DTEL (Department for Technology Enhanced Learning)

The Centre for Technology enabled Teaching & Learning

Teaching Innovation - Entrepreneurial - Global

NAGAR YUWAK SHIKSHAN SANSTHA'S
SHRI DATTA MEGHE POLYTECHNIC

DEPARTMENT OF COMPUTER TECHNOLOGY

Microprocessor and Programming

AUTHORS

MANOJ JETHWA

CONTENT: MICROPROCESSOR AND PROGRAMMING

1 CHAPTER 1: **Basics of Microprocessor**

2 CHAPTER 2: **16 Bit Microprocessor: 8086**

3 CHAPTER 3: **Instruction Set of 8086 Microprocessor**

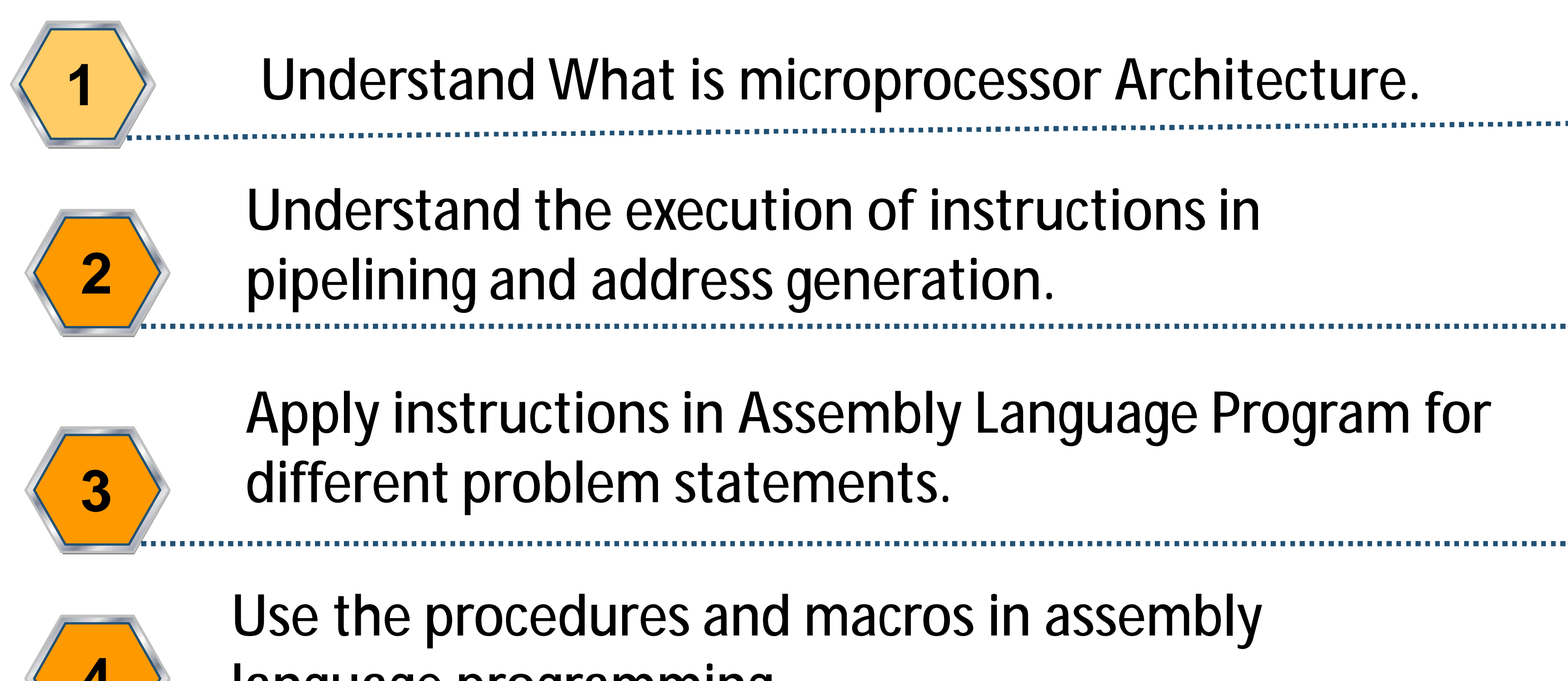
4 CHAPTER 4: **The Art of Assembly Language Programming**

5 CHAPTER 5: **8086 Assembly Language Programming.**

6 CHAPTER 6: **Procedure and Macro in Assembly Language Program**

SYLLABUS GENERAL OBJECTIVE

The student will be able to:



CHAPTER-1 Basics of Microprocessor

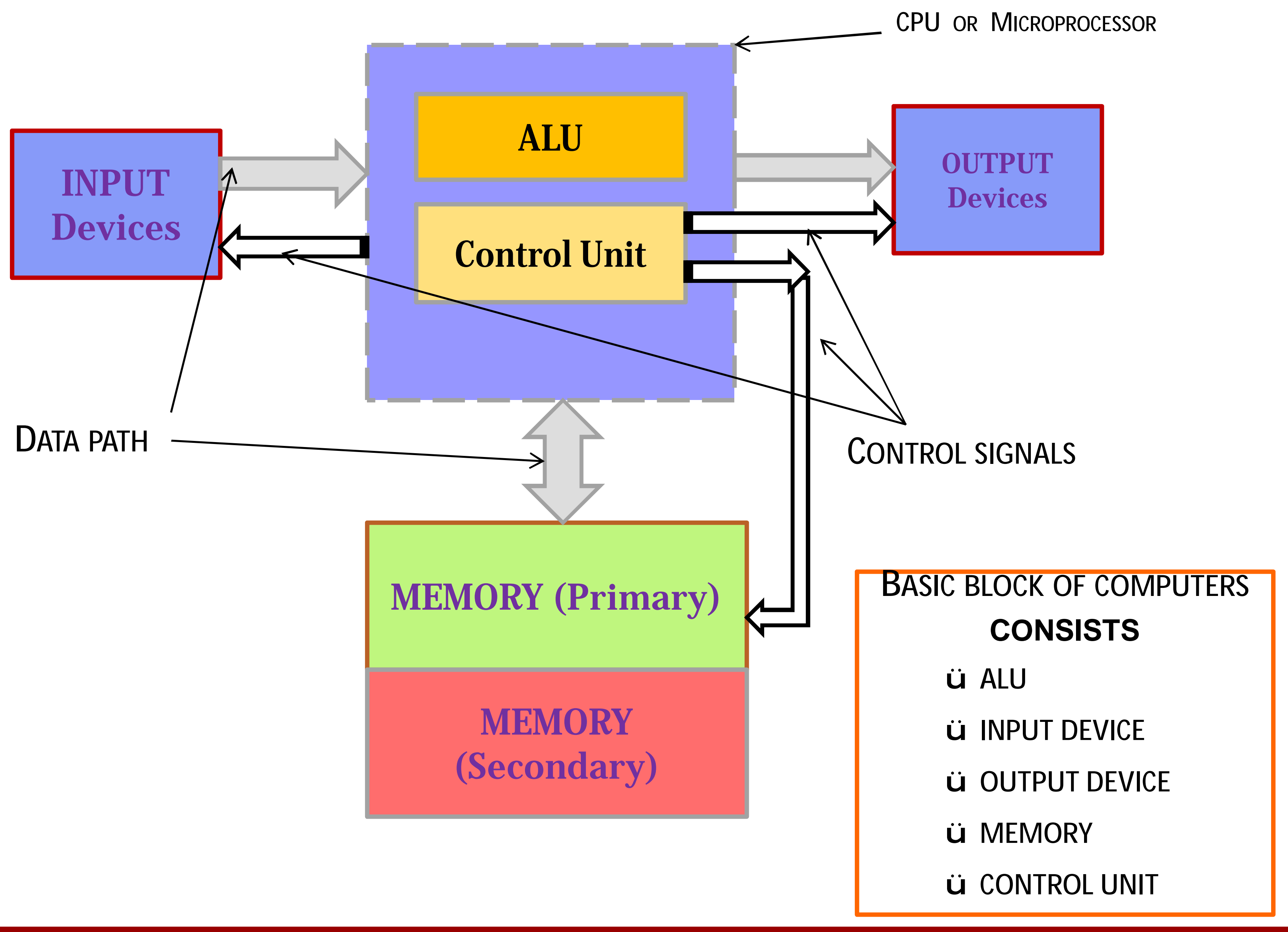
1. Topic 1: Evolution of Microprocessor and types
2. Topic 2: 8085 Microprocessor,
3. Topic 3: Salient features of 8085
4. Topic 4: Architecture of 8085 - Functional Block diagram,
5. Topic 5: Pin description,

The student will be able to:

Draw the architecture of 8085 and understand the functions of different pins of 8085

Identify status of different flags and understand register organisation of 8085

BASIC BLOCK OF COMPUTER



BASIC BLOCK OF COMPUTER

The typical Computer system consists of:

- CPU (central processing unit)
  - ALU (arithmetic-logic unit)
  - Control Logic
  - Registers, etc…
- Memory
- Input / Output interfaces

Interconnections between these units are through 3 basic buses:

- Address Bus
- Data Bus
- Control Bus

CPU

- The main function of ALU is to perform arithmetic and logical operations on binary numbers.
- The Input Device is used to feed data and command for the CPU.
- The output device is used for display of result /data /program etc.
- The memory is used for storing information.
- The control unit Synchronizes operation of ALU with IO and Memory.

BUS

The interconnections (known as Interfacing) between the 5 units of computer system is carried by 3 basic buses i) Address Bus ii) Data Bus iii) Control Bus. A bus(from the Latin *omnibus*, meaning "for all") is essentially a set of wires which is used in computer system to carry information of the same logical functionality. The function of the 3 buses is

- The address bus selects memory location or an I/O device for the CPU.
- The data bus transfers information between the microprocessor and its memory or I/O device. Data transfer can vary in size, from 8-bits wide to 64 bits wide in various members of microprocessors.
- The Control bus generates command signals to synchronise the CPU operation with IO and Memory devices.

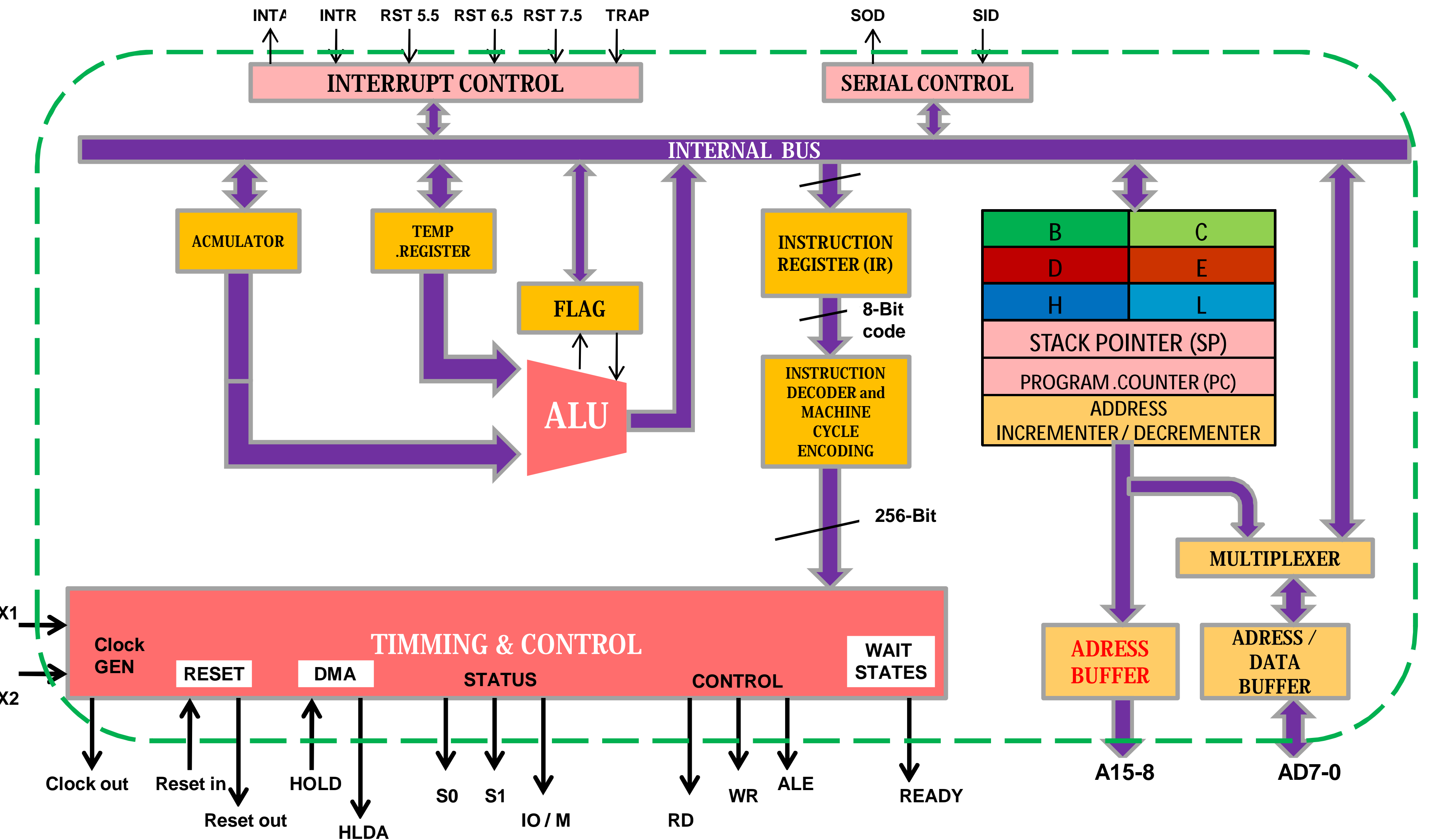
Evolution of Microprocessor

Processor	Date of Launch	Clock speed	Data Bus Width	Adress Bus	Addressable Memory Size
4004	1971	740 khz	4 bit	12	4 KB
8-BIT PROCESSOR					
8008	1972	800 Khz	8 bit	14	16 Kb
8080	1974	2 Mhz	8 bit	16	64 kb
8085	1976	3 Mhz	8 bit	16	64 kb
16-BIT PROCESSOR					
8086	1978	5 Mhz	16	20	1M
80286	1982	16 Mhz	16	24	16 M

Evolution of Microprocessor

Processor	Date of Launch	Clock speed	Data Bus Width	Adress Bus	Addressable Memory Size
32-BIT PROCESSOR					
80386	1985	33 Mhz	32	32	4 G
80486	1989	40 Mhz	32	32	4G+ 8k cache
Petium I	1993	100 Mhz	32	32	4G+16k cache
Petium II	1997	233 Mhz	32	32	4G+16k cache + L2 256 Cache
Petium III	1999	1.4 Ghz	32	32	4G+32k cache + L2 256 Cache
Petium IV	2000	2.66 Ghz	32 Internal 64 External	32	4G+32k cache + L2 256 Cache
64-BIT PROCESSOR					
Dual Core	2006	2.66 Ghz	64	36	64G+Independent L1 64 Kb+ Common L2 256 kb Cache
Core 2 Duo	2006	3 Ghz	64	36	64G+Independent L1 128 Kb+ Common L2 4 Mb Cache
I7	2008	3.33 Ghz	64	36	64G+Independent L1 64 Kb+ Common L2 256 kb Cache + 8 Mb L3 Cache

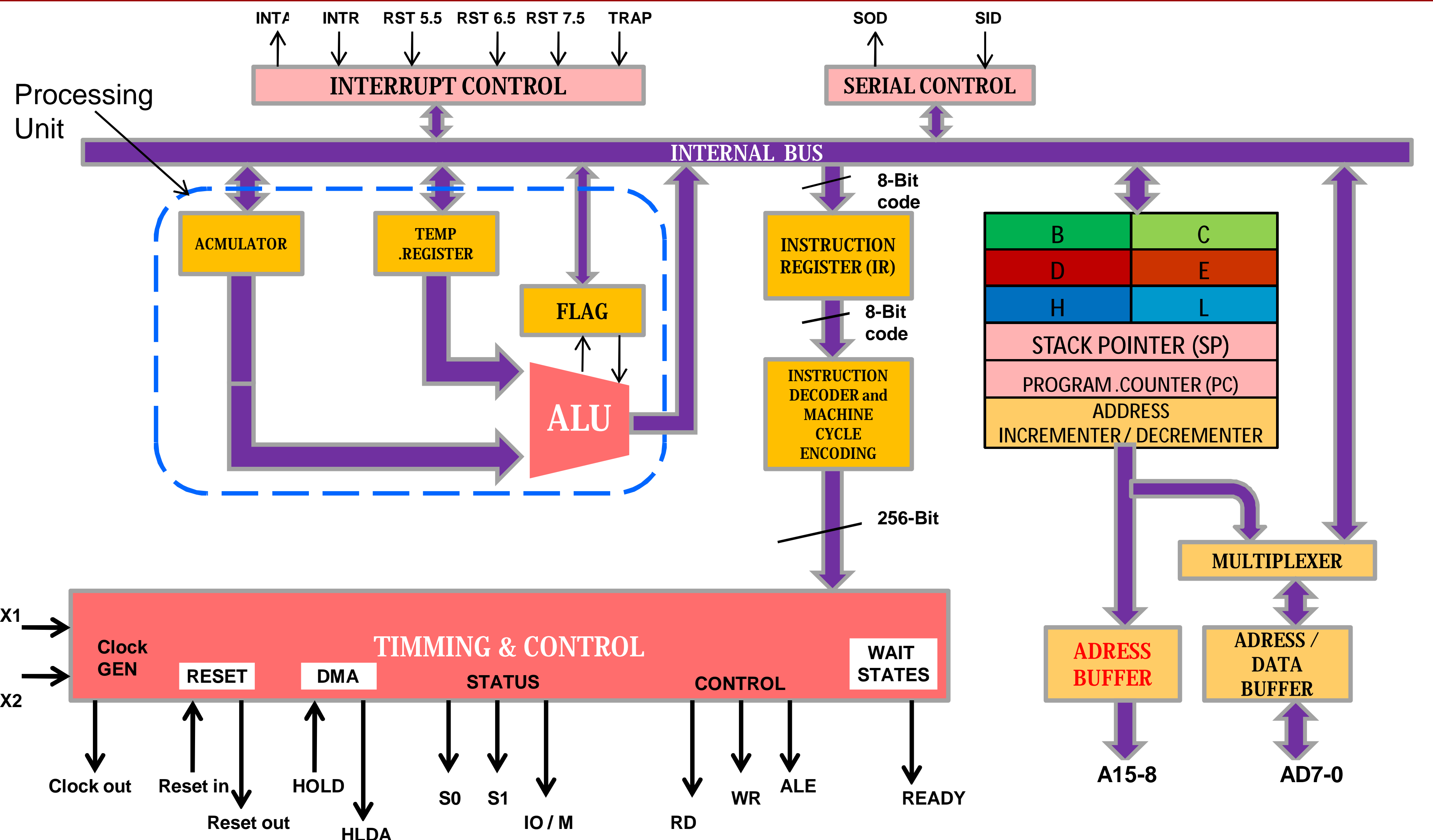
8085 ARCHITECTURE



Processing Unit

- Arithmetic and Logic Unit
- Accumulator
- Status Flags
- Temporary Register

Processing Unit



Arithmetic & Logic Unit (ALU)

- It performs Arithmetic and logic operations on binary nos.
- The result is stored in accumulator in most cases, hence A is known as accumulator.
- Arithmetic Operations:
  - Addition, Subtraction, Increment, Decrement .
- Logic Operations:
  - AND, OR, X-OR, Complement .

Arithmetic & Logic Unit (ALU)

ü It the main register of microprocessor directly connected with the ALU.

ü It is also called register 'A'.

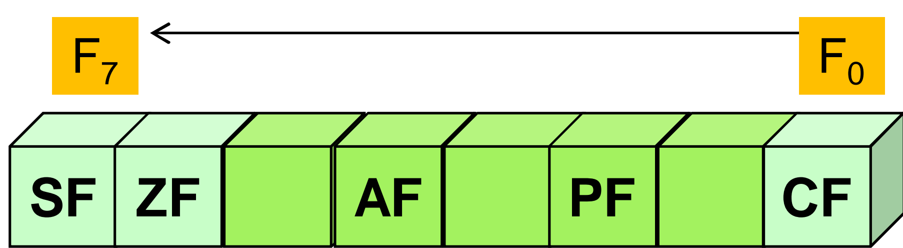
ü It is an 8-bit register.

ü It is used in the arithmetic and logic operations.

ü It always contains one of the operands on which arithmetic/logic has to be performed.

ü After the arithmetic/logic operation, the contents of accumulator are replaced by the result.

