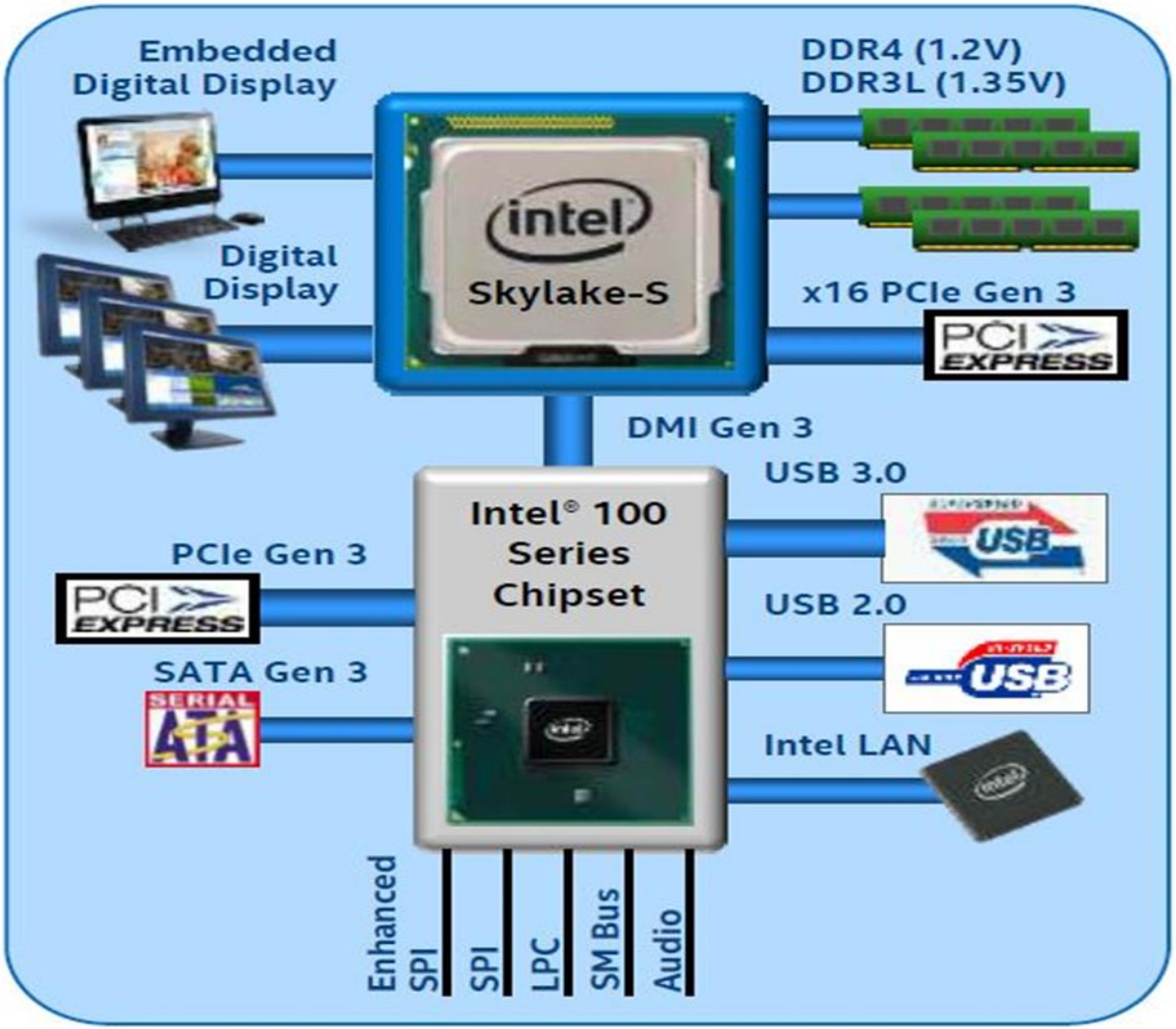
- As the workstation companies converted from commercial microprocessors to in-house designs, microprocessors entered their fourth generation with designs surpassing a million transistors.
- Leading-edge microprocessors such as Intel's 80960CA Intel's Haswell processors (core i7) and Motorola's 88100 could issue and retire more than one instruction per clock cycle.

Fifth Generation:-

- Some of the key characteristics of 5th Gen Intel dual core processors are as under:
- Intel HD Graphics 5500-6000 OR Intel Iris Graphics 6100.
- 22% higher 3d graphics performance than 4th generation processors.
- 14nm 2nd gen transistors are used causing a size shrink from last generation.
- 1.9 billion transistors as compared to 1.3 Billion under fourth generation systems.
- Example :Intel’s Broadwell processors launched in January 2015

Intel's Fourth and Fifth Generation Processors

Intel's Sixth Generation Processor
Skylake processor Launched in July 2015



Classification of Microprocessors

- Microprocessors are classified on the basis of
  - Bit size (Word Length)
  - Fabrication Technology used

Bit Size

4 bit:-Intel's 4004/4040/Toshiba TMS 1000/Rockwell's PPS-4

8 bit:- Intel's 8008/8080/8085/Zilog Z-80/Motorola M-6800/Fairchild's F-8/Rockwell's PPS-8

16 bit :- Intel's 8086/ 8088/ Zilog 8000/ Intel's 80186 80286/Motorola M68000

32 bit:- Intel 80386 / 80486 PENTIUM /PENTIUM PRO/Zilog 80000/M-68002

64 bit:- Dual core processors/Itanium/core i7 etc

Speed-Power Product (SPP)

- Speed (propagation delay) and power consumption are the two most important performance parameters of a digital IC. SPP is defined as the product of propagation delay expressed in nano sec and power consumption in milli watt, the unit for the same is pj (pico Joules)
- A simple means for measuring and comparing the overall performance of an IC family is the speed-power product (the smaller, the better).
- For example, an IC has
  - an average propagation delay of 10 ns
  - an average power dissipation of 5 mW
  - the speed-power product = (10 ns) x (5 mW)
    = 50 picoJoules (pJ)

- In 1968 Gordon Moore and Robert Noyce left Fairchild, to form a company ,they purchased the rights to use the name intel from a company called intelco.
- The “e” of intel was dropped to arrive at the intel logo
- Intel stands for **Int**egrated **el**ectronics

- In 1971 Intel launches its first microprocessor Intel-4004.
- In 1972 Intel launches its first 8 bit processor Intel-8008.
- In 1974 Intel launches its first 8 bit processor Intel-8080.
- In 1976 Intel launches its faster 8 bit processor Intel-8085.
- In the same year the first ever microcontroller Intel's 8048 was introduced and was used in cars and home appliances.
- In 1978 Intel launches its first 16 bit processor Intel-8086.

- In 1980 Intel launches the microcontroller 8051, which became the best selling microcontroller till date.
- In 1982 Intel launches its first high performance/multitasking 16 bit processor Intel-80286.
- In 1985 Intel launches its first high performance/multitasking 32 bit processor Intel-80386.
- In 1992 Intel launches 32 bit processor Intel-80486.
- In 1993 Intel launches 64 bit Pentium processor.

Moore’s Law

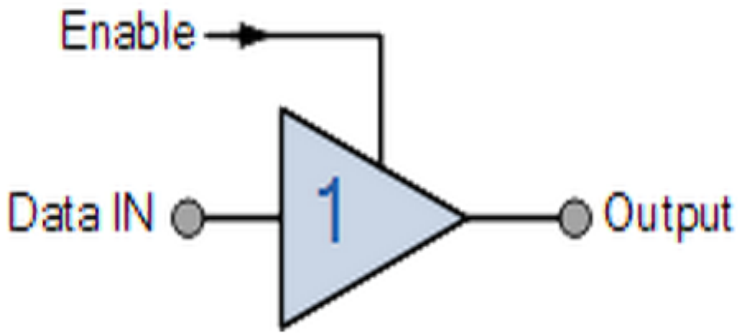
40 years ago, Intel co-founder Gordon Moore gave Moore’s Law , which states “The number of Transistors fabricated on a single chip will double every 18 months”

Tri-State Switch

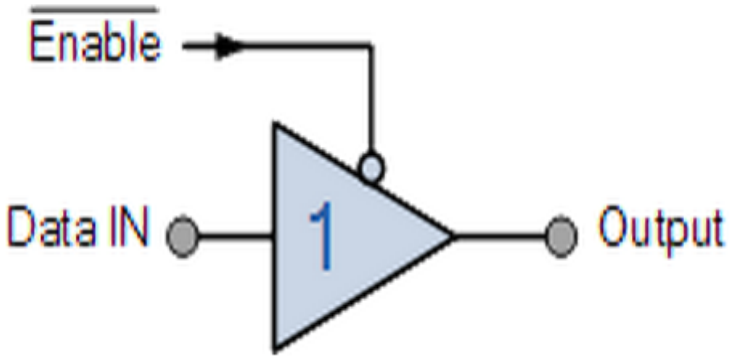
A Tri-state Switch can be thought of as an input controlled switch with an output that can be electronically turned “ON” or “OFF” by means of an external “Control” or “Enable” (EN) signal input. This control signal can be either a logic “0” or a logic “1” type signal resulting in the Tri-state switch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

Then a tri-state switch **requires two inputs**. One being the data input and the other being the

Active High Tri-State switch

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

Active low Tri-State switch

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

Need for Tri-State Switches

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

Internal Architecture of 8085

- Features of 8085:-

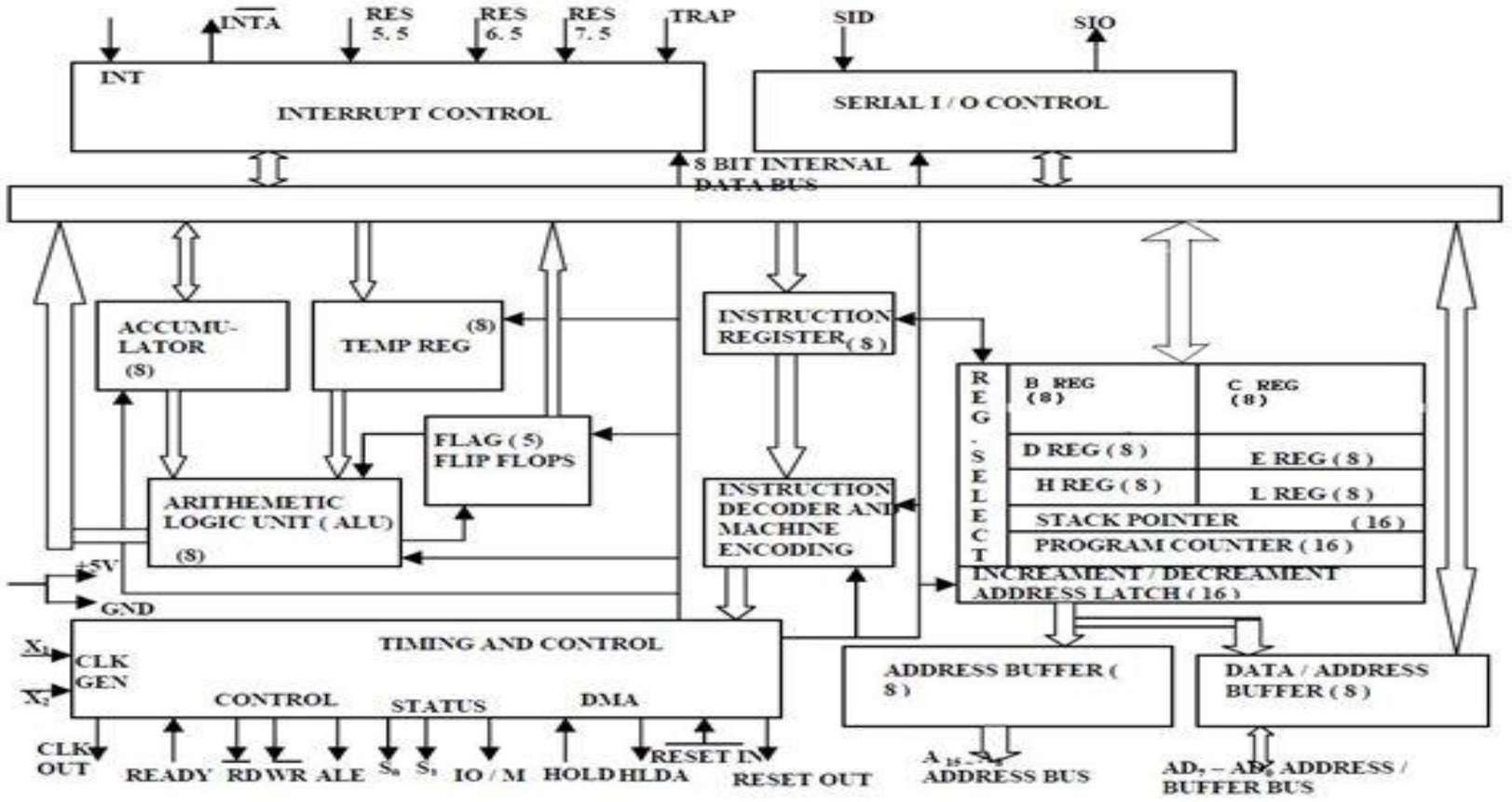
- It is an 8 bit microprocessor (each character is represented by 8 bits or a byte).
- It is manufactured with N-MOS (n-type Metal Oxide Semiconductor) technology implemented with 6200 transistors.
- It has 16-bit address lines - A0-A15 (to point the memory locations) and hence can point up to 2^16 = 65535 bytes (64KB) memory locations.
- The first 8 lines of address bus and 8 lines of data bus are multiplexed AD0-AD7. Data bus is a group of 8 lines D0-D7.
- It provides 5 level interrupts and supports external interrupt request.
- A 16 bit program counters (PC).
- A 16 bit stack pointer (SP).
- It provides 1 accumulator, 1 flag register, six 8-bit general purpose register arranged in pairs: BC, DE, HL.

- It consists of 74 instruction sets. (246 variations)
- It performs arithmetic and logical operations.
- It provides status for advanced control signals, On chip divide by 2 clock generator. (To synchronize external devices)
- It requires a signal +5V power supply and operates at 3.2 MHZ single phase clock with maximum clock frequency 6 MHz
- Serial input/output port.
- 1.3 micro sec instruction cycles.
- It is enclosed with 40 pins DIP (Dual in line package).

Functional block diagram of 8085

- 8085 consists of various units and each unit performs its own functions. The various units of a 8085 are listed below.
- Register Section
- Bus Section
- Arithmetic and logic Unit
- Timing and Control unit
- Interrupt Section
- Serial Input/output control

Functional Block Diagram of 8085



Register Section of 8085

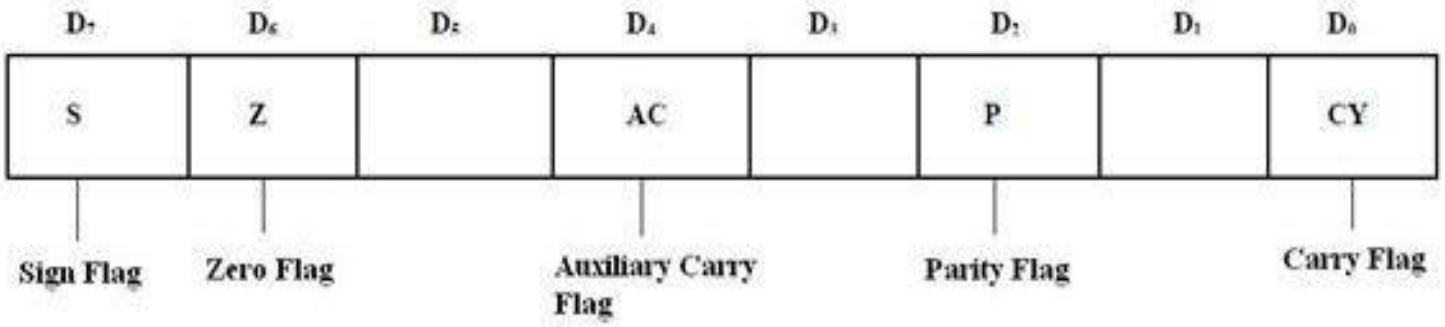
- Register section of 8085 contains the following registers:-
- 6 -8 bit GPR's (B,C,D,E,H,L)
- Accumulator Register (Acc/ A)
- Flag Register)(5/8)
- Temp Register(8 bits)
- PC(16 bits)
- SP(16 bits)
- Instruction Register(8 bits)

- 6 GPR’s
- These GPR’s are known as Scratch Pad Registers, they are of 8 bits each. They have designated numerical codes which are as under:-
- The above registers can be used in pairs forming the following 16 bit pairs
- BC
- DE
- HL
- Their code are as under:-

Accumulator Register

- Accumulator is nothing but a register which can hold 8-bit data. Accumulator aids in storing two quantities.
- The data to be processed by arithmetic and logic unit is stored in accumulator.
- It also stores the result of the operation carried out by the Arithmetic and Logic unit.
- The accumulator is a 8-bit register. The accumulator is connected to Internal Data bus and ALU (arithmetic and logic unit). The accumulator can be used to send or receive

Flag Register



There are five flip-flop in 8085 which are set or reset based on the outcome of an arithmetic and logical operation.

1.Sign flag(S): S=1, if D7=1; otherwise S=0;

2.Zero flag(Z): Z=1,if A=00H; otherwise Z=0;

3.Auxiliary flag(AC): AC=1, if a carry is generated from the addition of fourth bit from right of a 8-bit binary no. otherwise AC=0;

4.Parity flag(P): P=1, if A has even parity;

P=0, if A has odd parity;

5.Carry flag(CY): CY=1,Carry is generated;

CY=0,Carry is not generated;

Temporary Register:

As the name suggests this register acts as a temporary memory during the arithmetic and logical operations. Unlike other registers, this temporary register can only be accessed by the microprocessor and it is completely inaccessible to programmers. Temporary register is an 8-bit register.

Instruction Register:

This is a 8bit register. It is used to receive the 8 bit opcode of an instruction from memory. It may be noted that opcode of an instruction is only 8 bits, even though the instruction may be 3 bytes long. It is

W-Z Pair

- These are 8bit registers. They are not accessible to the programmer.
  They are used for temporary storage inside the 8085, the 16 bit address operand of an instruction. For example, when ‘LDA C234H, ins tructionis fetched, IR register will receive the op code for LDA, and W and Zregisters will receive C2H and 34H, respectively.

Program Counter & Stack Pointer

Program Counter(PC)

This 16-bit register deals with sequencing the execution of instructions. 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.

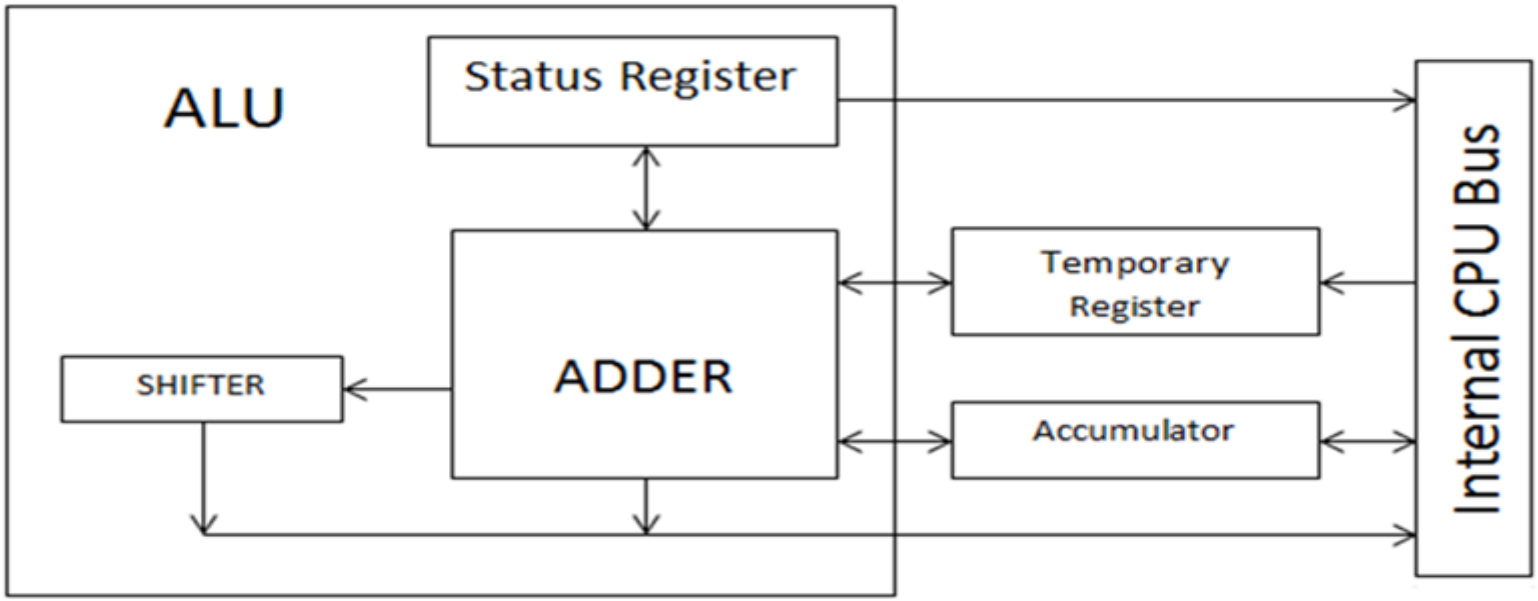
Stack Pointer(SP)

Stack pointer is also a 16-bit register which is used as a memory pointer.A stack is nothing but the portion of RAM (Random access memory).Stack pointer maintains the address of the last byte that is entered into stack.

Each time when the data is loaded into stack, Stack pointer gets decremented. Conversely it is incremented when data is retrieved from stack.

Arithmetic and Logic Unit

- It is multi operational combinational logic circuit.It performs arithmetic and logical operations like ANDing, ORing, EX-ORing, ADDITON, SUBTRACTION, etc.
- It is not accessible by user.
- The word length of ALU depends upon the width of the internal data bus.
- It is 8 bit. It is always controlled by timing and control circuits.
- It controls the status of flag register based on the outcome of the result.



