Unit-1

Topics Covered:

8086 Architecture	Functional diagram, Registers, Programming Model, Memory-addresses and organization, Signal Descriptions and Interrupts.
Instruction Set and Assembly Language Programming of 8086	Instruction formats, Addressing modes, Instruction set, Assembler Directives, Macros, Programming- Control flow instructions, sorting, and string manipulations.

8086 Architecture

Year	Processor	ALU size	Data	Address	Clock rate	Feature size
1971	4004	4-bit	4-bit	12-bit (Mux)	740 KHz	10 µm
1972	8008	8-bit	8-bit	14-bit	200 – 800 KHz	10 µm
1974	8080	8-bit	8-bit	16-bit	2 – 3.125 MHz	6 µm
1976	8085	8-bit	8-bit	16-bit	3 – 6 MHz	3 µm
1978	8086	16-bit	16-bit	20-bit	5 – 10 MHz	3 µm
1979	8088	16-bit	8-bit	20-bit	5 – 10 MHz	3 µm

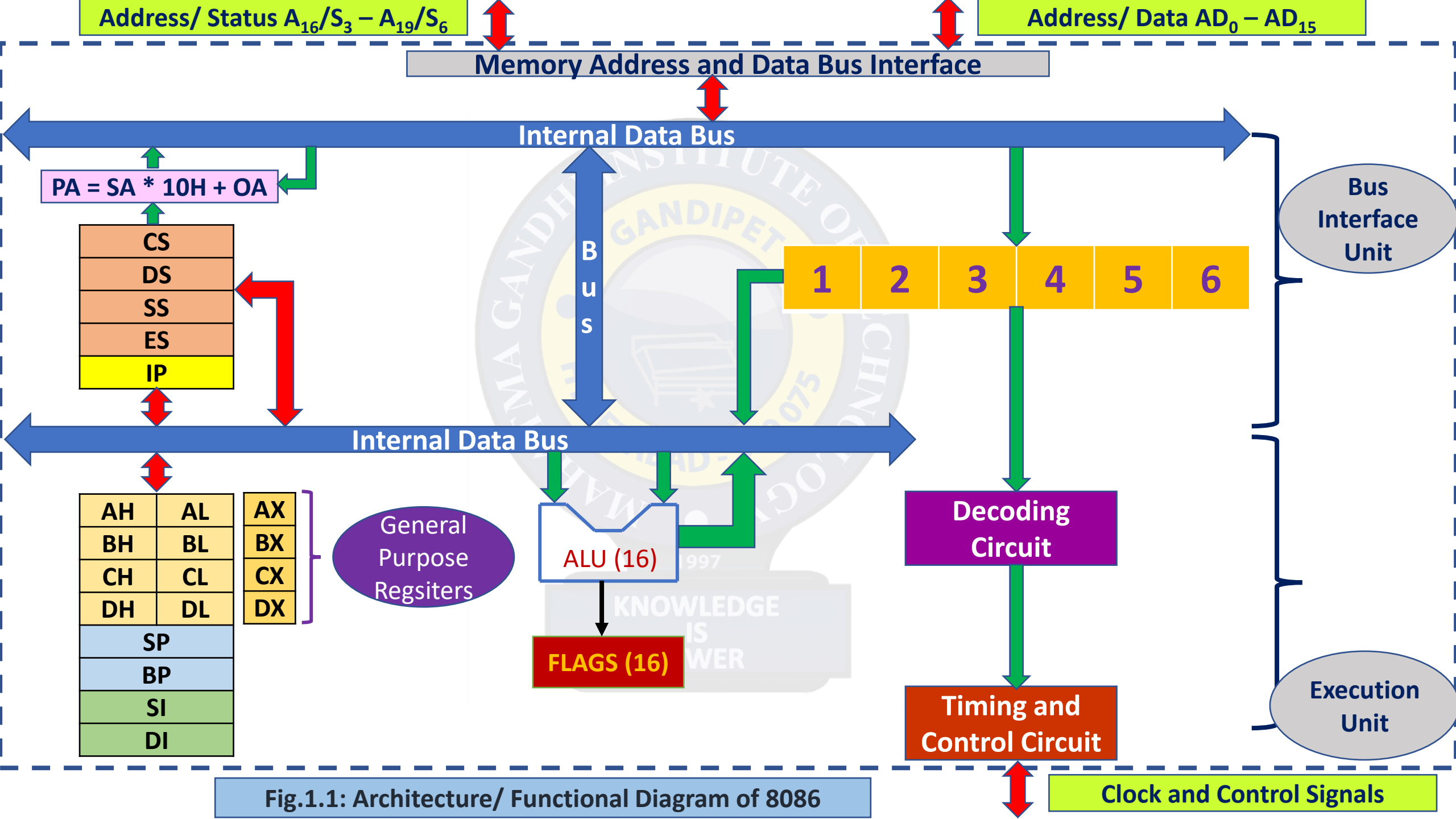


Fig.1.1: Architecture/ Functional Diagram of 8086

Register Organization

- General Purpose registers
- Segment registers
- Point and Index registers
- Flags/ Program Status Word (PSW)
- Instruction queue byte (6 bytes)

15 8	7 0
AH	AL
BH	BL
CH	CL
DH	DL

15 0
AX
BX
CX
DX

Fig.1.1(a): General purpose registers

General Purpose registers (GPR)

- 4 GPRs are available (AX, BX, CX, & DX)
- Each of 16-bit, can be used as two 8-bit registers each (H, L)
- Used for temporary data (variables & results) holding, counter, offset address
- Also used as implicit operand or destination

General Purpose registers (GPR)

- AX (16): AH (8) & AL (8): Accumulator
- BX (16): BH (8) & BL (8): Base register (stores the offset values)
- CX (16): CH (8) & CL (8): Counter
- DX (16): DH (8) & DL (8): Data

Segment registers (SR)

- 4 segment registers – CS, DS, SS, & ES
- Memory in 8086 is divided into 16 logical segments each of 64 Kb
- CS: used for addressing a memory location in the code segment
- SS: used for stack operations
- DS: locates an address in data segment
- ES: also points to a location in data (extra) segment

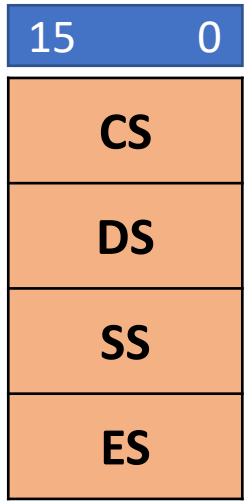


Fig.1.1(b): Segment registers

Pointers and Index Registers

- SP & BP: points to an offset in stack segment
- Index registers SI (source index) & DI (destination index) can be used as general data registers
- Particularly useful for string manipulation operations
- Also used for offset storage in Indexed, Based Indexed, and Relative Based Indexed addressing modes

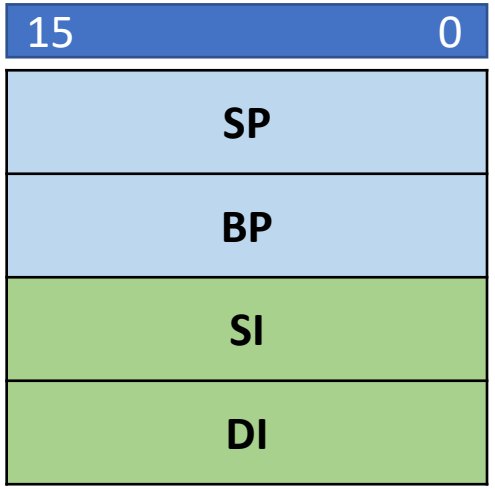


Fig.1.1(c): Pointers & Index registers

Flags/ PSW

- Indicates the status of the result
- Appeared as flag set (1) or reset (0)
- Widely used in control flow operations, interrupt handling, string manipulation, and ASCII operations

Flags/ PSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	O	D	I	T	S	Z	X	Ac	X	P	X	C

Fig.1.1(d): Flags

- Condition code/ Status flags – Lower byte of the flags register (**S, Z, Ac, P, C**) plus the overflow flag (**O**)
- Reflect the results of the ALU operations
- Control flags: higher byte of the PSW
- Consists of three flags – **D**, **I**, and **T**
- S-sign flag: sets when the result is negative and for signed operations, the MSB indicates to the sign bit.
- Z-zero flag: sets when the result of an operation (arithmetic or logical) is a zero.
- P-parity flag: sets when the lower byte of the result contains even number of 1s.

8086 Architecture

- C-carry flag: sets when the there is carry/ borrow out of the MSB and useful for sorting.
- O-overflow flag: sets if the result of a signed operation is large enough to fit in a destination register.
- T-trap flag: processor enters the single step execution when sets. It is an interrupt.
- I-interrupt flag: sets when a maskable interrupt is recognized by the CPU, else this will be ignored.
- D- direction flag:
  - used by string manipulation instructions. Indicates the direction of string accessing.
  - set: auto-decrement mode – from higher offset to lower offset.
  - Reset: auto-increment mode – from lower offset to higher offset.
- Ac- Auxiliary carry flag: sets if there is a carry from the lowest nibble.