Status Flag



The 5 Status Flags are affected immediately after an arithmetic or logical operation performed by the ALU. The SET or RESET condition of each flag is used to indicate the status of the result generated by the ALU.

- Status

CF:	Carry flag
PF:	Parity flag
AF:	Auxiliary carry flag
ZF:	Zero flag
SF:	Sign flag
OF:	Overflow flag

Status Flag

- Sign Flag: It is used to indicate whether the result is positive or negative. It will set (SF=1) if the result is –ve and if the result +ve then SF=0.

- Zero Flag: It is used to indicate whether the result is a Zero or non-zero. It will set (ZF=1) if the result is zero else ZF=0.

- Auxiliary carry Flag: It is used to indicate whether or not the ALU has generated a carry/Borrow from D3 bit position to D4 bit. It will set if there was a carry out from bit 3 to bit 4 of the result else AF=0. The auxiliary carry flag is used for binary coded decimal (BCD) operations.

- Parity Flag: It is used to indicate parity (Even or Odd) of the result. It will set if the parity is even else PF =0.

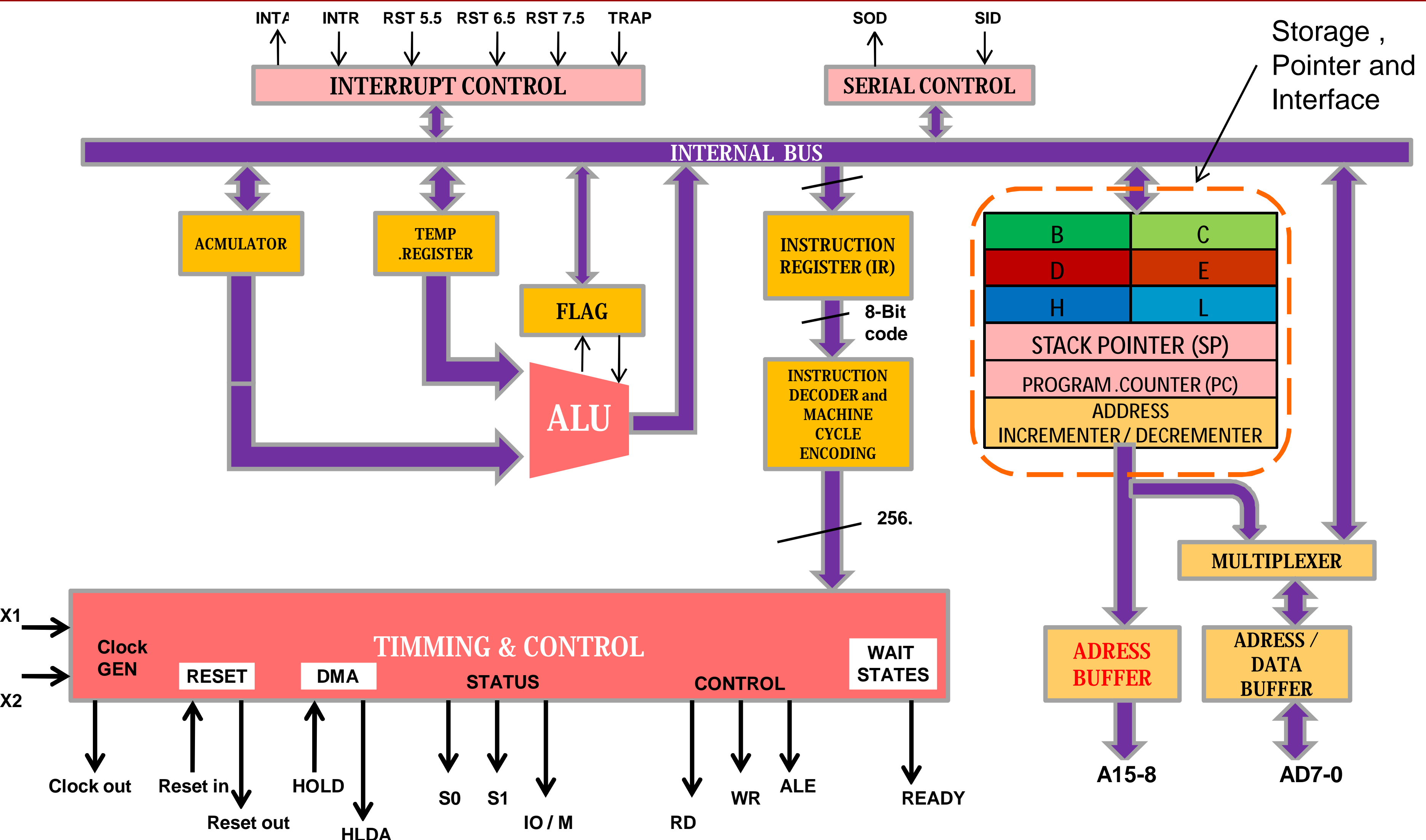
- Carry Flag: It is used to indicate whether a carry/Borrow has been generated /occurred during addition/subtraction It will set if there was a carry is generated from the MS-bit during addition, or borrow during subtraction/comparison else CF=0.

Program Status Word (PSW)

- **The Accumulator and Status Flags clubbed together is known as Program Status Word (PSW).**
- **It is a 16-bit word.**

Accumulator (8)	FLAGs (8)

Register sets & pointer



Register sets & pointer

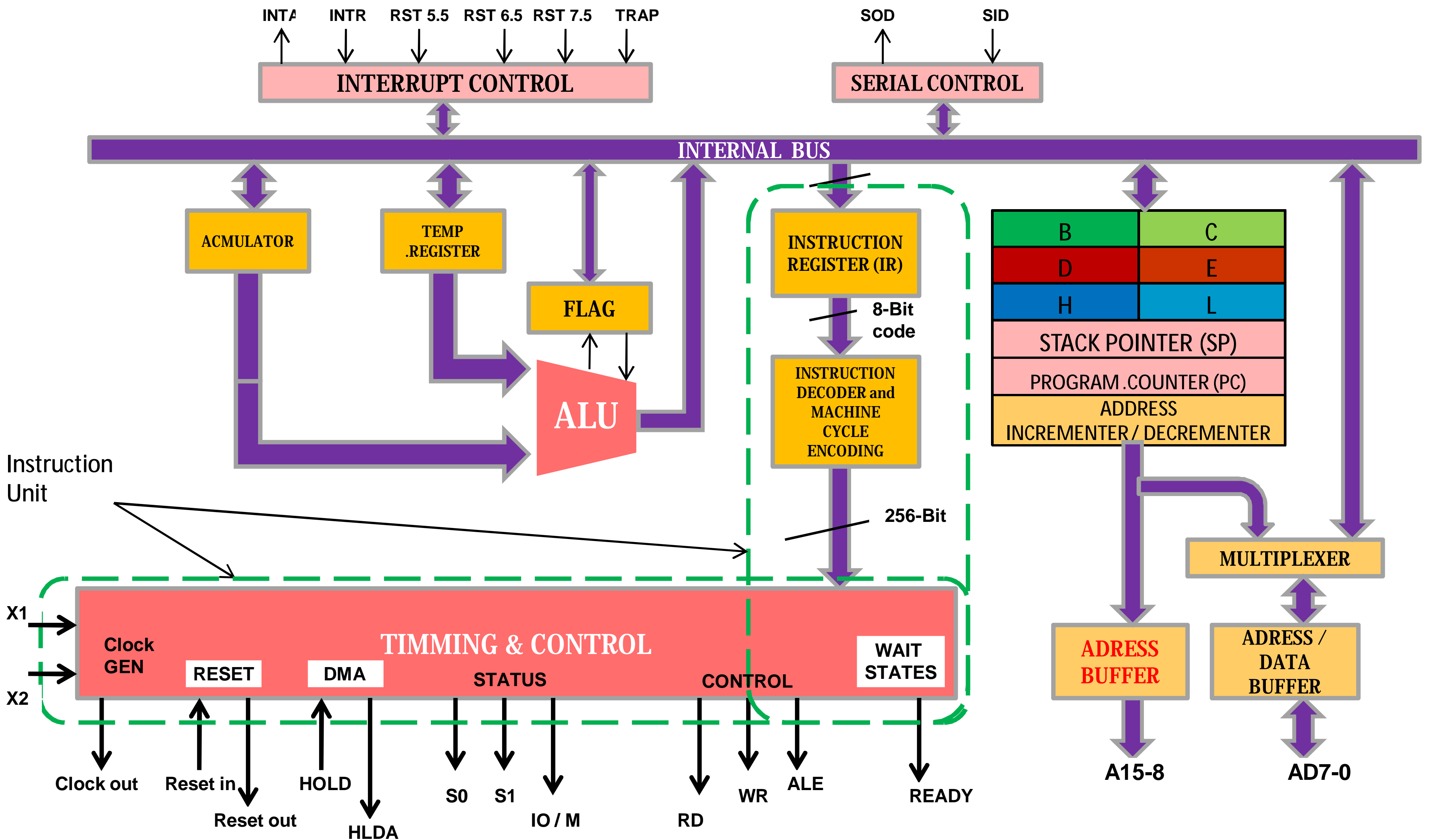
- The 8085 has set of 8 register (of 8-bit) and 2 memory pointers (of 16-bit) . The register A and flag are directly connected with ALU,While B,C,D,E,H,& L are indirectly connected through internal bus. The register A is used to store data as well as result of an operation performed by the ALU.The Flag is used to store status of result. The register B,C,D,E,H,L are used to store 8-bit data. It can also be paired to store 16-bit data. The pairing combination can be,

  B-C D-E H-L

  The register pairs can also be used to generate 16-bit address.

- The pointers are used to generate 16-bit address for selection of memory location. The PC generates address during execution of a program. It contains the memory address (16 bits) of the instruction that will be executed in the next step.
  While SP generates address during stack operation.

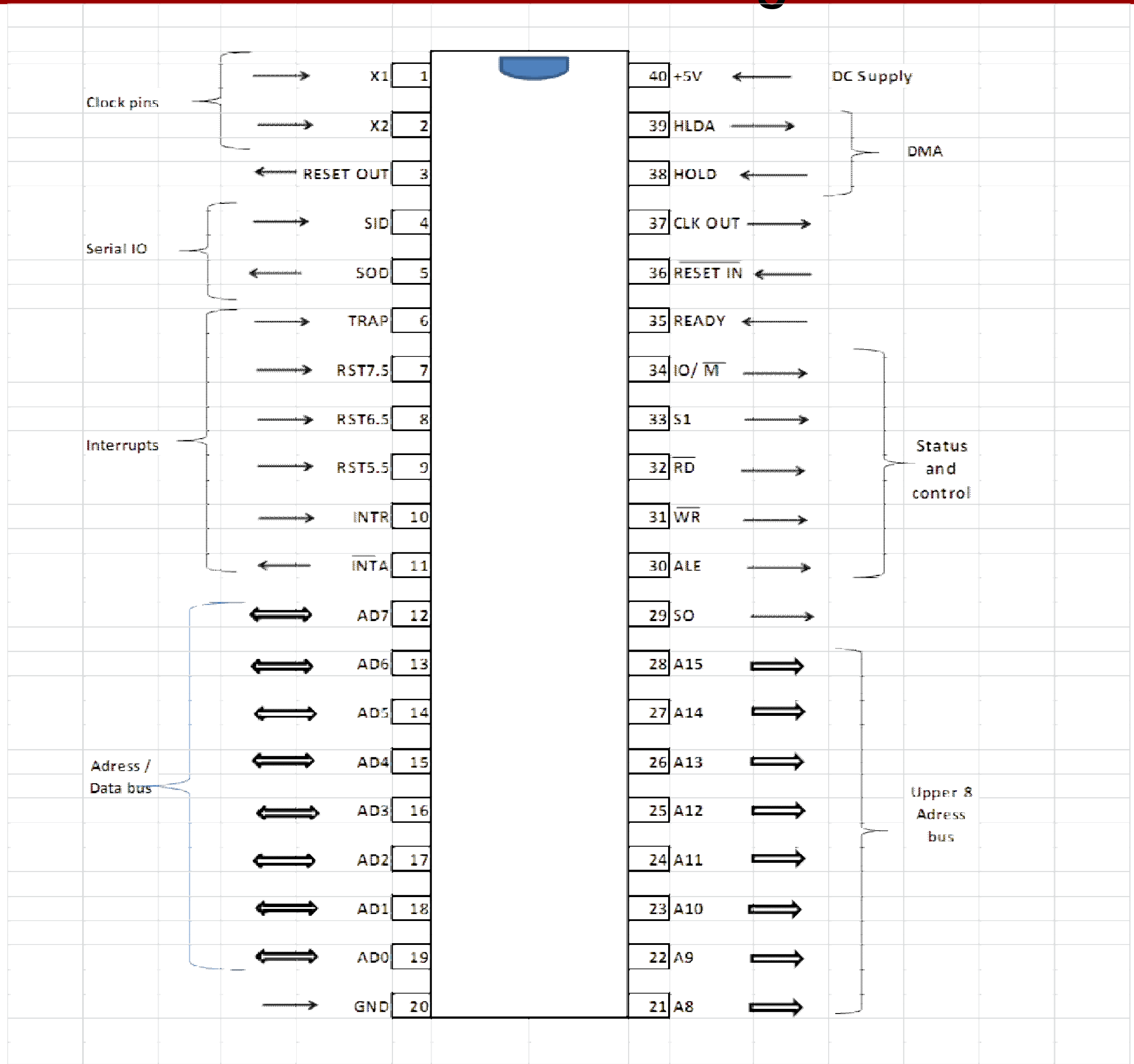
Timming-Control and IR -Decoder



- Instruction Register : It is used to store the current instruction code which is fetched from the memory It is an 8-bit register.
- Instruction Decoder : Instruction is decoded and the meaning in the form of signal is given to TC.
- Timing and Control Unit : **It generates internal and external signals to execute the code**

PIN DIAGRAM OF 8085

Pin diagram



Pin diagram

- It was introduced in 1977 by Intel.
- It is 8-bit microprocessor.
- It is NMOS device consisting of 6200 transistors .
- Its data bus is 8-bit and address bus is 16-bit.
- Its clock speed was 3 MHz. Could execute 7,69,230 instructions per second.
- Its data bus is 8-bit and address bus is 16-bit.
- It had 6,500 transistors.
- It is 40 pins Dual-Inline-Package (DIP).

X_1 & X_2 Pin 1 and Pin 2

- It is also called Oscillator Pins to which crystal of max 6.14 Mhz should be connected so that 8085 can generate clock signals internally.
- The internal Clock pulses will be ½ times crystal value i.e, 3.07 Mhz.

RESET IN and RESET OUT Pin 36 and Pin 3

- RESET IN is used to reset the microprocessor by making the pin Low. When the signal on this pin is low for at least 3 clock cycles, it forces the microprocessor to reset . Thereby ,
  1. Clear the PC and IR.
  2. Disable all the interrupts (except TRAP) and the SOD pin.
  3. HIGH output pulse to RESET OUT pin.

- Reset OUT is used to reset the external peripheral devices and ICs on the circuit. It is active high signal. The output on this pin goes high whenever RESET IN pin is made low .

SID and SOD Pin 4 and Pin 5

– SID (Serial Input Data): It receives 1-bit from external device and Stores the bit at the MSB of the Accumulator. RIM (Read Interrupt Mask) instruction is used to transfer the bit from SID MS Bit of Acc.

– SOD (Serial Output Data): It transmits MSB of Accumulator through this pin. SIM (Set Interrupt Mask) instruction is used to transfer the MS bit of Acc through SOD.

Interrupt Pins 6 to 11

– Interrupt: (INTR,RST5.5,RST6.6,RST7.5 and TRAP pins)
  - It allows external devices to i*nterrupt* the normal program execution of the microprocessor.
  - When microprocessor receives interrupt signal, it temporarily stops current program and starts executing new program indicated by the interrupt signal.
  - Interrupt signals are generated by external peripheral devices like keyboard , sensors, printers etc.
  - After execution of the new program, microprocessor returns back to the previous program.

Interrupt Pins 6 to 11

– The 5 Hardware Interrupt Pins are TRAP , RST 7.5 , RST 6.5 , RST 5.5 and INTR. Interrupts can be classified as,

1. Maskable and Non-Maskable
2. Vectored and Non-Vectored
3. Edge Triggered and Level Triggered
4. Priority Based Interrupts

Interrupt Pins 6 to 11

Maskable and Non-Maskable

Maskable interrupts are those interrupts which can be *enabled* or *disabled*. Enabling and Disabling can be done by software instructions like EI , DI and SIM. The interrupt pins RST7.5, RST6.5 ,RST5.5 and INTR are Maskable.

The interrupts which cannot be disa*bled* are called non-maskable interrupts. These interrupts can never be disabled by any software instruction. TRAP is a non-maskable interrupt.Such pins are normally used for emergency cases like fire alarming , fire extinguisher system ,intruder detector etc.

Interrupt Pins 6 to 11

Vectored and Non-Vectored

- Vectored interrupts which have particular memory location where program control is transferred when interrupt occur. Each vectored interrupt points to the particular location in memory. RST 7.5 , RST 6.5 ,RST 5.5, TRAP are vectored Interrupts.
  - The addresses to which program control is transferred are :

Name	Vectored Address
RST 7.5	003C H (7.5 x 0008 H)
RST 6.5	0034 H (6.5 x 0008 H)
RST 5.5	002C H (5.5 x 0008 H)
TRAP	0024 H (4.5 x 0008 H)

  - Absolute address is calculated by multiplying the RST no with 0008 H.

- Non-Vectored interrupts don't have fixed memory location for transfer of program control. The address of the memory location is given by interrupting device to the processor along with the interrupt. INTR is a non-vectored interrupt.

Interrupt Pins 6 to 11

Edge Triggered and Level Triggered

The interrupts that are triggered at leading or trailing edge are called edge triggered interrupts. RST 7.5 is an edge triggered interrupt. It is triggered during the leading (positive) edge.

The interrupts which are triggered at high or low level are called level triggered interrupts. RST 6.5,RST 5.5, INTR, are level triggered interrupt.

The TRAP is edge and level triggered interrupt.

Interrupt Pins 6 to 11

Priority Based Interrupts

When there is a simultaneous interrupt request at two or more interrupt pins then the microprocessor will execute program of that pin that has higher priority. To avoid confusion in such cases all microprocessor assigns priority level to each interrupt pins. Priority is considered by microprocessor only when there are simultaneous requests. The priority of 8085 pins are:

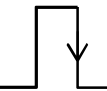
Interrupt	Priority
TRAP	1
RST 7.5	2
RST 6.5	3
RST 5.5	4
INTR	5

Address and Data Pins

- Address Bus pins $A_{15}–A_8$ **and** $AD_7–AD_0$: The address bus are used to send address for memory or IO device. It selects one of the many locations in memory or a particular IO port. 8085 has 16-bit bus.
- Data Bus **pins** $AD_7–AD_0$: It is used to transfer data between microprocessor and memory /IO Device. 8085 Data bus is of 8-bit.

ALE Pin 30

- On this pin 8085 generates a pulse in the T1 clock of each Machine cycle to latch lower byte adress from $AD_7 - AD_0$ pins.From mid of T2 to T3 the bus can transfer data.



- It indicates whether the bus has address or data .Since the bus $AD_7 - AD_0$ is used for Adress as well as data therefore it is known as Multiplexed bus. Due to multiplexing 8085 has less no of pins.

Status S1 ,So Pin 31 ,29

- S_0 and S_1 are called Status Pins.They indicate the status of current operation which is in progress by 8085.The 4 status indicated by 8085 are

S_0	S_1	Operation
0	0	Halt
0	1	Write
1	0	Read
1	1	Opcode Fetch

$IO/\overline{M}$ Pin 34

- This pin indicates whether I/O or memory operation is being performed by microprocessor.
- If $IO/\overline{M} = 1$ then
  - I/O operation is being performed.
- If $IO/\overline{M} = 0$ then
  - Memory operation is being performed.

$\overline{RD}$ Pin 32

- It is a control signal used to perform Read operation from memory or from Input device. It is active low signal. A low signal indicates that data on the data bus must be placed by the selected memory location or input device.

$\overline{WR}$ Pin 31

- It is a control signal used to perform Write operation into memory location or to output device. It is also active low signal. A low signal indicates that data on the data bus must be written into selected memory location or to output device.

READY Pin 35

- This pin is used to synchronize slower peripheral devices with high speed of microprocessor.
- A low pulse in T2 causes the microprocessor to enter into *wait state*.
- The microprocessor remains in wait state until the input at this pin goes high.

HOLD Pin 38

- **HOLD pin is used to request the microprocessor for DMA transfer.**
- **A high signal on this pin is a request to microprocessor, by external device, to relinquish the hold on buses.**
- **The request is sent by external device through DMA controller.**

HLDA Pin 39

- The HLDA signal is send to DMA Controller as acknowledgement to DMA controller to indicate that microprocessor has relinquished the system bus.
- After data transfer When HOLD is made low by DMA controller, HLDA is also made low by 8085 so that the microprocessor takes control of the buses again.