The ALU contains following blocks:
Adder: It performs arithmetic operations like addition, subtraction, increment, decrement, etc. The result of operation is stored into accumulator.

Shifter: It performs logical operations like rotate left, rotate right, etc. The result of operation is again stored into accumulator.

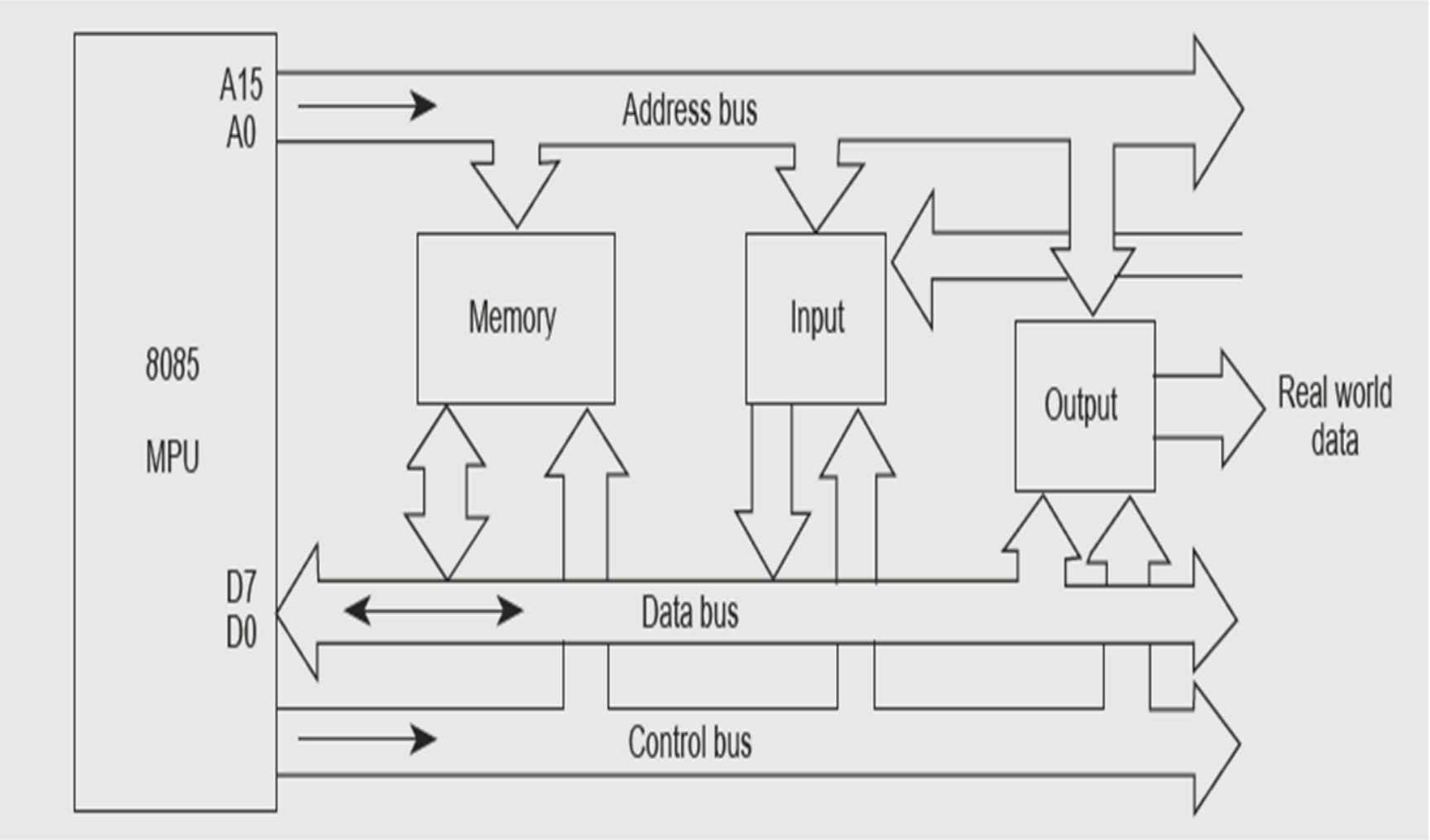
Status Register: Also known as flag register. It contains a no. of flags either to indicate conditions arising after last ALU operation or to control certain operations.

Bus Section of 8085

Address Bus:

It is a group of wires or lines that are used to transfer the addresses of Memory or I/O devices. It is unidirectional. In Intel 8085 microprocessor, Address bus is of 16 bits. This means that Microprocessor 8085 can transfer maximum 16 bit address which means it can address 65,536 different memory locations. This bus is multiplexed with 8 bit data bus. So the most significant bits (MSB) of address goes through Address bus (A15-A8) and LSB goes through multiplexed data bus (AD0-AD7).The

Bus Architecture of 8085



- **Timing and control unit**

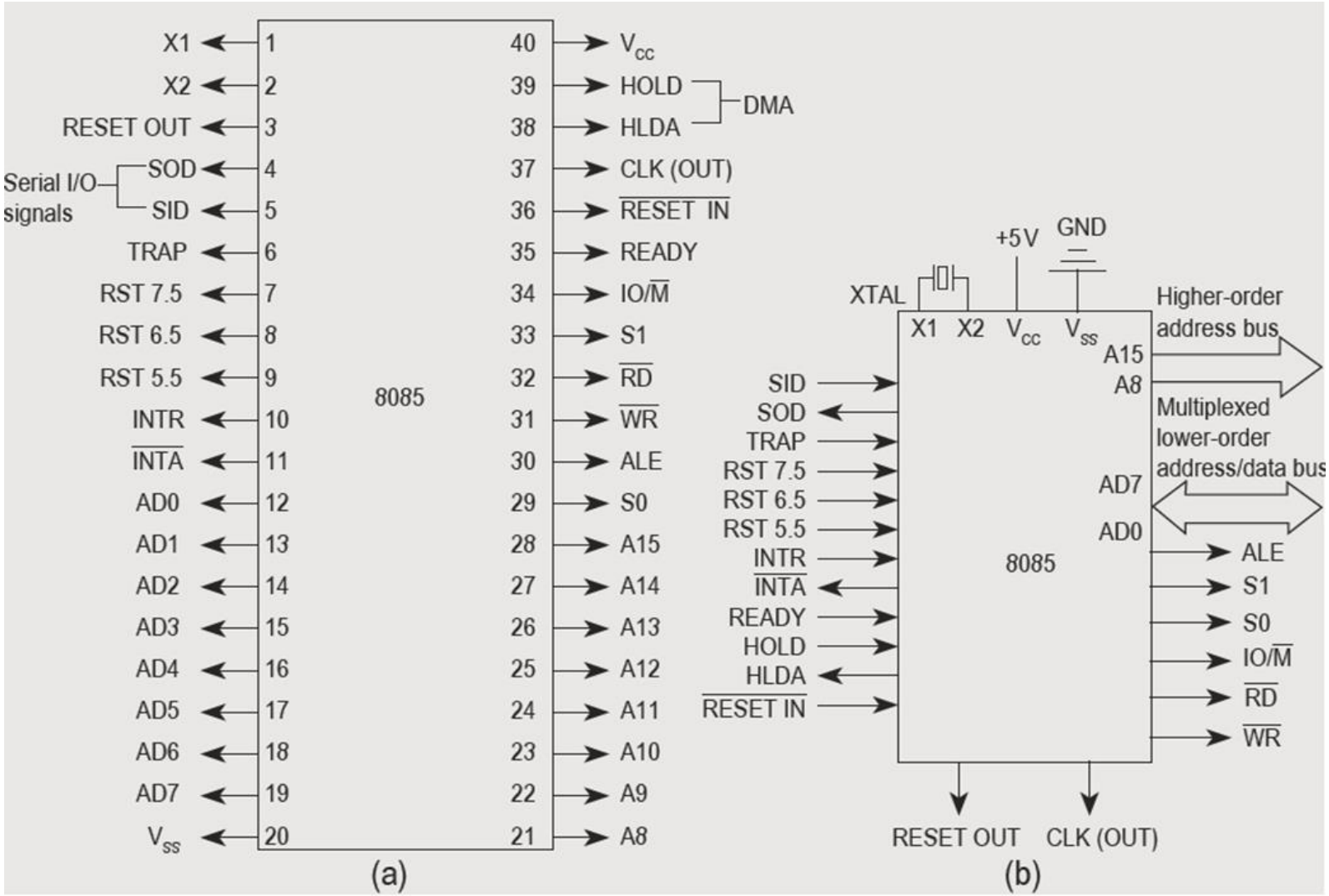
- Timing and control unit is a very important unit as it synchronizes the registers and flow of data through various registers and other units.
- This unit consists of an oscillator and controller sequencer which sends control signals needed for internal and external control of data and other units.
- The oscillator generates clock signals which aids in synchronizing all the registers of 8085 microprocessor.

- Signals that are associated with Timing and control unit are:

- Control Signals: READY, $\overline{RD}$,$\overline{WR}$, ALE
- Status Signals: S0, S1, IO/$\overline{M}$
- DMA Signals: HOLD, HLDA
- RESET Signals: $\overline{RESET\ IN}$,RESET OUT.

Pin Diagram of 8085

- Intel 8085 has 40 pins, operates 3MHz clock and requires +5V for power supply. The signals can be classified into six groups,
- Address Bus.
- Data Bus.
- Control and Status Signals.
- Power supply and System Clock.
- Externally Initiated Signals.
- Serial I/O signals


X1 1
X2 2
RESET OUT 3
Serial I/O signals
SOD 4
SID 5
TRAP 6
RST 7.5 7
RST 6.5 8
RST 5.5 9
INTR 10
INTA 11
AD0 12
AD1 13
AD2 14
AD3 15
AD4 16
AD5 17
AD6 18
AD7 19
Vss 20
8085
40 Vcc
39 HOLD
DMA
38 HLDA
37 CLK (OUT)
36 RESET IN
35 READY
34 IO/M
33 S1
32 RD
31 WR
30 ALE
29 S0
28 A15
27 A14
26 A13
25 A12
24 A11
23 A10
22 A9
21 A8
(a)
XTAL
X1 X2 Vcc Vss
+5V
GND
Higher-order address bus
A15
A8
Multiplexed lower-order address/data bus
AD7
AD0
SID
SOD
TRAP
RST 7.5
RST 6.5
RST 5.5
INTR
INTA
READY
HOLD
HLDA
RESET IN
8085
ALE
S1
S0
IO/M
RD
WR
RESET OUT
CLK (OUT)
(b)

Pin Functionality

- The functions of each pin are described below:
- Address and Data Buses:
- A8 - A15 (Output -3 state higher order address bus)-
- The most significant 8 bits of the memory address or the 8 bits of the I/0 addresses, tri-stated during Hold and Halt modes.
- AD0-AD7 (Input/output- 3 state multiplexed address/data bus)
- Multiplexed Address/Data Bus; Lower 8 bits of the memory address (or I/0 address) appear on the bus during the first clock cycle of a machine state.
- It then becomes the data bus during the second and third clock cycles.
- Tri- stated during Hold and Halt modes.

Control and Status Signals

- ALE (Output) Address Latch Enable
- a) This output signal indicates the availability of the valid address on the address/data lines.
- b) It occurs during the first clock cycle of a machine state and enables the address to get latched into the on chip latch of peripherals.
- SO, S1 (Output)
- Microprocessor Bus Status signals
- Encoded status of the bus cycle as explained in previous section.

Control and Status Signals

- (Output- 3state) READ
- indicates the selected memory or I/O device is to be read and that the Data Bus is available for the data transfer.
- (Output- 3state) WRITE
- indicates the data on the Data Bus is to be written into the selected memory or I/O location.
- IO/M (output :3 state)
- a) It indicates whether the read/write operation is being done with respect to the memory or an I/O device.
- b) Logic 1 indicates I/O device access
- c) Logic 0 indicates memory access
- d) This signal is tri-stated during hold and halt modes

8085 Machine Cycle Status and Control Signals

Machine Cycle	Status			Control Signals
	$IO/\overline{M}$	S_1	S_0	
Opcode Fetch	0	1	1	$\overline{RD} = 0$
Memory Read	0	1	0	$\overline{RD} = 0$
Memory Write	0	0	1	$\overline{WR} = 0$
I/O Read	1	1	0	$\overline{RD} = 0$
I/O Write	1	0	1	$\overline{WR} = 0$
Interrupt Acknowledge	1	1	1	$\overline{INTA} = 0$
Halt	Z	0	0	

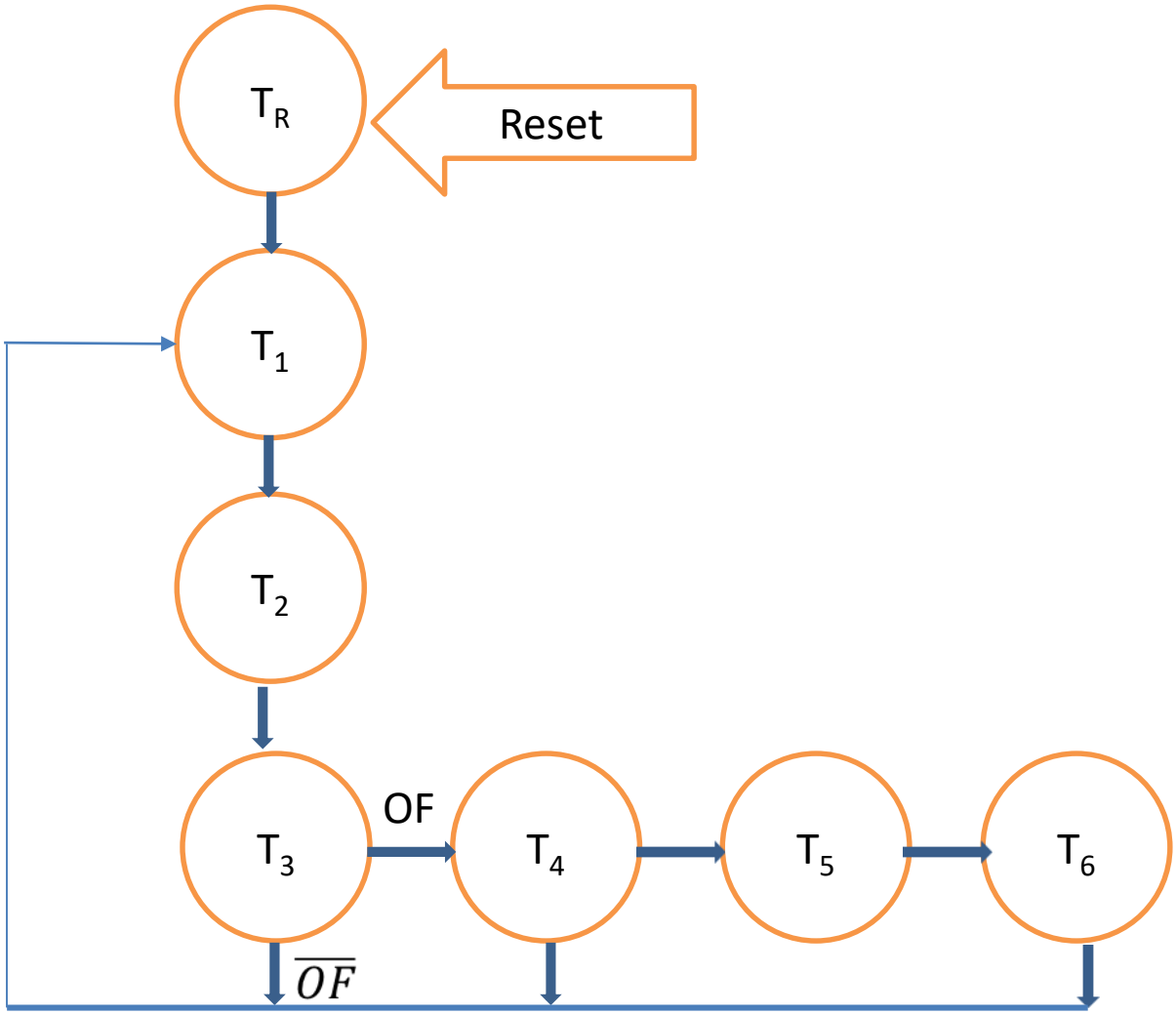
Externally Initiated Signals

- **Ready**
- It is used to interface slow peripheral devices with the fast microprocessor
- If Ready is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data.
- If Ready is low, the CPU will wait for Ready signal to go high before completing the read or write cycle.
- **HLDA (Output)**

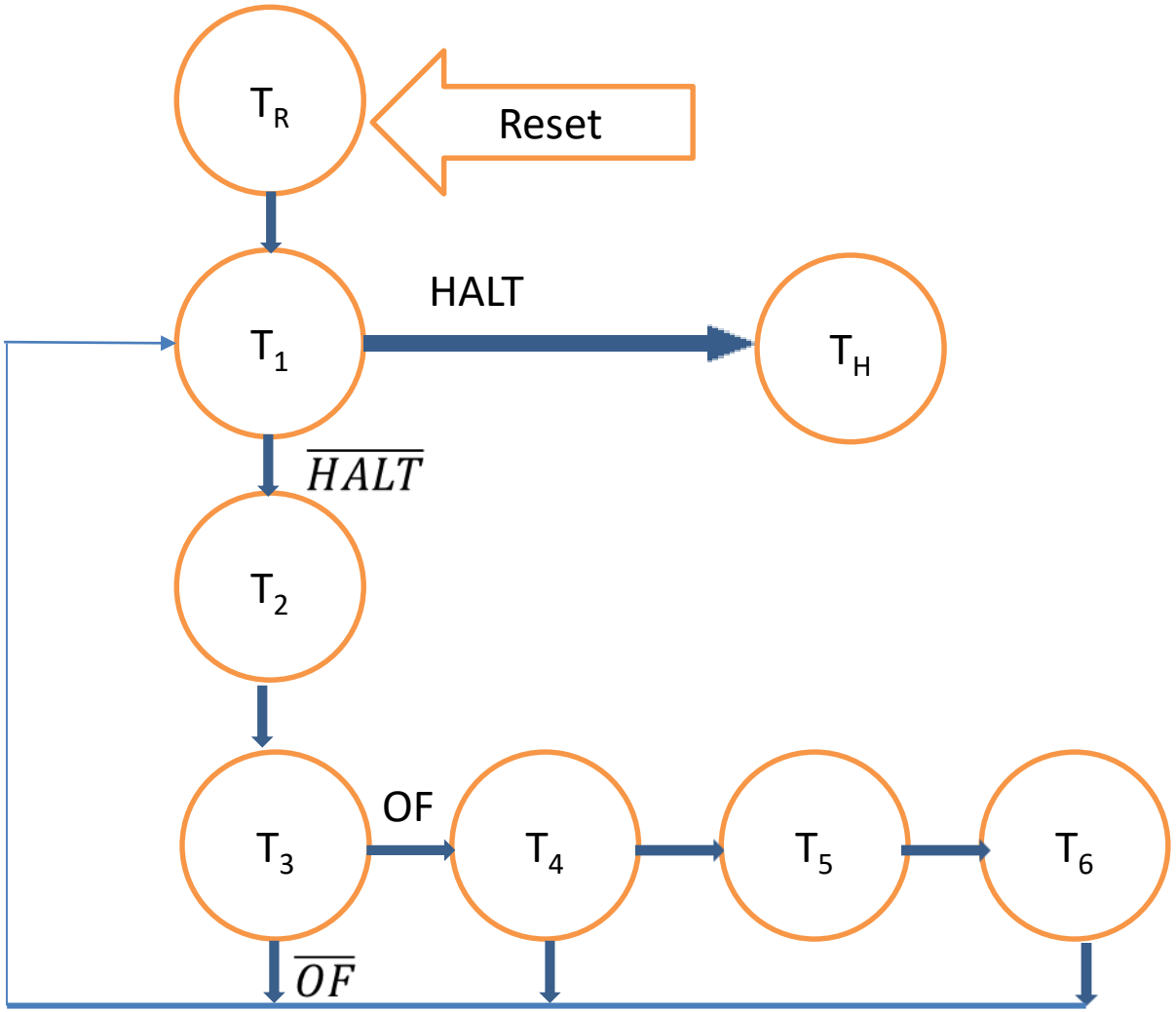
a) HOLD ACKNOWLEDGE, an active high signal, indicates that the CPU has received the Hold request and that it will relinquish the buses in the next clock cycle.

b) HLDA goes low after the Hold request is removed.