Physical Address (PA) generation

Segment address (SA)	1234H					
Offset address (OA)	5678H					
SA	1234H		0001	0010	0011	0100
SA * 10H (shifted by 4-bits)	12340H	0001	0010	0011	0100	0000
	+					
OA	5678H		0101	0110	0111	1000
PA = SA * 10H + OA	179B8H	0001	0111	1001	1011	1000
Physical address (20)		**1**	**7**	**9**	**B**	**8**

Memory Segmentation

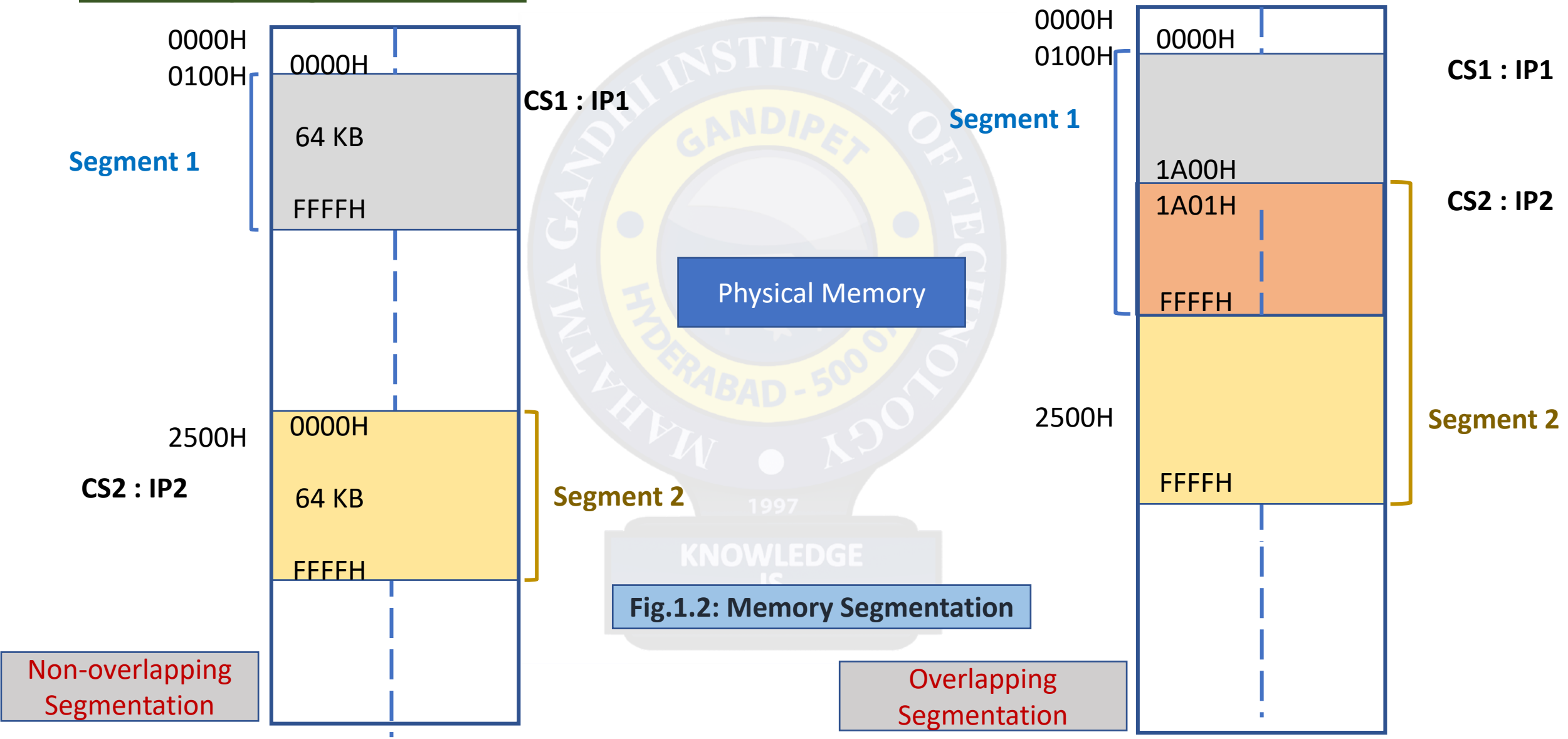


Fig.1.2: Memory Segmentation

Physical Memory Organization

- 1 MB Memory: Odd bank (512 KB) and Even bank (512 KB)
- Byte data with an even address transferred on $D_7 - D_0$ while the odd address transferred on $D_{15} - D_8$.

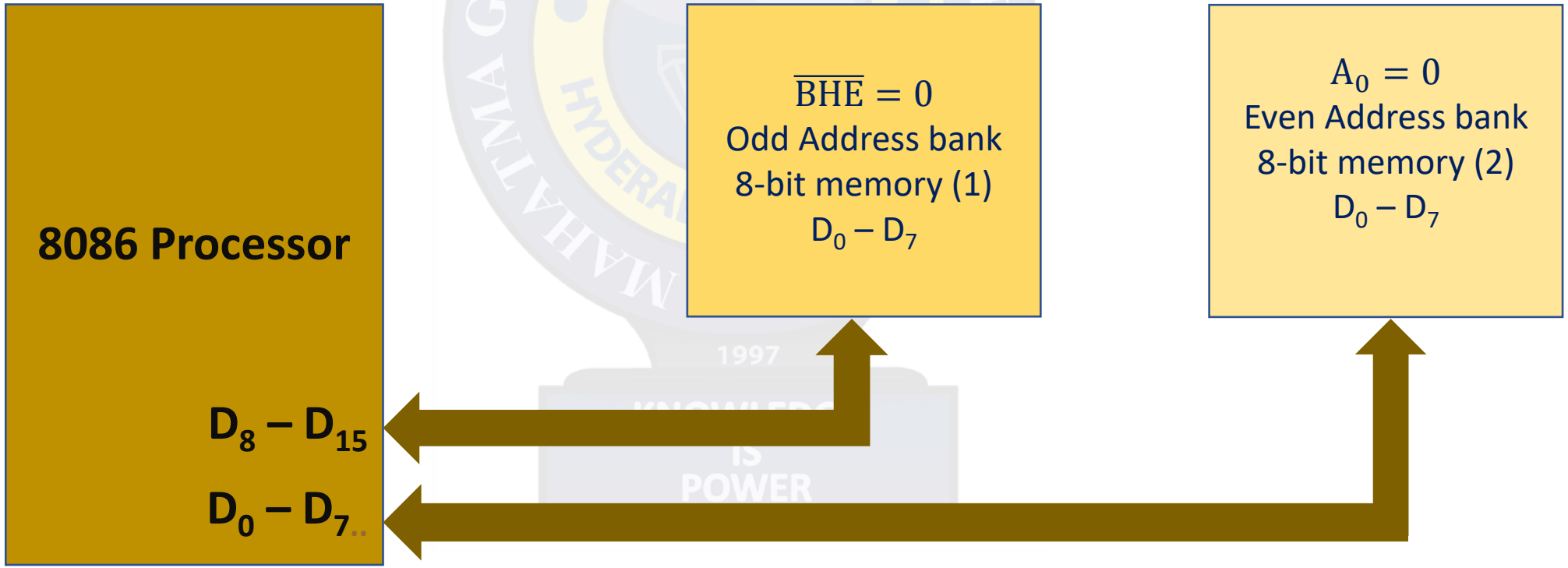


Fig.1.3: Address banks and 16-bit data access

Physical Memory Organization

- 8086 has $\overline{BHE}$ and A_0 for the selection of even or odd or both.
- The instruction fetched form memory as words (2 bytes) where
  - Both the bytes may be data operands
  - Both may contain opcode
  - One is opcode while the other is data
- Internal decoder circuit will take care.
- In case referring to word data, BIU requires one or two cycles depending upon the location of starting byte (even/ odd).
- It always recommended to initialize word at an even address to improve the efficiency.
- All the higher bytes (D_8 – D_{15}) will be stored in memory bank (1) and consequently, all the lower bytes (D_0 – D_7) will be stored in memory bank (2).

Physical Memory Organization

Tab.1.1: Bus HIGH enable and Address

$\overline{BHE}$	A_0	Indication
0	0	Whole word (2 bytes)
0	1	Upper byte from/to odd address
1	0	Lower byte from/to even address
1	1	None

- Locations from FFFF0H to FFFFFH are reserved for operations jump to initialization programme and I/O processor initialization .
- Locations from 00000H to 003FFH are reserved for Interrupt Vector Table (IVT).
- Each interrupt routine needs CS : IP i.e., 4-byes each and 256 interrupts require 256 * 4 bytes = 1024 bytes = 1KB.
- All the higher bytes ($D_8 – D_{15}$) will be stored in memory bank (1) and consequently, all the lower bytes ($D_0 – D_7$) will be stored in memory bank (2).