V_{SS} and V_{CC} Pin 20 and Pin 40

- **+5V DC power supply is connected to V_{CC} to bias internal circuit.**
- **Ground signal is connected to V_{SS}.**

Chapter 1 Question Bank

- What are the technical features of 8085?
- Explain the function of ALU section of 8085.
- Describe the function of the following blocks of 8085
- ALU ii) Timming & control iii) Instruction Decoder
- Explain the function of various registers of 8085.
- Draw the Block (Architecture) of 8085 and explain IR, stack pointer and programme counter.
- What are the various Flag of 8085?
- What are the pointers of 8085.Explain the function of Pointers of 8085?
- Explain the function of Interrupt section of 8085.
- List Maskable and non-maskable Interrupts of 8085.
- Explain the function of SID & SOD of 8085.
- Describe microprocessor evolution with suitable example?
- Differentiate, any six ,between 8085 & 8086.

Summary

1.The typical Computer system consists of:

- ü ALU (arithmetic-logic unit)
- ü Control Logic
- ü Memory
- ü Input devices
- ü Output devices

2. A bus(from the Latin *omnibus*, meaning "for all") is essentially a set of wires which is used in computer system to carry information of the same logical functionality. The interconnections (known as Interfacing) between the 5 units of computer system is carried by 3 basic buses

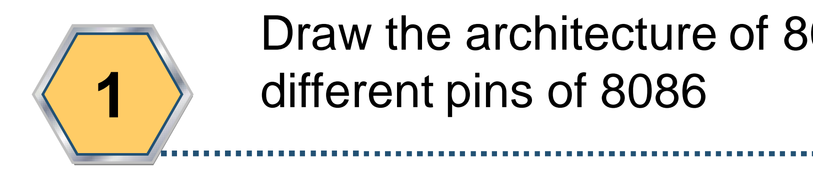
i) Address Bus ii) Data Bus iii) Control Bus.

3. 8085 can be divided into sections like i) Processing unit ii) Register & pointers iii) instruction register-decoder-timming & control.

4.The 8085 has 8-bit flag but 5 are affected by Arithmetic / logical operation.

1. Topic 1: Salient features of 8086
2. Topic 2: Architecture of 8086 - Functional Block diagram,
3. Topic 3: Register organization,
4. Topic 4: Concepts of pipelining, Memory segmentation
5. Topic 5: Operating Modes of 8086

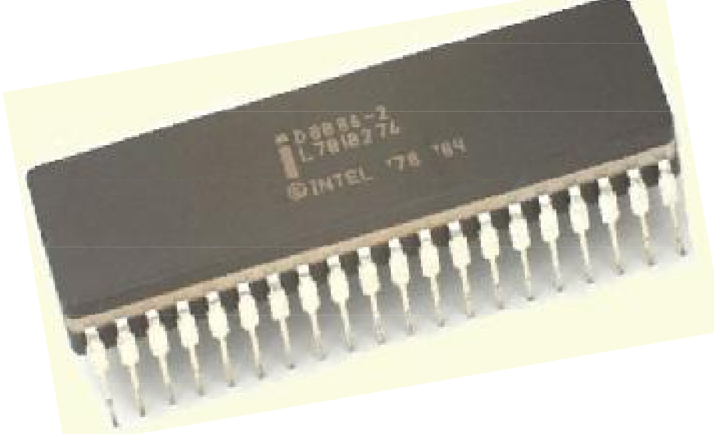
The student will be able to:



1. Draw the architecture of 8086 and understand the functions of different pins of 8086

2. Understand the operating modes of 8086

Intel 8086 Microprocessor



Key Features:

ü Introduction date: March 1978

ü It is 16-bit HMOS microprocessor implemented with 29,000 transistors

ü It can be operated with clock Frequency of 5MHz

ü Technology: HMOS

ü Number of Pins : 40

ü It has 20-bit Address lines and hence it can address 2^{20} = 1 Mbytes memory location.

ü It can generate 16-bit address for IO devices and can address 2^{16} = 64K IO ports.

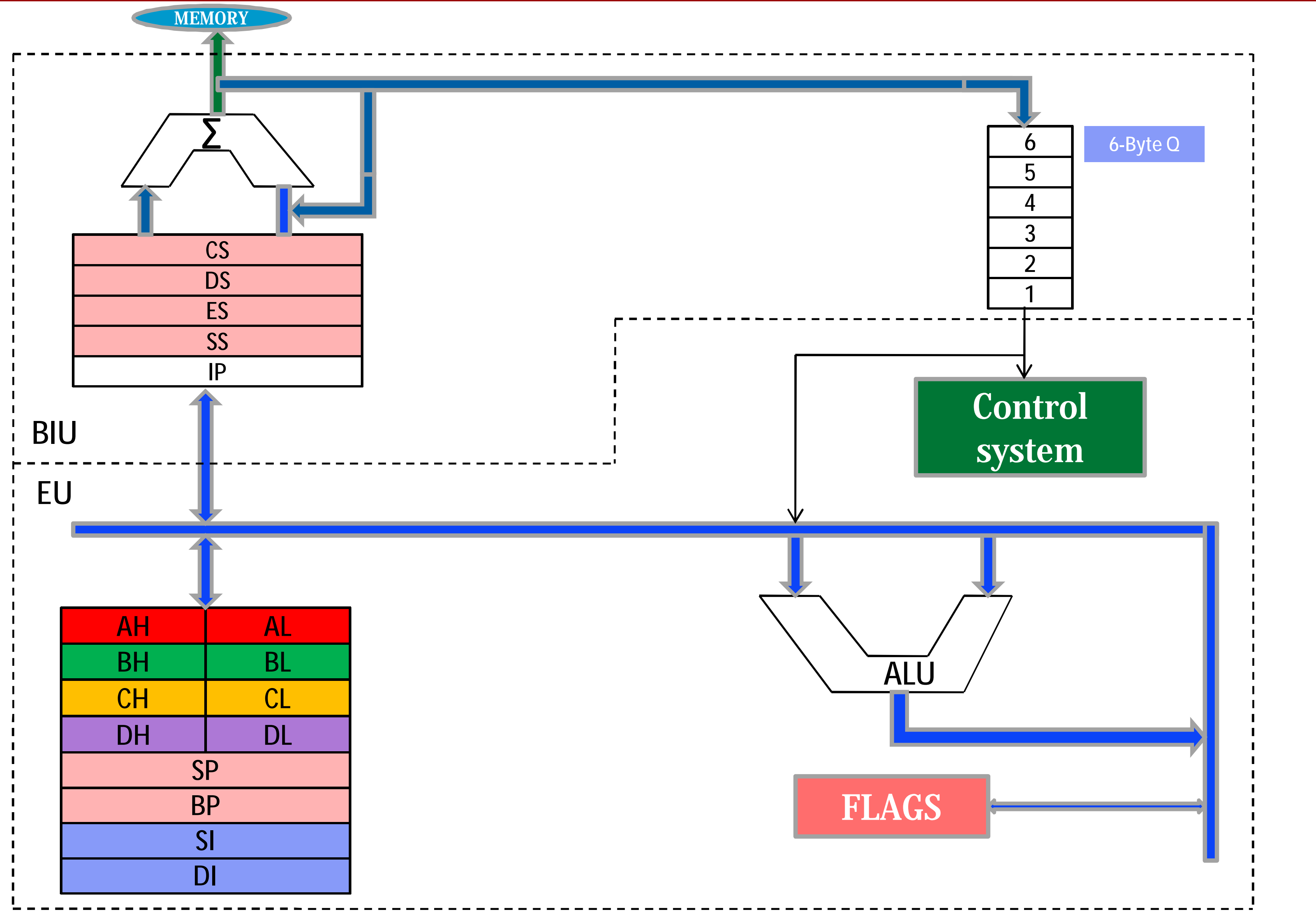
ü It can be operated in two Modes : Maximum and Minimum

ü It has two stage pipeline architecture.

ü Number of instructions: 135 instructions with eight 8-bit registers and eight 16-bit registers

ü DC Power Supply +5v

Architecture of 8086



Architecture of 8086

- The architecture of 8086 provides a number of improvements over 8085 architecture. It supports a 16-bit ALU, a set of 16-bit registers and provides segmented memory addressing capability, a rich instruction set, powerful interrupt structure, fetched instruction queue for overlapped fetching and execution.
- The complete architecture of 8086 can be logically divided into two units a) Bus Interface Unit (BIU) and (b) Execution Unit (EU). Both units operate asynchronously to provide the 8086 to overlap instruction fetch and execution operation , which is called as parallel processing. This results in efficient use of the system bus and enhance system performance.

Pipelining and parallel processor

An instruction pipeline is a technique used in the design of microprocessors to increase the number of instructions that can be executed in a unit of time. Pipeline technique is used in advanced microprocessors where the microprocessor begins operation on next instruction before it has completed operation on the previous. That is, several instructions are simultaneously in the *pipeline* at a different stage of processing. The pipeline is divided into different Stages and each Stage can perform its particular operation simultaneously with the other stages. When a stage completes an operation, it passes the result to the next stage in the pipeline and fetches the next operation from the preceding stage. The final results of each instruction emerge at the end of the pipeline in rapid succession. Since all units perform operation concurrently on different instructions , it is known as parallel processor.

Pipelining and parallel processor

Pipelining of 8086

INSTRUCTION NO.	EXECUTION PHASES							
1	Fetch-1	Decode-1	Execute-1					
2		Fetch-2	Decode-2	Execute-2				
3			Fetch-3	Decode-3	Execute-3			
4				Fetch-4	Decode-4	Execute-4		
5					Fetch-5	Decode-5	Execute-5	
6						Fetch-6	Decode-6	Execute-6
Machine cycle	1	2	3	4	5	6	7	8

Non-Pipelining Process of 8085

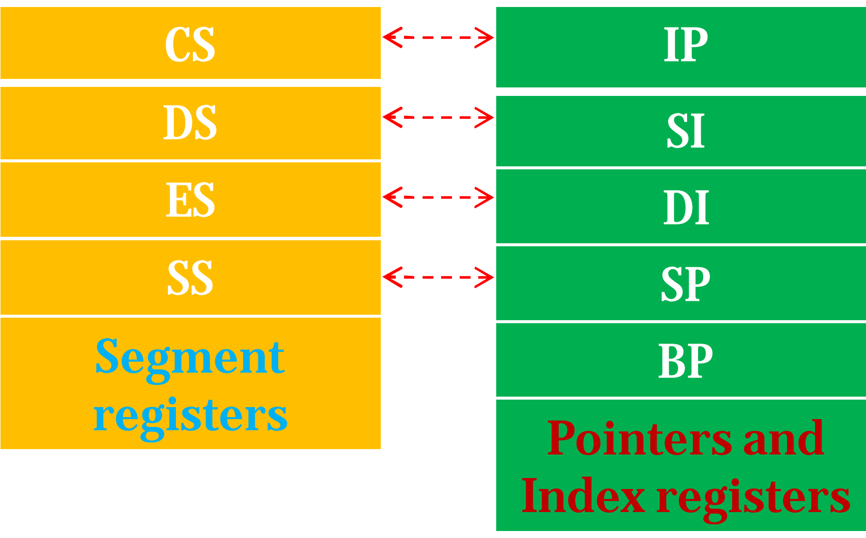
	Inst ruction-1			Inst ruction-2			Inst ruction-3		
	Fetch-1	Decode-1	Execute-1	Fetch-2	Decode-2	Execute-2	Fetch-3	Decode-3	Execute-3
M. cycle	1	2	3	4	5	6	7	8	9

Register of 8086

8086 has a powerful set of registers that can be grouped as

- General Data register
- Segment registers
- Pointers & Index registers
- FLAG
- Only GPRs can be accesses as 8/16-bit while others as 16-bit only

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL
	General Data registers	



FLAGS / PSW

Special Purpose Registers:

- Special Purpose Registers: The special purpose registers are
  - Ø Segment registers
  - Ø Pointers and index registers

- Segment Registers : Unlike 8085, the 8086 addresses a segmented memory of 1MB, which the 8086 is able to address. The 1 MB is divided into 16 logical segments (16 X 64 KB = 1024 KB = 1 MB). Each segment thus contains 64 Kbytes of memory. There are four segment registers, viz. Code Segment Register (CS), Data Segment Register (DS), Extra Segment Register (ES) and Stack Segment Register (SS).

Special Purpose Registers:

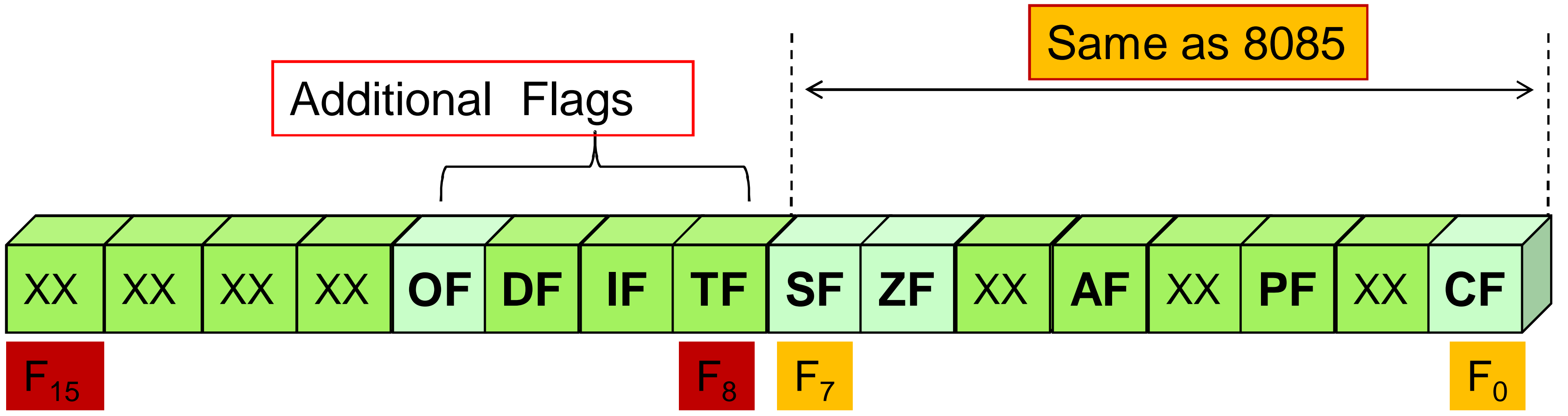
Pointers and Index Registers

The pointers contain offset within the particular segments. The pointers IP, BP and SP usually contain offsets within the code, data and stack segments respectively. The index registers are used as general purpose registers as well as for offset storage in case of indexed, based indexed and relative based indexed addressing modes. The register SI is generally used to store the offset of source data in DMS while the register DI is used to store the offset of destination in DMS or EMS. The index registers are particularly useful for string manipulations.

Flag Register

- The FLAG is nothing but group of flip-flops which are affected (SET or RESET) immediately after an arithmetic or logical operation performed by the ALU.
- The flags of 8086 can be divided into two types: Conditional Flags and Control Flags
- Conditional Flags are affected immediately after an arithmetic or logical operation performed by the ALU. The SET or RESET condition of each flag is used to indicate the status of the result generated by the ALU.The 8086 has 6 conditional flags, out of which 5 are similar to the 8085 while Overflow flag is the additional flag.
- Control Flag are not affected by Arithmetic or logical operation performed by the ALU but programmer can SET or RESET these Flags to Control certain operation/Instructions.

Flag Register



- Control Flags

IF:	Interrupt enable flag
DF:	Direction flag
TF:	Trap flag
XX:	Don't Care

- Status Flags

CF:	Carry flag
PF:	Parity flag
AF:	Auxiliary carry flag
ZF:	Zero flag
SF:	Sign flag
OF:	Overflow flag

Conditional Flag

The 6 Status or Conditional Flags are affected immediately after an arithmetic or logical operation performed by the ALU. The SET or RESET condition of each flag is used to indicate the status of the result generated by the ALU.

- Sign Flag: It is used to indicate whether the result is positive or negative. It will set (SF=1) if the result is –ve and if the result +ve then SF=0.
- Zero Flag: It is used to indicate whether the result is a Zero or non-zero. It will set (ZF=1) if the result is zero else ZF=0.
- Auxiliary carry Flag: It is used to indicate whether or not the ALU has generated a carry/Borrow from D3 bit position to D4 bit. It will set if there was a carry out from bit 3 to bit 4 of the result else AF=0. The auxiliary carry flag is used for binary coded decimal (BCD) operations.

Conditional Flag

•Parity Flag: It is used to indicate parity (Even or Odd) of the result. It will set if the parity is even else PF =0.

•Carry Flag: It is used to indicate whether a carry/Borrow has been generated /occurred during addition/subtraction It will set if there was a carry is generated from the MS-bit during addition, or borrow during subtraction/comparison else CF=0.

•Overflow Flag: The OF indicates a signed arithmetic result overflow. If result of an operation is too large a positive number or too small a negative number to fit in the destination then OF will SET, else it will RESET.

Control Flag

TF (Trap Flag) : It is used for Single step operation .If TF=1 then 8086 executes single instruction at a time and stop momentarily. If TF=0 then 8086 executes the given programme in natural sequence.

IF (Interrupt-enable flag) : When IF=1 then maskable Interrupt INTR will cause the CPU to transfer control to an interrupt vector location.

DF (Direction flag) : Causes string instructions to auto decrement/ increment the index registers (SI/DI) by 1 (for byte operation) or 2 by word operation). If DF=1 will decrement and DF=0 will increment index registers.

20-bit Physical address generation

- Since the 8086 can generate 20-bit physical address therefore it can access 2^{20}= 1048576 locations or 1024 Kbytes location or 1 Mbytes locations addressed from 00000h TO FFFFFh .
- For programme flexibility the 1Mbytes location is logically segmented (divided or organized) into
  - Code Memory Segment (CMS),
  - Data Memory Segment (DMS),
  - Extra Memory Segment (EMS) and
  - Stack Memory Segment (SMS).

20-bit Physical address generation

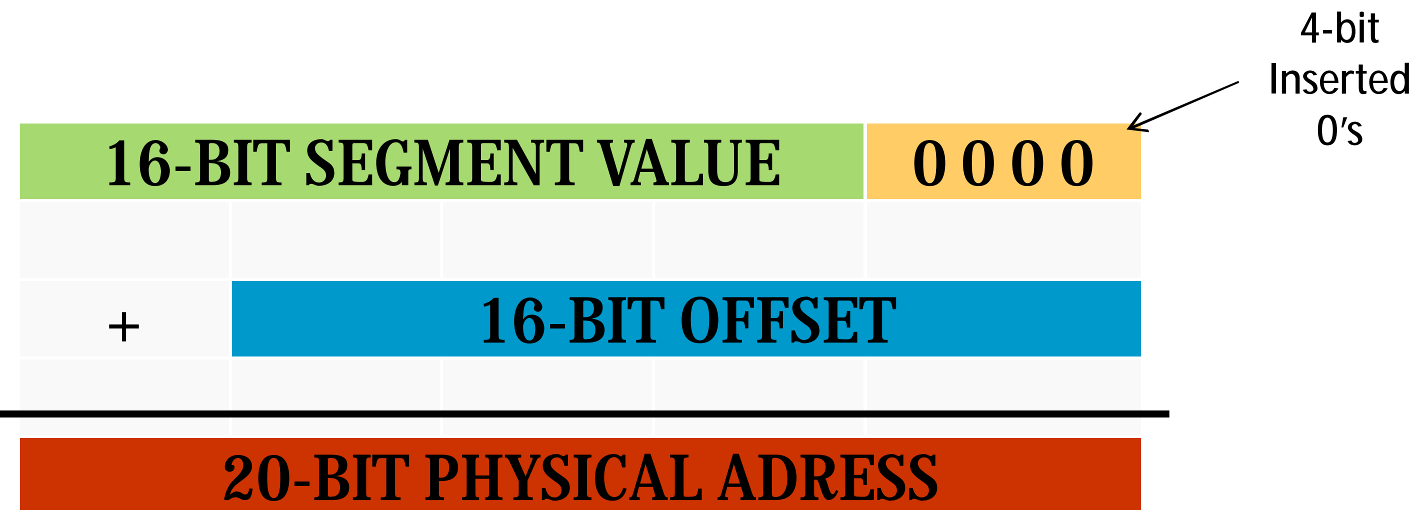
- Each memory segment can be maximum of 64 Kbytes.
- To access a particular location of memory segment the 20-bit physical address is generated by the addition of Base Address (BA) provided by the segment register and 16/8 bit offset address/displacements (OA) is provided by Pointers/index registers.