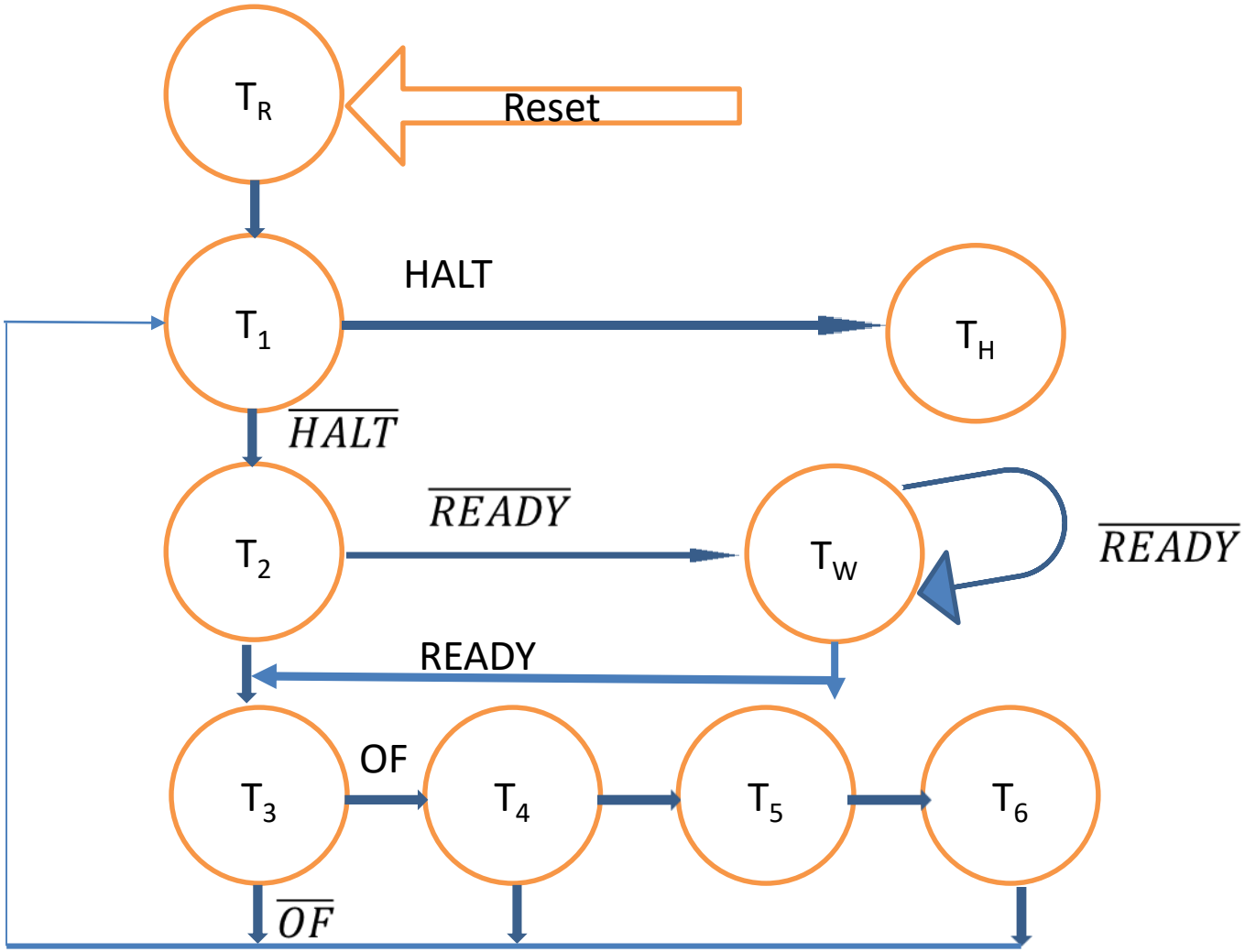
c) The CPU takes the control of buses after HLDA goes low.



Simplified State Transition Diagram of 8085



Simplified State Transition Diagram of 8085 with HALT State



Simplified State Transition Diagram of 8085 with HALT & WAIT State

Activities associated with the T-States of 8085

- T1:A memory or I/O device address is placed on address/data bus (Ad0-Ad7) and address bus (A8-A15).ALE signal goes high to facilitate the latching of lower 8 bits of address on Ad0-Ad7.Status information is placed on IO/M and s0,s1 to define the type of machine cycle. The HALT Flag is checked.
- T2:Ready and Hold inputs are sampled. PC is incremented .Ready line is also sampled if it is low it moves into Wait state. The states of address/data/control signals remains as they were at the end of T2
- T3:An instruction / data byte is transferred to or from the processor.
- T4:The contents of instruction Register are decoded.
- T5-T6: These states are used to complete the execution of some instructions which require more than 4 T-states in their OPCODE.

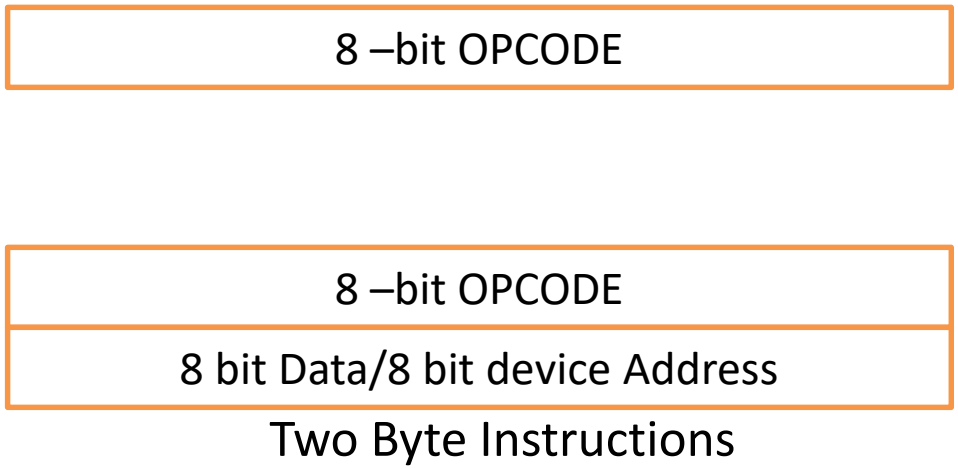
Instruction Formats

- Based on the length of the instructions, the instruction set can be classified into three or more types.

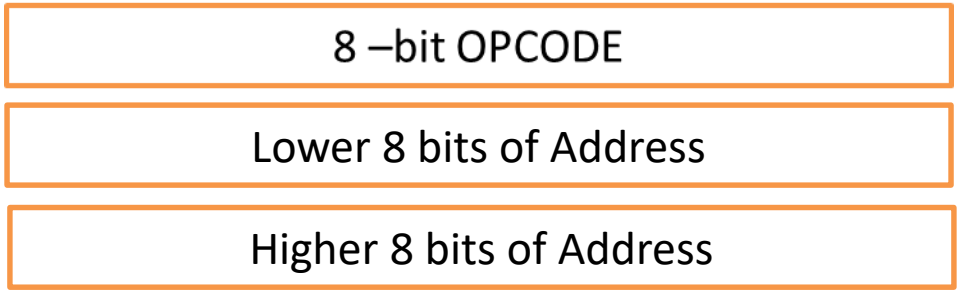
a) One-byte instructions,

b) Two-byte instructions and

c) Three-byte instructions etc.



Two Byte Instructions



Three Byte Instruction Format

One-Byte Instructions

- Instructions that require only one byte in machine language are called one-byte instructions.
- These instructions just have the machine code or opcode alone to represent the operation to be performed.
- The common examples are the instructions that have their operands within the processor itself.
- Even though the instruction ADD M adds the content of a memory location to that of the accumulator, its machine code requires only one byte.

Example - One-Byte Instructions

Opcode	Operand	Machine code/Opcode/ Hex code
MOV	A, B	78
ADD	M	86
XRA	A	AF

Two-Byte Instructions

Opcode	Operand	Machine code / Opcode/ Hex code	Byte description
MVI	A, 7FH	3E 7F	First Byte Second Byte
ADI	0FH	C6 OF	First Byte Second Byte
IN	40H	DB 40	First Byte Second Byte

In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand.

Two-Byte Instructions contd..

Mnemonics	Hex code
MVI A, 32H	3E 32

Assume that the data byte is 32H. The assembly language instruction is written as

The instruction would require two memory locations to be stored in consecutive memory locations.

MVI r, data ; r <--data

Two-Byte Instructions Example

- ADI data A <-- A + data
- OUT port, where port is an 8-bit device address. (Port) <-- A.
- Since the byte is not the data but points directly to where it is located, this is called direct addressing.

Three-Byte Instructions

- Instructions that require three bytes in machine code are called three-byte instructions.
- In 8085 machine language, the first byte of the three-byte instructions is the opcode which specifies the operation to be performed.
- The next two bytes refer to the 16-bit operand, which is either a 16-bit number or the address of a memory location.

Three-Byte Instructions Examples

Opcode	Operand	Hex Code	Byte description
JMP	2085H	C3 85 20	First byte Second Byte Third Byte
LDA	8850H	3A 50 88	First byte Second Byte Third Byte
LXI	H, 0520H	21 20 05	First byte Second Byte Third Byte

Three-Byte Instructions contd..

- This instruction would require three memory locations in memory. Three byte instructions - opcode + data byte + data byte
- Examples:
- LXI rp, 16-bit data where rp is one of the pairs of registers BC, DE, HL used as 16-bit registers. The two data bytes are 16-bit data to be stored in L and H in sequence.
- LXI H,0520H coded as 21H 20H 50H in three bytes. This is also immediate addressing.

Three-Byte Instructions contd..

- LDA addr
- A <-- (addr) Accumulator is loaded with the memory content of the address given in the instruction.
- Addr is a 16-bit address.
- Example: LDA 2134H coded as 3AH 34H 21H. This is also an example of direct addressing.

ADDRESSING MODES OF 8085

- Addressing mode is defined as the way by which an operand is specified within the instruction.
- Efficient software development for the microprocessor requires complete familiarity with the addressing mode employed for each instruction.

Types of Addressing Modes

- The 8085 has the following 5 different types of addressing.

a) Immediate Addressing
b) Direct Addressing
c) Register Addressing
d) Register Indirect Addressing
e) Implied/Implicit Addressing

Immediate Addressing

- Immediate addressing transfers the operand given in the instruction.
- A byte or a word into the destination register or Register pair
- Operand is part of the instruction itself.
- Format of Immediate Addressing
- Instructions

Example - Immediate Addressing

- MVI A, 9AH

(a) The operand is part of the instruction.

(b) The operand is stored in the register mentioned in the instruction.

- ADI 05H

(a) Add 05H to the contents of the accumulator.

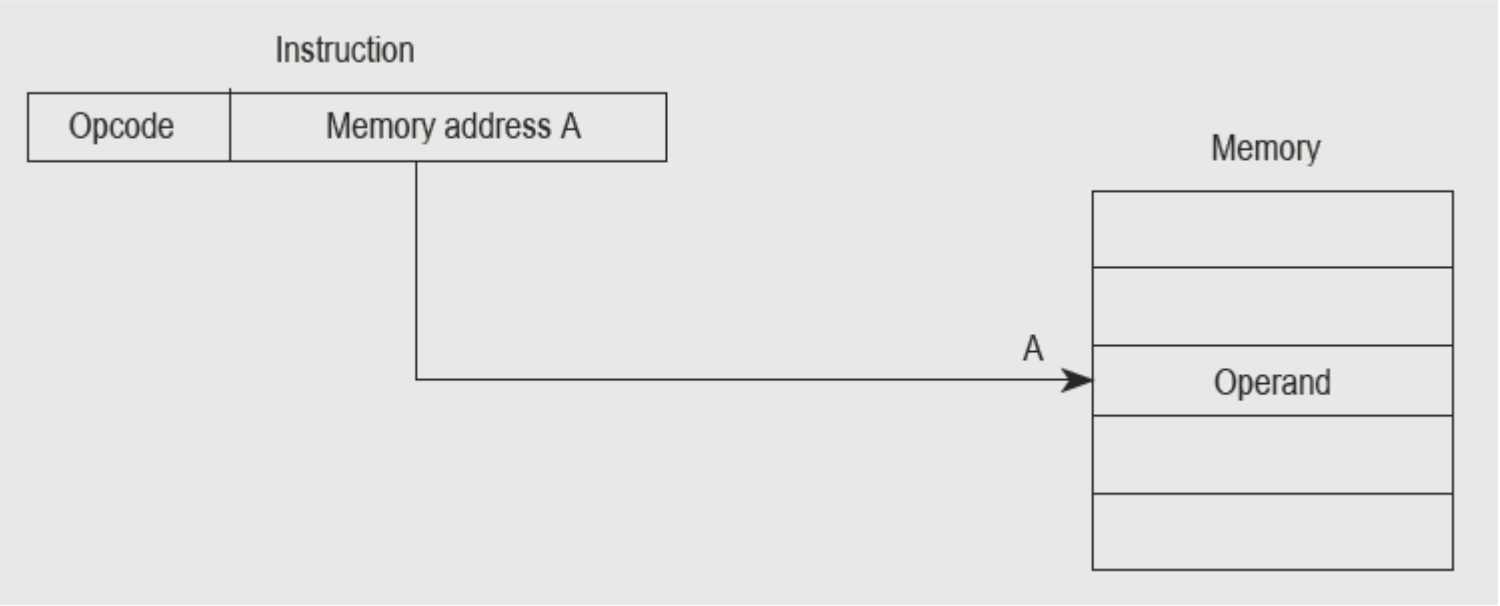
(b) 05H is the operand.

- It executes faster.

Memory Direct Addressing

- Memory Direct addressing moves a byte or word between a memory location and a register.
- The memory location address is given in the instruction.
- The instruction set of 8085 does not support a memory to memory transfer.

Format of memory Direct Addressing



Example - Memory Direct Addressing

- LDA 850FH

This instruction is used to load the contents of the memory location 850Fh into the accumulator.

- STA 9001H

a) This instruction is used to store the contents of the accumulator to the memory address 9001H.

b) In these instructions the memory address of the operand is given in the instruction.

Memory Direct Addressing contd..

- Direct addressing is also used for data transfer between processor and output / input devices.
- For example,

a) IN instruction is used to receive data from the input port and store it into the accumulator.

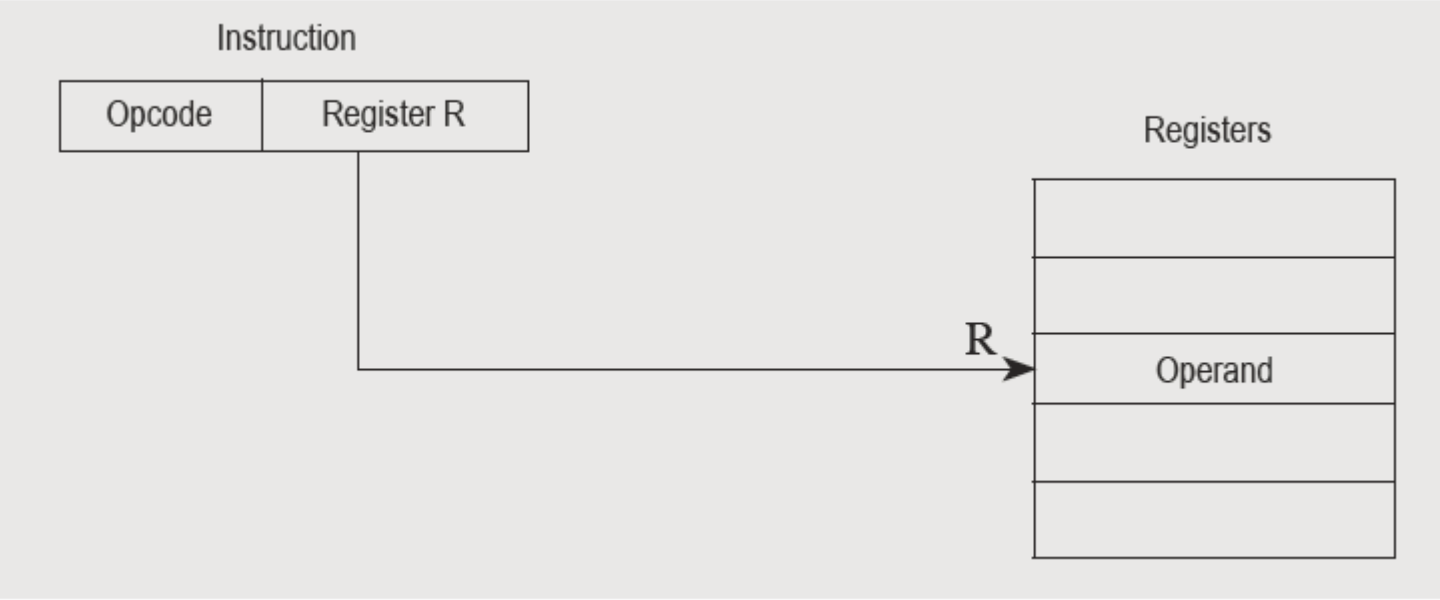
b) OUT instruction is used to send the data from the accumulator to the output port.

e.g., IN 00H and OUT 01H

Register Addressing

- Register addressing transfers a copy of a byte from the source register to the destination register.
- Operand is in register named in the instruction.
- It features very fast execution, very limited register space and requires good assembly programming.
- Mostly the operand is within in the processor itself and so the execution is faster.

Format of Register Direct Addressing



Example - Register Addressing

- MOV Rd, Rs

MOV B, C ; Copy the contents of C register to B register.

- ADD B

Add contents of register mentioned – B register content to A, accumulator

Indirect Addressing

- Indirect addressing transfers a byte or word between a register and a memory location.
- The memory location address is stored in a register and that register is specified in the instruction.
- Effective Address is calculated by the processor using contents of the register specified in the instruction.
- This type of addressing employs several accesses—two accesses to retrieve the 16-bit address and a further access (or accesses) to retrieve the data which is to be loaded in the register.

Example - Indirect Addressing

- MOV A, M

a) Here, the data is in the memory location pointed to by the contents of the HL pair.

b) The data is moved to the accumulator

Implied or Implicit Addressing

- In implied addressing mode, the instruction itself specifies the data to be operated.
- For example CMA – complement the content of the accumulator. CMC, STC – complement carry, set carry etc.
- No specific data is mentioned in the instruction.
- The instruction does not need any specific operand

UNIT II

INSTRUCTION SET OF 8085

- An instruction is a binary bit pattern that can be decoded inside a microprocessor to perform a specific function.
- The assembly language mnemonics are the codes for these binary patterns so that the user can easily understand the function performed by these instructions.
- The entire group of instructions is called the instruction set, and this determines the functionalities the microprocessor can perform.
- Intel 8085 processor has its own set of instructions listed both in mnemonics and machine code, also called as object code.
- As 8085 is an 8-bit processor, the machine codes for the instructions are also 8-bits wide.

Classification of Instruction Set

- Instructions can be classified into five categories based on the functionality provided by the instructions.
- a) Data Transfer (copy) operations,
- b) Arithmetic operations,
- c) Logical operations,
- d) Branching operations
- e) Machine-control operations

- This group of instructions copy data from a location called a source register to another location called a destination register.
- The contents of the source register are not modified.

Data Transfer Operations

Mnemonic	Operand	M/C format	Addressing mode	Number of bytes	T states
MOV	Reg destination, Reg source	01dddsss	Register direct	1	4
MOV	Reg destination,M	01ddd110	Memory Indirect	1	7
MOV	M,Reg source	01110sss	Memory Indirect	1	7
MVI	Reg, byte	00ddd110	Immediate	2	7
MVI	M, byte	00110110,data	Immediate	3	10
LXI	rp, Dble (rp=BC,DE,HL, SP)	00rp0001 data	Immediate	3	10
LDA	adr	00111010,addr	Memory direct	3	13
STA	adr	00110010,addr	Memory direct	3	13
LDAX	Rp (rp=BC,DE)	000x1010	Memory Indirect	1	7
STAX	Rp (rp=BC,DE) x=0->B,1->D	000x0010	Memory Indirect	1	7
XCHG	HL & DE		Implicit	1	4

Mnemonic	Operand	Addressing mode	Number of bytes	T states
SPHL	SP and HL pair	Register	1	6
XTHL	Stack and HL pair	Memory Indirect	1	7
PCHL		Register	1	6
IN	8 bit Address	Direct	2	10
OUT	8 bit Address	Direct	2	10

- **LDA 16-bit address**

- The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator (A).
- The contents of the source are not altered.

- Example: LDA 2000H


BEFORE EXECUTION
A
30
2000H
LDA 2000H
AFTER EXECUTION
A
30
30
2000H

- **LDAX Register Pair**
- Load accumulator (A) with the contents of memory location whose address is specified by BC or DE or register pair.
- The contents of either the register pair or the memory location are not altered.

- **Example: LDAX D**


BEFORE EXECUTION
A
F
B
C
D
20
E
30
2030H
80
LDAX D
AFTER EXECUTION
A
80
F
B
C
D
20
E
30
2030H
80

- STA 16-bit address

- The contents of accumulator are copied into the memory location i.e. address specified by the operand in the instruction.

- Example: STA 2000 H


BEFORE EXECUTION
A
50
2000H
STA 2000H
AFTER EXECUTION
A
50
50
2000H

- **STAX Register Pair**

- **Store the contents of accumulator (A) into the memory location whose address is specified by BC Or DE register pair.**

- **Example: STAX B**


BEFORE EXECUTION
A 50 F
B 10 C 20
D E
1020H
STAX B
AFTER EXECUTION
A 50 F
B 10 C 20
D E
1020H
50

- **SHLD 16-bit address**

- Store H-L register pair in memory.
- The contents of register L are stored into memory location specified by the 16-bit address.
- The contents of register H are stored into the next memory location.

- Example: SHLD 2500 H


BEFORE EXECUTION
H
30
L
60
204FH
2500H
2502H
SHLD 2500H
AFTER EXECUTION
H
30
L
60
204FH
2500H
2502H
60
30

- XCHG

- The contents of register H are exchanged with the contents of register D.
- The contents of register L are exchanged with the contents of register E.

- Example: XCHG

BEFORE EXECUTION

D	20	E	40
H	70	L	80

XCHG

AFTER EXECUTION

D	70	E	80
H	20	L	40

- SPHL

- Move data from H-L pair to the Stack Pointer (SP)
- This instruction loads the contents of H-L pair into SP.

- Example: SPHL

BEFORE EXECUTION

SP			
H	25	L	00

SPHL

AFTER EXECUTION

SP	2500		
H	25	L	00

- **XTHL**
- Exchange H–L with top of stack
- The contents of L register are exchanged with the location pointed out by the contents of the SP.
- The contents of H register are exchanged with the next location (SP + 1).

- Example: XTHL

L=SP

H=(SP+1)

BEFORE EXECUTION

SP	2700		
H	30	L	40

2700H	50
2701H	60
2702H	

XTHL

AFTER EXECUTION

SP	2700		
H	60	L	50

2700H	40
2701H	30
2702H	

- **PCHL**

- Load program counter with H-L contents
- The contents of registers H and L are copied into the program counter (PC).
- The contents of H are placed as the high-order byte and the contents of L as the low-order byte.

- Example: PCHL

BEFORE EXECUTION

PC			
H	60	L	00

PCHL

AFTER EXECUTION

PC	6000		
H	60	L	00

- **IN 8-bit port address**

- Copy data to accumulator from a port with 8-bit address.
- The contents of I/O port are copied into accumulator.

- Example: IN 80 H


BEFORE EXECUTION
PORT 80H
10
A
IN 80H
AFTER EXECUTION
PORT 80H
10
A
10

- **OUT 8-bit port address**

- Copy data from accumulator to a port with 8-bit address
- The contents of accumulator are copied into the I/O port.