8086 Pin Description

- 40-pin Dual-In-Line (DIP).
- Operates in single processor (Min) mode and multiprocessor (Max) mode.
- Signals categorised into three
  - Common
  - Min mode
  - Max mode
- 1 & 20: Ground (GND)
- 19: CLK. Async with 33% duty cycle(5-10MHz)
- 21: Reset
- 40: VCC

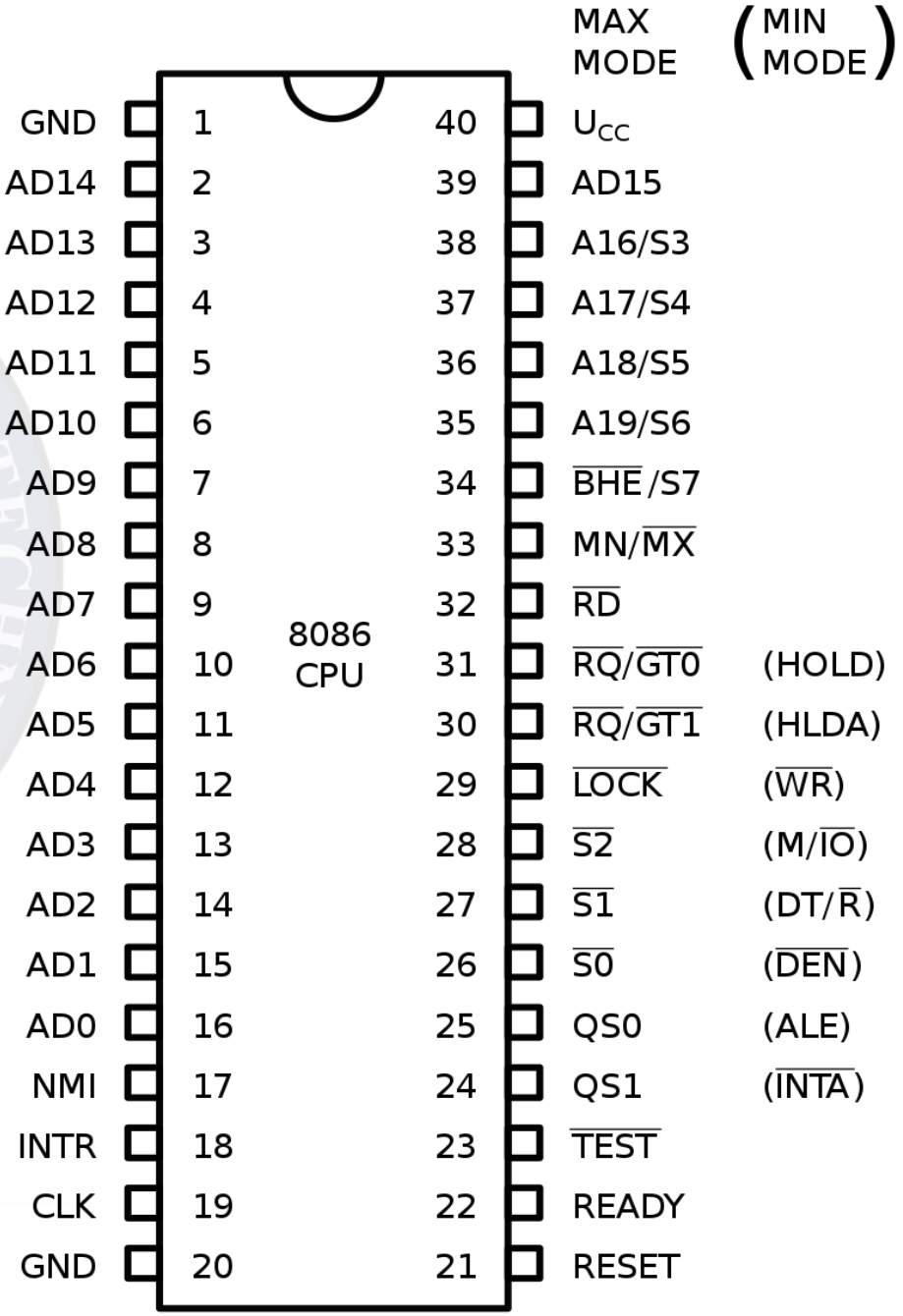


Fig.1.4: Pin Diagram/ Signal Description

8086 Pin Description

- $AD_0 – AD_{15}$: 16 – 2 & 39: Time multiplexed memory I/O address and Data Lines (A & D).
- During T_1 state, address is transferred through the bus.
- During T_3 & T_4 data transfer takes place.
- T_2 is used to change the direction of the bus.
- T_w is inserted when the addressed device is not ready.
- $A_{19} / S_6 – A_{16} / S_3$: 35 – 38: Time multiplexed address and status lines.
- These lines are the most significant address lines during T_1 state.
- During I/O operations, these lines are low.
- During memory and I/O operations, status information (S_6 – S_3) is available on these lines for T_2, T_3,T_4 & T_w.

8086 Pin Description

- S_5 : The status of the interrupt enable flag bit is updated at the beginning of each clock cycle.
- S_6 : is active LOW (0) always.
- S_3 & S_4 are encoded as shown in the table.
- These lines float to 3-state OFF during local bus "hold acknowledge".
- The address bits separated by the status bits using ALE signal.

Tab.1.2: Status signals functions

S_4	S_3	Characteristics
0	0	Alternate data
0	1	Stack
1	0	Code/ none
1	1	Data

8086 Pin Description

- $\overline{BHE}/S_7$: 34: Bus High Enable/ Status: used to enable the data transfer through $D_8 – D_{15}$. (Refer Tab.1.1.)
- $\overline{BHE}$ is active LOW (0) during T_1 for read, write, and interrupt acknowledge cycles.
- The status information is available during T_2, T_3 & T_4.
- Becomes tristated during 'hold'.
- S_7 is not used.
- $\overline{RD}$-Read: 32: when LOW, the processor performs memory or I/O read operation.
- Shows the state for T_2, T_3 & T_w of any read cycle.
- Remains tristated during "hold acknowledge".

8086 Pin Description

- READY: 22: an acknowledgement from the slower devices and memory.
- Synchronized by the 8284A clock generator.
- INTR: 18: Interrupt request: is a level-triggered input.
- Sampled during the last cycle of each instruction.
- If any interrupt is pending, processor enters the INTA cycle.
- Can be masked by resetting the IF.
- $\overline{\text{TEST}}$:23: examined by a WAIT instruction.
- If LOW, execution will continue, else, the processor will remain in idle state.
- Synchronized internally during each clock cycle on leading edge of the clock.

8086 Pin Description

- NMI: 17: an edge-triggered input causes a Type-2 interrupt.
- Non-maskable internally by software.
- Input is internally synchronized.
- RESET: 21: Causes the processor to terminate its current activity and start execution from FFFF0H.
- Is active high and must be HIGH for at least four clock cycles.
- Processor restarts execution when RESET goes LOW.
- MN/ $\overline{MX}$: 33: used to select the processor mode.