The selection of segment registers and pointers/index registers is according to the rule given in the table.

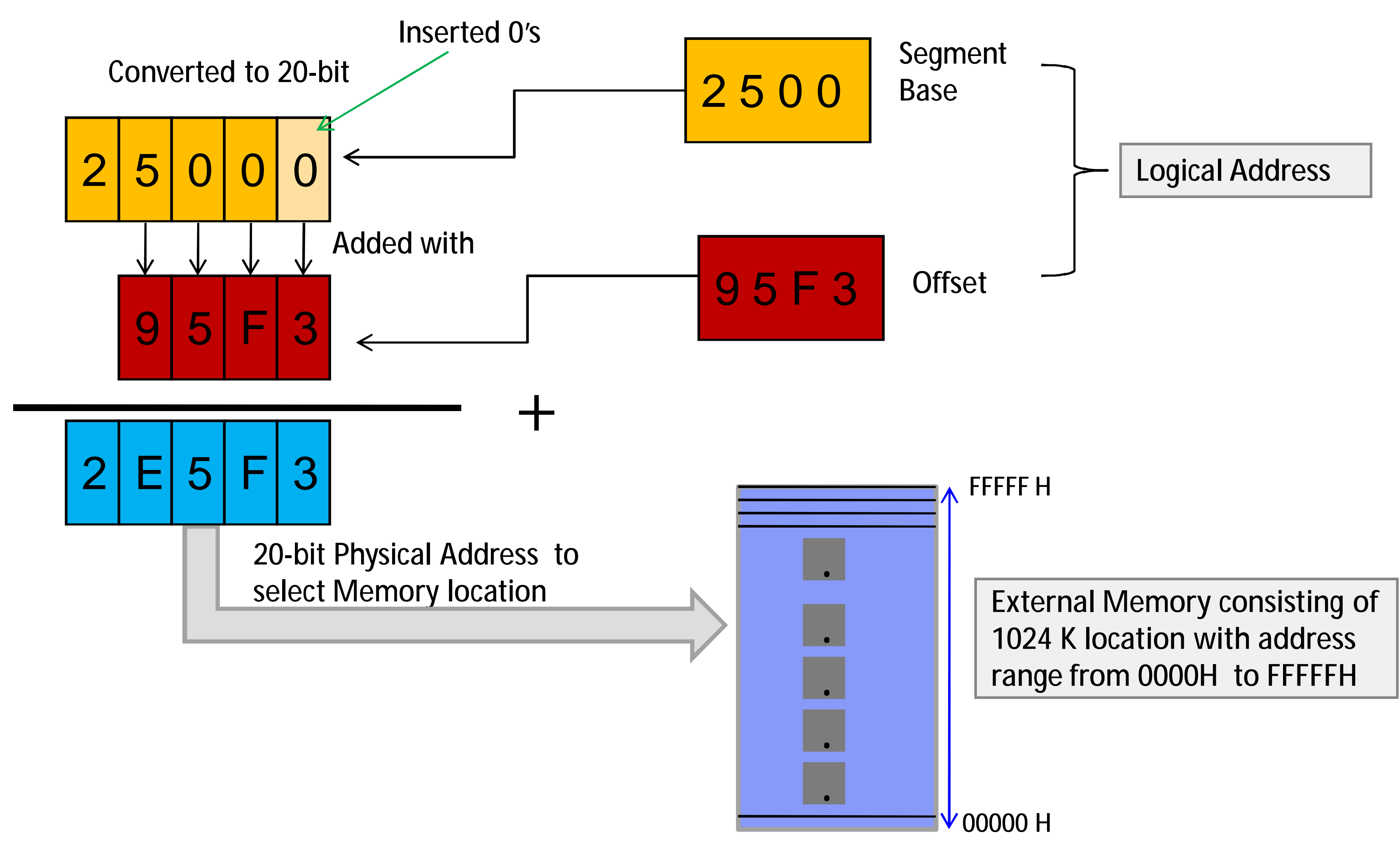
Default combination of seg reg & pointer

Name of Memory segment	Segment register used for base value	Default Pointers/ Index register used for offset address	Memory segment used for	Segment selection rule
CMS (Code memory Segment)	CS	IP	Instructions	Automatic during execution of a programme to prefetch code.
DMS (Data memory Segment)	DS	BX/SI/16/8bit displacement	Local data	During execution of a string instruction or data transfer.
(Data memory Segment)	ES	DI/16/8bit displacement	External data	During execution of a string instruction or data transfer from IO.
SMS (Data memory Segment)	SS	SP/BP	Stack	During execution of a stack instruction.

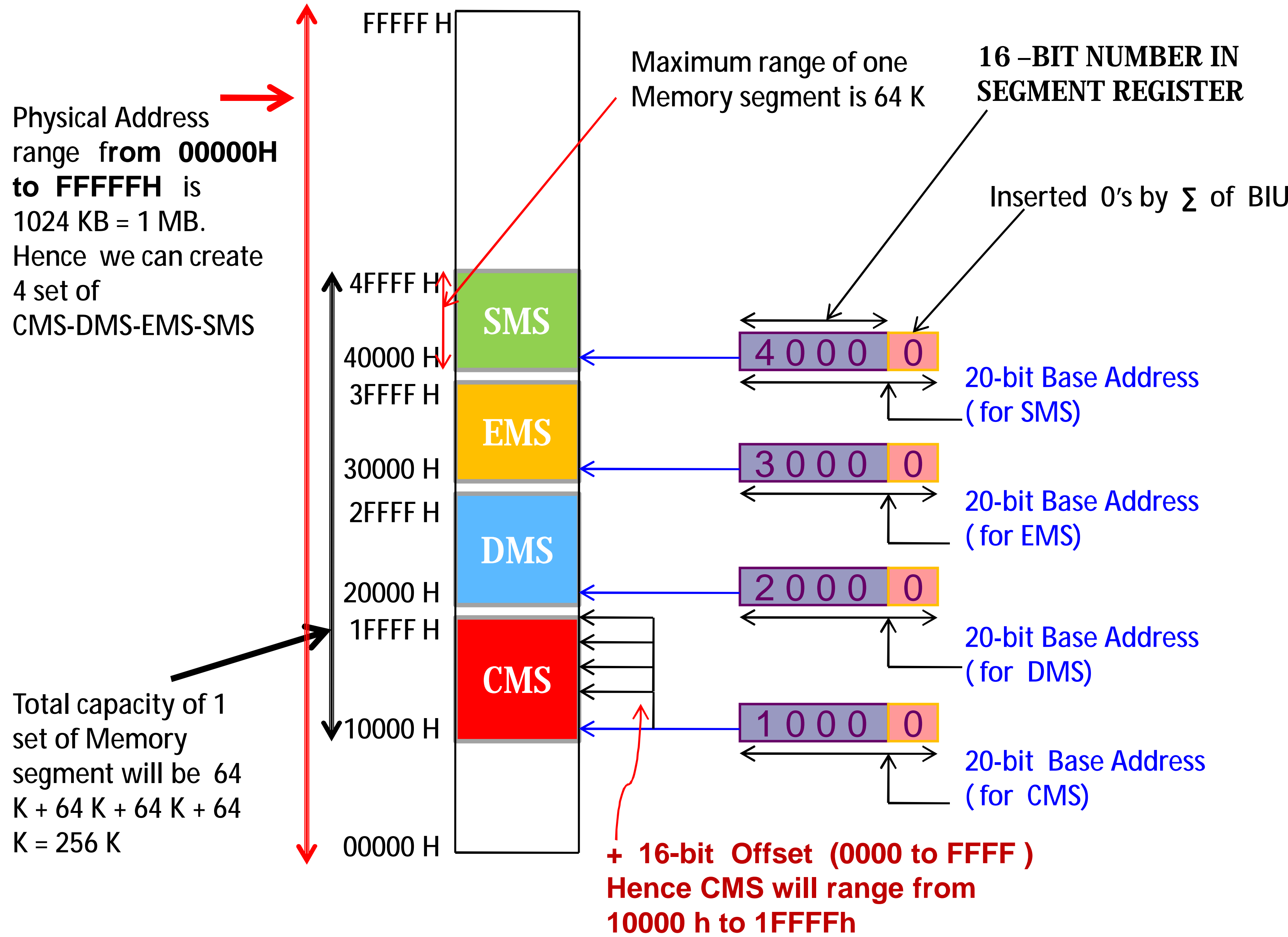
Memory Address generation



Memory Address generation Example



RANGE OF CMS-DMS-EMS-SMS



Advantages of memory segmentation

- Allows the memory capacity to be 1Mb even though the addresses associated with the individual register / instructions are only 16 bits wide.

- Facilitate the use of separate memory areas for the program, its data and the stack and allows a program and/or its data to be put into different areas of memory each time the program is executed. Due to which relocatibility of information becomes efficient.

Advantages of memory segmentation

ü The greatest advantage of segmented memory is that programs that reference logical addresses only can be loaded and run anywhere in memory. This is because the logical addresses always range from 00000h to 0FFFFh, independent of the code segment base. Such programs are said to be relocatable, meaning that they can be executed at any location in memory. The requirements for writing relocatable programs are

1. No reference should be made to physical addresses, and
2. No changes to the segment registers be allowed once initialised.

ü Since more than 1 set of CMS-DMS-EMS-SMS can be created therefore multiprogramming can be implemented easily. Also sharing of segments by different process is also possible.

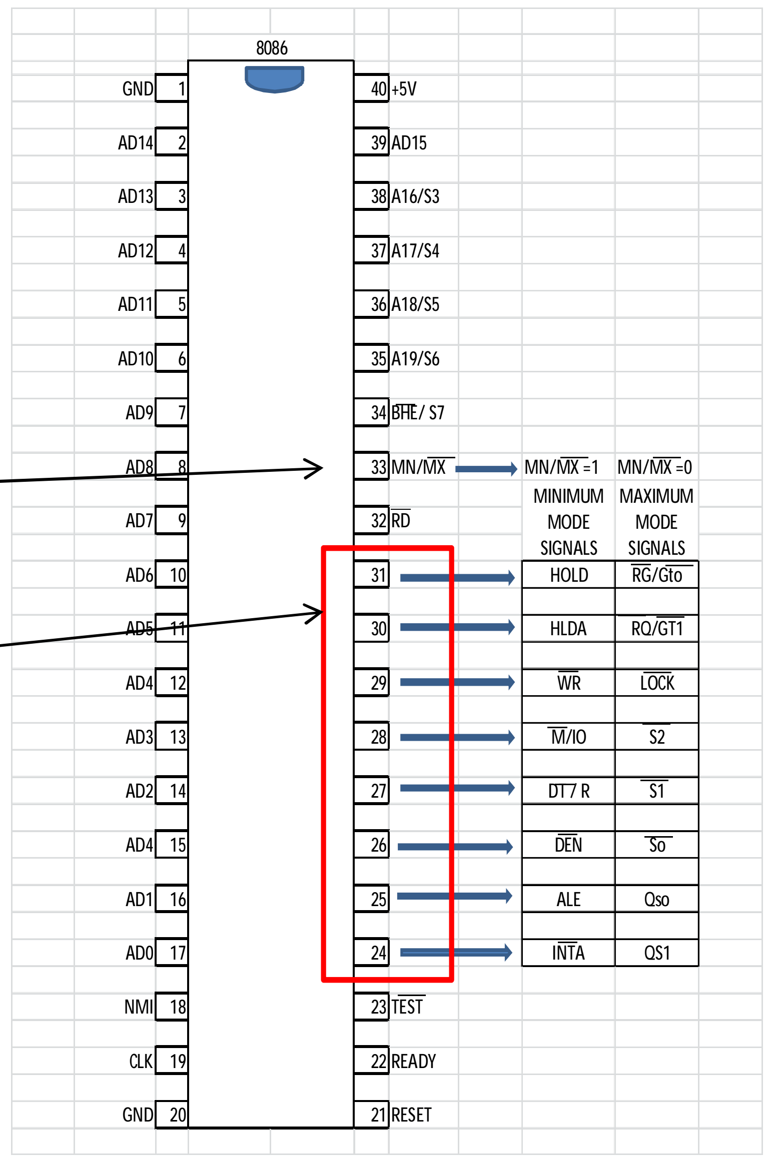
MINIMUM & MAXIMUM MODE

8086 works in two modes:

1) Minimum Mode and 2) Maximum Mode

If pin 33 $MN/\overline{MX}$ is high, it works in minimum mode and If pin 33 $MN/\overline{MX}$ is low, it works in maximum mode.

Pins 24 to 31 generates two different sets of signals. One set of signals is generated in minimum mode. Other set of signals is generated in maximum mode.



Pin Description for Minimum Mode

- Pin 24 is an interrupt acknowledge. When microprocessor receives INTR signal, it uses this pin to send acknowledgment by generating 3 active low signal.
- Pin 25 is an Address Latch Enable signal. It indicates that valid address is generated on bus AD_{15} – AD_0.It generates a pulse during T_1 state.It is connected to enable external latch .
- Pin 26 is a Data Enable signal. This signal is used to enable the external transceiver like 8286. Transceiver is used to separate the data from the address/data bus AD_{15} – AD_0.It is an active low signal.
- Pin 27 is a Data Transmit/Receive signal. It controls the direction of data flow through the transceiver. When it is high, data is transmitted out.When it is low, data is received in.

Pin Description for Minimum Mode

- Pin 28 is issued by the microprocessor to distinguish whether memory or /O access. When it is high, memory can be accessed. When it is low, I/O devices can be accessed.
- Pin 29 is a Write signal. It is used to write data in memory or output device depending on the status of M/IO signal. It is an active low signal.
- Pin 30 is a Hold Acknowledgement signal.It is issued after receiving the HOLD signal.It is an active high signal.
- Pin 31 During DMA operation microprocessor receives HOLD signal from DMA controller.

Pin Description for Maximum Mode

QS_1 and QS_0
Pin 24 and 25

- These pins provide the status of internal instruction queue.

QS_1	QS_0	Status
0	0	No operation
0	1	1st byte of opcode from queue
1	0	Empty queue
1	1	Subsequent byte from queue

Pin Description for Maximum Mode

$\overline{S_0}, \overline{S_1}, \overline{S_2}$

Pin 26, 27, 28

- These status signals indicate the operation being to be performed by the microprocessor.
- These information decoded by the Bus Controller 8288 which generates all memory and I/O control signals.

S_2	S_1	S_0	Status
0	0	0	Interrupt Acknowledge
0	0	1	I/O Read
0	1	0	I/O Write
0	1	1	Halt
1	0	0	Opcode Fetch
1	0	1	Memory Read
1	1	0	Memory Write
1	1	1	Passive

Pin Description for Maximum Mode

$\overline{\text{LOCK}}$

Pin 29

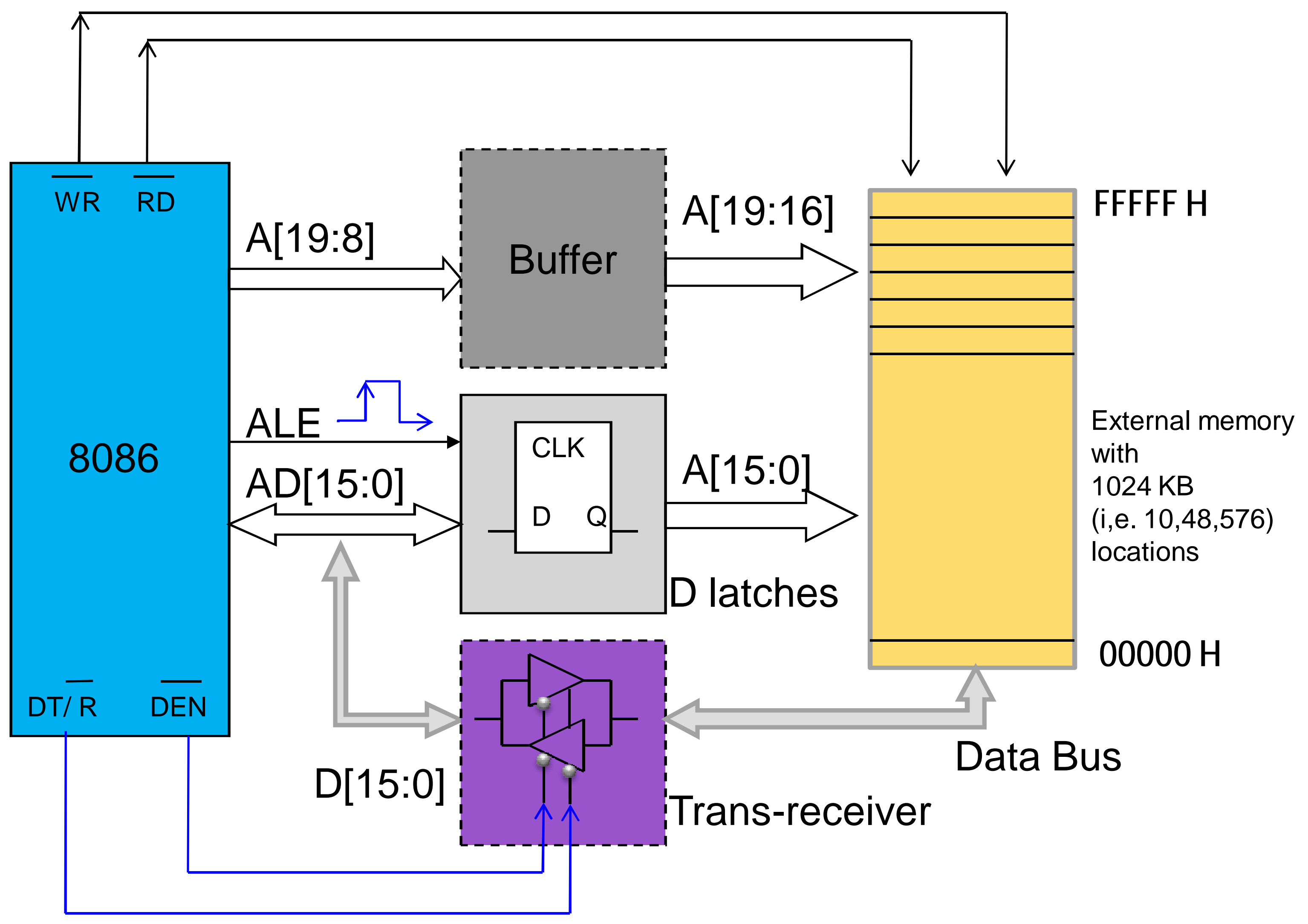
- **This signal indicates that external processors like 8087 should not request CPU to relinquish the system bus as it is locked with important operation. This pin is activated by using LOCK prefix before any instruction.**
- **When it goes low, all interrupts are masked and HOLD request is not granted.**

$\overline{RQ}/\overline{GT}_1$ and $\overline{RQ}/\overline{GT}_0$

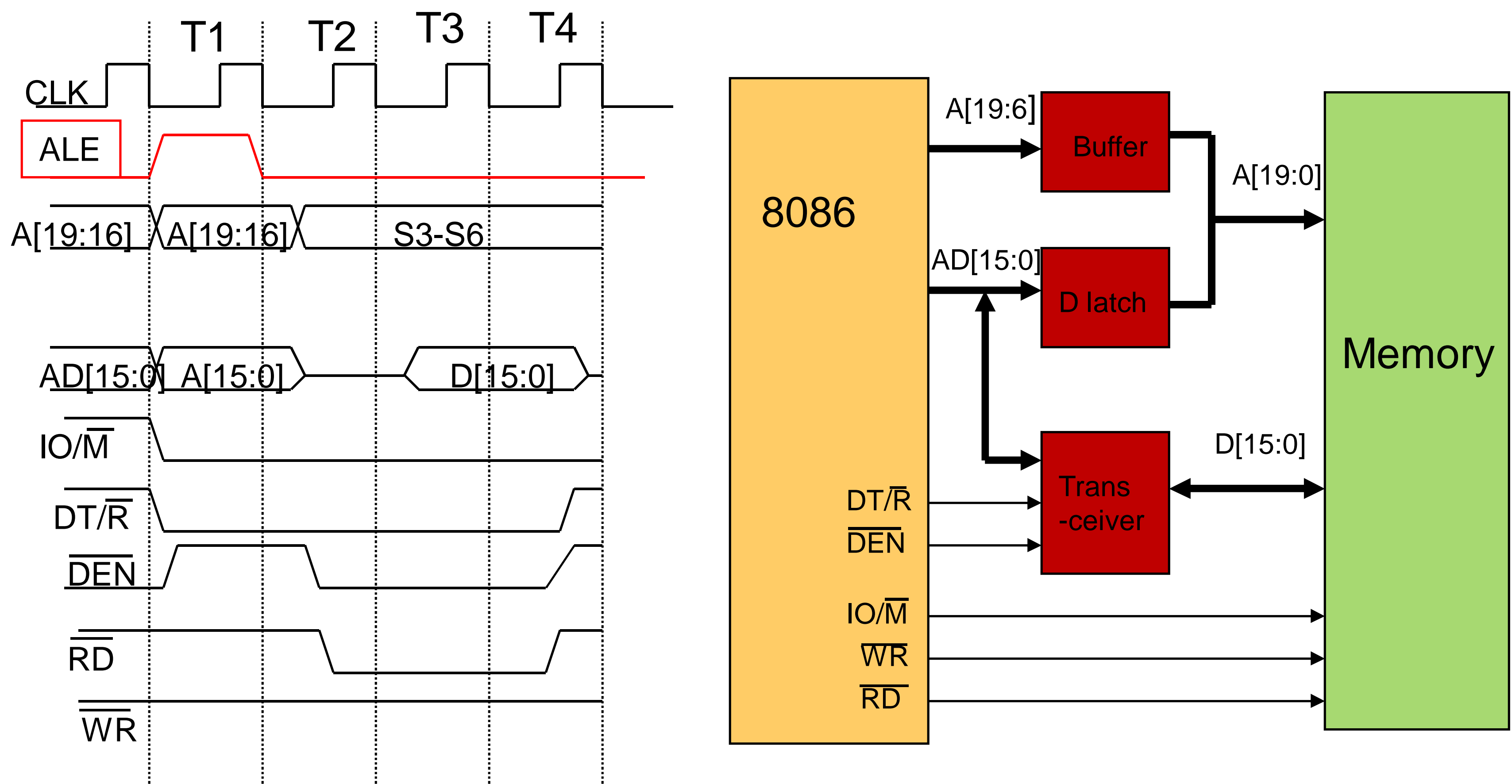
Pin 30 and 31 (Bi-directional)