- Example: OUT 50 H


BEFORE EXECUTION
PORT 50H
10
A
40
OUT 50H
AFTER EXECUTION
PORT 50H
40
A
40

Arithematic Instructions

- These instructions perform the operations like:

- Addition
- Subtraction
- Increment
- Decrement

(1) Arithematic Instructions

- **ADD R**
- **ADD M**

- The contents of register or memory are added to the contents of accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.

- Example: ADD C or ADD M

BEFORE EXECUTION

A	20		
B		C	30
D		E	
H		L	

ADD C

A=A+R

AFTER EXECUTION

A	50		
B		C	30
D		E	
H		L	

BEFORE EXECUTION

A	20		
B		C	
D		E	
H	20	L	50

10

2050

ADD M

A=A+M

AFTER EXECUTION

A	30		
B		C	
D		E	
H	20	L	50

10

2050

(2) Arithematic Instructions

- **ADC R**
- **ADC M**

- The contents of register or memory and Carry Flag (CY) are added to the contents of accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair. All flags are modified to reflect the result of the addition.

- Example: ADC C or ADC M


BEFORE EXECUTION
CY 1
A 50
B C 20
D E
H L
ADC C
A=A+R+CY
AFTER EXECUTION
CY 0
A 71
B C 20
D E
H L
BEFORE EXECUTION
CY 1
A 20
2050H
30
H 20 L 50
ADC M
A=A+M+CY
AFTER EXECUTION
CY 0
A 51
2050H
30
H 20 L 50

(3) Arithematic Instructions

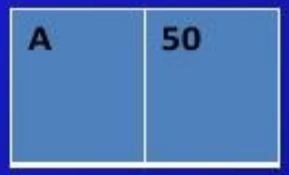
- **ADI 8-bit data**

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

- Example: ADI 10 H

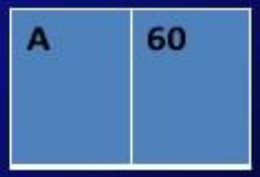
BEFORE EXECUTION

AFTER EXECUTION



ADI 10H

A=A+DATA(8)



(4) Arithematic Instructions

- **ACI 8-bit data**

- The 8-bit data and the Carry Flag (CY) are added to the contents of accumulator.
- The result is stored in accumulator.

- Example: ACI 20 H

BEFORE EXECUTION

CY	1
A	30

ACI 20H
A=A+DATA (8)+CY

AFTER EXECUTION

CY	0
A	51

(5) Arithematic Instructions

- **DAD Register pair**

- The 16-bit contents of the register pair are added to the contents of H-L pair.
- The result is stored in H-L pair.
- If the result is larger than 16 bits, then CY is set.

- Example: DAD D

BEFORE EXECUTION

CY	0

SP			
B		C	
D	10	E	20
H	20	L	50

DAD D

HL=HL+R

AFTER EXECUTION

CY	0

SP			
B		C	
D	10	E	20
H	30	L	70

(6) Arithematic Instructions

- **SUB R**
- **SUB M**

- The contents of the register or memory location are subtracted from the contents of the accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.

- Example: SUB B or SUB M

BEFORE EXECUTION

A	50		
B	30	C	
D		E	
H		L	

SUB B

A=A-R

AFTER EXECUTION

A	20		
B	30	C	
D		E	
H		L	

BEFORE EXECUTION

A	50		
H	10	L	20

1020H

10

SUB M

A=A-M

AFTER EXECUTION

A	40		
H	10	L	20

1020H

10

(7) Arithematic Instructions

- **SBB R**
- **SBB M**

- The contents of the register or memory location and Borrow Flag (i.e.CY) are subtracted from the contents of the accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.

- **Example: SBB C or SBB M**


BEFORE EXECUTION
CY 1
A 40
B C 20
D E
H L
SBB C
A=A-R-CY
AFTER EXECUTION
CY 0
A 19
B C 20
D E
H L
BEFORE EXECUTION
CY 1
A 50
H 20 L 50
2050H
10
SBB M
A=A-M-CY
AFTER EXECUTION
CY 0
A 39
H 20 L 50
2050H
10

(8) Arithematic Instructions

- SUI 8-bit data

- OPERATION: A=A-DATA(8)
- The 8-bit immediate data is subtracted from the contents of the accumulator.
- The result is stored in accumulator.

- Example: SUI 45 H

(9) Arithematic Instructions

- **SBI 8-bit data**

- The 8-bit data and the Borrow Flag (i.e. CY) is subtracted from the contents of the accumulator.
- The result is stored in accumulator.

- Example: SBI 20 H

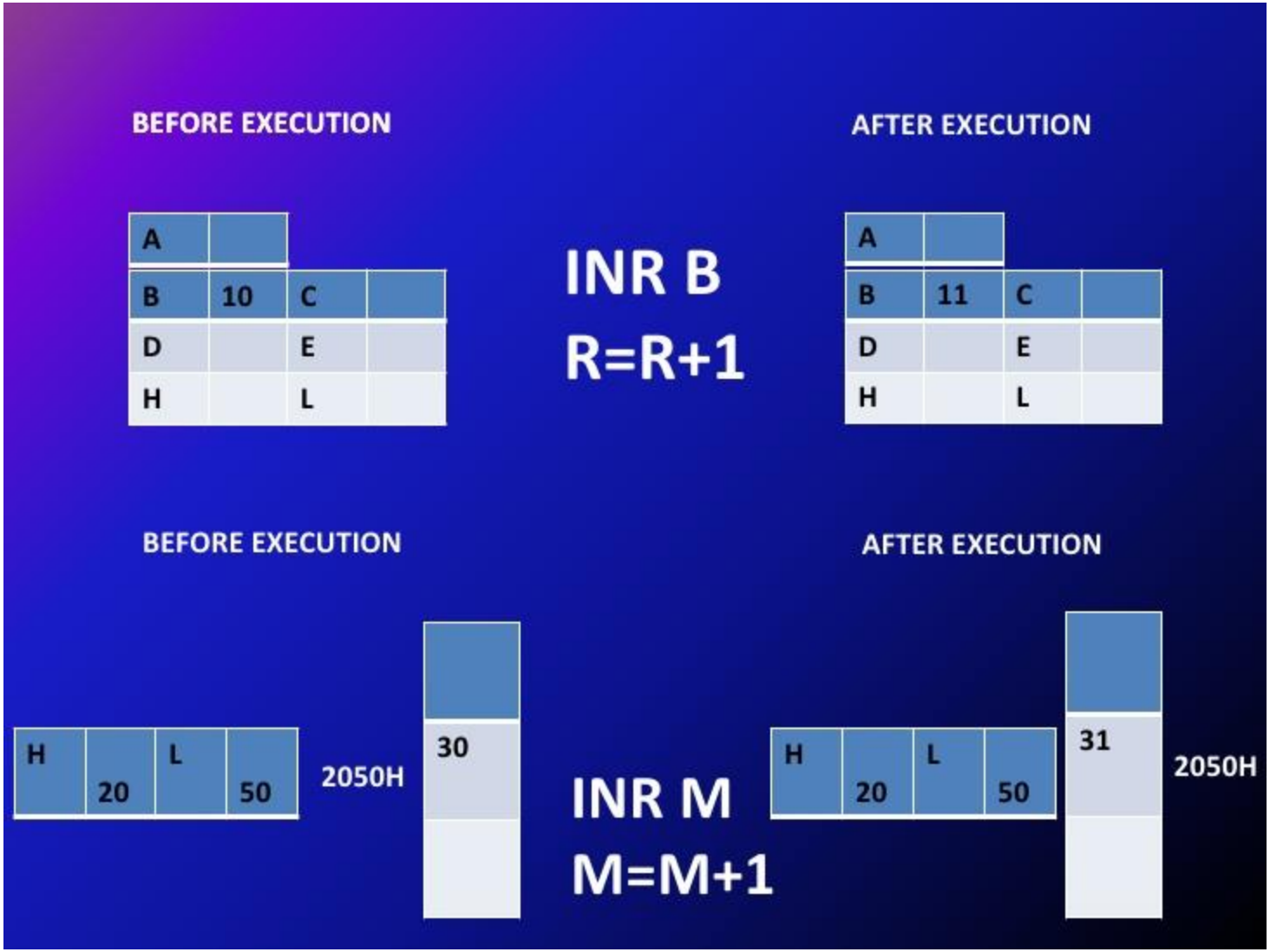

BEFORE EXECUTION
AFTER EXECUTION
CY 1
A 50
SBI 20H
A=A-DATA(8)-CY
CY 0
A 29

(10) Arithematic Instructions

- **INR R**
- **INR M**

- The contents of register or memory location are incremented by 1.
- The result is stored in the same place.
- If the operand is a memory location, its address is specified by the contents of H-L pair.

- Example: INR B or INR M


BEFORE EXECUTION
A
B
10
C
D
E
H
L
INR B
R=R+1
AFTER EXECUTION
A
B
11
C
D
E
H
L
BEFORE EXECUTION
H
20
L
50
2050H
30
INR M
M=M+1
AFTER EXECUTION
H
20
L
50
31
2050H

(11) Arithematic Instructions

- **INX Rp**

- The contents of register pair are incremented by 1.
- The result is stored in the same place.

- Example: INX H

BEFORE EXECUTION

SP			
B		C	
D		E	
H	10	L	20

INX H

RP=RP+1

AFTER EXECUTION

SP			
B		C	
D		E	
H	11	L	21

(12) Arithematic Instructions

- **DCR R**
- **DCR M**

- The contents of register or memory location are decremented by 1.
- The result is stored in the same place.
- If the operand is a memory location, its address is specified by the contents of H-L pair.
- Example: DCR E or DCR M

BEFORE EXECUTION

A			
B		C	
D		E	20
H		L	

DCR E

R=R-1

AFTER EXECUTION

A			
B		C	
D		E	19
H		L	

BEFORE EXECUTION

H		L	
	20		50

2050H

21

DCR M

M=M-1

AFTER EXECUTION

H		L	
	20		50

2050H

20

(13) Arithematic Instructions

- **DCX Rp**

- The contents of register pair are decremented by 1.
- The result is stored in the same place.

- Example: DCX D

BEFORE EXECUTION

SP			
B		C	
D	10	E	20
H		L	

DCX D

RP=RP-1

AFTER EXECUTION

SP			
B		C	
D	10	E	19
H		L	

(1) Logical Instructions

- **ANA R**
- **ANA M**

- **AND specified data in register or memory with accumulator.**
- **Store the result in accumulator (A).**

- Example: **ANA B, ANA M**


BEFORE EXECUTION
CY
AC
A
AA
B
0F
C
D
E
H
L
1010 1010=AAH
0000 1111=0FH
0000 1010=0AH
ANA B
A=A and R
AFTER EXECUTION
CY
0
AC
1
A
0A
B
0F
C
D
E
H
L
BEFORE EXECUTION
CY
AC
A
55
2050H
H
20
L
50
B3
0101 0101=55H
1011 0011=B3H
0001 0001=11H
ANA M
A=A and M
AFTER EXECUTION
CY
0
AC
1
A
11
2050H
H
20
L
50
B3

(2) Logical Instructions

- **ANI 8-bit data**

- **AND 8-bit data with accumulator (A).**
- **Store the result in accumulator (A)**

- Example: **ANI 3FH**

BEFORE EXECUTION

AFTER EXECUTION

1011 0011=B3H
0011 1111=3FH

0011 0011=33H

CY		AC	

A	B3

ANI 3FH
A=A and DATA(8)

CY	0	AC	1

A	33

(3) Logical Instructions

- **XRA Register (8-bit)**

- **XOR specified register with accumulator.**
- **Store the result in accumulator.**

- Example: **XRA C**

BEFORE EXECUTION

1010 1010=AAH
0010 1101=2DH

1000 0111=87H

AFTER EXECUTION

CY		AC	

A	AA		
B		C	2D
D		E	
H		L	

XRA C
A=A xor R

CY	0	AC	0

A	87		
B		C	2D
D		E	
H		L	

(4) Logical Instructions

- **XRA M**

- XOR data in memory (memory location pointed by H-L pair) with Accumulator.
- Store the result in Accumulator.

- Example: XRA M


BEFORE EXECUTION
0101 0101=55H
1011 0011=B3H
1110 0110=E6H
AFTER EXECUTION
CY
AC
A
55
H
20
L
50
2050H
B3
XRA M
A=A xor M
CY
0
AC
0
A
E6
H
20
L
50
2050H
B3

(5) Logical Instructions

- **XRI 8-bit data**

- XOR 8-bit immediate data with accumulator (A).
- Store the result in accumulator.

- Example: XRI 39H

1011 0011=B3H
0011 1001=39H

1000 1010=8AH

BEFORE EXECUTION

CY		AC	

A	B3

XRI 39H

A=A xor DATA(8)

AFTER EXECUTION

CY	0	AC	0

A	8A

(6) Logical Instructions

- **ORA Register**

- OR specified register with accumulator (A).
- Store the result in accumulator.

- Example: ORA B

1010 1010=AAH
0001 0010=12H

1011 1010=BAH

BEFORE EXECUTION

CY		AC	

A	AA		
B	12	C	
D		E	
H		L	

ORA B

A=A or R

AFTER EXECUTION

CY	0	AC	0

A	BA		
B	12	C	
D		E	
H		L	

(7) Logical Instructions

- **ORA M**

- OR specified register with accumulator (A).
- Store the result in accumulator.

- Example: ORA M

0101 0101=55H
1011 0011=B3H

1111 0111=F7H

BEFORE EXECUTION

CY		AC	

A	55

2050H

H	20	L	50

B3

ORA M

A=A or M

AFTER EXECUTION

CY	0	AC	0

A	F7

2050H

H	20	L	50

B3

(8) Logical Instructions

- **ORI 8-bit data**

- OR 8-bit data with accumulator (A).
- Store the result in accumulator.

- Example: ORI 08H

1011 0011=B3H
0000 1000=08H

1011 1011=BBH

BEFORE EXECUTION

CY		AC	

A	B3

ORI 08H

A=A or DATA(8)

AFTER EXECUTION

CY	0	AC	0

A	BB

(9) Logical Instructions

- **CMP Register**
- **CMP M**

- Compare specified data in register or memory with accumulator (A).
- Store the result in accumulator.

- Example: CMP D or CMP M

BEFORE EXECUTION

CY		Z	

A	B8

B		C	
D	B9	E	
H		L	

A>R: CY=0,Z=0
A=R: CY=0,Z=1
A<R: CY=1,Z=0

CMP D
A-R

AFTER EXECUTION

CY	0	Z	0

A	B8

B		C	
D	B9	E	
H		L	

BEFORE EXECUTION

CY		Z	

A	B8

2050H

H	20	L	50

B8

A>M: CY=0,Z=0
A=M: CY=0,Z=1
A<M: CY=1,Z=0

CMP M
A-M

AFTER EXECUTION

CY	0	Z	1

A	B8

2050H

H	20	L	50

B8

(10) Logical Instructions

- **CPI 8-bit data**

- Compare 8-bit immediate data with accumulator (A).
- Store the result in accumulator.

- Example: CPI 30H

A>DATA: CY=0,Z=0
A=DATA: CY=0,Z=1
A<DATA: CY=1,Z=0

BEFORE EXECUTION

CY		Z	

A	BA

CPI 30H
A-DATA

1011 1010=BAH

AFTER EXECUTION

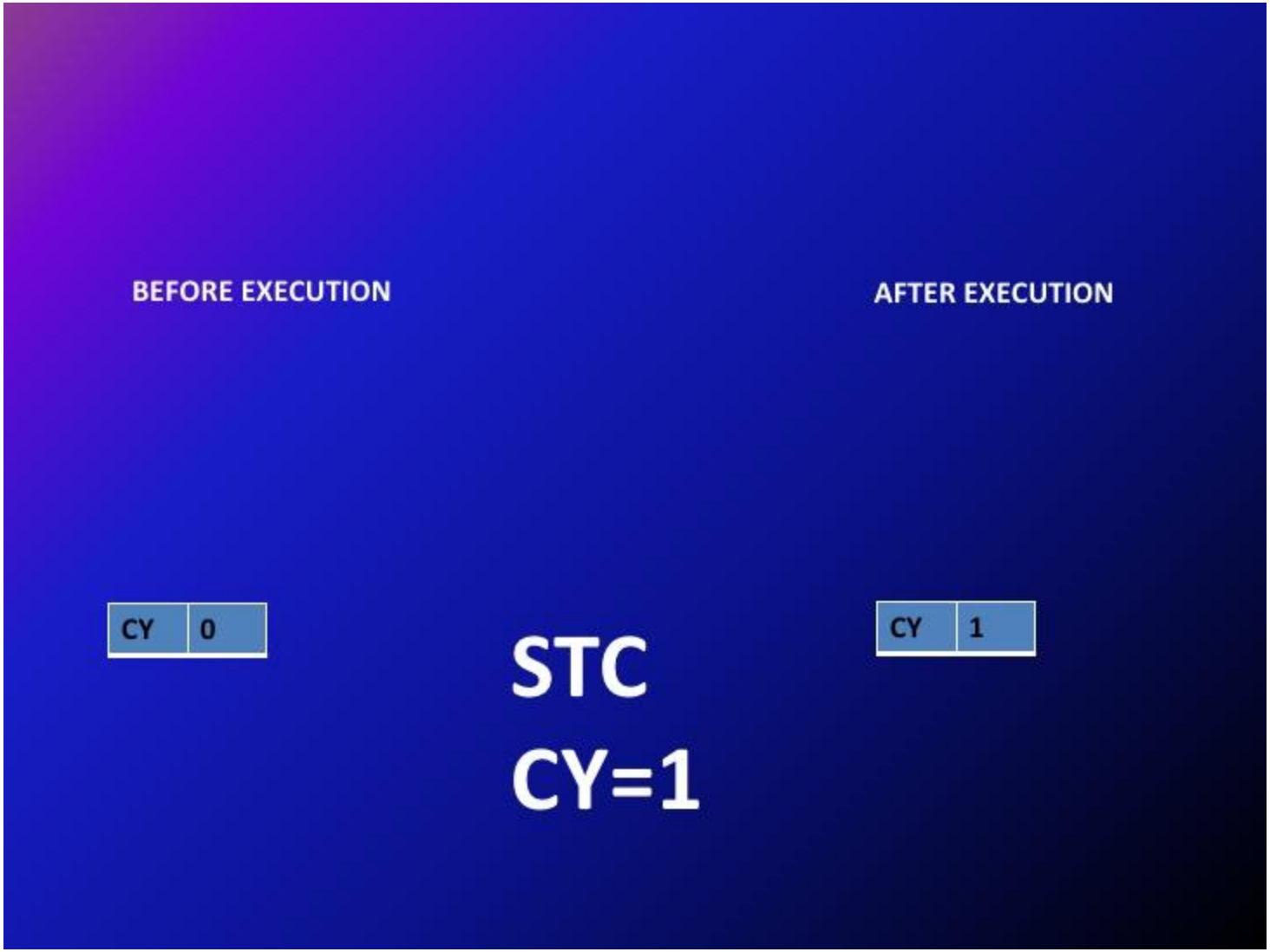
CY	0	AC	0

A	BA

(11) Logical Instructions

- **STC**

- It sets the carry flag to 1.
- Example: STC


BEFORE EXECUTION
AFTER EXECUTION
CY
0
CY
1
STC
CY=1

(12) Logical Instructions

- **CMC**

- It complements the carry flag.
- Example: CMC


BEFORE EXECUTION
AFTER EXECUTION
CY
1
CMC
CY
0

(13) Logical Instructions

- **CMA**

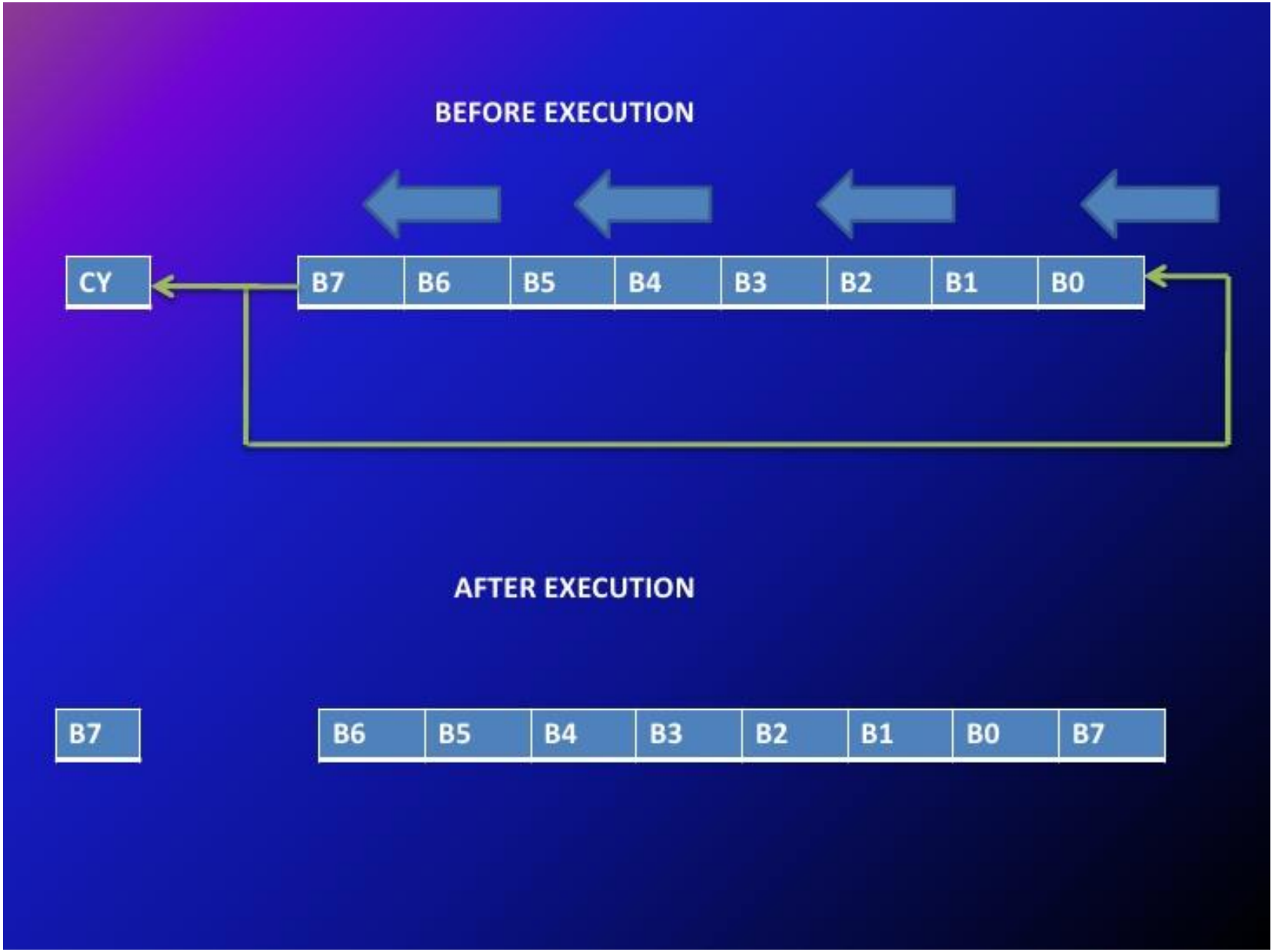
- It complements each bit of the accumulator.
- Example: CMA

(14) Logical Instructions

- **RLC**

- Rotate accumulator left
- Each binary bit of the accumulator is rotated left by one position.
- Bit D7 is placed in the position of D0 as well as in the Carry flag.
- CY is modified according to bit D7.