8086 Pin Description

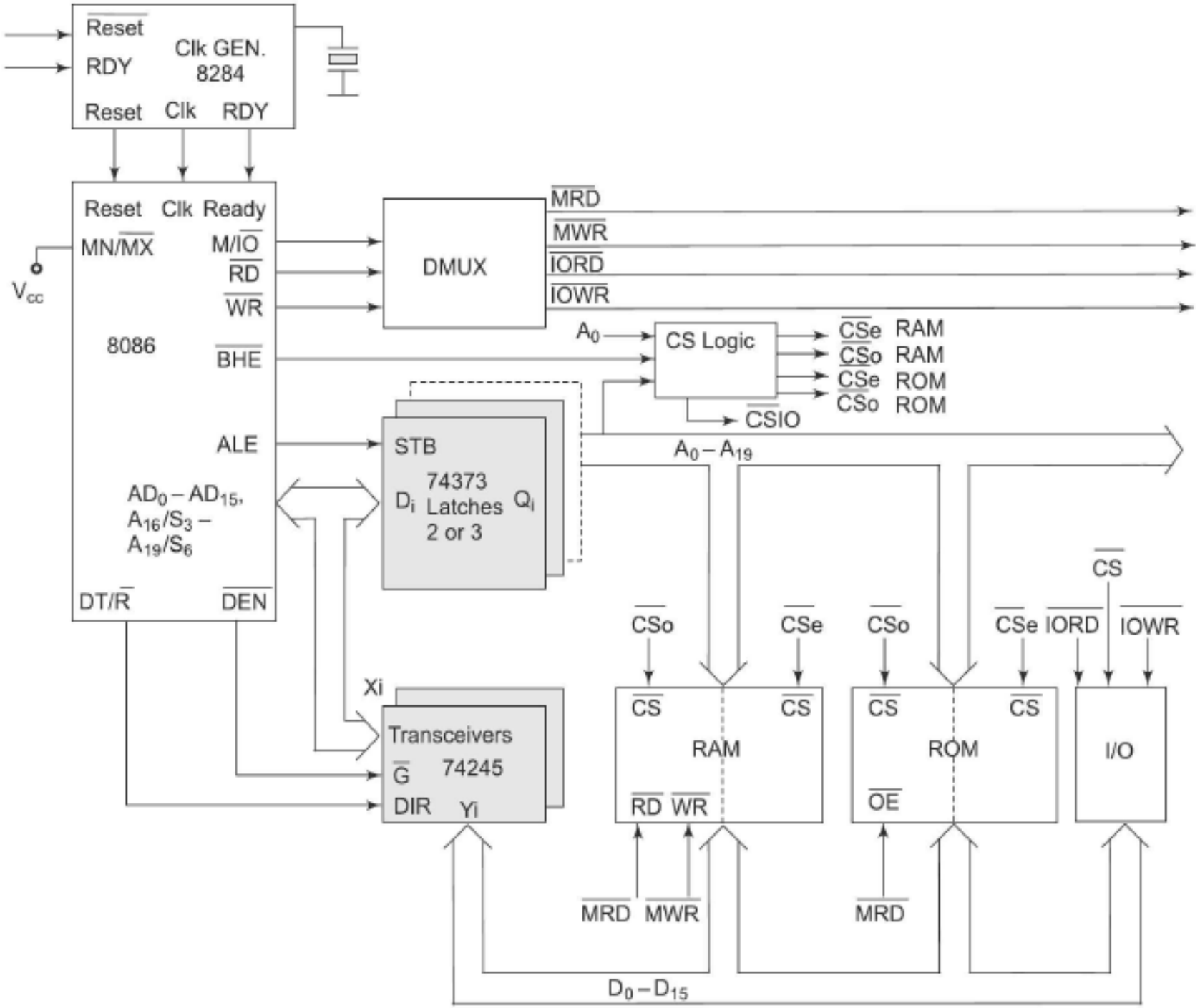
Minimum Mode

- $M/\overline{I/O}$: 28: 1: Memory operation and 0: I/O operation.
- Gets activated during previous T_4 and remains active until the current T_4.
- Tristated during local bus 'HLDA'.
- $\overline{INTA}$: 24: Used as a read strobe for interrupt acknowledgement.
- Goes low when the interrupt accepted by the processor.
- Remain low during T_2, T_3 & T_w.
- ALE: 25: Indicates the availability of the valid address on the $AD_0 - AD_{15}$.
- $DT/\overline{R}$: 27: Decides the direction of data flow trough the bidirectional buffers.
- Timing is similar to $M/\overline{I/O}$.

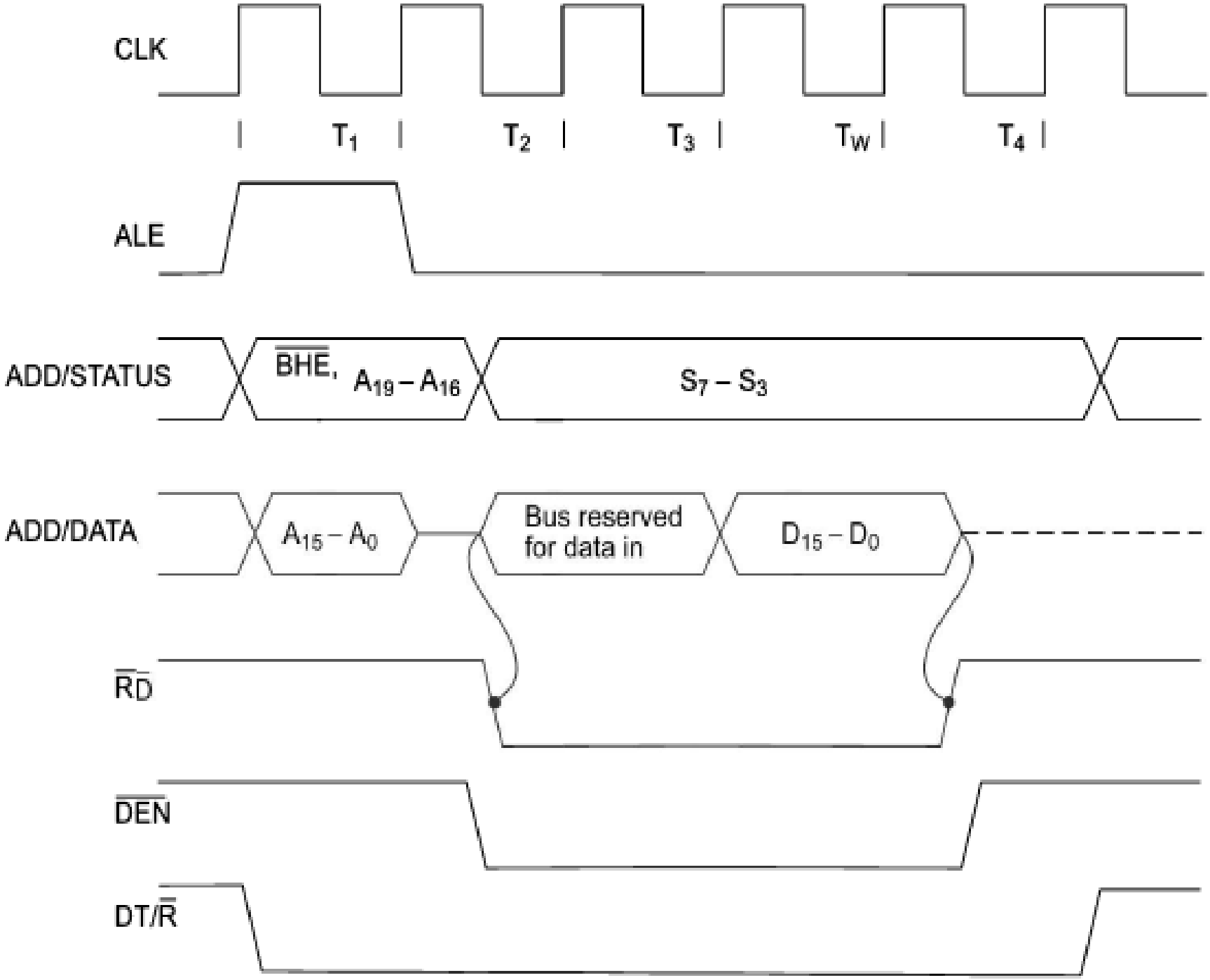
8086 Pin Description

- $\overline{DEN}$: 26: Indicates the availability of the valid address on the $AD_0 - AD_{15}$.
- Active from middle of T_2 until the middle of T_4.
- Tristated during local bus 'HLDA'.
- $\overline{WR}$: 29: Indicates the processor is performing memory or I/O write operation.
- Active during T_2, T_3 & T_w.
- Tristated during local bus 'HLDA'.
- HOLD & HLDA: 30 & 31: Indicates another master is requesting local bus.
- Processor acknowledges through HLDA.
- HOLD is an asynchronous input and requires external synchronization.
- $\overline{RQ}/\overline{GT_0}$ & $\overline{RQ}/\overline{GT_1}$: 31 & 30: used by local bus masters to force the processor to release the local bus at the end of the present processing cycle.