- **These are Request/Grant pins.**
- **External processors like 8087 can request the CPU through these lines to release the system bus.**
- **After receiving the request, CPU sends acknowledge signal on the same lines.**
- **RQ/GT_0 has higher priority than RQ/GT_1.**

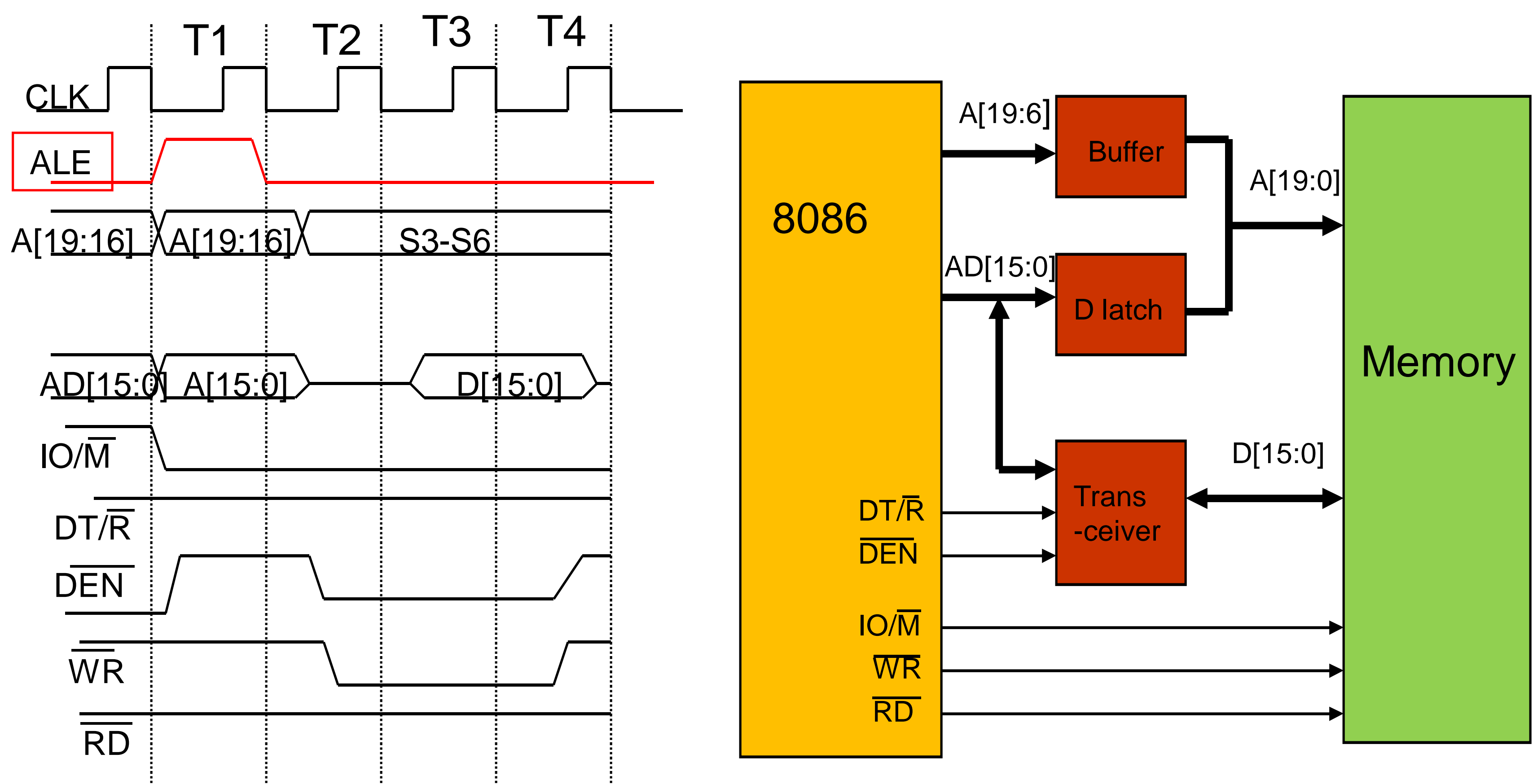
De-multiplexing Address/Data Pin Description



Memory Read Timing Diagrams



Memory Write Timing Diagrams



Chapter 2 Question Bank

- What is pipeline?
- Explain the function of Q 8086.
- Describe the function EU & BIU.
- Explain the function of various registers of 8086.
- Explain function of segment register & pointer .
- What are the various Flag of 8086?
- How 20-bit address is generated in 8086?
- Explain the Minimum and Maximum mode of 8086.
- Explain the timming diagram of memory read 8086.
- Explain the timming diagram of memory write 8086.

Summary

1.The 8086 logically divided into:

- BIU &
- EU

Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining.

2. For programme flexibility the 1Mbytes location is logically segmented (divided or organized) into Code Memory Segment (CMS), Data Memory Segment (DMS), Extra Memory Segment (EMS) and Stack Memory Segment (SMS).

3.To access a particular location of memory segment the 20-bit physical address is generated by the addition of Base Address (BA) provided by the segment register and 16/8 bit offset address/displacements (OA) is provided by Pointers/index registers.

4. The flags of 8086 can be divided into two types: Conditional Flags and Control Flags

5. 8086 works in two modes: 1) Minimum Mode 2) Maximum Mode.

CHAPTER-3 Instruction Set of 8086 Microprocessor

1. Topic 1: Machine Language Instruction format, addressing modes
2. Topic 2: Instruction set,
3. Topic 3: Groups of Instructions,

The student will be able to:

Understand the different types of instructions

Identify the addressing modes of instruction and the operation of an instructions

A programme is nothing but set of Instructions written sequentially one below the other and stored in computers memory for execution by microprocessor.

***Instruction** consists of a mnemonic and one or two operands (data).*

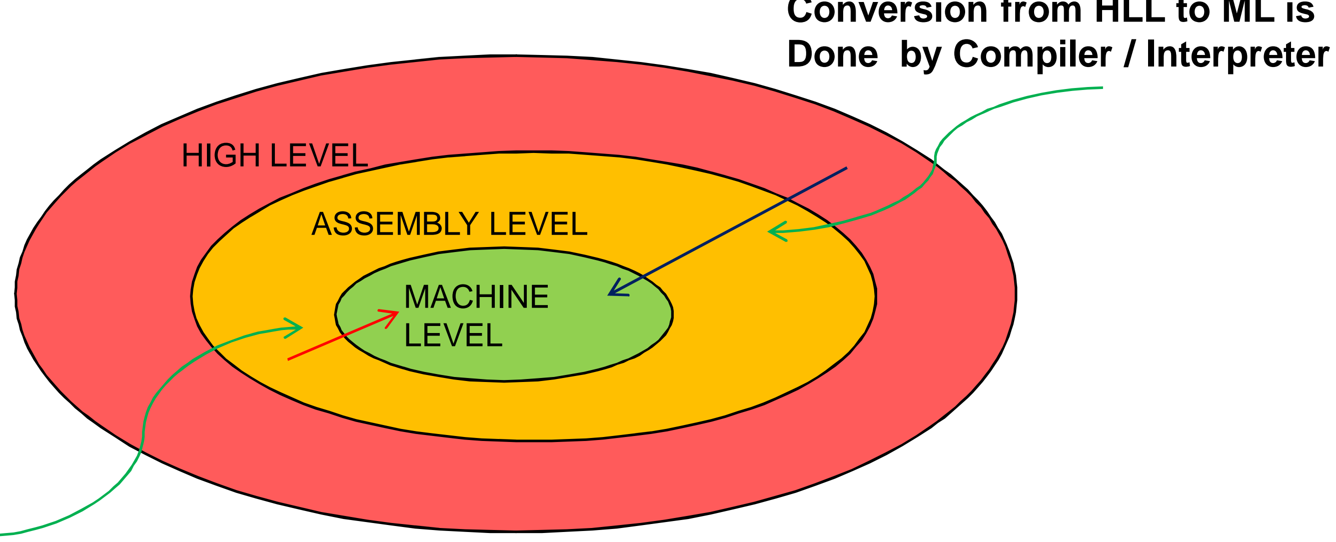
- ***Machine Language:*** In this Programs consist of 0s and 1s.

- ***Assembly Languages :*** It uses short form notations , called , ***mnemonics ,*** to write a programme .The Mnemonics are like MOV , ADD , SUB, etc.

- ***High level languages:*** It uses English like sentences with proper syntax to write a programme.

- ***Assembler** translates Assembly language program into machine code.*

- **Compilers** like Pascal, Basic, C etc ***translate the HLL program into machine code.*** The programmer does not have to be concerned with internal details of the CPU.



Conversion from ALL to ML is Done by Assembler like MASM

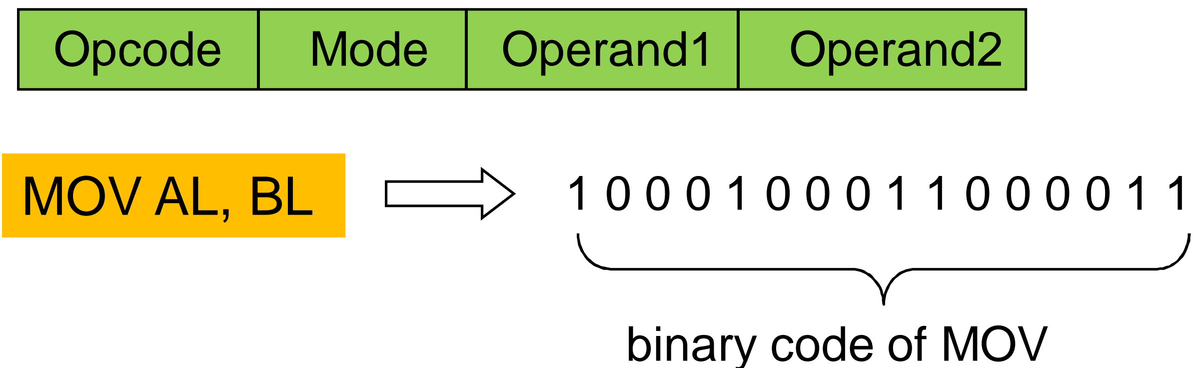
Instruction Format

- General Format of Instructions is

  Label: Opcode Operands ; Comment

  - **Label**: It provides a symbolic address that can be used in branch instructions
  - **Opcode:** It specifies the type of instructions
  - **Operands**: Instructions of 8086 family can have one, two, or zero operand
  - **Comments**: programmers can use for effective reference

- Machine Code Format



What is Addressing Modes?

- Addressing modes is define as the way in which data is addressed in the operand part of the instruction. It indicates the CPU finds from where to get data and where to store results

- When a CPU executes an instruction, it needs to know where to get data and where to store results. Such information is specified in the operand fields of the instruction.

1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 1 ⇒ MOV AL, BL

- An operand can be:
  - — A datum
  - — A register
  - — A memory location

Types of Addressing Modes

Addressing Modes	*Examples*
1] Immediate addressing	MOV AL, 1FH
2] Register addressing	MOV AL, BH
3] Direct addressing	MOV [0200H], AH
4] Register Indirect addressing	MOV DX, [SI]
5] Relative Based addressing	MOV BX, [SI+4]
6] Relative Indexed addressing	MOV [DI+8], BL
7] Based indexed addressing	MOV [BP+SI], AL
8] Relative Based indexed with displacement addressing	MOV AL, [BX+SI+2]

1] Immediate Addressing Mode

In this AM 8/16-bit Data is specified in the operand part of instruction immediately

- MOV AL,1Fh
- MOV AX,0FC8h
- MOV AH,4Eh
- MOV DX,1F00h

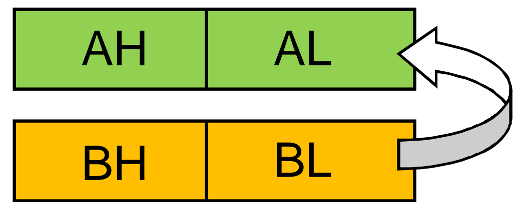
2] Register Addressing

- In this data is specified through register in operand part of instruction

- Operands are the names of internal register. The processor gets data from the register specified by instruction .

For Example: *move the value of register BL to register AL*

MOV AL, BL



- If AX = 1F00H and BX=8086H, after the execution of MOV AL, BL what are the new values of AX and BX?

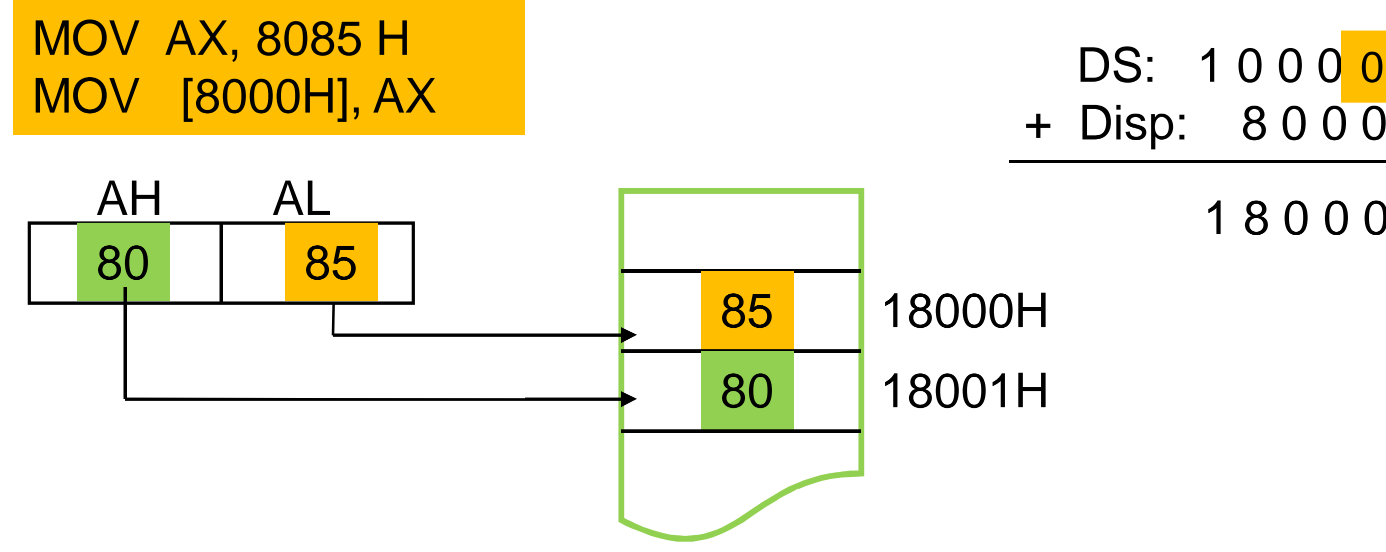
3] Direct Addressing

- In this AM 16-bit OFFSET address is specified , with symbol [ ] , in the operand part of the instruction.

- The processor will access memory location by adding this OFFSET with Base address given by DS.

DS × 10H + OFFSET = Memory location

— Example: If *DS = 1000H, then explain the operation*



3] Direct Addressing example

— Example: If *DS = 1000H, then explain the operation*

1
```
MOV  AX, F2A5 H
MOV   [8000H], AH
```

2
```
MOV  AX, F2A5 H
MOV   [8000H], AL
```

4] Register Indirect Addressing

- In this AM OFFSET address is specified indirectly through one of the registers BX, SI, DI in the instruction operand.
- The index register is specified using symbol [ ].
- This value is added with DS to generate 20-bit Physical address

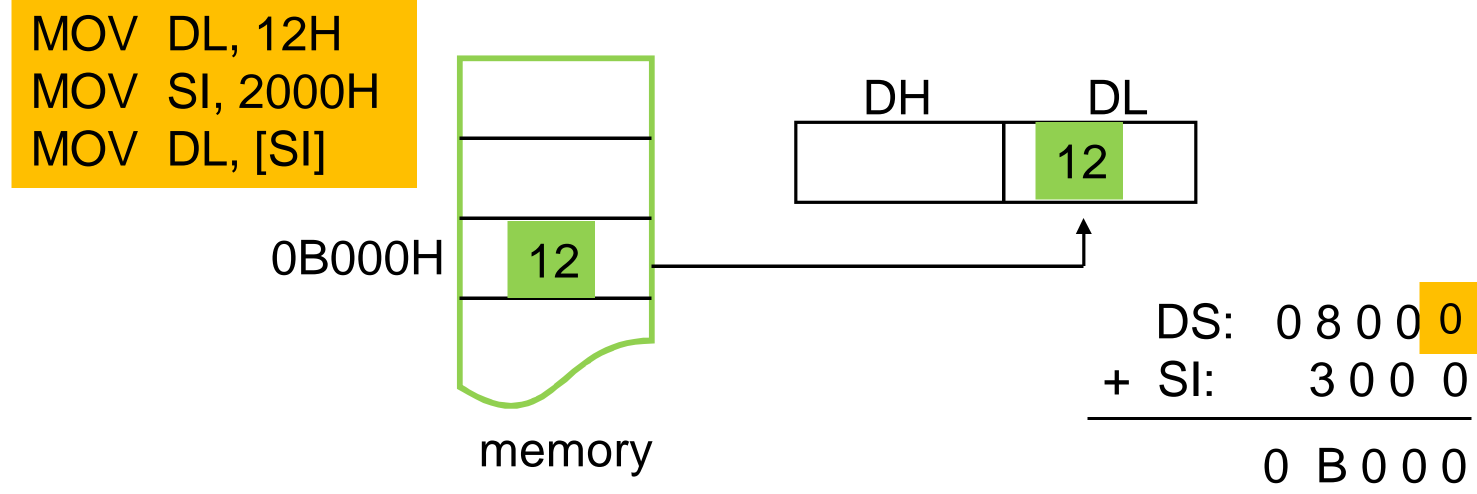
For Example: MOV DL, [SI]

- Memory address is calculated as following:

$$\begin{bmatrix} DS \end{bmatrix} \times 10H + \begin{bmatrix} BX \\ SI \\ DI \end{bmatrix} = \text{20-bit Memory address}$$

4] Register Indirect Addressing example

- Example 1: *assume DS = 0800H then explain the operation*



- Example 2: *assume DS = 0900H, BX=3000H*

```
MOV DL,E7H
MOV [BX], DL
```

5] Relative Based Addressing

- In this AM OFFSET address is specified indirectly by adding an 8-bit (or 16-bit) constant (displacement) with one of the registers BX, BP in the instruction operand.
- If BX appears in the instruction operand field, segment register DS is used in address calculation and If BP appears in the instruction operand field, segment register SS is used in address calculation

- Calculation of memory address

$$\begin{bmatrix} DS \\ SS \end{bmatrix} \times 10H + \begin{bmatrix} BX \\ BP \end{bmatrix} + \begin{bmatrix} \text{8/16-bit Displacement} \end{bmatrix} = \text{20-bit Memory address}$$