- Example: RLC.


BEFORE EXECUTION
CY
B7
B6
B5
B4
B3
B2
B1
B0
AFTER EXECUTION
B7
B6
B5
B4
B3
B2
B1
B0
B7

(15) Logical Instructions

- **RRC**

- Rotate accumulator right
- Each binary bit of the accumulator is rotated right by one
- position.
- Bit D0 is placed in the position of D7 as well as in the Carry flag.
- CY is modified according to bit D0.

- Example: RRC.

BEFORE EXECUTION

AFTER EXECUTION

(16) Logical Instructions

- **RAL**

- Rotate accumulator left through carry
- Each binary bit of the accumulator is rotated left by one position through the Carry flag.
- Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0.
- CY is modified according to bit D7.

- Example: RAL.


BEFORE EXECUTION
CY
B7
B6
B5
B4
B3
B2
B1
B0
AFTER EXECUTION
B7
B6
B5
B4
B3
B2
B1
B0
CY

(17) Logical Instructions

- **RAR**

- Rotate accumulator right through carry
- Each binary bit of the accumulator is rotated left by one position through the Carry flag.
- Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0.
- CY is modified according to bit D7.

- Example: RAR

BEFORE EXECUTION

AFTER EXECUTION

Branching Instructions

- The branch group instructions allows the microprocessor to change the sequence of program either conditionally or under certain test conditions. The group includes,
- (1) Jump instructions,
- (2) Call and Return instructions,
- (3) Restart instructions,

(1) Branching Instructions

- **JUMP ADDRESS**

- BEFORE EXECUTION

JMP 2000H

AFTER EXECUTION

- Jump unconditionally to the address.
- The instruction loads the PC with the address given within the instruction and resumes the program execution from specified location.
- Example: JMP 200H

Conditional Jumps

Instruction Code	Decription	Condition For Jump
JC	Jump on carry	CY=1
JNC	Jump on not carry	CY=0
JP	Jump on positive	S=0
JM	Jump on minus	S=1
JPE	Jump on parity even	P=1
JPO	Jump on parity odd	P=0
JZ	Jump on zero	Z=1
JNZ	Jump on not zero	Z=0

(2) Branching Instructions

- **CALL address**

- Call unconditionally a subroutine whose starting address given within the instruction and used to transfer program control to a subprogram or subroutine.

- Example: CALL 2000H

Conditional Calls

Instruction Code	Description	Condition for CALL
CC	Call on carry	CY=1
CNC	Call on not carry	CY=0
CP	Call on positive	S=0
CM	Call on minus	S=1
CPE	Call on parity even	P=1
CPO	Call on parity odd	P=0
CZ	Call on zero	Z=1
CNZ	Call on not zero	Z=0

(3) Branching Instructions

- **RET**

- Return from the subroutine unconditionally.
- This instruction takes return address from the stack and loads the program counter with this address.

- Example: RET

BEFORE EXECUTION

SP	27FD
PC	

27FDH	00
27FEH	62
27FFH	

RET

AFTER EXECUTION

SP	27FF
PC	6200

27FDH	00
27FEH	62
27FFH	

(4) Branching Instructions

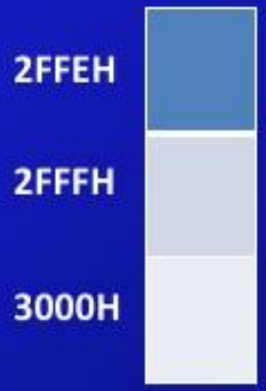
- **RST n**

- Restart n (0 to 7)
- This instruction transfers the program control to a specific memory address. The processor multiplies the RST number by 8 to calculate the vector address.

- Example: RST 6

BEFORE EXECUTION

SP	3000
PC	2000



AFTER EXECUTION

SP	2999
PC	0030

SP-1

RST 6

ADDRESS OF THE NEXT INSTRUCTION IS 2001H

Vector Address For Return Instructions

Instruction Code	Vector Address
RST 0	0*8=0000H
RST 1	0*8=0008H
RST 2	0*8=0010H
RST 3	0*8=0018H
RST 4	0*8=0020H
RST 5	0*8=0028H
RST 6	0*8=0030H
Rst 7	0*8=0038H

(1) Control Instructions

- **NOP**

- No operation
- No operation is performed.
- The instruction is fetched and decoded but no operation is executed.

- Example: NOP

(2) Control Instructions

- **HLT**

- Halt
- The CPU finishes executing the current instruction and halts any further execution.
- An interrupt or reset is necessary to exit from the halt state.

- Example: HLT

PUSH: - This instruction pushes the register pair onto stack. The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the highorder register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location.

Eg: - PUSH B
PUSH A
Parameters:
Addressing mode:Register
Bytes:1
Machine cycle (3) OFMC*, MW,MW
Flags/;None

POP: - This instruction pop off stack to register pair. The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1

Eg: - POP H (Normal OFMC) 4 T-States
POP A

- Write a program in assembly of 8085 to read and compliment the contents of the flag register.
- Save PSW
- Restore in H
- Transfer contents of L into A
- Compliment the contents of ACC
- Transfer A into L
- Save H
- Restore PSW
- HLT

Example Data Transfer (Copy) Operations / Instructions

– **Load** a 8-bit number 4F in register B	**MVI B, 4FH**
– **Copy** from Register B to Register A	**MOV A,B**
– **Load** a 16-bit number 2050 in Register pair HL	**LXI H, 2050H**
– **Copy** from Register B to Memory Address 2050	**MOV M,B**
– **Copy** between Input/Output Port and Accumulator	**OUT 01H** **IN 07H**

Program 8085 in Assembly language to add two 8-bit numbers and store 8-bit result in register C.

1. Analyze the problem
   - Addition of two 8-bit numbers to be done
2. Program Logic
   - Add two numbers
   - Store result in register C
   - Example

```
  10011001   (99H) A
+ 00111001   (39H) D
  11010010   (D2H) C
```

5. Assembly Language Program

1. Get two numbers
   - Load 1st no. in register D
   - Load 2nd no. in register E

   MVI D, 2H
   MVI E, 3H
4. Add them
   a) Copy register D to A
   b) Add register E to A

   MOV A, D
   ADD E
7. Store result
   a) Copy A to register C

   MOV C, A
9. Stop
   a) Stop processing

   HLT

Program 8085 in Assembly language to add two 8-bit numbers. Result can be more than 8-bits.

1. Analyze the problem

- Result of addition of two 8-bit numbers can be 9-bit
- Example

```
  10011001   (99H) A
+ 10011001   (99H) B
 100110010 (132H)
```

- The 9th bit in the result is called CARRY bit.

5. Assembly Language Program

- Load registers D, E

- Copy register D to A
- Add register E to A

- Copy A to register C

- Use Conditional Jump instructions

- Clear register B
- Increment B

- Stop processing

```
        MVI D, 2H
        MVI E, 3H
        MOV A, D
        ADD E
        MOV C, A
        JNC END
        MVI B, 0H
        INR B
END:    HLT
```

- Write a Program to exchange contents of memory location using direct addressing.

```
LDA 9000
MOV C,A
LDA 9050
STA 9000
MOV A,C
STA 9050
HLT
```

- Write a Program to add two 16 bit numbers using add and adc instructions.

LHLD 9000;Load first 16 bit number in HL
XCHG; Save the first 16 bit number in DE
LHLD 9002; Load the second 16 bit number in the HL Pair.
MOV A,E ;Move lower order byte of first to acc
ADD L;Add Lower order bytes.
MOV L,A; Store the result in L
MOV A,D;Mov higher order byte of the first number in Acc
ADC H;Add the higher order byte along with the carry

- Write a Program to add two 16 bit numbers using DAD instruction

LHLD 9000;Load first 16 bit number in HL
XCHG; Save the first 16 bit number in DE
LHLD 9002; Load the second 16 bit number in the HL Pair.
DAD D; Add DE and HL reg Pair
SHLD 9004 ; Store the 16 bit result in 9004 and 9005
HLT

- Write a Program to find one’s Compliment of a number

LDA 9000;Load the number to be complimented
CMA ; Compliment the number
STA 9001; Store the result
HLT

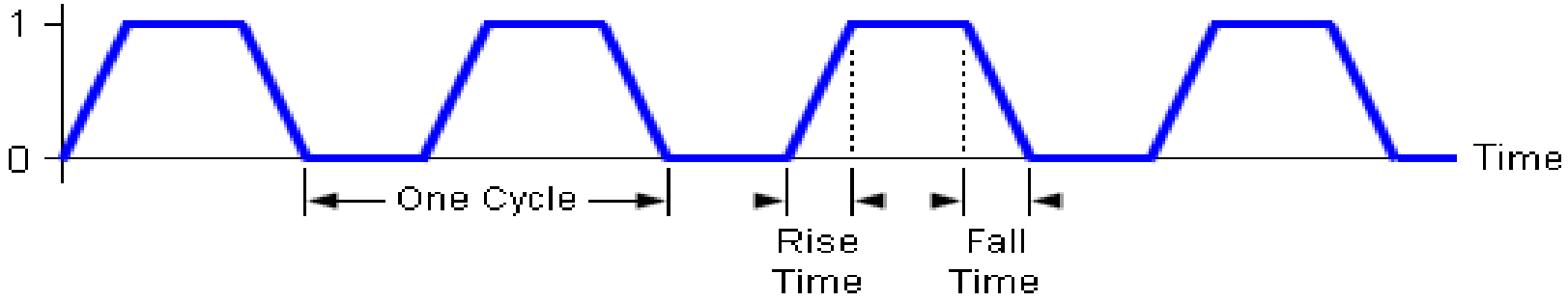
- Write a Program to find Two's Compliment of a number

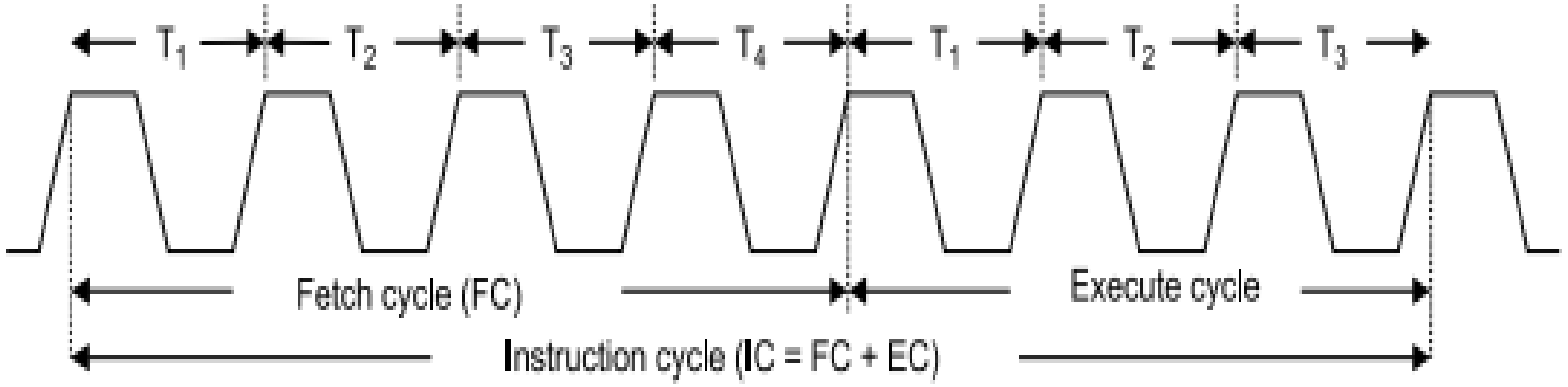
```
LDA 9000;Load the number to be complimented
CMA ; Compliment the number
ADI 01; Add one to the compliment
STA 9001; Store the result
HLT
```

- Write a Program to shift an 8-bit number right by 4-bits.

```
LDA 9000;Load the number to be complimented
RRC;Rotate Right without Carry
RRC;Rotate Right without Carry
RRC;Rotate Right without Carry
RRC;Rotate Right without Carry
STA 9001; Store the result
HLT
```

- Write a program to transfer 10 bytes of data from one memory block to another memory block. (Source 9000 and destination 9100)
- LXI H 9000; Initialize source memory pointer
- LXI D 9100;Initialize target memory pointer
- MVI B, 0A; initialize counter to count 10 bytes
- LOOP:MOV A, ; Get data from source
- STAX D; Store it in Destination Address
- INX H ; Increment the source address
- INX D ; Increment the Destination address
- DCR B; Decrement the counter
- JNZ Loop; if the counter is not zero jump to loop
- HLT


1
0
Time
One Cycle
Rise Time
Fall Time


T1
T2
T3
T4
T1
T2
T3
Fetch cycle (FC)
Execute cycle
Instruction cycle (IC = FC + EC)

MPU Communication and Bus Timing

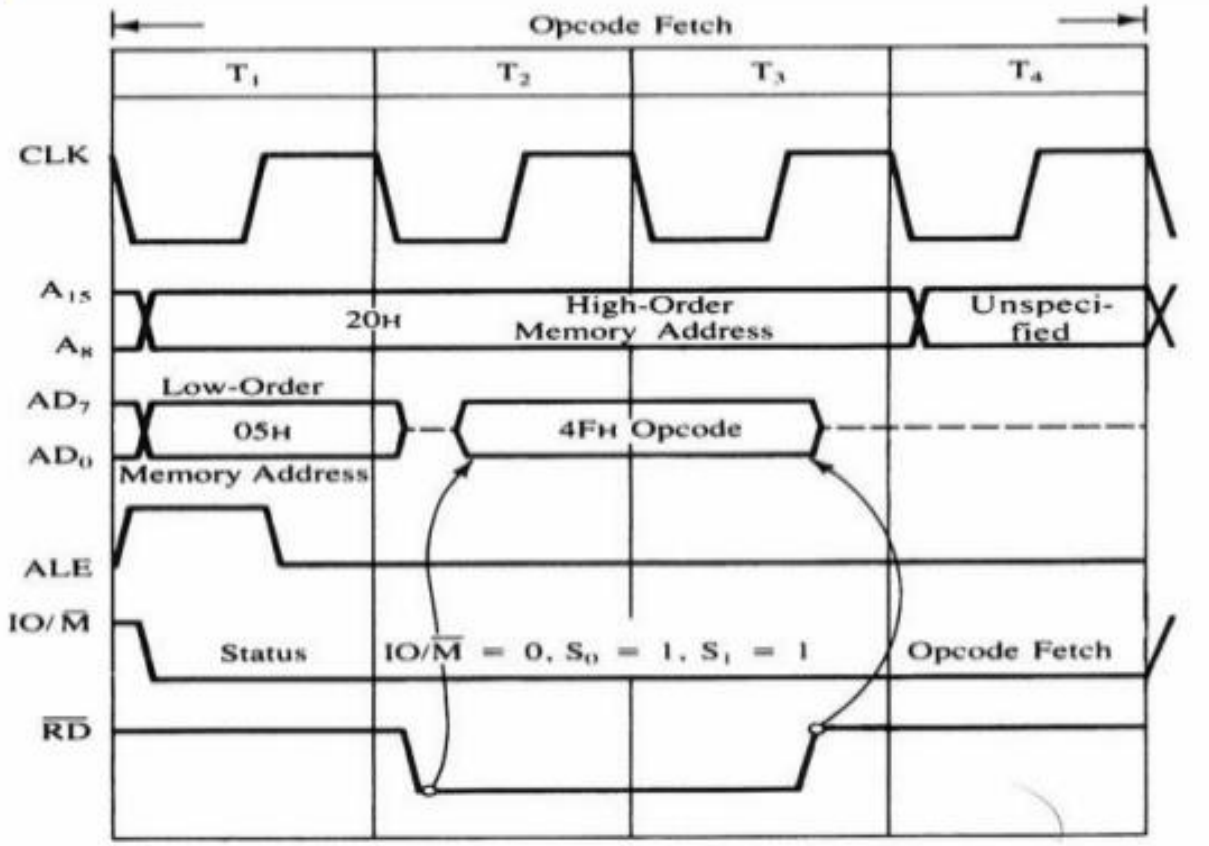
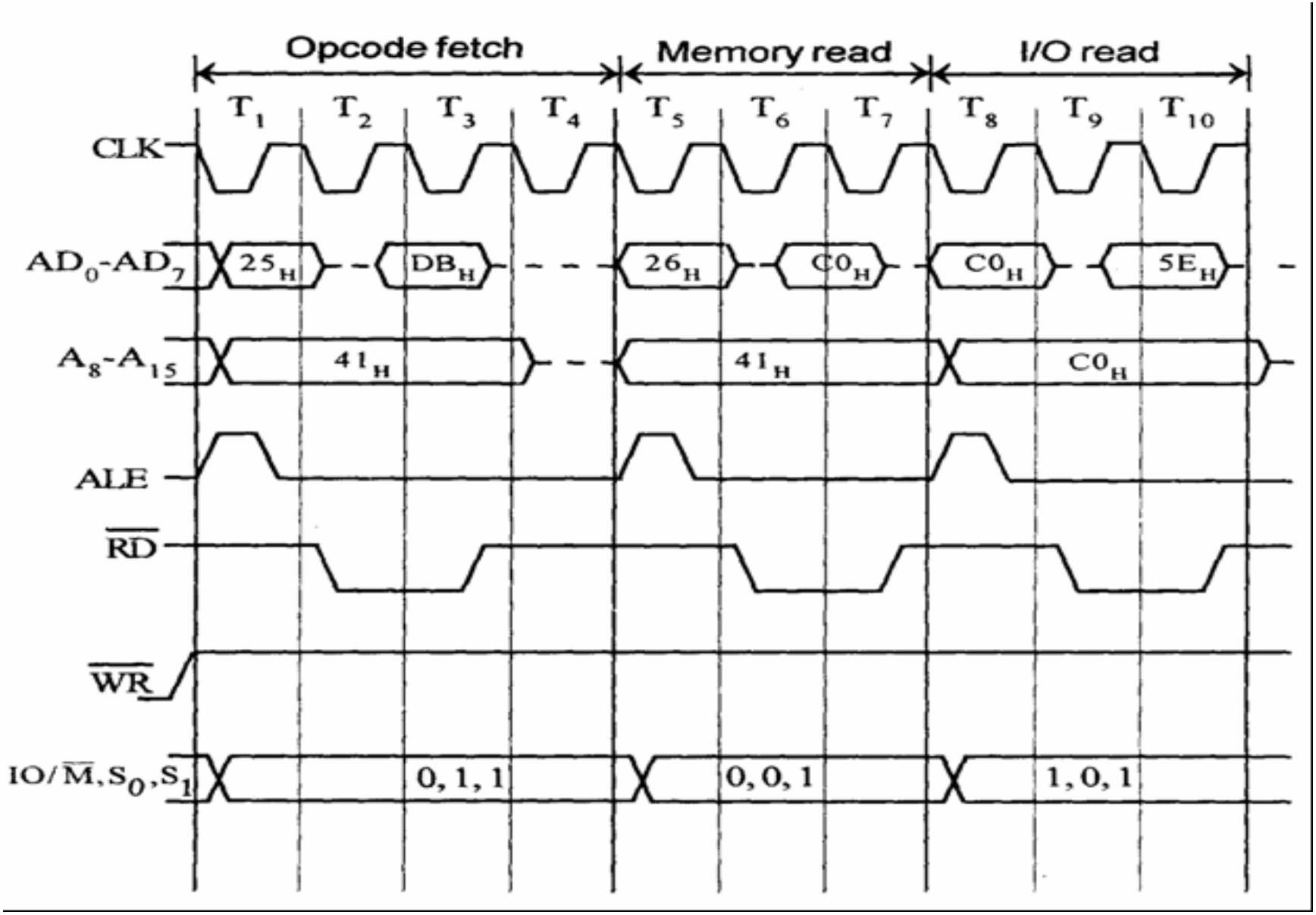
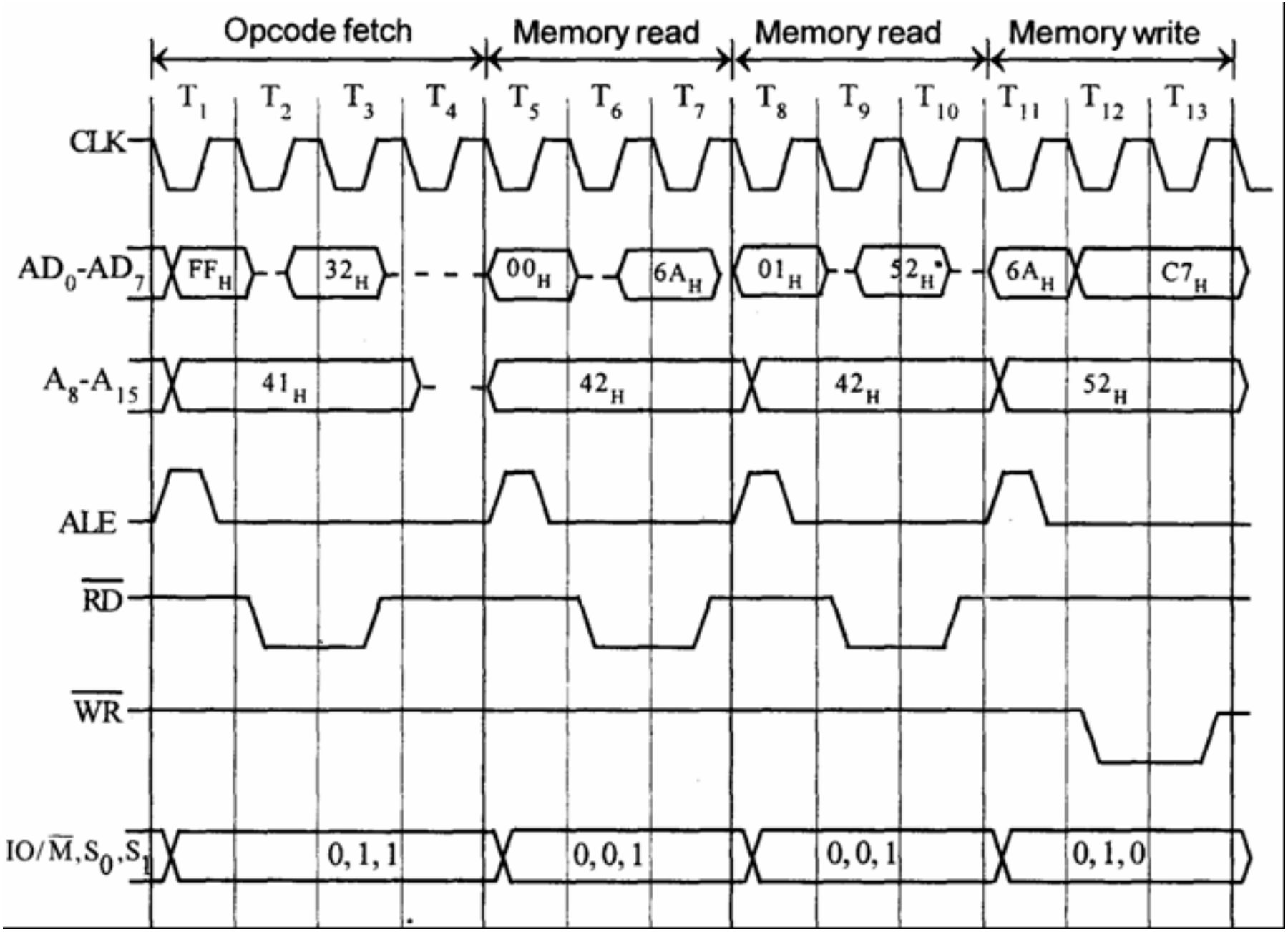


Figure 4: 8085 timing diagram for Opcode fetch cycle for MOV C, A .