8086 Minimum mode operation

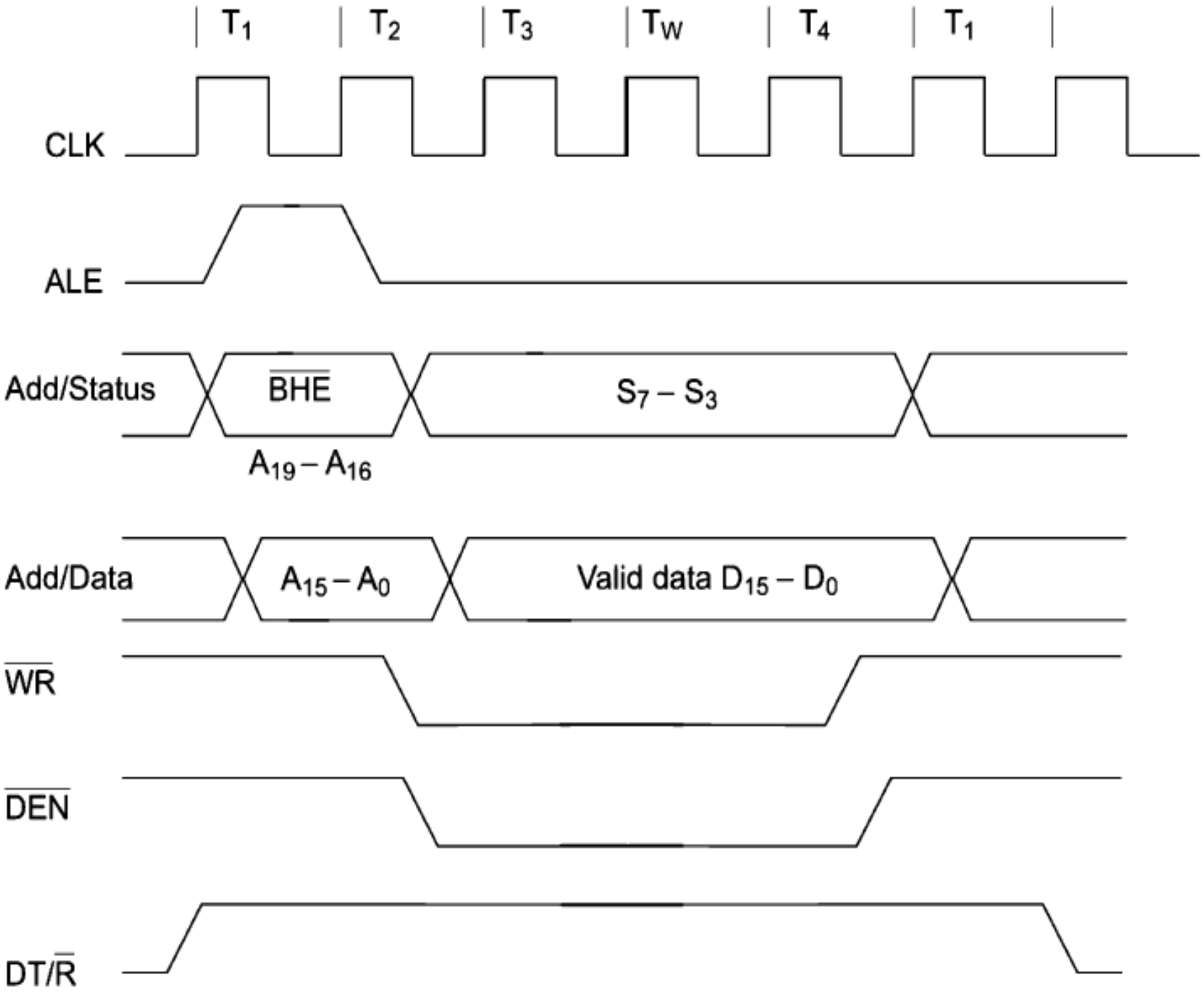


8086 Minimum mode operation



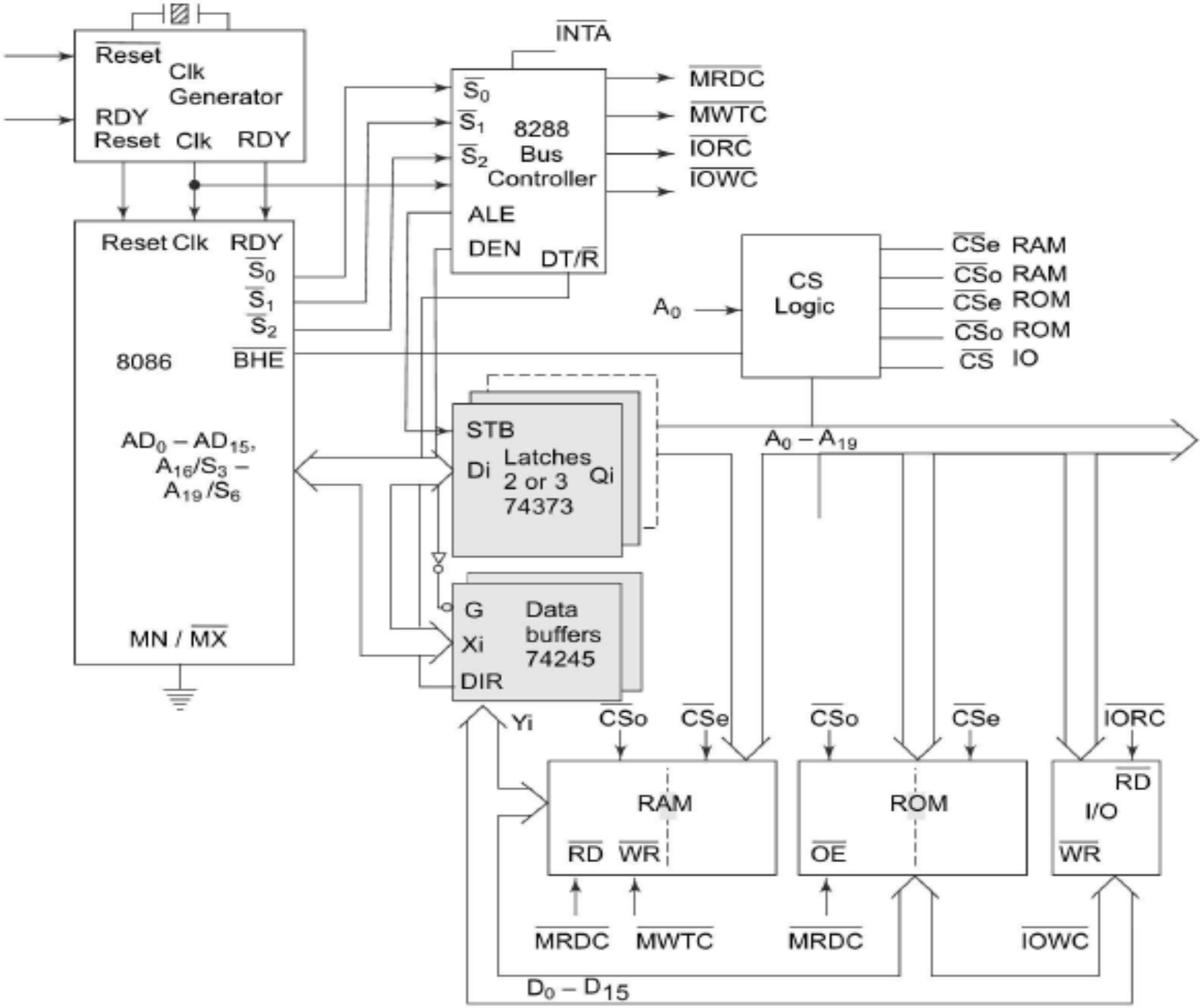
Read cycle timing- Minimum mode operation