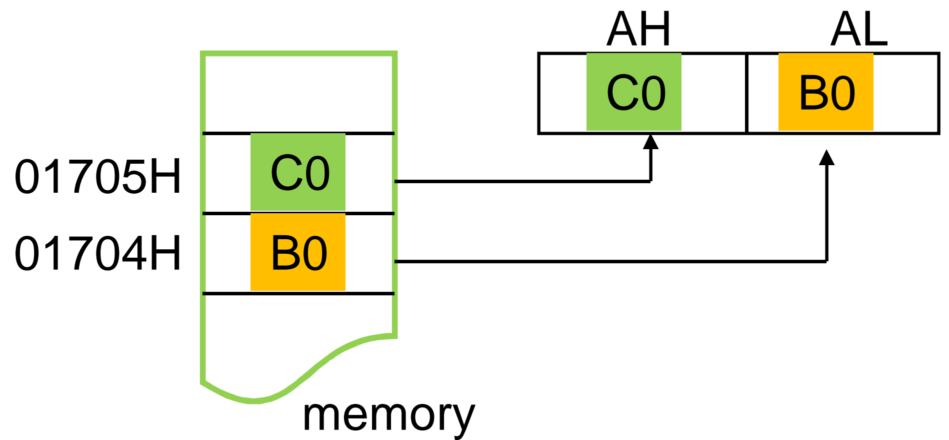
For Example: If DS = 2000H then explain MOV AX, [BX+4]

5]Relative Based Addressing example

- Example 1: *assume DS = 0100H, BX=0700H*

DS: 0 1 0 0 0
\+ BX: 0 7 0 0
\+ Disp.: 0 0 0 4

0 1 7 0 4



- Example 2: *assume SS = 0A00H, BP=0012H, CH = ABH*

MOV [BP +7], CH

6] Relative Indexed Addressing

- In this AM OFFSET address is specified indirectly by adding an 8-bit (or 16-bit) constant (displacement) with one of the Index registers SI, DI in the instruction operand. This value is then added with DS to generate 20-bit Physical address
  - Calculation for memory address

$$DS \times 10H + \begin{bmatrix} SI \\ DI \end{bmatrix} + \begin{bmatrix} 8 / 16 \text{ bit Displacement} \end{bmatrix} = \text{Memory address}$$

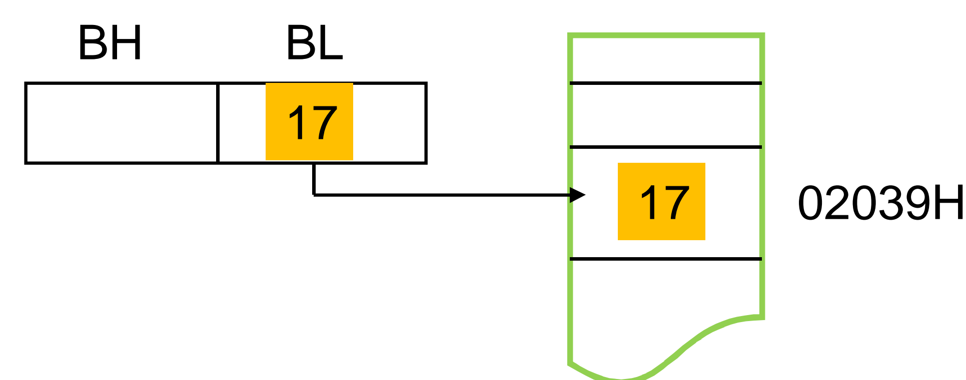
For Example:

```
MOV  BL,17H
MOV  DI,0030H
MOV  [DI+9],BL
```

- Example: *assume DS = 0200H*

MOV [DI+9], BL

	DS:	0 2 0 0 0
+	DI:	0 0 3 0
-	Disp.:	0 0 0 9
		0 2 0 3 9



7] Based Indexed Addressing

- In this AM OFFSET address is specified indirectly by adding one of the Index registers SI /DI with based register BX / BP in the instruction operand. This value is added with DS to generate 20-bit Physical address.

  - Calculation for memory address

$$\begin{bmatrix} DS \\ SS \end{bmatrix} \times 10H + \begin{bmatrix} BX \\ BP \end{bmatrix} + \begin{bmatrix} SI \\ DI \end{bmatrix} = \text{20-bit Memory address}$$

For Example:

MOV [BX] [SI], AH
or
MOV [BX+SI], AH

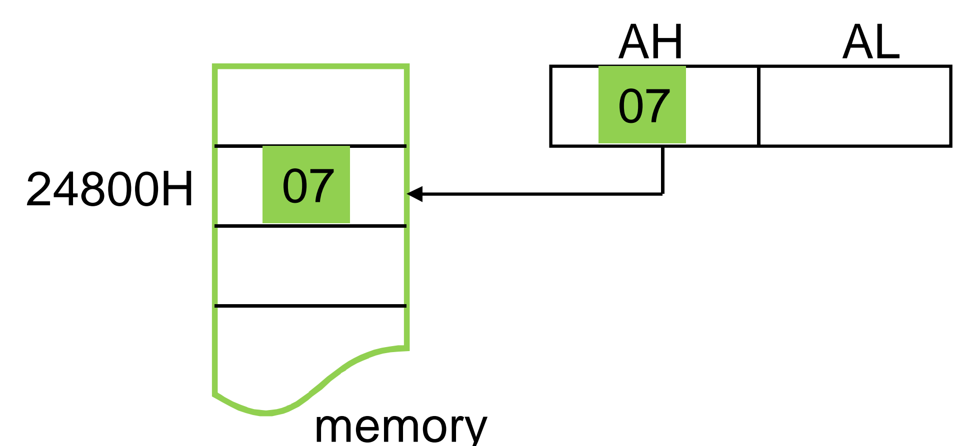
- If BX appears in the instruction operand field, then segment register DS is used in address calculation
- If BP appears in the instruction operand field, segment register SS is used in address calculation

7] Based Indexed Addressing example

- Example 1: *assume SS = 2000H explain the operation*

```
MOV  AH,07H
MOV  SI, 0800H
MOV  BP,4000H
MOV  [BP] [SI], AH
```

	SS:	2 0 0 0 0
+	BP:	4 0 0 0
+	SI.:	0 8 0 0
		2 4 8 0 0



- Example 2: *assume DS = 0B00H, BX=0112H, DI = 0003H, CH=ABH*

MOV [BX+DI], CH

8] Relative Based Indexed Addressing

- In this AM OFFSET address is specified indirectly by adding 8/16-bit displacement with one of the Index registers SI /DI with based register BX / BP in the instruction operand. This value is added with DS to generate 20-bit Physical address.

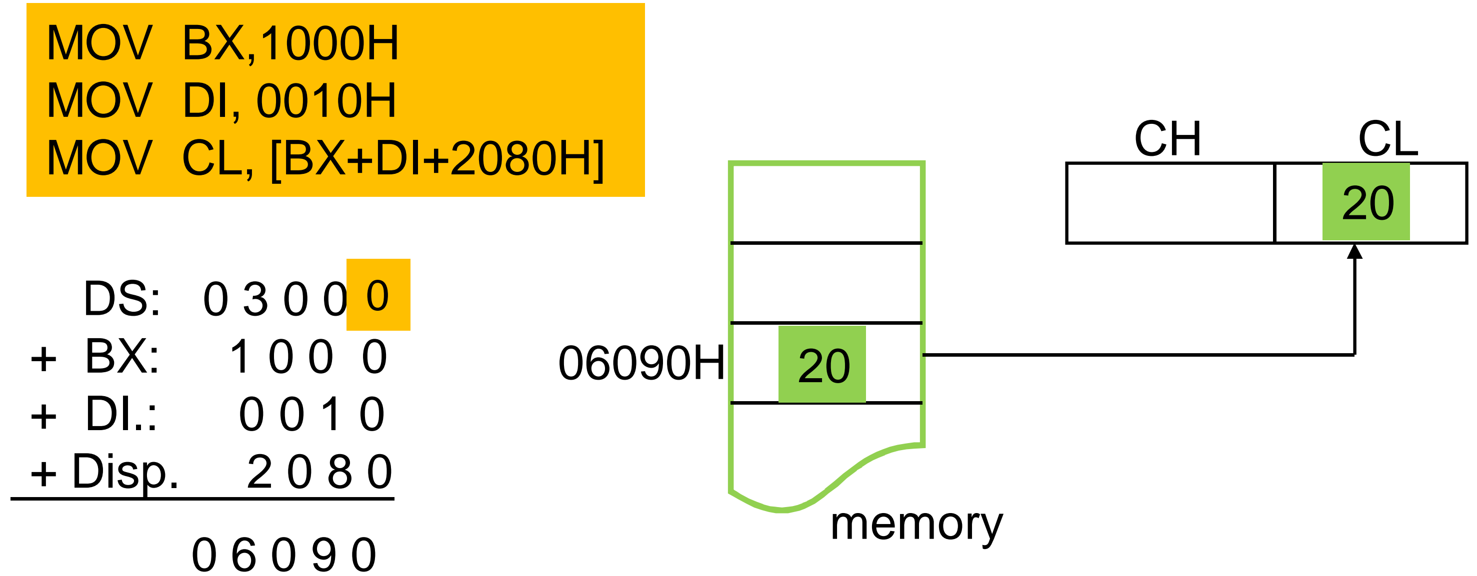
- Calculate memory address

$$\begin{bmatrix} DS \\ SS \end{bmatrix} \times 10H + \begin{bmatrix} BX \\ BP \end{bmatrix} + \begin{bmatrix} SI \\ DI \end{bmatrix} + \begin{bmatrix} \text{8/16-bit} \\ \text{displacement} \end{bmatrix}$$

For Example: MOV CL, [BX+DI+2080H]

8] Relative Based Indexed Addressing example

- Example 1: *assume DS = 0300H, BX=1000H, DI=0010H*



- Example 2: *assume SS = 1100H, BP=0110H, SI = 000AH, CH=ABH*

MOV [BP+SI+0010H], CH

Instruction Types

1] Data transfer instructions

2] Arithmetic instructions

3] String instructions

4] Bit manipulation instructions

5] Loop and jump instructions

6] Subroutine and interrupt instructions

7] Processor control instructions

1] Data transfer instructions

Various Data transfer Instructions are

a) Memory/Register Transfers

MOV	Move byte or word to register or memory
XCHG	Exchange byte or word
XLAT	Translate byte using look-up table

b) Stack Transfers

PUSH	Push data onto stack
PUSHF	Push flags onto stack
POP	Pop data from stack
POPF	Pop flags off stack

c) AH/Flags Transfers

LAHF	Load AH from flags
SAHF	Store AH into flags

1] Data transfer instructions

d) Address Translation

LEA	Load effective address
LDS	Load pointer using data segment
LES	Load pointer using extra segment

e) Port I/O Instructions

IN	Input byte or word from port
OUT	Output word to port

2] Arithmetic Instructions

Addition	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
Subtraction	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
Multiplication	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiplication
Division	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to double word

3] Bit Manipulation Instructions

a) Logical Instructions

- NOT *Destination*
  - Inverts each bit of the destination
  - Destination can be a register or a memory location
- AND *Destination, Source*
  - Performs logic AND operation for each bit of the destination with corresponding source bit and stores result into destination
  - Source can be immediate no while destination can be register or memory
  - Destination and source can not be both memory locations at the same time
- OR *Destination, Source*
  - Performs logic OR operation for each bit of the destination with source; stores result into destination
  - Source can be immediate no while destination can be register or memory
  - Destination and source can not be both memory locations at the same time

3] Bit Manipulation Instructions

a) Logical Instructions

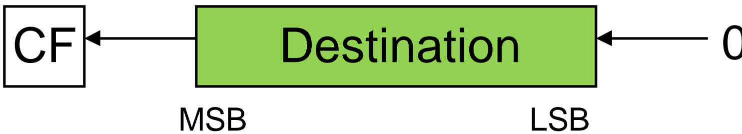
- XOR *Destination, Source*
  - Performs logic XOR operation for each bit of the destination with source; stores result into destination
  - Source can be immediate no while destination can be register or memory
  - Destination and source can not be both memory locations at the same time

- TEST *Destination, Source*
  - Performs logic AND operation for each bit of the destination with source
  - Updates Flags depending on the result of AND operation
  - Do not store the result of AND operation anywhere

3] Bit Manipulation Instructions

b) Shift Instruction

- SHL *Destination, Count*
  - SHift LEFT destination bits; the number of times bits shifted is given by CL
  - During the shift operation, the MSB of the destination is shifted into CF and zero is shifted into the LSB of the destination
  - Destination can be a register or a memory location



- SHR *Destination, Count*
  - SHift RIGHT destination bits; the number of times bits shifted is given by CL
  - During the shift operation, the LSB of the destination is shifted into CF and zero is shifted into the MSB of the destination
  - Destination can be a register or a memory location

3] Bit Manipulation Instructions

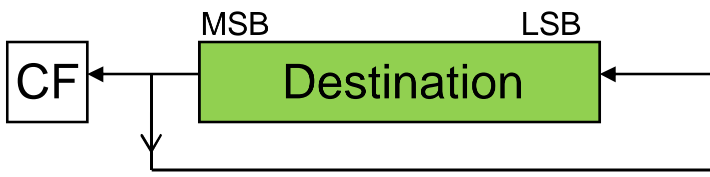
b) Shift Instructions

- SAR *Destination, Count*
  - Shift RIGHT destination bits; the number of times bits shifted is given by CL
  - The LSB of the destination is shifted into CF and the MSB of the destination is copied in the MSB itself i.e, it remains the same
  - Destination can be a register or a memory location

3] Bit Manipulation Instructions

c) Rotate Instructions

- ROL *Destination, Count*
  - Left shift destination bits; the number of times bits shifted is given by CL
  - The MSB of the destination is shifted into CF, it is also rotated into the LSB .
  - Destination can be a register or a memory location

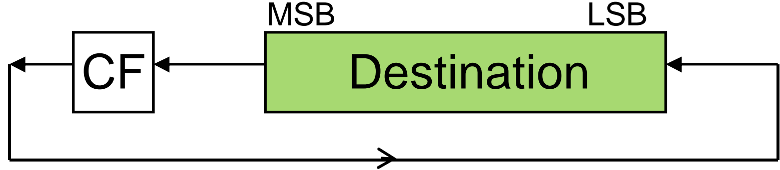


- ROR *Destination, Count*
  - Right shift destination bits; the number of times bits shifted is given by CL
  - The LSB of the destination is shifted into CF, it is also rotated into the MSB .
  - Destination can be a register or a memory location

3] Bit Manipulation Instructions

c) Rotate Instructions

- RCL *Destination, Count*
  - Left shift destination bits; the number of times bits shifted is given by CL
  - The MSB of the destination is shifted into CF; the old CF value is rotated into the LSB.
  - Destination can be a register or a memory location



- RCR *Destination, Count*
  - Right shift destination bits; the number of times bits shifted is given by CL
  - The LSB of the destination is shifted into CF, the old CF value is rotated into the MSB.
  - Destination can be a register or a memory location

4] String Instructions

- String is a collection of bytes or words stored in successive memory locations of DMS or EMS that can be up to 64KB in length .

- String instructions can have two operands. One is source string and the second is destination string .
  - Source string is located in Data Segment and SI register points to the current element of the source string
  - Destination string is located in Extra Segment and DI register points to the current element of the destination string

DS : SI	Source String
0510:0000	5F
0510:0001	4E
0510:0002	4A
0510:0003	5B
0510:0004	D0
0510:0005	CA
0510:0006	55

ES : DI	Destination String
02A8:2000	5F
02A8:2001	4E
02A8:2002	4A
02A8:2003	5B
02A8:2004	D0
02A8:2005	CA
02A8:2006	55

4] String Instructions

Repeat Prefix Instructions

- REP *String Instruction*

— The prefix instruction repeatedly execute the instruction until CX AUTO-decrements to 0 (During the execution, CX is decremented by one after execution of the string instruction).

— For Example: **MOV CX, 09**
REP MOVSB

By the above two instructions, the microprocessor will execute MOVSB 9 times.

— Execution flow of REP MOVSB is as below:

```
While (CX!=0)                     Check_CX:  If CX!=0 Then
  {                                              CX = CX –1;
    CX = CX –1;         OR                       MOVSB;
    MOVSB;                                       goto Check_CX;
  }                                          next instruction
```

4] String Instructions

Repeat Prefix Instructions

- REPZ *String Instruction*
  - Repeatedly execute the string instruction until CX=0 OR zero flag is clear
- REPNZ *String Instruction*
  - Repeatedly execute the string instruction until CX=0 OR zero flag is set
- REPE *String Instruction*
  - Repeatedly execute the string instruction until CX=0 OR zero flag is clear
- REPNE *String Instruction*
  - Repeatedly execute the string instruction until CX=0 OR zero flag is set

4] String Instructions

- MOVSB (MOVSW)
  - Move a byte (word) at source memory location of DMS (DS:SI) to destination memory location (ES:DI) and update SI and DI according to status of DF.
  - After transfer Increment SI/DI by 1 (or 2) if DF=0 and Decrement SI/DI if DF=1.

Example:

```
MOV  AX, 0510H
MOV DS, AX
MOV SI, 0
MOV AX, 0300H
MOV ES, AX
MOV DI, 100H
CLD
MOV CX, 5
REP MOVSB
INT 21
```

Source String

DS : SI	
0510:0000	5E
0510:0001	48
0510:0002	4F
0510:0003	50
0510:0004	50
0510:0005	45
0510:0006	52

Destination String

ES : DI	
0300:0100	5E
0300:0101	?
0300:0102	?
0300:0103	?
0300:0104	?

4] String Instructions

- MOVSB (MOVSW)
  - Move a byte (word) at source memory location of DMS (DS:SI) to destination memory location (ES:DI) and update SI and DI according to status of DF.
  - After transfer Increment SI/DI by 1 (or 2) if DF=0 and Decrement SI/DI if DF=1.
  - Example:

```
MOV  AX, 0510H
MOV DS, AX
MOV SI, 0
MOV AX, 0300H
MOV ES, AX
MOV DI, 100H
CLD
MOV CX, 5
REP MOVSB
INT 21
```

Source String

DS : SI	
0510:0000	5E
0510:0001	48
0510:0002	4F
0510:0003	50
0510:0004	50
0510:0005	45
0510:0006	52

Destination String

ES : DI	
0300:0100	5E
0300:0101	48
0300:0102	4F
0300:0103	50
0300:0104	50

4] String Instructions

- CMPSB (CMPSW)
  - Compare bytes (words) at memory locations DS:SI and ES:DI; update SI and DI according to DF and the width of the data being compared
  - Example:

Assume: ES = 02A8H
DI = 2000H
DS = 0510H
SI = 0000H

```
CLD
MOV CX, 7
REPZ CMPSB
INT 21
```

DS : SI		
0510:0000	4D	M
0510:0001	4A	J
0510:0002	45	E
0510:0003	54	T
0510:0004	48	H
0510:0005	57	W
0510:0006	41	A

Source String

ES : DI		
02A8:2000	4D	M
02A8:2001	4A	J
02A8:2002	45	E
02A8:2003	54	T
02A8:2004	48	H
02A8:2005	57	W
02A8:2006	4E	N

Destination String

What will be the values of CX after The execution?