Opcode fetch
Memory read
I/O read
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
CLK
AD0-AD7
25H
DBH
26H
C0H
C0H
5EH
A8-A15
41H
41H
C0H
ALE
RD
WR
IO/M, S0, S1
0, 1, 1
0, 0, 1
1, 0, 1


Opcode fetch
Memory read
Memory read
Memory write
T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13
CLK
AD0-AD7
FFH
32H
00H
6AH
01H
52H
6AH
C7H
A8-A15
41H
42H
42H
52H
ALE
RD
WR
IO/M, S0, S1
0, 1, 1
0, 0, 1
0, 0, 1
0, 1, 0

Unit III

Hardware Interfacing , Data Transfer Schemes& Support Chips

Data transfer mechanisms

- Data transfer in any microprocessor based system is essential.
- The data transfer can take place between processor and memory or between processor and input device or between processor and output device or between memory and input device or between memory and output device.
- Data can be transferred in many different ways in all these data transfers.

Data transfer mechanisms contd..

- The data transfer mechanism differs from each other based on issues like the addressing of the device, amount of data transferred, way of data transfer, and interaction among the devices.
- The data transfer mechanism is divided into many types as followed

a) Based on the addressing of the device

i) I/O mapped I/O access

ii) Memory mapped I/O access

Types of Data Transfer contd..

b) Based on the program and hardware involved

i) Programmed data transfer

- Polled mode of data transfer
- Interrupt driven data transfer

ii) Direct memory access

- Burst mode
- Cycle stealing mode

Types of Data Transfer contd..

c) Based on the way how data is transferred and accessed

i) Parallel data transfer

- Simple Data transfer
- Handshake mode data transfer

ii) Serial data transfer

- Synchronous data transfer
- Asynchronous data transfer

Memory mapped and I/O mapped data transfer

- In I/O mapped device data transfer method, the I/O devices are treated separately from memory.
- Separate address range will be assigned for the input and output devices.
- The control signals for read and write from I/O devices are completely separate from the control signals used for memory access.

Memory mapped and I/O mapped data transfer

- The microprocessor will have separate instructions for Input and output device access such as IN instruction and OUT instruction of 8085.
- As the memory and I/O device access is completely different, a single address can be assigned to both an I/O device and a memory location.

Memory Mapped I/O

- In memory mapped I/O, each input device or output device is treated as if it is a memory location.
- The control signal for read and write operation of I/O device is same as that of memory chips.
- Each input or output device is identified by a unique address in the memory address range.

Memory Mapped I/O

- All the memory related instructions used to read data from memory are used to access input and output device.
- Since the I/O devices use some of the memory address space, the maximum memory addressing capacity will be reduced in a microprocessor based system.

Serial No.	Characterstics	Memory Mapped I/O	I/O Mapped I/O
1	Device length	16 In this Device address is 16 bit, lines A0-A15 are used for generating the device address	8 In this only either A0-A7 are used or A8-A15 are used for address generation
2	Control Signals	MEMR' and MEMW' signals are used to control the read and write operations	IOR' and IOW' are used for read and write operations
3	Instructions Supported	Instructions like LDA,STA,LDAX RP,STAX RP, MOV M,R, MOV R,M	IN and OUT two distinct instructions supported.
4	Data Transfer	IT is possible between any Register and Device.	Data Transfer is possible between I/O device and ACC.
5	Maximum No. of I/O Devices Supported	2^16 Devices are supported	2 ^ 8 Devices supported
6	Execution Speed	7 T-states / 13 T-states	10 T-States
7	Hardware Requirements	Complex as it has to decode 16 bit Address	Less complex as it has to decode only 8 bit Address

Programmed data transfer

- Programmed data transfer is written and controlled by programmer and executed by the processor.
- The data transfer between processor and I/O devices or vice versa takes place by executing the corresponding instruction.
- Programmed I/O data transfers are identical to read and write operations for memories or device registers.
- An example of programmed I/O is a device driver writing one data byte at a time directly to the device's memory.

Programmed data transfer contd..

- The execution of programmed data transfer can take place at predefined period determined by the programmer.
- Based on the time of execution of the data transfer instruction, the programmed data transfer is divided into two types namely

a) Polled mode of data transfer

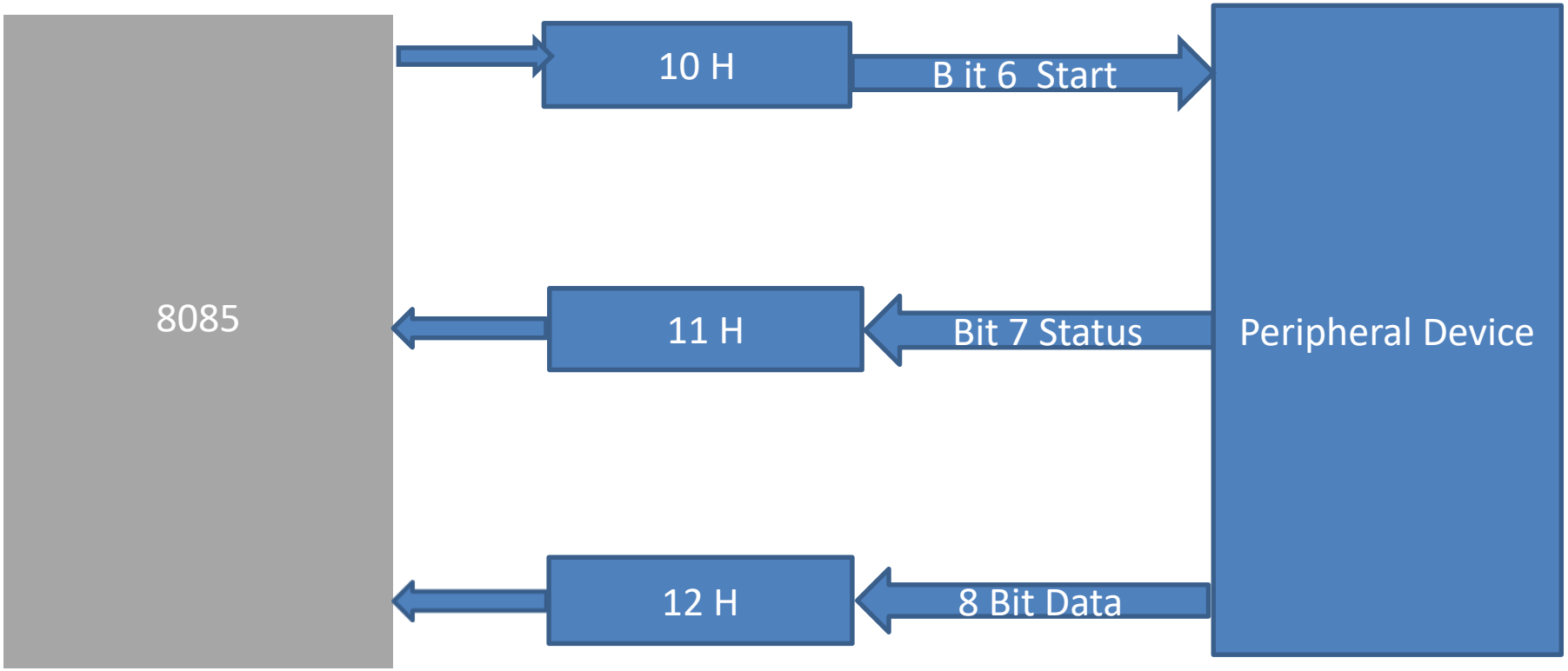
b) Interrupt driven data transfer

Polled mode of data transfer

- In polled mode of data transfer, the data is read from an input device when the processor or CPU is ready and executes the data transfer instruction.
- If the input device is not ready the processor will wait until the device is ready with data.

- Similarly, the data is written into an output device by the processor when it executes the data ‘write’ instruction to the corresponding output device.
- The program is written in such a way that the processor will wait in a loop until the output device is ready to receive data.
- As it can be seen clearly, the processor time is wasted in this polled mode of data transfer as it waits for the device to be ready.

Programmed Input



- Assumptions:

1)8085 is connected to a P.D through ports 10,11, and 12 .
2)Port 10h is used to send start bit to PD
3)Port 11h is used for sending the status bit.
4)Port 12h is used to input one byte of data.

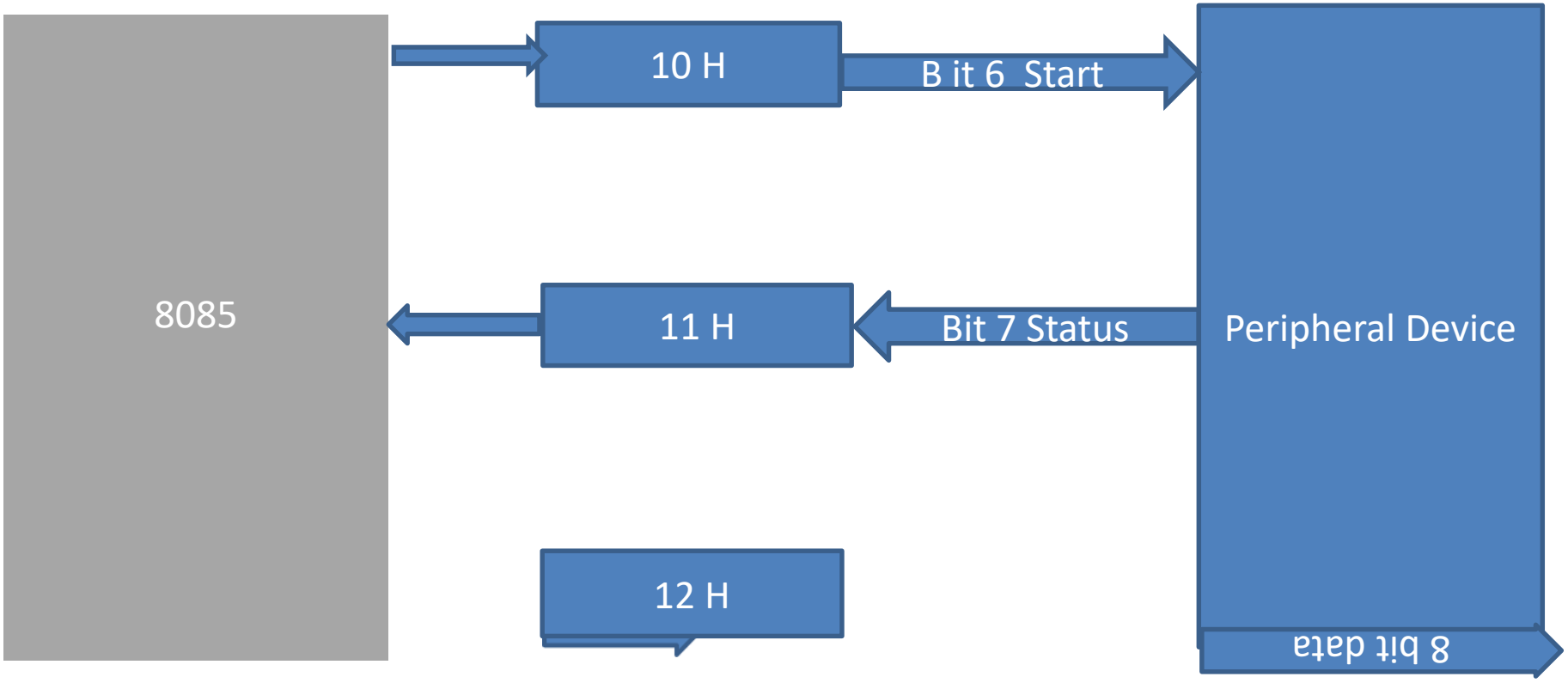
Working:

When the CPU wants to transfer data byte,it sends a high start signal to the PD through port 10h.After this the processor waits for the PD to get ready and t he CPU continuously monitors the status line of the PD through Port 11h.Once the device gets ready it sends a high status signal through status line by making it high(bit

Program to input 100 bytes of data .

```
LXI H, 2000;Initialize HL Pointer
MVI C, 64 h;Initialize Counter (100 Dec)
START:MVI A, 40h;Set START bit
Out 10h; Send high Start bit
Wait:IN 11h;Get status bit
ANI 80h;Isolate Status bit
JZ wait;Wait if device not ready
IN 12h;Input Data
MOV M,A;Store data
INX H;Update HL pointer
MVI A,00h;Reset Start bit
OUT 10h;Send low start bit
DCR C; Decrement counter
```

Programmed Output



- Assumptions:

1)8085 is connected to a P.D through ports 10,11, and 12 .

2)Port 10h is used to send start bit to PD

3)Port 11h is used for sending the status bit.

4)Port 12h is used to input one byte of data.

Working:

When the CPU wants to transfer data byte,it will latch 8 bit data into port 12 h. A high start signal to the PD through port 10h indicating that valid 8 bit data is waiting at port 12h.After the PD has loaded the data it sends a high status signal indicating that the data byte is successfully loaded.

Program to output 10 bytes of data .

```
LXI H, 3000;Initialize HL Pointer
MVI C, 0A h;Initialize Counter (10 Dec)
LOOP:MOV A,M; Get data byte
OUT 12h; Latch data in port 12h
MVI A,40H;Set start bit
OUT 10h;send high start bit
Wait:IN 11h;Get status bit
ANI 80h;Isolate Status bit
JZ wait;Wait if device not ready
INX H;Update pointer
MVI A,00h;Reset Start bit
OUT 10h;Send low start bit
DCR C; Decrement counter
```

8085 is receiving data @ 100 bytes per second from a PD. For the programmed input discussed earlier, calculate the wasted by CPU in waiting for the PD to get ready(assume 3 MHZ internal clock)

T=1/100 HZ 0.01 sec
It takes 0.01 sec for PD to get ready
Useful time does not include time for start bit and waiting in loop.

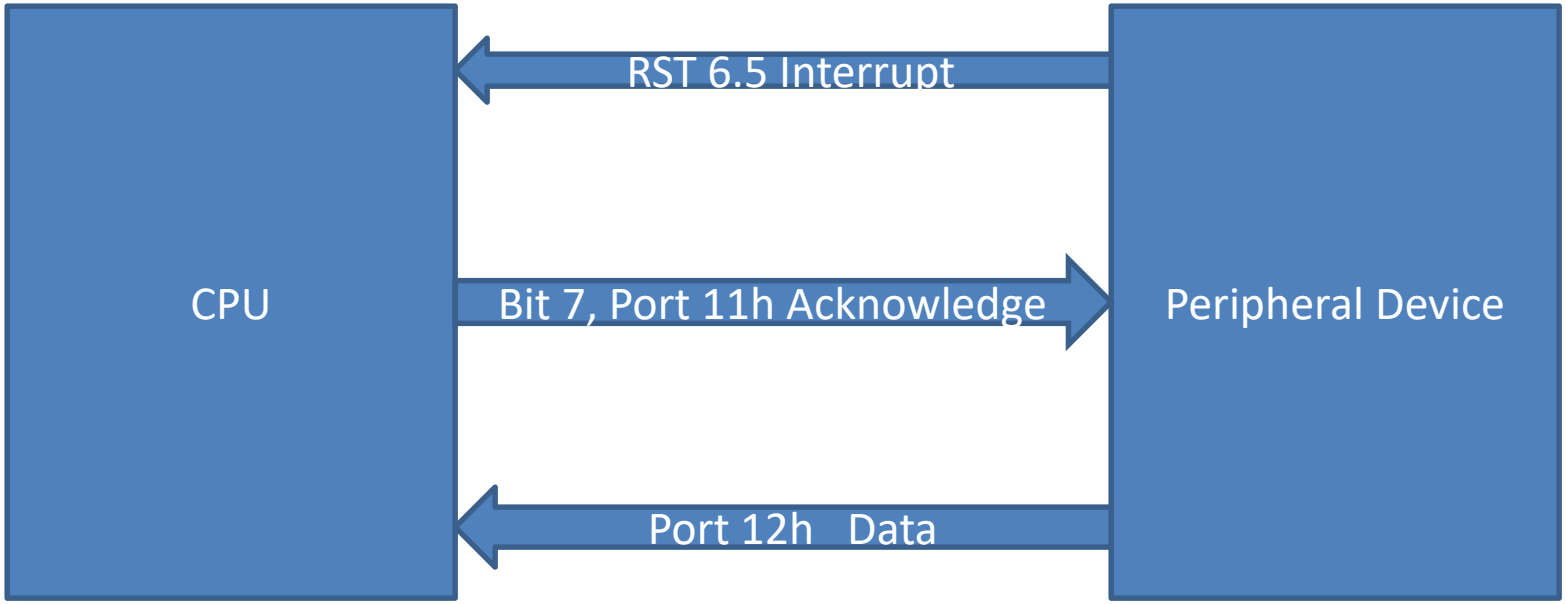
Useful T-states are:-
IN 12H
MOV M,A
INX H
DCR C
JNZ LOOP

Useful CPU time is 330 nanosec * 37= 12.21 micro sec
0.01 sec =10,000 micro sec
Useful time 0.12 %
Wasted time 99.88 %

Interrupt driven data transfer

- In interrupt driven data transfer, the data is read from the input device only when the input device is ready with data.
- When the device is ready, it will give a signal to interrupt the processor indicating that the data is ready.
- In the interrupt service routine, the program is written to read the data from the corresponding input device.
- Similarly, the output device will also give an interrupt to the processor when it can accept a data.
- The programmers have to write an interrupt service routine for data transfer to the corresponding Input/output device.

Interrupt Driven I/O



- The above fig shows PD connected to RST 5.5 Interrupt.
- After the CPU receives a data word in port 12h, it sends a high acknowledge bit (bit 7) through port 11h back to the PD.
- The starting address in the RST 5.5 (002Ch) is F100h, the starting address of the Service subroutine which is used to input one byte of data.