8086 Minimum mode operation

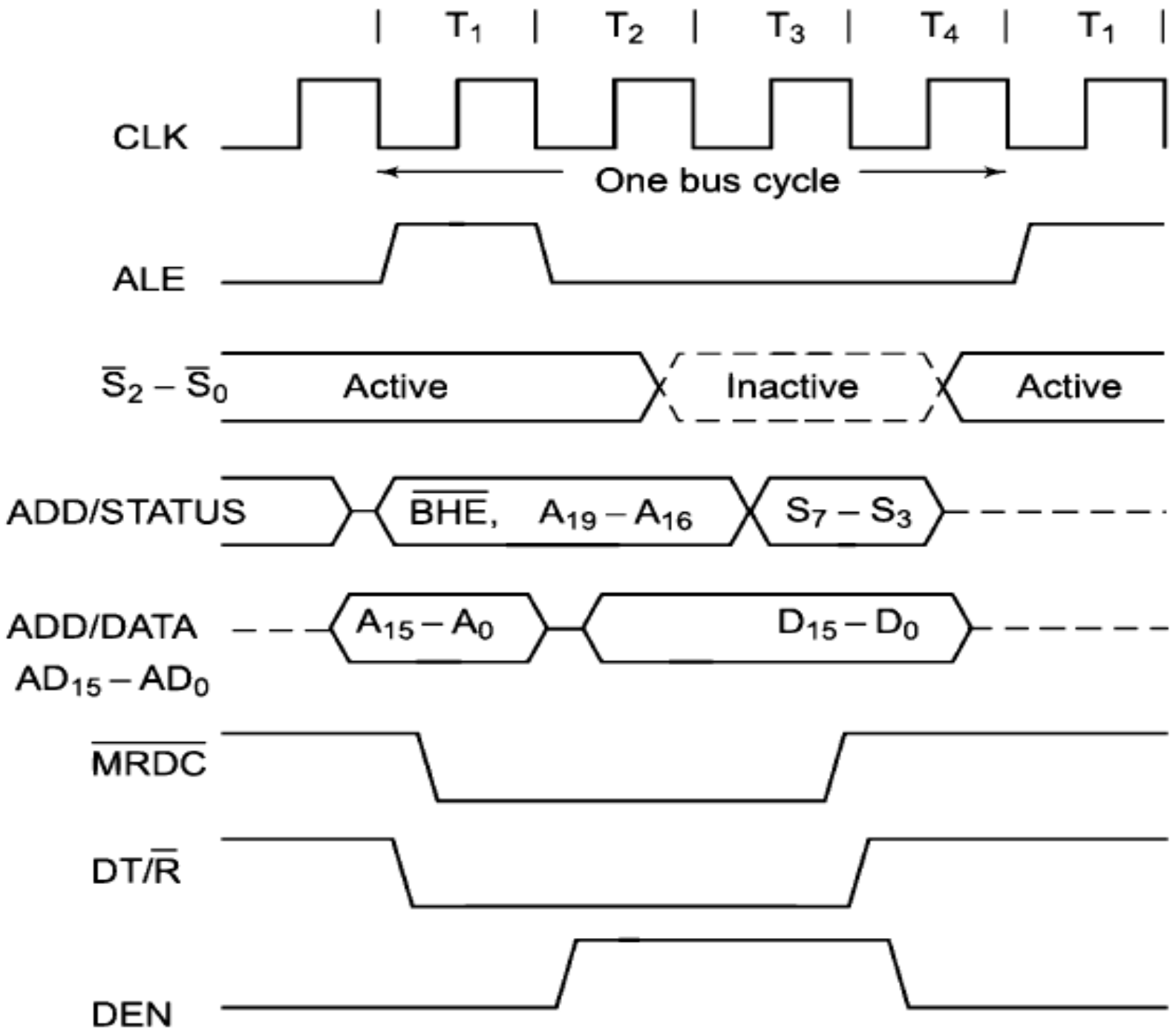


Write cycle timing- Minimum mode operation

8086 Maximum mode operation

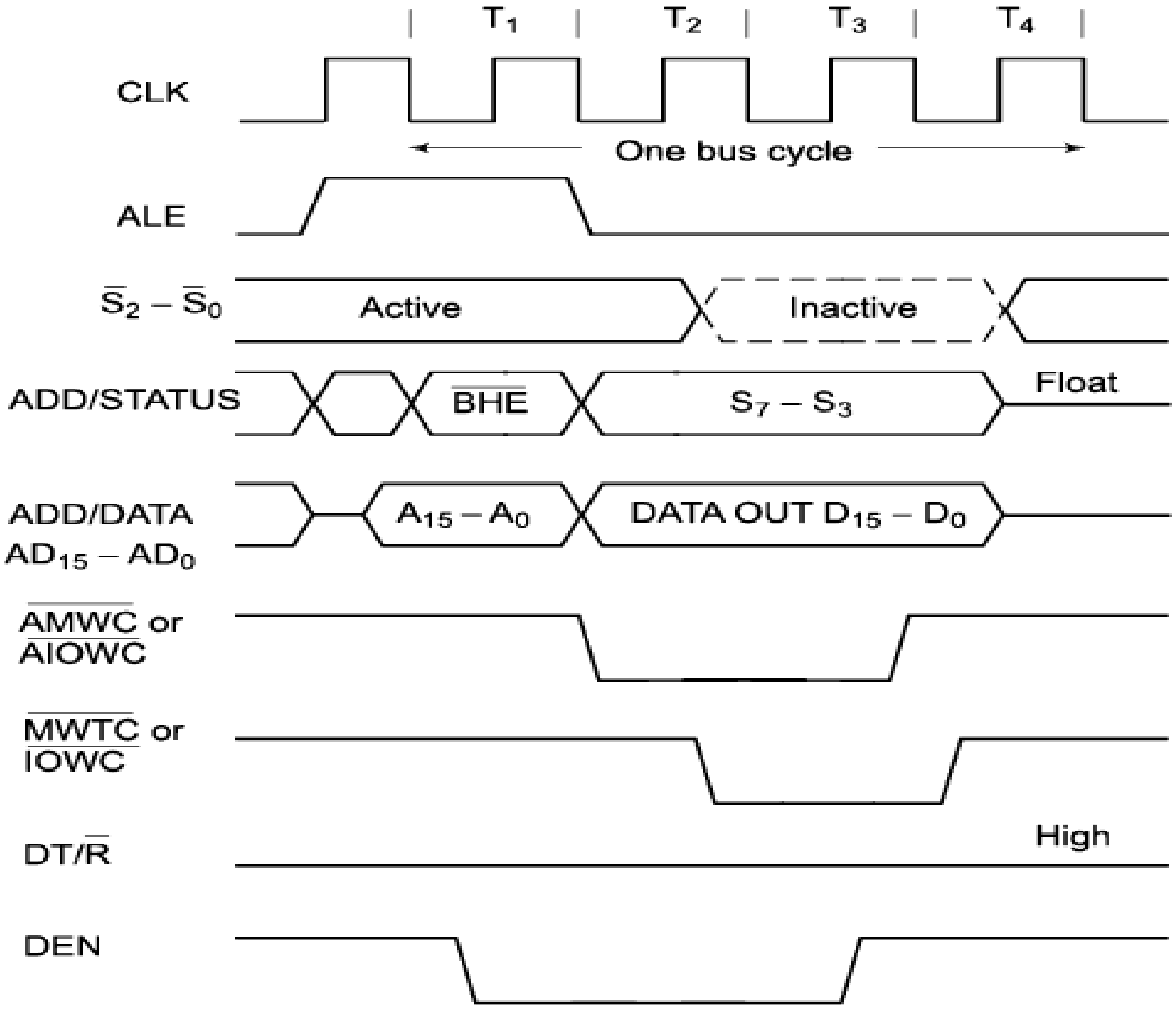


8086 Maximum mode operation



Memory Read timing-Maximum mode

8086 Maximum mode operation



Memory Write timing-Maximum mode

8086 Pin Description

Maximum Mode

- $\overline{S_0}$, $\overline{S_1}$, & $\overline{S_2}$: 26, 27, &28:

Tab.1.3: Status signals description

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Indication	Timing
0	0	0	Interrupt acknowledge	Active during T_4 of the previous cycle and remain active during T_1 and T_4 of the current cycle
0	0	1	Read I/O port	
0	1	0	Write I/O port	
0	1	1	Halt	
1	0	0	Code access	
1	0	1	Read memory	
1	1	0	Write memory	
1	1	1	Passive	T_3

8086 Pin Description

- $\overline{LOCK}$: 29: Indicates the other system bus masters will be prevented from gaining the system bus.
- Activated by LOCK instruction prefix and remains active until the completion of the next instruction.
- Tristated during local bus 'HLDA'.
- QS_1 &QS_0: 24 & 25: Indicates queue status of the instruction pre-fetch queue.

Tab.1.4: Queue Status signals description

QS_1	QS_0	Indication
0	0	No operation
0	1	First byte of the opcode from the queue
1	0	Empty queue
1	1	Subsequent byte from the queue

8086 Interrupts

Tab.1.5: 8086 interrupts

Interrupt Type	Content (16 bit)	Address	Comments
Type 0	ISR IP	0000:0000	Divided by zero
	ISR CS	0000:0002	
Type 1	ISR IP	0000:0004	Single step interrupt
	ISR CS	0000:0006	
Type 2	ISR IP	0000:0008	NMI
	ISR CS	0000:000A	
Type 3	ISR IP	0000:000C	INT single byte instruction
	ISR CS	0000:000E	
Type 4	ISR IP	0000:0010	INTO
	ISR CS	0000:0012	
		0000:0014	Two byte instruction type INT TYPE
		0000:0016	
Type 3FF	ISR IP	0000:03FE	
	ISR CS	0000:03FF	

Assembly Language Instruction Formats
&
Addressing Modes

Machine language Instruction Formats

- Machine language instruction consists of
  - Operational code or opcode
  - Operands

Opcode	Destination Operand	Source Operand

- Operational code/opcode:

W	D	S	V	Z	REG

Fig.1.5: Single byte instruction format

- W-bit: indicates the length of the operands
- 0: 8-bit operands
- 1: 16-bit operands
- D-bit: valid in case of double operand instructions
- One operand must be register specified by the REG field
- 0: REG is source operand and 1: REG is the destination operand

Machine language Instruction Formats

- S-bit: sign extension bit
- **One byte Instruction:** Only of 8-bit length as shown in Fig.1.5.

Tab.1.6: S & W bits description

S	W	Indication
0	**0**	8-bit operation with 8-bit immediate operand
0	**1**	16-bit operation with 16-bit immediate operand
1	**0**	8-bit operation with sign extended immediate data
1	**1**	16-bit operation with sign extended immediate data

Machine language Instruction Formats

- V-bit: indicates the no. of shift and rotations to take place on the given operand.
- 0: single shift/ rotation
- 1: CL contains the count
- Z-bit: used by REP instruction to control the loop.
- 1: instruction with REP prefix will execute until the zero flag matches the Z-bit.
- 0: REP prefix will not be executed.

Machine language Instruction Formats

Tab.1.7: Register codes

W	Reg address	Reg	W	Reg address	Reg
0	000	AL	1	000	AX
0	001	CL	1	001	CX
0	010	DL	1	010	DX
0	011	BL	1	011	BX
0	100	AH	1	100	SP
0	101	CH	1	101	BP
0	110	DH	1	110	SI
0	111	BH	1	111	DI

Machine language Instruction Formats

- In general, all the addressing modes have DS as the default segment register.
- Addressing modes using SP and BP use SS as the default segment register.
- **Register to Register:** 2 bytes long
- First byte specifies the opcode and length of the operation is specified by W-bit.
- Second byte has the register operands and R/M field.
- R/M specifies the another operand as whether register or memory.