4] String Instructions

- SCASB (SCASW)
  - Compare byte in AL (or word in AX) with data at memory location ES:DI; It updates DI depending status of DF and the length of the data being compare

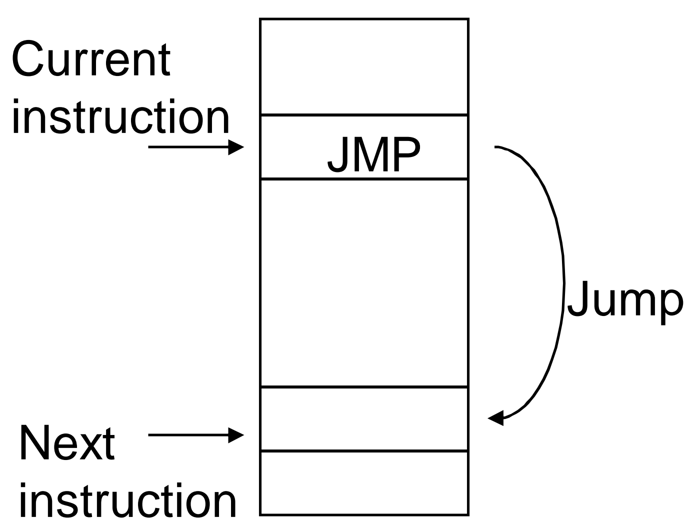
- LODSB (LODSW)
  - Load byte (word) at memory location DS:SI to AL (AX);
    It updates SI depending status of DF and the length of the data being transferred

- STOSB (STOSW)
  - Store byte (word) at in AL (AX) to memory location ES:DI;
    It updates DI depending status of DF and the length of the data being transferred

5] Program Transfer Instructions unconditional

- JMP *Label*
  - Unconditionally Jump to specified Label or address location.
  - Label can be represented by a word or Alphabet with no.

```
    MOV CX, 0007h
    MOV AX,F2FEh
    ADD AH,CL
    JMP  L1
    SUB AH,CL
L1: MOV [0200],AH
     INT21
```



5] Program Transfer Instructions conditional

- Conditional Jumps
  - JZ: *Label_1*
    - If ZF =1, jump to the target address labeled by *Label_1; else* do not jump
  - JNZ: *Label_1*
    - If ZF =0, jump to the target address labeled by *Label_1; else* do not jump

- Other Conditional Jumps

JNC	JAE	JNB	JC	JB	JNAE	JNG
JNE	JE	JNS	JS	JNO	JO	JNP
JPO	JP	JPE	JA	JBNE	JBE	JNA
JGE	JNL	JL	JNGE	JG	JNLE	JLE

5] Program Transfer Instructions conditional

Mnemonic	Meaning	Jump Condition
JA	Jump if Above	CF = 0 and ZF = 0
JAE	Jump if Above or Equal	CF = 0
JB	Jump if Below	CF = 1
JBE	Jump if Below or Equal	CF = 1 or ZF = 1
JC	Jump if Carry	CF = 1
JE	Jump if Equal	ZF = 1
JNC	Jump if Not Carry	CF = 0
JNE	Jump if Not Equal	ZF = 0
JNZ	Jump if Not Zero	ZF = 0
JPE	Jump if Parity Even	PF = 1
JPO	Jump if Parity Odd	PF = 0
JZ	Jump if Zero	ZF = 1

5] Program Transfer Instructions (Looping)

- LOOP *Label*

CX = CX –1
If CX != 0 Then
JMP Label else
Next Instruction

- LOOPE/LOOPZ *Label*

CX = CX –1
If CX != 0 & ZF=1 Then
JMP Label else
Next Instruction

- LOOPNE/LOOPNZ *Label*

CX = CX –1
If CX != 0 & ZF=0 Then
JMP Label else
Next Instruction

6] Processor Control Instructions

- CLC *Clear carry flag*
- STC *Set carry flag*
- CMC *Complement carry flag*
- CLD *Clear direction flag*
- STD *Set direction flag*
- CLI *Clear interrupt-enable flag*
- STI *Set interrupt-enable flag*
- HLT *Halt microprocessor operation*
- NOP *No operation*
- LOCK *Lock Bus During Next Instruction*

7] Subroutine Instructions

- A subroutine or procedure is a collection of instructions, written separately from main program, and can be called from a program.
- Instruction used is CALL *Procedure-Name*
- RET instruction lets the microprocessor to return from a subroutine to the called program.

Example

```
•••
MOV AL, 1
CALL  M1
MOV BL, 3
•••


M1 PROC
     MOV CL, 2
     RET
M1 ENDP
```

The order of execution will be :

```
MOV AL, 1
MOV CL, 2
MOV BL, 3
```

Chapter 3 Question Bank

- **What Opcode & Operand?**
- **List various Instructions of 8086.**
- **Describe the Data transfer Instruction.**
- Describe the Arithmetic Instruction.
- Describe the Data Bit manipilation Instruction.
- **Explain Various Program control Instruction.**
- Describe the Processor Control Instruction.
- **What Adressing mode ?**
- **Explain various Adressing modes of 8086.**

Summary

1. Addressing modes is define as the way in which data is addressed in the operand part of the instruction. It indicates the CPU finds from where to get data and where to store results.

2. 8086 has 8 Adressing modes a] Immediate addressing b] Register addressing c] Direct addressing d] Register Indirect addressing e] Relative Based f] Relative Indexed addressing g] Based indexed addressing h] Relative Based indexed with displacement addressing

3. 8086 Instructions cab be grouped as ,

1] Data transfer instructions

2] Arithmetic instructions

3] String instructions

4] Bit manipulation instructions

5] Loop and jump instructions

6] Subroutine and interrupt instructions

7] Processor control instructions

CHAPTER-4 The Art of Assembly Language Programming

1.	Topic 1:	Program development steps
2	Topic 2:	Assembly Language Programming Tools
3	Topic 3:	Assembler directives and Operators

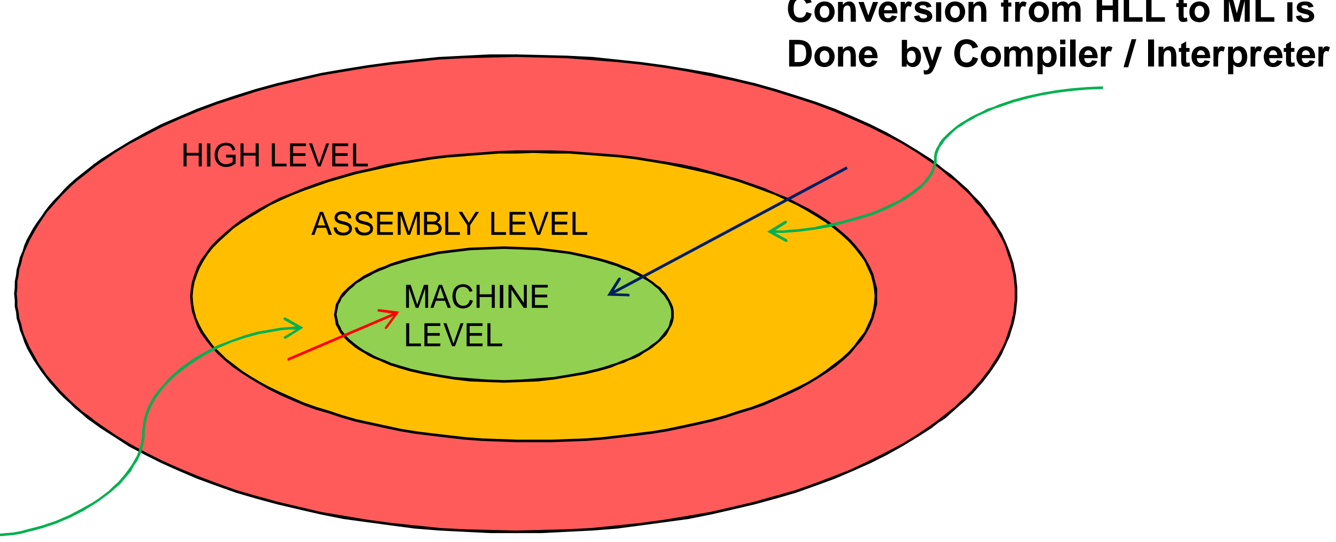
The student will be able to:

A programme is nothing but set of Instructions written sequentially one below the other and stored in computers memory for execution by microprocessor.

***Instruction* consists of a mnemonic and one or two operands (data).**

- ***Machine Language:*** In this Programs is written in 0s and 1s.
- ***Assembly Languages :*** It uses short form notations , called , ***mnemonics ,*** to write a programme .The Mnemonics are like MOV , ADD , SUB, etc.
- ***High level languages:*** It uses English like sentences with proper syntax to write a programme.
- ***Assembler* translates Assembly language program into machine code.**

Conversion from HLL to ML is Done by Compiler / Interpreter

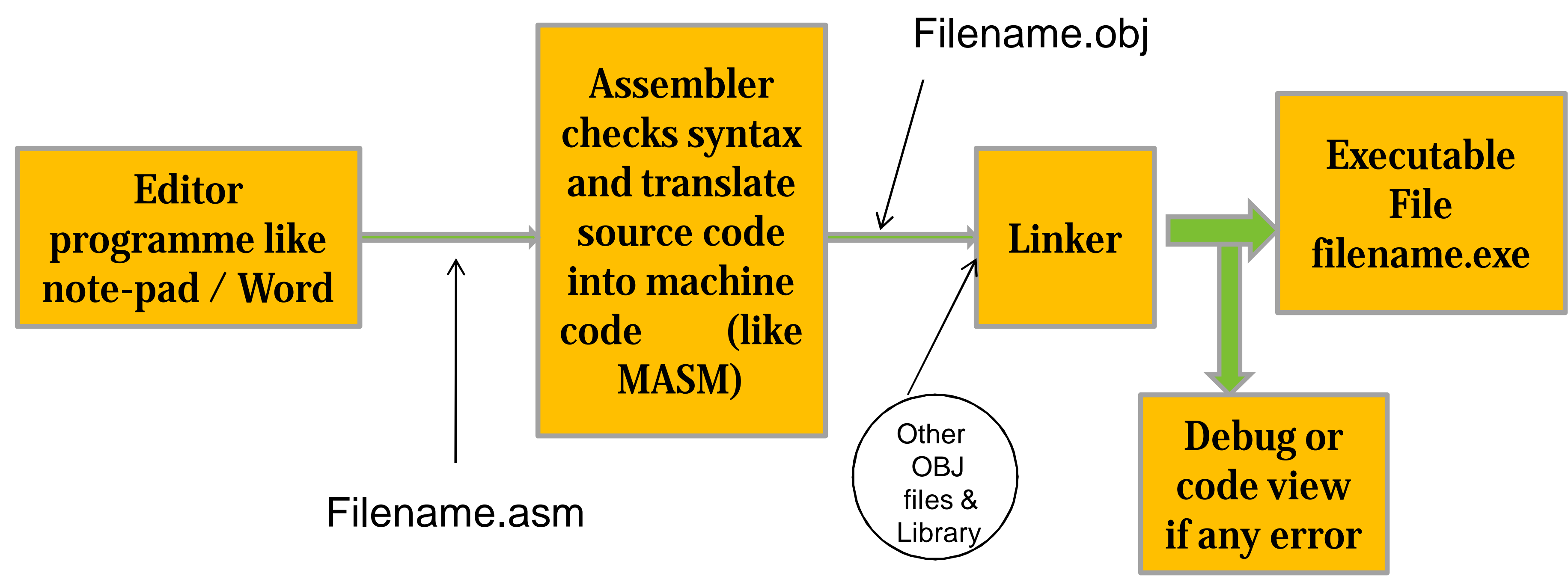


Conversion from ALL to ML is Done by Assembler like MASM

Assemble, Link and execute Program

Step & operation	Input	Software	Output
1.Editing / writing programme	Note pad or Word	Note pad / MS-Word	Filename.asm
2.Assemble	Filename.asm	MASM	Filename.obj
3.Link	Filename.obj	LINK	Filename.exe

How to Build Executable Programs



How to Build Executable Programs

Editor

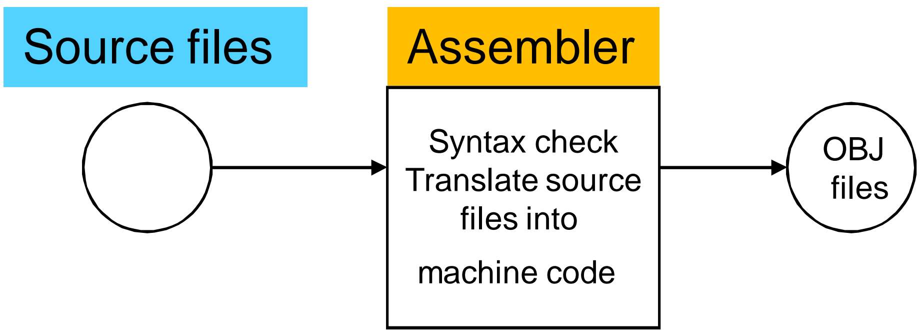
A text editor is required in order to create assembly language source files, where you'll be writing your code. You can use Notepad, DOS editor, or any other editor of your choice that produces plain ASCII text files.

Debugger

A debugger program allows tracing of program execution and examination of registers and memory content.

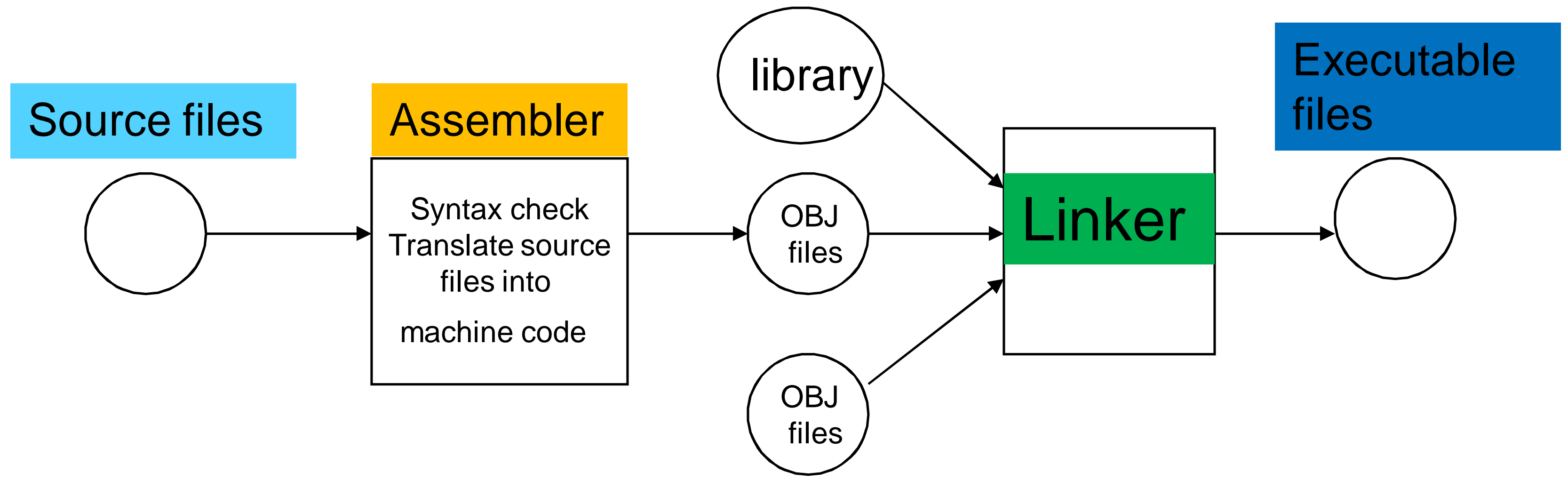
For *16-bit programs, MASM's debugger named CodeView can be used to debug .*

Assembler



An assembler is a program that converts source-code programs written in assembly language into object files in machine language. Popular assemblers include MASM (Macro Assembler from Microsoft), TASM (Turbo Assembler from Borland), NASM (Netwide Assembler for both Windows and Linux), and GNU assembler distributed by the free software foundation.

Linker

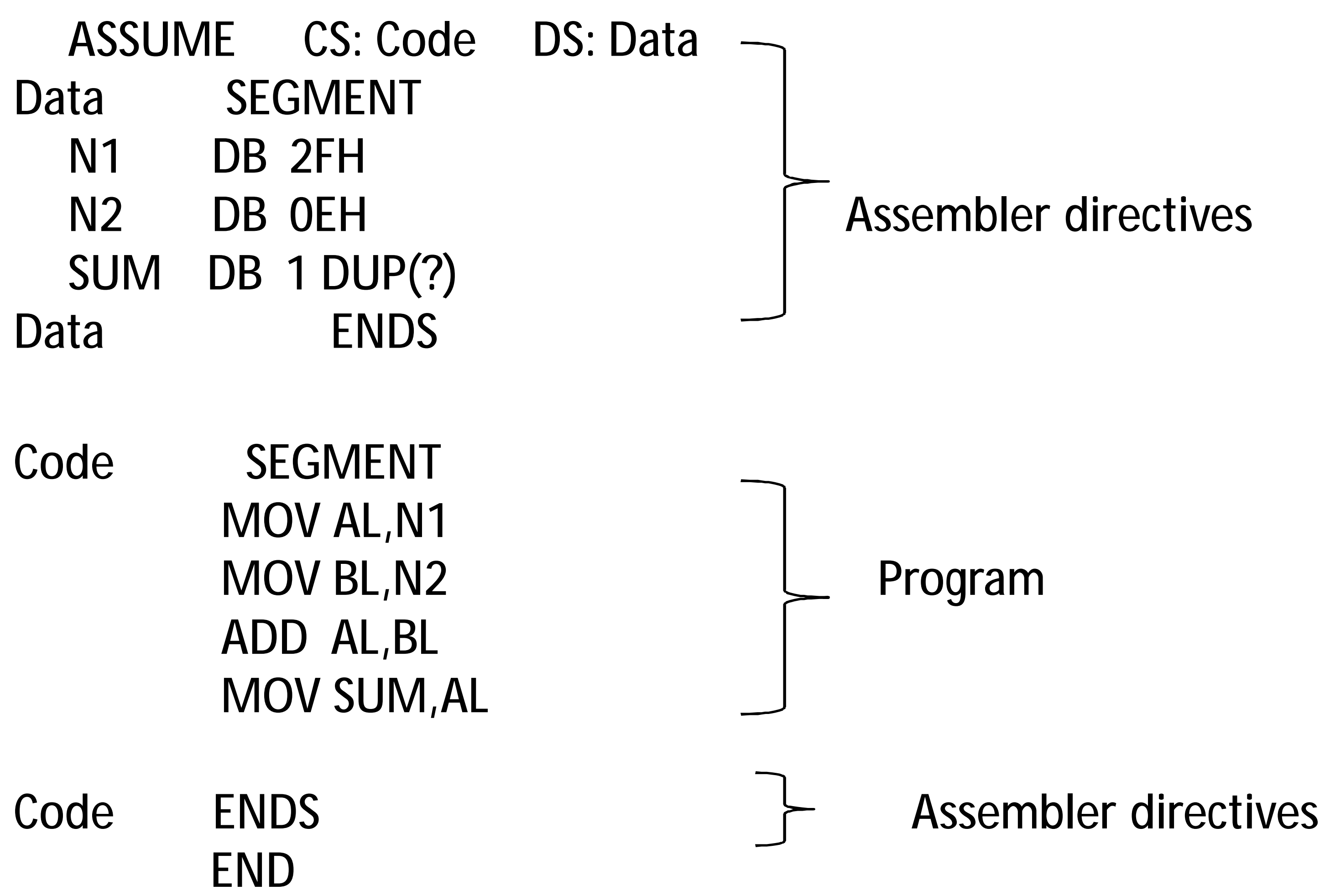


A linker program combines your program's object file created by the assembler with other object files and link libraries, producing a single executable program. You need a linker utility to produce executable files.
Two linkers: LINK.EXE and LINK32.EXE are provided with the MASM 6.15 distribution to link 16-bit real-address mode and 32-bit protected-address mode programs respectively.

Assembler Directives

It provides information to assist the assembler in producing executable code. It creates storage for a variable and initialize it. Assembler directives (pseudo-instructions) give directions to the assembler about how it should translate the Assembly language instructions into machine code, Where as other instructions discussed in the above section give command to the 8086 microprocessor. Assembler directives are specific for a particular assembler. However all the popular assemblers like the Intel 8086 macro assembler, the turbo assembler and the IBM macro assembler use common assembler directives. The basic structure of a program in ASM will look like this

Program format



ASSUME

1) The ASSUME directive tells the assembler the name of the logical segment it should use for a specified segment. It associates segment names with segment registers. For example

ASSUME CS: Code tells assembler that the instructions for programme are in the logical segment named Code.

Similarly for

ASSUME DS: Data tells assembler that for any program if instruction refers to the data segment then it should use the logical segment named Data. For example in the instruction MOV AX, [BX] the memory segment referred to by [BX] is in logical segment Data.

ASSUME

The 8086 contains a segment register (DS) that is dedicated to a data memory segment. This register is the default segment register used for all memory references used for data. The user is responsible loading the DS register with the appropriate value and telling the assembler where the DS register points so that it can calculate the offsets correctly. The standard is to define a segment to be a data segment. This is a convenient way of keeping data and code separate. The most common way of doing this is:

```
Data  SEGMENT
          ...
Data   ENDS          ;indicates the end of the data segment
```

Hence the Assume directive is required to inform assembler of location of DS pointer by : ASSUME DS: Data;

Data directive

2) Data storage directive

Each variable has a data type and is assigned a memory address by the program. Data directives are used to reserve and provide name for memory location in data segments. The symbols used for data types are:

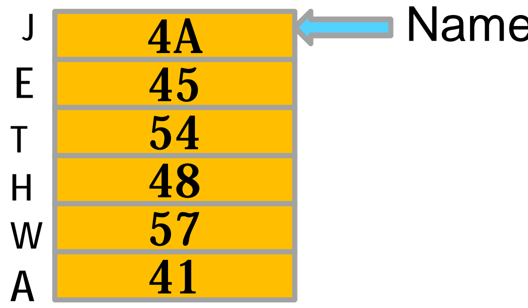
Data type	Symbol
Byte	B
Word	W
Double word	D
Quad Word	Q
Ten Bytes	T

Data directive Example

For byte variable we should use

- DB for declaration

N1 DB 4 ; initialise variable N1 with value 4 in decimal
N2 DB AFH ; initialise variable N2 with value AF in Hex
Name DB "JETHWA" ; allocates 6 byte with variable Name



DUP Operator

DUP Operator is used to create arrays of elements whose initialize value is same.