Service Subroutine:-

```
F100    PUSH PSW    ; Save Acc and Flags
F101    PUSH H      ;Save HL Contents
F102    IN 12h  ;Input data from device
F104    LXI H,3000  ;Set Pointer
F107    MOV M,A     ;Store data
F108    MVI A,80    ;Set Acknowledge bit
F10A    OUT 11      ;Acknowledge Data Arrival
F10C    POP H  ;Restore HL
F10D    POP PSW     ;Restore Acc and Flag
F10E    EI          ;Enable Interrupts
F10F    RET         ; Return
```

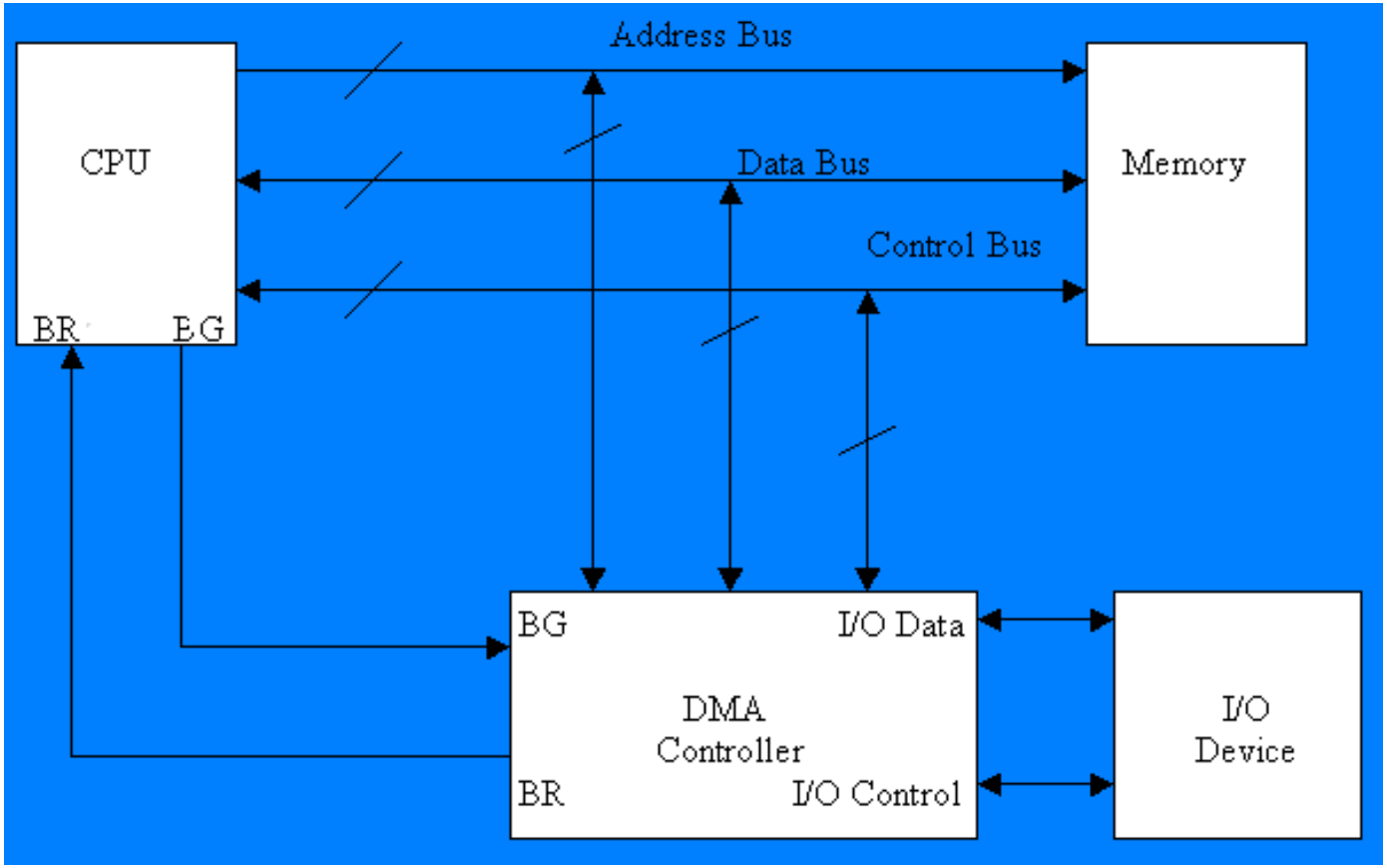
Direct Memory Access (DMA)

- In programmed I/O data transfer, processor is actively involved in the entire data transfer process. The processor is tied up and processor time is wasted. So, the data transfer rate is limited.
- To overcome these disadvantages of programmed data transfer and to increase the speed of data transfer, DMA method of data transfer is used.

Direct Memory Access

- Introduction
  - An important aspect governing the Computer System performance is the transfer of data between memory and I/O devices.
  - The operation involves loading programs or data files from disk into memory, saving file on disk, and accessing virtual memory pages on any secondary storage medium.
  - BR (HOLD) Bus Request
  - BG(HOLDA) Bus Grant

Computer System with DMA



Implementing DMA in a Computer System

- A DMA controller implements direct memory access in a computer system.
- It connects directly to the I/O device at one end and to the system buses at the other end. It also interacts with the CPU, both via the system buses and two new direct connections(HOLD and HOLDA)

Data Transfer using DMA Controller

- To transfer data from an I/O device to memory, the DMA controller first sends a Bus Request to the CPU by setting BR to 1. When it is ready to grant this request, the CPU sets it’s Bus grant signal, BG to 1.
- The CPU also tri-states it’s address,data, and control lines thus truly granting control of the system buses to the DMA controller.
- The CPU will continue to tri-state it’s outputs as long as BR is asserted.

Process of DMA Transfer

- To initiate a DMA transfer, the CPU loads the address of the first memory location of the memory block (to be read or written from) into the DMA address register. It does this via an I/O output instruction, It then writes the no. of bytes to be transferred into the DMA count register in the sane manner.

DMA Transfer Modes

Modes vary by how the DMA controller determines when to transfer data, but the actual data transfer process is the same for all the modes.

- BURST mode
  - Sometimes called Block Transfer Mode.
  - An entire block of data is transferred in one contiguous sequence. Once the DMA controller is granted access to the system buses by the CPU, it transfers all bytes of data in the data block before releasing control of the system buses back to the CPU.
  - This mode is useful for loading programs or data files into memory, but it does render the CPU inactive for relatively long periods of time.

- CYCLE STEALING Mode
  - Viable alternative for systems in which the CPU should not be disabled for the length of time needed for Burst transfer modes.
  - DMA controller obtains access to the system buses as in burst mode, using BR & BG signals. However, it transfers one byte of data and then deasserts BR, returning control of the system buses to the CPU. It continually issues requests via BR, transferring one byte of data per request, until it has transferred it's entire block of data.

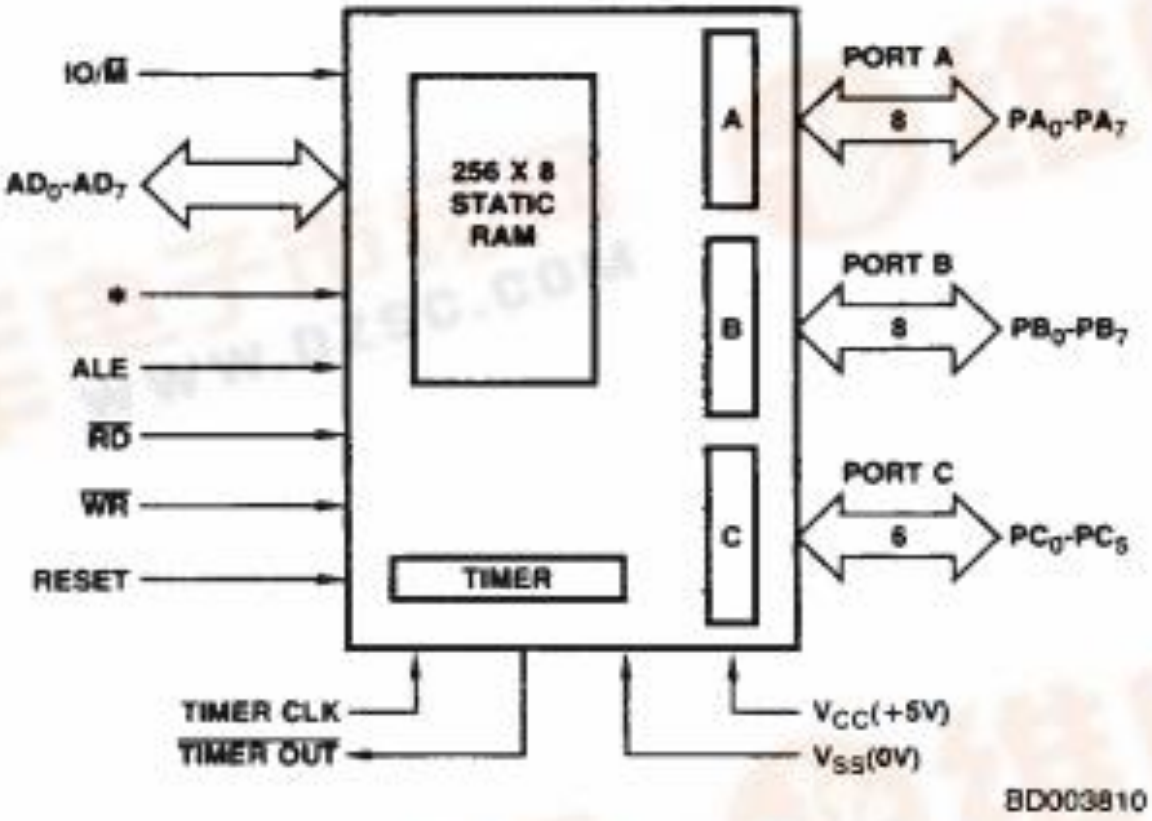
- By continually obtaining and releasing control of the system buses, the DMA controller essentially interleaves instruction & data transfers. The CPU processes an instruction, then the DMA controller transfers a data value, and so on.
- The data block is not transferred as quickly as in burst mode, but the CPU is not idled for as long as in that mode.
- Useful for controllers monitoring data in real time.

- TRANSPARENT Mode
  - This requires the most time to transfer a block of data, yet it is also the most efficient in terms of overall system performance.
  - The DMA controller only transfers data when the CPU is performing operations that do not use the system buses.

8155/8156 support chip

- Characteristics of 8155/8156
- 256 Bytes of memory organized as 8-bit words
- Single +5 V supply
- 2-Programmable 8 –bit I/O ports
- 1-Programmable 6-bit I/O port
- Programmable 14 bit down Timer
- 8155 is active low whereas 8156 is active high

Block Diagram of 8155/8156



*8155H = $\overline{CE}$, 8156H = CE

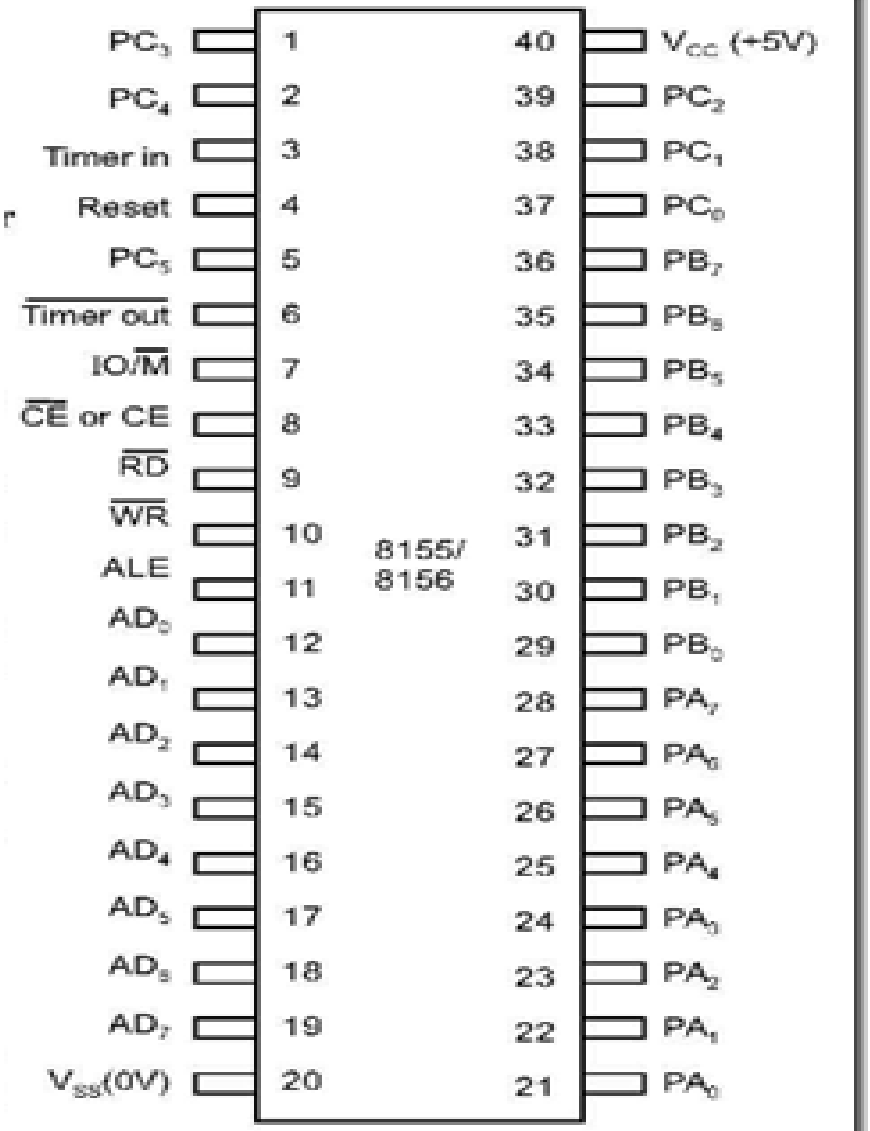


Fig. 9b.1: Pin diagram of 8155 (*Source:* Intel Corporation)

The 8155 is functionally divided into two Sections. These are:

(i) Memory Section,
(ii) I/O ports, and Timer Section.

(i) Memory

The 8155 RAM is designed with 2048 static memory cells arranged as a 256 × 8 matrix. The access time is of the order of 400 ns so that it can be used without the use of WAIT cycles.

(ii) I/O Ports and Timer

This consists of two 8-bit ports (A and B), one 6-bit port C, a Command/Status register, and two timer registers.

Each of the three ports can be individually programmed for simple I/O operation, or, Port A and/or Port B can be programmed for I/O operation in the handshake mode, using pins of Port C for handshaking.

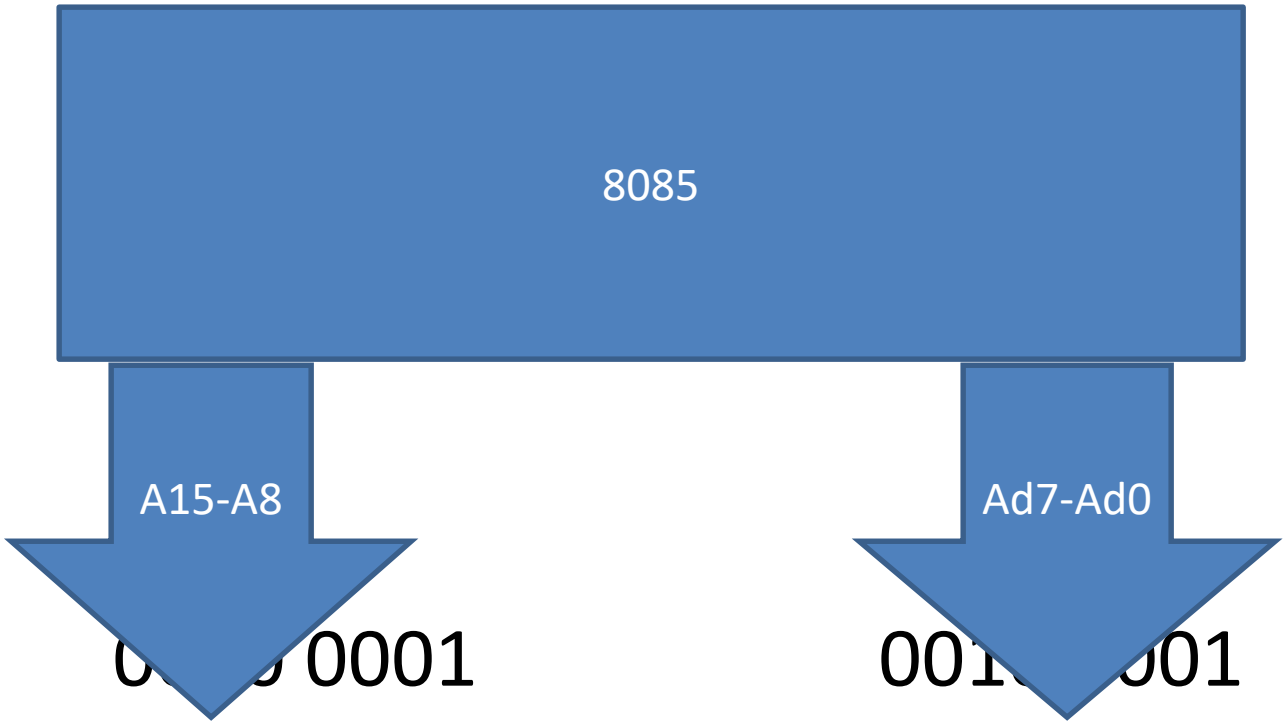
The Command register is used to program the ports and the Timer for various modes of operation. The CPU can *read* the Status word in order to check the status of the Timer and I/O ports.

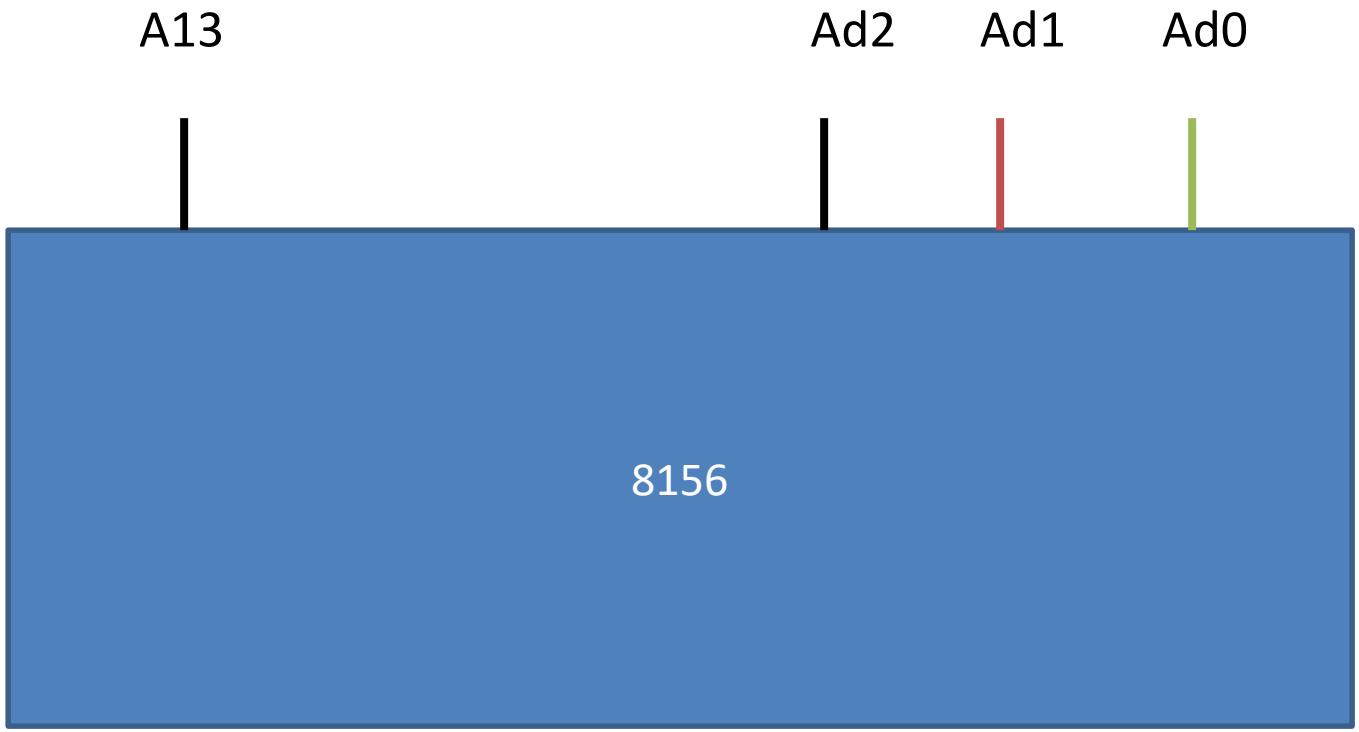
The Timer consists of a 14-bit counter which is loaded with a count. The TIMER IN pulses are counted and a square wave(s) or a pulse(s) is/are sent out on the $\overline{\text{TIMER OUT}}$ pin when Terminal Count (TC) is reached.

The Command/Status (C/S) register, the three ports, and the timer registers are accessed only when IO/$\overline{M}$ is high *and* the $\overline{CE}$ is low. One of these alternatives is selected depending on the I/O address input to the device. The 8155 has an internal decoder which uses the A_2 A_1 A_0 bits

Port Numbers for 8156

- Port numbers are duplicated on buses



- Active bits during I/O addressing
- 

- When an IN or OUT instruction is executed, 8085 duplicates the port number on the address bus and address-data bus, for example during the execution of
- IN 21 h, 8085 sends out
- 0010 0001 on its address and address data bus.
- As an equation the above can be written as

AD15-AD8=AD7-AD0

- During the execution of IN and OUT instructions , 8156 uses only three lower address bits AD2,AD1,and AD0), the following table shows the bit combination.