Tab.1.8: Segment registers

Reg code (2-bit)	Reg
00	ES
01	CS
10	SS
11	DS

Opcode	Destination Operand	Source Operand

Machine language Instruction Formats

D7 … D_1	D_0	D_7 D_6	D_3 D_4 D_5	D_2 D_1 D_0
Opcode	W	11	REG	R/M

Fig.1.6: Two byte instruction format

- **Register to/from memory with displacement:** 4 bytes long

D7 … D_0	D_7 D_6	D_3 D_4 D_5	D_2 D_1 D_0	D7 … D_0	D7 … D_0
Opcode	MOD	REG	R/M	Lower byte of displacement	Upper byte of displacement

Fig.1.7: Four byte instruction format

- **Immediate operand to register:** 4 bytes long

D7 … D_0	D_7 D_6	D_3 D_4 D_5	D_2 D_1 D_0	D7 … D_0	D7 … D_0
Opcode	11	Opcode	R/M	Lower byte DATA	Upper byte DATA

Fig.1.8: Four byte immediate mode

Machine language Instruction Formats

Tab.1.9: Addressing Modes

operands	Memory operands			Register operands	
	No Displacement	Displacement 8 bits	Displacement 16 bits		
MOD	0 0	0 1	1 0	11	
R/M					
0 0 0	$(BX)+(SI)$	$(BX)+(SI)+D8$	$(BX)+(SI)+D16$	AL	AX
0 0 1	$(BX)+(DI)$	$(BX)+(DI)+D8$	$(BX)+(DI)+D16$	CL	CX
0 1 0	$(BP)+(SI)$	$(BP)+(SI)+D8$	$(BP)+(SI)+D16$	DL	DX
0 1 1	$(BP)+(DI)$	$(BP)+(DI)+D8$	$(BP)+(DI)+D16$	BL	BX
1 0 0	(SI)	$(SI)+D8$	$(SI)+D16$	AH	SP
1 0 1	(DI)	$(DI)+D8$	$(DI)+D16$	CH	BP
1 1 0	$D16$	$(BP)+D8$	$(BP)+D16$	DH	SS
1 1 1	(BX)	$(BX)+D8$	$(BX)+D16$	BH	DI

Machine language Instruction Formats

- **Immediate operand to memory with 16-bit displacement:** 6 bytes long

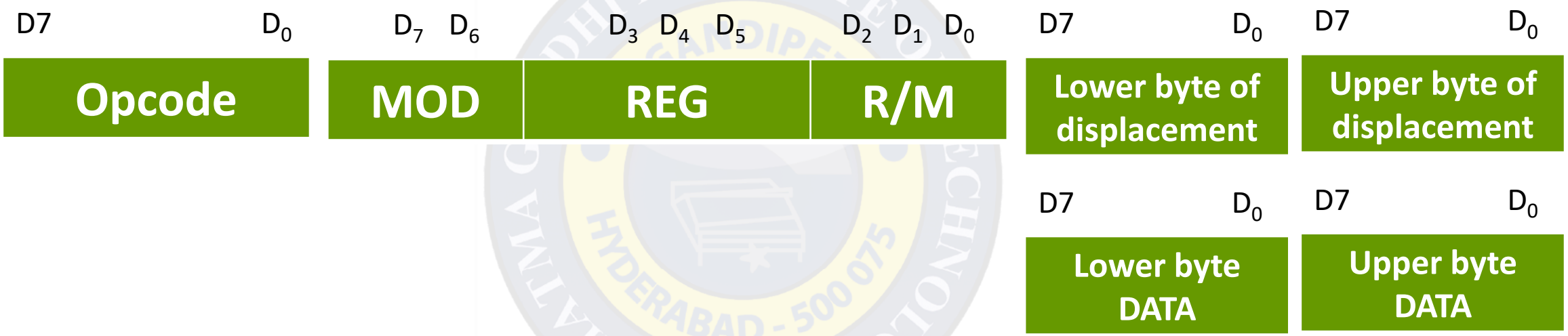


Fig.1.9: Six byte instruction format

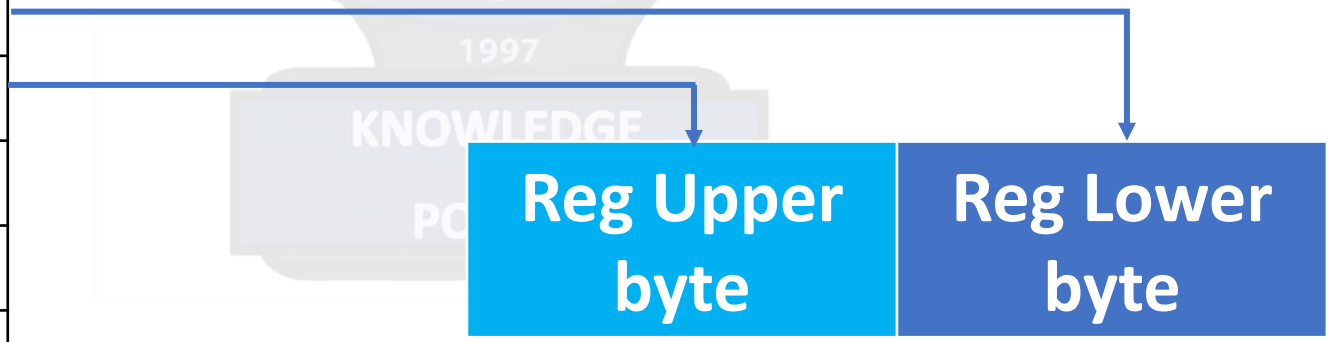
8086 Addressing Modes

- Indicate mode of locating data or operands.
- Depending upon the data type any instruction may belong to one or more addressing modes or may not belong to any.
- **Sequential control flow instructions**
- **Immediate:** immediate data is part of the instruction.
- Immediate data can not be a destination operand.
- MOV AX, 1234H (1234H is 16-bit immediate data)
- MOV DL, 77H (77H is 8-bit immediate data)
- MOV 1234H, AX (Invalid)
- MOV 77H, DL (Invalid)
- MOV segment register, immediate data (Invalid)

8086 Addressing Modes

- **Direct:** 16-bit memory address (offset) or an I/O address is directly specified.
- MOV AX, [1234H] (Moves 16-bit data from offsets 1234H & 1235H)
- MOV DL, [1234H] (Moves 8-bit data from offset 1234H)
- IN 80H
- MOV [1234H], DL (Moves 8-bit data to offset 1234H)

Offset	Data
1234H	ABH
1235H	21H
1236H	7AH
1237H	90H
:	:



8086 Addressing Modes

- **Indexed:** Offset of the operand stored in any of the indexed registers.
- DS is the default segment register for SI and DI.
- In string instructions DS and ES are default segment registers for SI and DI, respectively.
- MOV AX, [SI] (Moves 16-bit data from offset specified by SI)
- MOV DL, [DI] (Moves 8-bit data from offset specified by DI)
- PA = 10H * DS + SI or PA = 10H * ES + DI
- **Register Relative:** effective address is formed by adding an 8-bit or 16-bit displacement with the contents of any of the registers BX, BP, SI, and DI.
- MOV AX, 50H[BX] (PA = 10H * DS + 50H + BX)
- MOV 10H[SI], BL (PA = 10H * DS + 10H + SI)

8086 Addressing Modes

- **Based Indexed:** Effective address is formed by adding content of a base register (BX or BP) to the content of the index register (SI or DI).
- DS is the default segment register for SI and DI.
- DS and ES are default segment registers.
- MOV AX, [BX] [SI] (PA = 10H * DS + SI + BX)
- MOV [BP] [DI], DX (PA = 10H * DS + BP + SI)
- **Relative Based Indexed:** Effective address is formed by adding an 8-bit or 16-bit displacement with the sum contents of any of the registers BX or BP and any of the index registers SI or DI.
- MOV AX, 50H [BX] [SI] (PA = 10H * DS + 50H + BX + SI)
- ADD 50H [BX] [SI], BP (PA = 10H * DS + 50H + BX + SI)

Control transfer instructions

- Intersegment
  - Direct
  - Indirect
- Intrasegment
  - Direct
  - Indirect
- Intrasegment Direct: address lies in the same segment and appears directly in the instruction as an immediate displacement value.
- The displacement is computed relative to the content of the IP.
- EA = CS * 10H + IP + 8-bit/ 16- bit disp.
- 8-bit displacement: -128 < d < +127 : Short jump
- 16-bit displacement: -32768 < d < +32767 : Long jump

8086 Addressing Modes

- Example: JMP SHORT LABEL
- LABEL lies within -128 to +127 from the current IP address.
- 16-bit jump should not exceed relative address 32767 in forward jump and -32768 in backward jump.
- Let IP = 5000H, then forward jump in the same segment allows IP + DISP = FFFFH or max DISP = FFFFH - 5000H = AFFFH.
- So the forward jump occurs for all the displacement values from 0000H to AFFFH.
- If the DISP > AFFFH (B000H – FFFFH), then all such jump are treated as backward jumps.
- NEAR PTR is the opcode for all such jumps.

8086 Addressing Modes

- Example: JMP NEAR PTR LABEL
- Intrasegment Indirect: address lies in the same segment and passed to the instruction indirectly.
- The displacement is computed relative to the content of the BX.
- Used in unconditional branch instructions.
- Example: JMP [BX] here EA = BX
- JMP [BX + 5000H] here EA = BX + 5000H
- Intersegment Direct: address to which the control is to be transferred is in a different segment.
- Allows the transfer from one code segment to another code segment.
- CS and IP of the destination address are specified directly.