The basic syntax is count DUP (initial value)

Example :

N1 DB 100 DUP(0) ; create 100 bytes arrays with value 0 in each

N2 DW 5 DUP (?) ; CREATE 5 arrays of uninitialized words

L1 DB 5, 4, 3 DUP(2,3 DUP(0),1) is same as

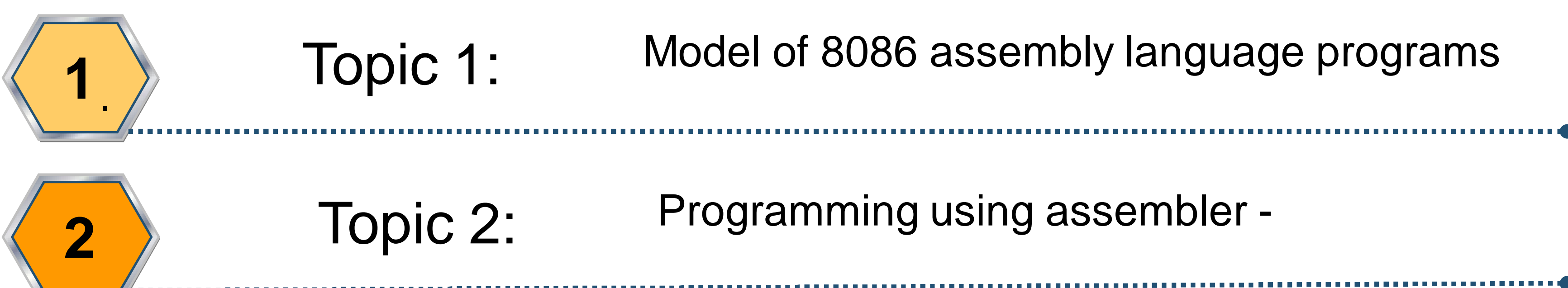
L1 DB 5,4,2,0,0,0,1,2,0,0,0,1,2,0,0,0,1

Chapter 4 Question Bank

- **What is Machine language?**
- What is Assembly language?
- What is High level language?
- Describe the function of Linker.
- Describe the function of Assembler & debugger.
- **What is Assemble directives?**
- **Explain various** Assemble directives.

Summary

1. A programme is nothing but set of Instructions written sequentially one below the other and stored in computers memory for execution by microprocessor. Program can be written in 3 levels a) Machine Language b) Assembly Languages c) High level languages
2. **Assembler** translates Assembly language program into machine code.
3. **Compilers** like Pascal, Basic, C etc ***translate the HLL program into machine code. T***he programmer does not have to be concerned with internal details of the CPU.
4. A text editor is required in order to create assembly language source files, where you'll be writing your code. You can use Notepad, DOS editor, or any other editor of your choice that produces lain ASCII text files.
5. A debugger program allows tracing of program execution and examination of registers and memory content.
6. An assembler is a program that converts source-code programs written in assembly language into object files in machine language.
7. A linker program combines your program's **object file created by the assembler with other object files and link libraries, producing a single executable program. You need a linker utility to produce executable files.**
8. **Assembler directives** provides information to assist the assembler in producing executable code. It creates storage for a variable and initialize it. **Assembler directives** (pseudo-instructions) give directions to the assembler about how it should translate the Assembly language instructions into machine code


1.
Topic 1:
Model of 8086 assembly language programs
2
Topic 2:
Programming using assembler -

CHAPTER-5 SPECIFIC OBJECTIVE / COURSE OUTCOME

The student will be able to:

Write a appropriate programs using editor

Run program using assembler & linker and Debug program using debugger

Program for 8-bit addition

```
; Program for addition of two 8-bit nos.    Comments / Remark

    ASSUME      CS: Code     DS: Data
Data          SEGMENT
    N1        DB  2FH
    N2        DB  0EH
    SUM       DB  1 DUP(?)
Data                 ENDS

Code           SEGMENT
              MOV AX, Data
              MOV DS,AX
              MOV AL,N1
              MOV BL,N2
              ADD  AL,BL
              MOV SUM,AL

Code          ENDS
              END
```

Program for 16-bit addition

; Program for addition of two 16-bit nos.   Comments / Remark

```
    ASSUME     CS: Code     DS: Data
Data          SEGMENT
    N1        DW  002FH
    N2        DW  000EH
    SUM       DW  1 DUP (?)
Data               ENDS

Code          SEGMENT
              MOV AX, Data
              MOV DS,AX
              MOV AX,N1
              MOV BX,N2
              ADD  AX,BX
              MOV SUM,AX
 Code         ENDS
              END
```

Program for 16-bit Multiplication

; Program for MULTIPLICATION of two 16-bit nos. Comments / Remark

```
    ASSUME     CS: Code     DS: Data
Data          SEGMENT
    N1        DW  002FH
    N2        DW  000EH
    RES       DW  2 DUP (?)
Data                ENDS

Code          SEGMENT
              MOV AX, Data
              MOV DS,AX
              MOV AX,N1
              MOV BX,N2
              MUL BX
              MOV RES,AX
              MOV RES+2,BX
  Code        ENDS
              END
```

Program for 16-bit Division

; Program for DIVISION of 16-bit no. Comments / Remark

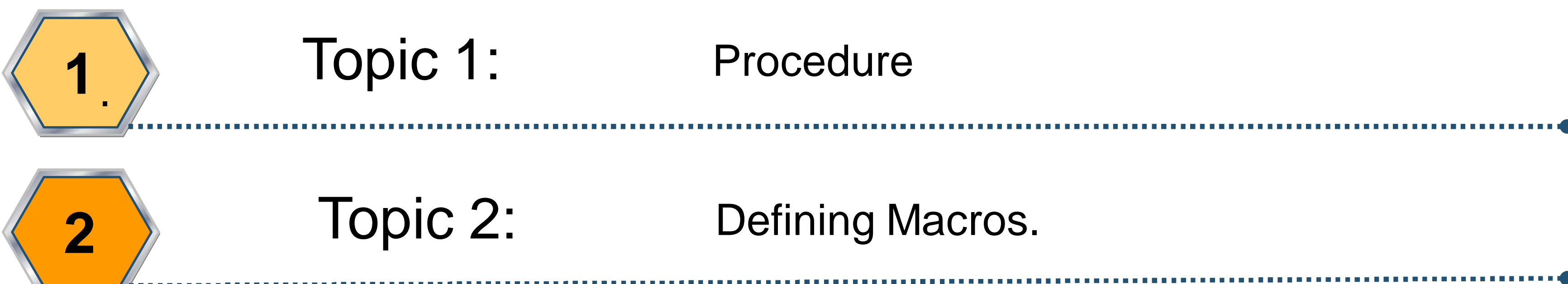
```
    ASSUME    CS: Code    DS: Data
Data        SEGMENT
    N1      DW  0F2FH
    N2      DB  0EH
    RESQ    DB  2 DUP (?)
Data              ENDS

Code         SEGMENT
             MOV AX, Data
             MOV DS,AX
             MOV AX,N1
             MOV BL,N2
             DIV BL
             MOV RESQ,AL
             MOV RESQ+1,AH
 Code        ENDS
             END
```

Program for 32-bit Division

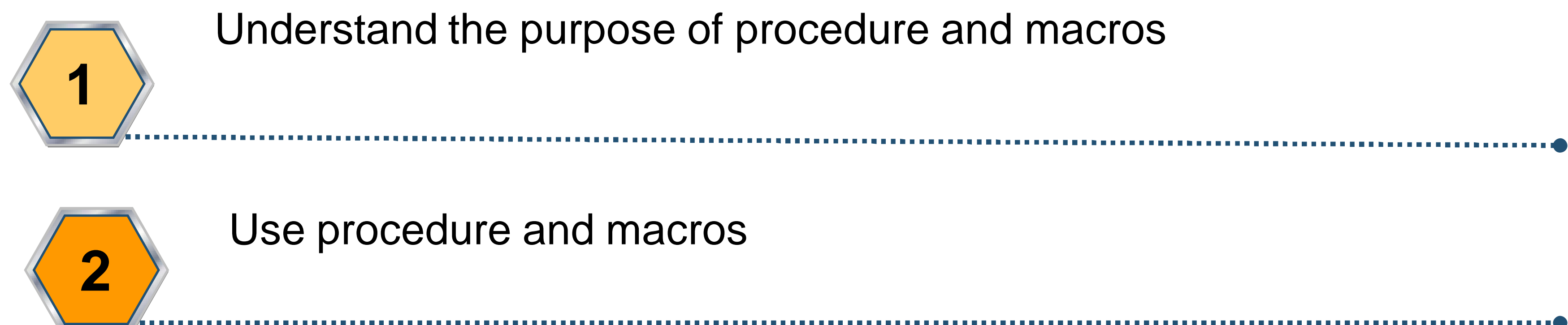
```
; Program for DIVISION of 32-bit no.    Comments / Remark
    ASSUME     CS: Code     DS: Data
 Data          SEGMENT
    N1        DD  FE000F2FH
    N2        DW   E40EH
    RESQ     DW  2 DUP (?)
 Data                ENDS
Code           SEGMENT
              MOV AX, Data
              MOV DS,AX
              MOV AX,N1
              MOV DX,N1+2
              MOV BX,N2
              DIV BX
              MOV RESQ,AX
              MOV RESQ+2,DX
Code          ENDS
              END
```

Chapter 5 Question Bank

- Write program to transfer a block of 50 bytes B1 to another block B2.The block B1 begins with offset address 1000h and block B2 from 2000h?
- Write program to exchange data of block of 10 bytes B1 to with another block B2.The block B1 begins with offset address 0200h and block B2 from 0300h?
- Write program to arrange a block of 50 bytes in ascending order. The block begins with offset address 1000h?
- Write program to arrange a block of 50 bytes in descending order. The block begins with offset address 1000h?


1.
Topic 1:
Procedure
2
Topic 2:
Defining Macros.

The student will be able to:



Procedures

Procedure is a part of code that can be called from your program in order to make some specific task. Procedures make program more structural and easier to understand. Generally procedure returns to the same point from where it was called.

The syntax for procedure declaration:

```
name PROC

    ; here goes the code
    ; of the procedure ...

RET
name ENDP
```

Procedures

name - is the procedure name, the same name should be in the top and the bottom, this is used to check correct closing of procedures.

RET instruction is used from procedure

PROC and ENDP are compiler directives, so they are not assembled into any real machine code. Compiler just remembers the address of procedure.

CALL instruction is used to call a procedure.

Procedures example

```
ORG    100h
MOV  AL,2FH
MOV   BL,F2H
CALL   m1
MOV  [SI] , AX
RET                    ; return to operating system.
```

Procedure for multiplication

```
m1     PROC
MUL  BL
RET                    ; return to caller.
m1     ENDP
END
```

Procedures example

There are several ways to pass parameters to procedure, the easiest way to pass parameters is by using registers, here is another example of a procedure that receives two parameters in AL and BL registers, multiplies these parameters and returns the result in AX register:

```
ORG    100h
MOV    AL, 1
MOV    BL, 2
CALL   m2
CALL   m2
CALL   m2
CALL   m2
RET                   ; return to operating
system.


m2     PROC
MUL    BL               ; AX = AL * BL.
RET                   ; return to caller.
m2     ENDP
END
```

In this example value of **AL** register is update every time the procedure is called, **BL** register stays unchanged, so this algorithm calculates **2** in power of **4**,

so final result in **AX** register is **16** (or **10h**).

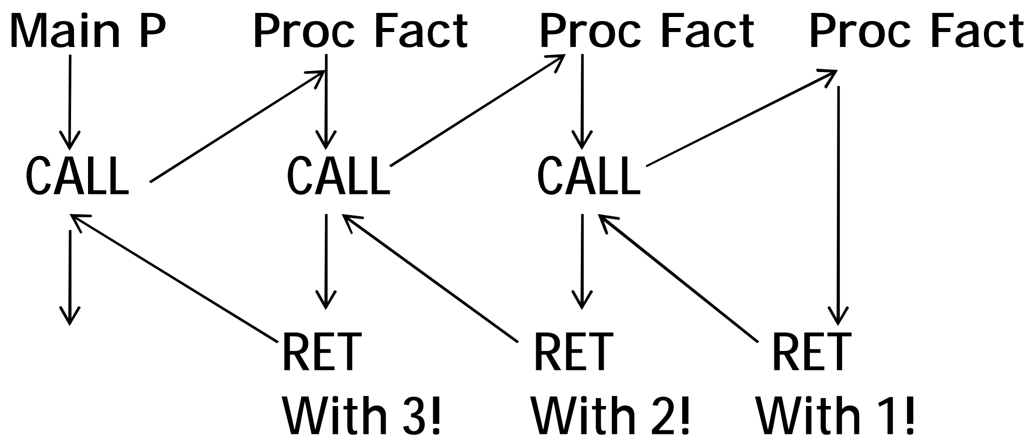
NEAR CALL & FAR CALL

A near CALL is a call to a procedure which is in the same code memory segment as that of CALL instruction in this the 8086 decrements Stack pointer by 2 and copies the IP on the STACK.

A far CALL is a call to a procedure which is in different code segment as that of CALL instruction. . In this the 8086 decrements Stack pointer by 2 and copies the CS first on the STACK and then again decrement SP by 2 to copy IP on the STACK.

Recursive Procedure

Recursive Procedure: A recursive procedure is procedure which calls itself. It is used to work with complex data structures called trees. If the procedure is called with N (known as recursion depth) =3 then the n is decremented by 1 after each procedure CALL and the procedure is called until n=0 as shown in the diagram below:



To find out factorial of number

```
;Program to find out factorial of number  Using Recursion
Data         SEGMENT
    NUMBER              DB 03H
    FACTORIAL           DW 1DUP(?)
ENDS

Stack        SEGMENT
   DW   128  DUP(0)
ENDS

Code         SEGMENT
   ASSUME CS:Code, DS:Data, SS:Stack

; INITIALISE SEGMENT REGISTERS:
              MOV AX, Data
              MOV DS, AX
              MOV AX, Stack
              MOV SS, AX

              MOV CX,NUMBER
              CALL  FACT
              RET
```

To find out factorial of number

```
;PROCEDURE FOR FACTORIAL PROGRAM
;CX CONTAINS INPUT NUMBER
;DX CONTAINS RESULT
FACT PROC  NEAR
        CMP CX, 01H
      JNE CONT
        MOV DX,01H
        RET
CONT:  PUSH CX        ; FOR BACKUP
        DEC CX
        CALL FACT
        POP AX          ; BACKUP OF CX IE N
        MUL DX                  ; N*(N-1)!
        MOV DX, AX ; RESULT INTO DX
        RET
FACT ENDP
ENDS
```

Reentrant Procedure

Reentrant Procedure: A program or subroutine is called reentrant if it can be interrupted in the mid (i.e. the control flow is transferred outside of the subroutine, either due to an internal action such as a jump or call, or by an external action such as a hardware interrupt or signal), and then can then safely be called again before its previous invocation has been completed, and once the reentered invocation completes, the previous invocations should be able to resume execution correctly.

If procedure1 is called from main program and procedure2 is called from procedure1 and procedure1 again from procedure2 then such is called as reentrant procedure as shown below:

Macros

Macros

Macros are just like procedures, but not really. Macros look like procedures, but they exist only until your code is compiled, after compilation all macros are replaced with real instructions. For Macro assembler generates the code in program each time where the macro is “called”. If you declared a macro and never used it in your code, compiler will simply ignore it.

Macro definition:

```
name    MACRO
[parameters,...]
<instructions>
ENDM
```

Macros example

```
MyMacro    MACRO  p1, p2, p3
MOV AX, p1
MOV BX, p2
MOV CX, p3
ENDM

ORG 100h
MyMacro 1, 2, 3
MyMacro 4, 5, DX
RET
```

The code is expanded into:

```
MOV AX, 00001h
MOV BX, 00002h
MOV CX, 00003h
MOV AX, 00004h
MOV BX, 00005h
MOV CX, DX
```

Compare Procedure & Macro

Procedure	Macro
Accessed by CALL & RET instruction	Accessed during assembly with name given during program execution to macro when defined
Machine code for instruction is put only once in the memory	Machine code is generated for instruction each time when macro is called.
With procedures less memory is required	With macro more memory is required
Parameters can be passed in registers, memory locations or stack	Parameters passed as part of statement which calls macro

Advantages and Disadvantages of MACRO

Advantages of MACRO

- üProgram written with MACRO is more readable
- üMACRO can be called by just writing its name along with its parameters;
  hence no extra code is required like CALL & RET.
- üExecution time is less as compared to Procedure
- üFinding errors is easy

Disadvantages of MACRO

- üObject code is generated every time Macro is called, hence object file
  becomes lengthy
- üFor large group of instruction macro is not preferred

Chapter 6 Question Bank

- What is Procedures?
- What are the instructions to implement Procedures?
- What is Re-entrant Procedures?
- Describe the function MACROS.
- What are the differences between Procedures & MACROS.
- List various Advantages and disadvantages of MACROS.

Summary

1. Procedure is a part of code that can be called from your program in order to make some specific task. Procedures make program more structural and easier to understand. Generally procedure returns to the same point from where it was called. CALL instruction is used to call a procedure.

2. A near CALL is a call to a procedure which is in the same code memory segment as that of CALL instruction in this the 8086 decrements Stack pointer by 2 and copies the IP on the STACK.

3. A far CALL is a call to a procedure which is in different code segment as that of CALL instruction. . In this the 8086 decrements Stack pointer by 2 and copies the CS first on the STACK and then again decrement SP by 2 to copy IP on the STACK.

4. Reentrant Procedure: A program or subroutine is called reentrant if it can be interrupted in the middle (i.e. the control flow is transferred outside of the subroutine, either due to an internal action such as a jump or call, or by an external action such as a hardware interrupt or signal), can then safely be called again before its previous invocations have been completed, and once the reentered invocation completes, the previous invocations should be able to resume execution correctly.

5. Macros are just like procedures, but not really. Macros look like procedures, but they exist only until your code is compiled, after compilation all macros are replaced with real instructions. For Macro assembler generates the code in program each time where the macro is “called”. If you declared a macro and never used it in your code, compiler will simply ignore it.

Recommended Books:

1. Advanced Microprocessor and Peripherals (Architecture, Programming & Interfacing) by A.K. Roy & K.M. Bhurchandi, Tata Mcgraw Hill
2. Fundamentals of MIcroprocessors by B RAM, Dhanpat Rai Publications
3. Microprocessors by A.P.Godse, Technical Publications
4. 8085 Microprocessor: Programming And Interfacing 1st Edition Author: N.K.Srinath, PHI Learning Private Limited
5. Microprocessor 8085 And Its Interfacing, Author A.P.Mathur , PHI Learning Pvt. Ltd.
6. The 8088 and 8086 Microprocessors: , Author: Walter A. Triebel, Avtar Singh,
7. Microprocessor 8085, 8086 , by Abhishek Yadav
8. Microprocessors: Theory and Applications : Intel and Motorola by Mohamed Rafiquzzaman
9. "Microcomputer Systems: The 8086/88 Family", Liu, Gibson, 2nd Edition, PHI Learning Private Limited

References Books:

1. Microprocessor Architecture,Programming and Applications with the 8085 by Ramesh S. Gaonkar , Penram International Publishing (India)
2. Microprocessor & interfacing (programming & hardware) Revised Second Edition by Douglas V. Hall , Tata McGraw Hill
3. The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386 Barry Bray , Pearson Education; Eighth edition (2011)
4. The 8086/8088 Family: Design, Programming And Interfacing 1st Edition, John Uffenbeck, Prentice-Hall
5. Microcomputer Systems the 8086/8088 Family : Architecture, Programming and Design 2nd Edition Author: Gleen A.Gibson , Prentice-Hall
6. 8085 Microprocessor: Programming And Interfacing 1st Edition Author: N.K.Srinath, PHI Learning Private Limited
7. The 8086 Microprocessor :Programming & Interfacing the PC with CD Kenneth Ayala, Publisher: Cengage Learning
8. Assembly programming and the 8086 microprocessor, Douglas Samuel Jones ,Oxford University Press

References Web:

1. www.intel.com
2. www.pcguide.com/ref/CPU
3. www.CPU-World.com /Arch /
4. www.techsource .com / Engineering parts/ microprocessor.html
5. *www.slideshare.net*
6. *www.powershow.com*
7. *www.authorstream.com*
8. *www.youtube.com*
9. *www.scribd.com*
10. *www.eazynotes.com*
11. *www.electronicstutorialsblog.com*
12. *ece.uprm.edu*