AD2	AD1	AD0	Location
0	0	0	Command and Status Register
0	0	1	Port A
0	1	0	Port B
0	1	1	Port C
1	0	0	Lower 8 bits of the timer
1	0	1	2 bits of timer mode and upper 6 bits of timer

Determination of Port Addresses using duplication Equation

- Port address of **Port A**:
- A13 must be in high state for chip to be in active state, bit pattern for Port A is 001
- AD7-AD0= XXXX X001 (for port A)
- AD15-AD8=XX1X XXXX (As A13 must be high for chip to be in active state),the above can be written as (using duplication equation)

XX1X XXXX=XXXX X001, pattern must repeat itself hence

XX1X X001=XX1X X001, solve the above equation by putting all Don't cares as 0, so the address obtained is

Programming the I/O ports

Command Register

D7	D6	D5	D4	D3	D2	D1	D0

- D0-- > Port A
- D1-- > Port B
- D2&D3 -- > Port C
- D4 -- > Interrupt Enable at A
- D5 -- > Interrupt Enable at B
- D6 & D7 -- > Timer

- Port A

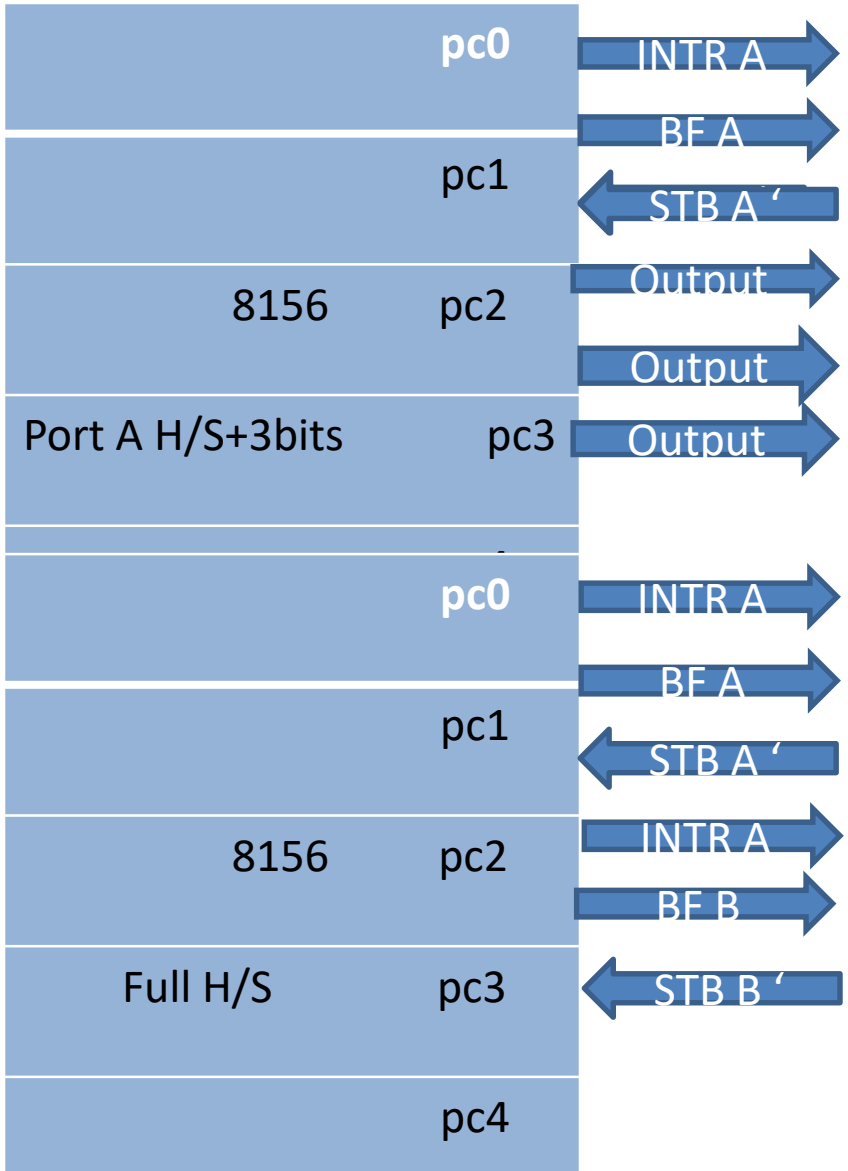
D0	Effect
0	Input
1	output

Port B

D1	effect
0	Input
1	Output

- Port C

D3	D2	Effect
0	0	Input Port (6 bits)
0	1	Port A Handshaking and output (3 bits)
1	0	Port A and B Full Handshaking
1	1	Output port (6 bits)


pc0
INTR A
BF A
pc1
STB A '
8156
pc2
Output
Output
Port A H/S+3bits
pc3
Output
pc0
INTR A
BF A
pc1
STB A '
8156
pc2
INTR A
BF B
Full H/S
pc3
STB B '
pc4

- When D3 and D2 is 01 half of the port C is H/S and other half as output.PC0 sends out a bit called INTR A, this stands for Interrupt at PORT A.
- PC1 sends out BF A, this stands for buffer register full at port A.
- PC2 receives the bit STB A’, this is a strobe signal for PORT A

Show the instructions needed to make A as input Port , B as output port, C as full H/S, the A and B interrupts should be enabled and timer bits reset to 00.

0011 1010 ---- > 3A

MVI A,3A

OUT 20h

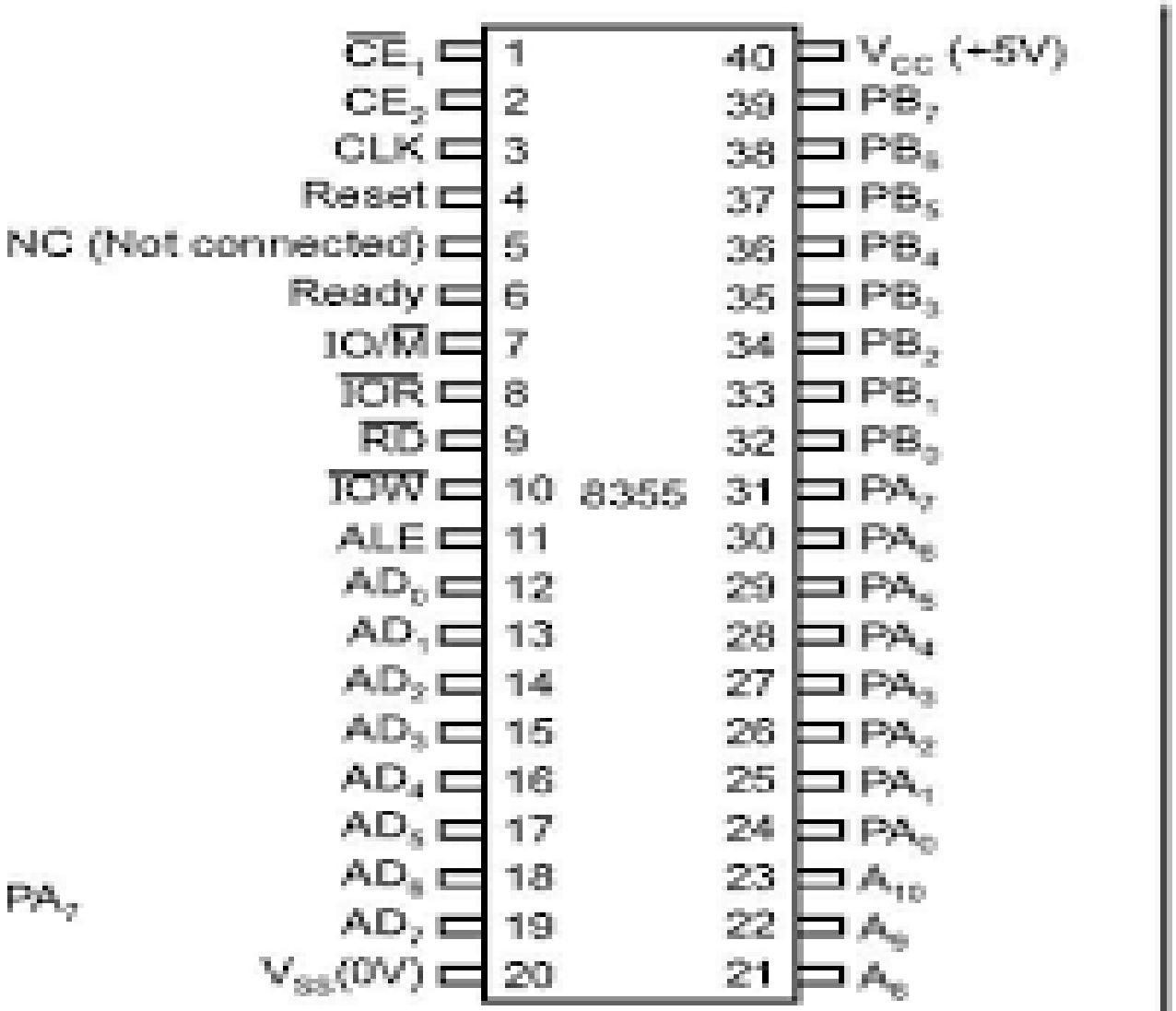


Fig. 9c.1: Pin diagram of 8355

(Source: Intel Corporation)

Unit IV

8086 Microprocessor

Internal

Introduction

- 1978 - Intel released its first 16-bit microprocessor - 8086 - executes the instructions at 2.5 MIPS (i.e 2.5 million Instruction per second).
- execution time of one instruction - 400ns (=1/MIPS=1/($2.5x10^6$))
- 8086 can also address one megabytes (1MB=2^{20} bytes) of memory.
- Another feature in 8086 - presence of a small 6-byte instruction queue - so instructions fetched from memory are placed in it before they are executed

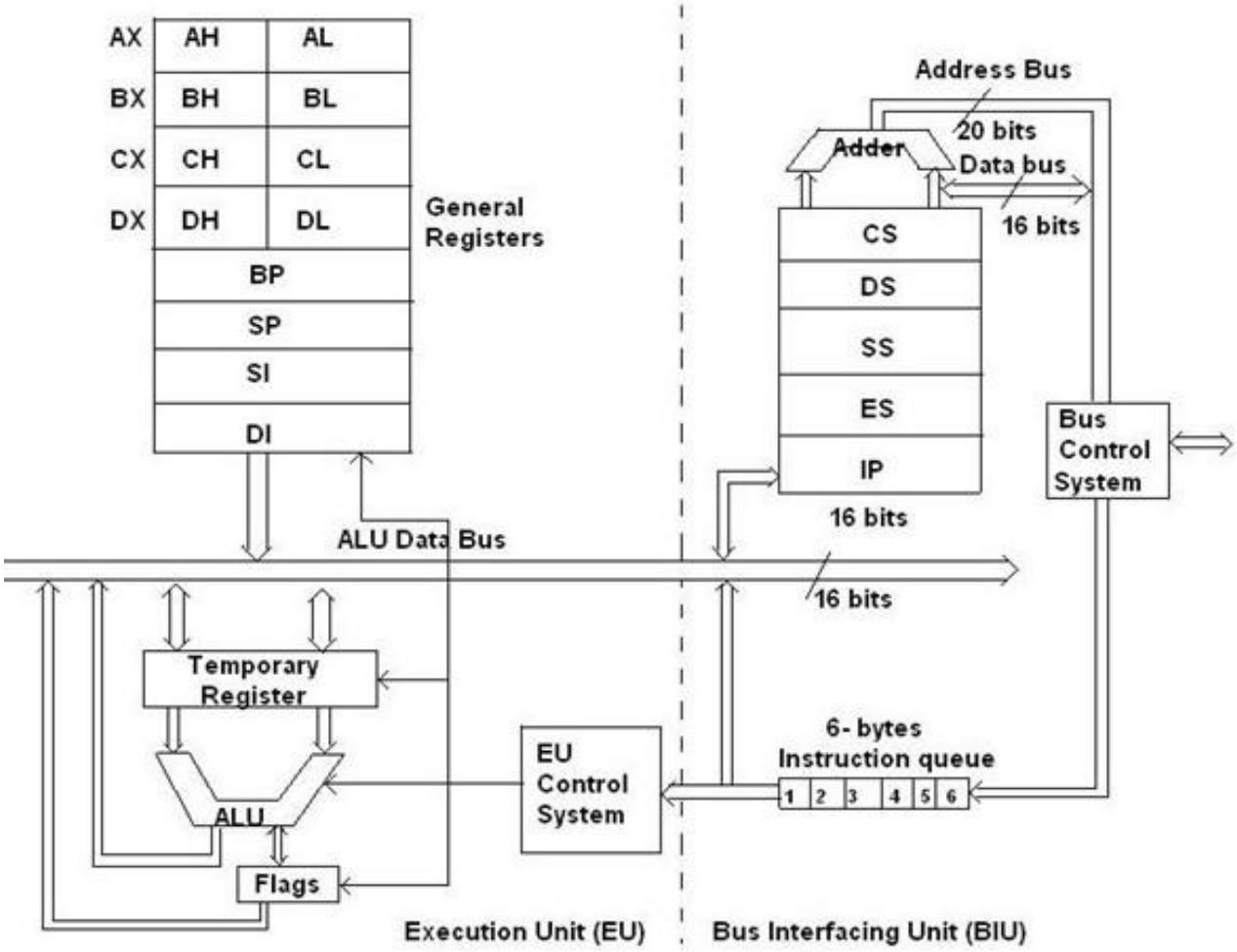
Architecture of 8086

- It is subdivided into two units –
- **Execution unit (EU)**
- **bus interface unit (BIU)**
- The **execution unit (EU)** includes :
- ALU
- Eight 16-bit general purpose registers
- A 16 bit flag register
- A control unit.
- The **bus interface unit (BIU)** includes :
- adder for address calculations
- four 16-bit segment registers (CS, DS, SS and ES)
- A 16 bit instruction pointer (IP)

Execution Unit (EU)

- The EU consists of eight 16-bit general purpose registers - AX, BX, CX, DX, SP, BP, SI and DI.
- AX, BX, CX and DX - can be divided into two 8-bit registers - AH, AL, BH, BL, CH, CL, DH and DL
- General purpose registers - can be used to store 8 bit or 16 bit data during program execution.

Functional Block diagram of 8086



Special functions of registers

- AX/AL - used as accumulator - multiply, divide, input/output (I/O) and some of the decimal and ASCII adjustment instructions.
- BX - holds the offset address of a location in memory - also used to refer the data in memory using lookup table technique.
- CX - used to hold the count while executing repeated instructions (REP/REPE/REPNE) and LOOP instruction - also used to hold the count while executing the shift and rotate instructions - count value indicates the number of times the same instructions has to be executed.
- DX - used to hold a part of the result during multiplication and part of the dividend before a division - also used to hold the I/O device address while executing IN and OUT instructions
- SP - stack pointer - used to hold the offset address of the data stored at the top of stack segment - used along with **SS** register to decide the address at which data is pushed or popped during the execution of PUSH and POP instructions.

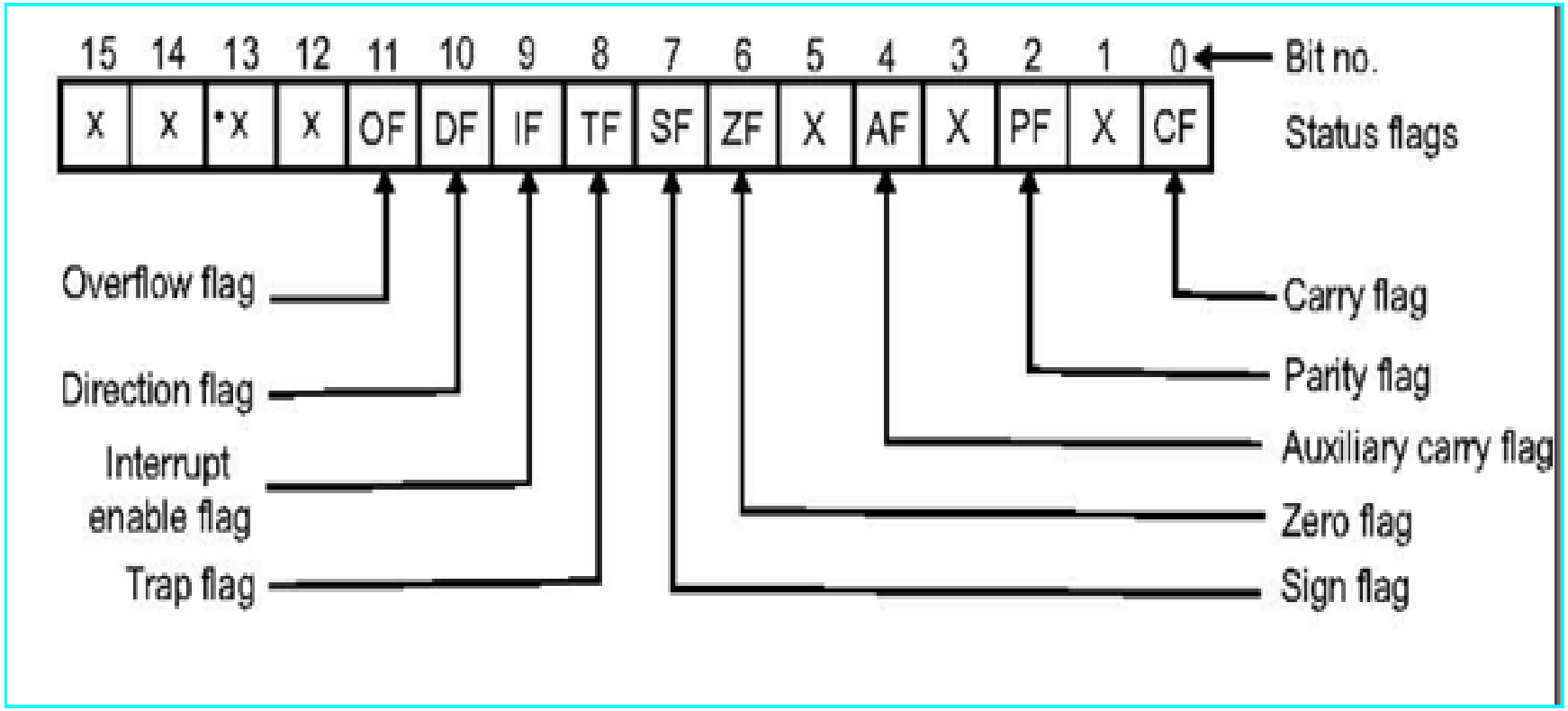
Special functions of registers

- BP - Base Pointer - used to hold the offset address of the data to be read from or write into the stack segment.

- SI - Source Index register - used to hold the offset address of source data in data segment while executing string instructions.

- DI - Destination Index register - used to hold the offset address of destination data in extra segment while executing String instructions.

Flag Register of 8086



Flag register of 8086

- The flags - classified into status flags and control flags
- CF, PF, AF, ZF, SF and OF - status flags - they indicate the status of the result that is obtained after the execution of arithmetic or logic instruction
- DF, IF and TF - control flags - they can control the operation of CPU

Function of different flags

- CF (Carry Flag) - holds the carry after 8 bit or 16 bit addition - holds borrow after 8 bit or 16 bit subtraction performed
- PF (Parity Flag) - If the lower 8 bit of the result is having odd parity (i.e. odd number of 1s) - PF is set to 0 - PF is set to 1 if the lower 8 bit of result is having even parity.
- AF (Auxiliary Carry Flag) - holds the carry after addition - the borrow after subtraction of the bits in bit position 3 - (LSB is treated as bit position 0) - used by DAA instructions to adjust the value in AL after a BCD addition or subtraction.
- ZF (Zero Flag) - indicates that the result of an arithmetic or logic operation is zero - If Z=1, the result is zero - if Z=0, the result is not zero.
- SF (Sign flag) - holds the arithmetic sign of the result after an arithmetic or logic instruction is executed - If S-0, the sign bit is 0 and the result is negative.

Function of different flags

- TF (Trap Flag) - used to debug a program using single step technique - If T flag is set (i.e. TF=1) - 8086 gets interrupted (Trap or single step interrupt) after the execution of each instruction in the program - If TF is cleared (i.e. TF=0) - the trapping or debugging feature is disabled
- DF (Direction Flag) - selects either the increment or decrement made for the DI

Function of different flags

- IF (Interrupt Flag) - controls the operation of the 'INTR' interrupt pin of 8086 - If I=0 - INTR pin is disabled - if I=1 - INTR pin is enabled - I flag can be set or cleared using the instruction STI or CLI respectively.
- OF (Overflow flag) - Signed numbers are represented in 2's complement form when the number is negative in microprocessor - When signed numbers are added or subtracted - overflow may occur - indicating that the result has exceeded the capacity of the machine - For example if the 8-bit signed data 7EH (= +126) is added with the 8-bit signed data 02H(= +2), the result is 80H(= -128 in 2's complement form). This result indicates an overflow condition - overflow flag is set during the above signed addition - In an 8-bit register, the minimum and maximum value of the signed number that can be stored in -128 (=80H) and +127 (=7FH) respectively - In a 16 bit register, the minimum and maximum value of the signed number that can be stored is -32768 (=8000H) and +32767 (=7FFFH) respectively - For operation an unsigned data, OF is ignored.

Bus Interface Unit (BIU)

- There are four segment registers - CS, DS, SS and ES.
- The function of the CS, DS, SS and ES register - indicate the starting address or base address of code segment, data segment, stack segment and extra segment in memory
- The code segment - contains the instructions of a program - data segment contains - data for the program
- The stack segment - holds the stack of a program - which is needed while executing CALL and RET instructions - also to handle interrupts - extra segment - additional data segment - used by some of the string instructions - minimum size of a segment - one byte - maximum size of a segment is 64 Kbytes.
- The base address - can be obtained by adding four binary 0s to the right most portion of the content of corresponding segment register which is same as adding a hexadecimal digit 0 to the right most portion of a segment register..

Bus Interface Unit (BIU)

- If the size of two different segments is less than 64Kbytes then it is possible for two segments to get overlapped (i.e. within 64 Kbytes allocated to a segment, another segment can start).
- Let a particular application in 8086 requires code segment of 1 Kbyte size and data segment of 2 Kbytes size. If the code segment is stored in memory from the address 20000H then it will end at the memory address 203FFH.
- The data segment can be stored from the address 20400H (which is the immediate next 16-byte boundary in memory).
- The CS and DS registers are loaded with the value 2000H and 2040H respectively for running this application in 8086.

Accessing memory locations

- Each address in physical memory (ROM/EPROM chips) is known as physical address. In order to access an operand (either data or instruction) from memory from a particular segment, 8086 has to first calculate the physical address of that operand.
- To find the physical address of that operand, the 8086 adds the base address of the corresponding segment with an offset address which may be either the content of a register or an 8 bit or 16 bit displacement given in the instruction or combination of both, depending upon the addressing mode used by the instruction.
- The 8086 designers have assigned certain register(s) as default offset register(s) for certain segment register.
- But this default assignment can be changed by using segment override prefix in the instruction which is explained in the next chapter (section 14.2).

Fetching of an instruction from memory

- Let us assume that the CS register is having the value 3000H and the IP register is having the value 2000H.
- To fetch an instruction from memory, the CPU calculates the memory address from where the next instruction is to be fetched, as shown below:
- CS X 10H =30000H + Base address of code segment

Function of pins common to minimum and maximum mode

- RD- Whenever the read signal ( ) is a logic 0, the 8086 reads data from memory or I/O device through the data bus.
- TEST – The test pin is an input that is tested by the WAIT instruction. If pin is at logic 0, then WAIT instruction functions as a NOP (No operation) instruction. If pin is at logic 1, then WAIT instruction waits for pin to become a logic 0. This pin is often connected to the BUSY input of 8087 numeric coprocessor to perform floating point operations.
- READY: This input is used to insert wait states into the timing of the 8086. If the READY pin is at logic 1 then it has no effect on the operation of the microprocessor. If the ready pin is at logic 0, the 8086 enters into wait state and remains idle. This pin is used to interface slowly operating peripherals with 8086.