8086 Addressing Modes

- Example: JMP 5000H : 2000H
- EA = CS * 10H + 2000H = 50000H + 2000H = 52000H.
- Intersegment indirect: address lies in the different segment and is passed to the instruction indirectly.
- Contents of memory block contains four bytes i.e., IP (LSB), IP (MSB), CS (LSB), CS (MSB).
- Example: JMP [2000H]
- Jumps to effective address 2000H in DS, where the IP & CS values are being stored and those memory contents give the effective address of the next instruction to which the control is to be transferred in the CS.

- Example: The contents of different registers are given below. Form the EA for different addressing modes. Offset(DISP) = 5000H, AX = 1000H, BX = 2000H, SI = 3000H, DI = 4000H, BP = 5000H, SP = 6000H, CS = 0000H, DS = 1000H, SS = 2000H, IP = 7000H.

(i) **Direct:** MOV AX, [5000H]

DS : Offset =	1000H : 5000H
10H * DS =	10000H
Offset =	+ 5000H
Disp =	± 0000H
EA/PA =	15000H

(ii) **Register indirect:**

MOV AX, [BX]

DS : BX =	1000H : 2000H
10H * DS =	10000H
BX =	+ 2000H
Disp =	± 0000H
EA/PA =	12000H

(iii) **Register relative:**

MOV AX, 5000H [BX]

DS : BX =	1000H : 2000H
10H * DS =	10000H
BX =	+ 2000H
Disp =	+ 5000H
EA/PA =	17000H

8086 Addressing Modes

(iv) **Based indexed:** MOV AX, [BX] [SI]

DS : BX =	1000H : 2000H
10H * DS =	10000H
BX =	+ 2000H
SI =	+ 3000H
EA/PA =	15000H

(v) **Relative based indexed:** MOV AX, 5000H [BX] [SI]

DS : BX =	1000H : 2000H
10H * DS =	10000H
BX =	+ 2000H
SI =	+ 3000H
Disp =	+ 5000H
EA/PA =	1A000H

8086/88 Instruction Set

8086 Instructions: Types

a) Data copy/transfer
b) Arithmetic and Logical
c) Conditional/unconditional Branch
d) Conditional/unconditional Loop
e) Machine control
f) Flag manipulation
g) Shift and rotate
h) String

8086 Instructions: Data copy/transfer

a) **Data copy/transfer**: Used for data transferring to/from Memory, I/O, or Registers

MOV Move

Syntax: MOV destination operand, source operand

- Both the operands must be of same size.
- No flags get affected.

	Destination operand	Source operand
Valid	Register	Register
Valid	Any register except segment registers	Immediate data
Valid	Register	Offset (with/without displacement)
Valid	Offset (with/without displacement)	Register
Invalid	Offset (with/without displacement)	Immediate data

8086 Instructions: Data copy/transfer

	Mnemonic	Addressing Mode
Valid	MOV AX, 5000H	Immediate
Valid	MOV AX, BX	Register
Valid	MOV AX, [SI]	Indirect
Valid	MOV AX, [2000H]	Direct
Valid	MOV AX, 50H [BX]	Relative Based Indexed
Valid	MOV AX, [BX]	Based Indexed
Invalid	MOV DS, 5000H	Immediate

PUSH Push on to the Stack

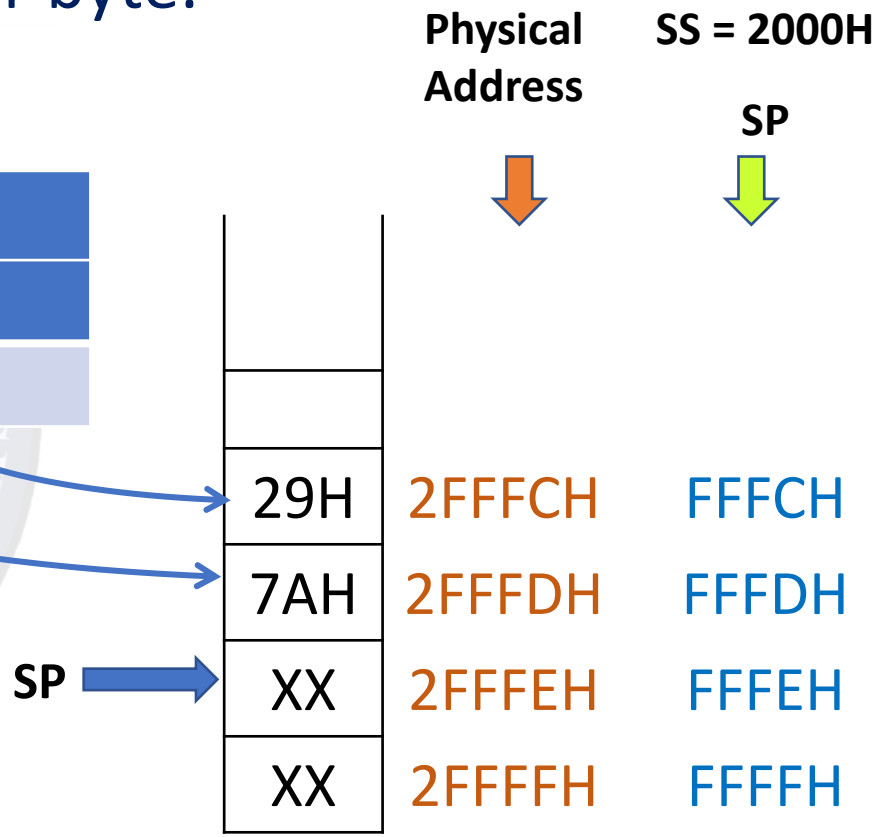
Syntax: PUSH register/offset (direct/indirect)

- Stack pointer (SP) is decremented by 2.
- No flags get affected.

8086 Instructions: Data copy/transfer

- Higher byte is pushed first and then the lower byte.
- SS : SP points to the top of the stack.

PUSH AX	
AH	AL
7AH	29H



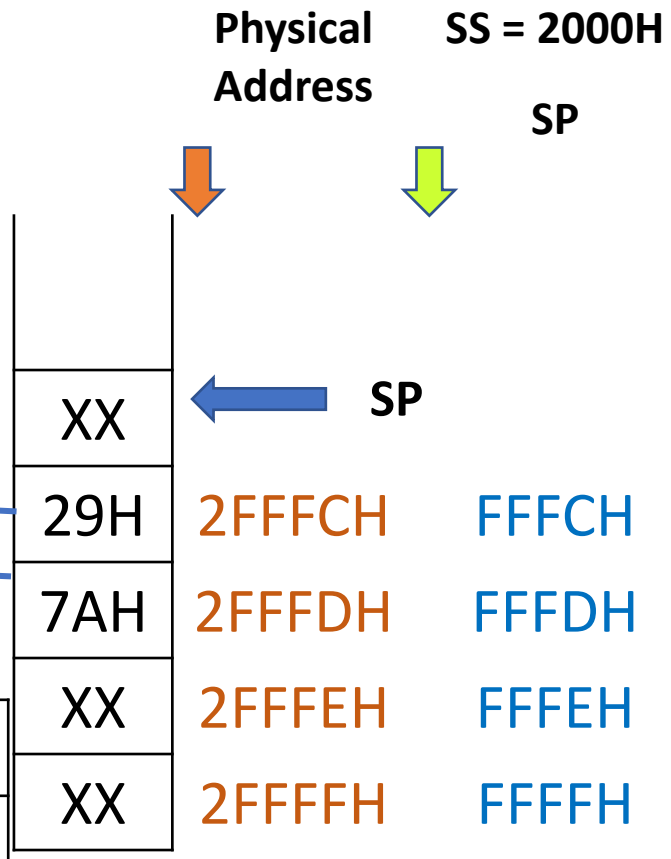
	Mnemonic	Addressing Mode
Valid	PUSH AX	Register
Valid	PUSH DS	Register
Valid	PUSH [5000H]	Direct
Invalid	PUSH 5000H	Immediate

POP Pop from the Stack

Syntax: POP register/offset (direct/indirect)

- Lower byte is popped first and then the higher byte.
- SS : SP points to the top of the stack.
- SP is incremented by 2.
- No flag gets affected.

POP AX	
AH	AL
7AH	29H



	Mnemonic	Addressing Mode
Valid	POP AX	Register
Valid	POP DS	Register
Valid	POP [5000H]	Direct
Invalid	POP 5000H	Immediate

XCHG Exchange

Syntax: XCHG destination operand, source operand

- No flags get affected.

	Mnemonic	Addressing Mode
Invalid	XCHG AX, 5000H	Immediate
Valid	XCHG AX, BX	Register
Valid	XCHG AX, [SI]	Indirect
Valid	XCHG AX, [2000H]	Direct
Valid	XCHG [2000H], AX	?
Invalid	XCHG [AX], [BX]	Based Indexed
Invalid	XCHG [2000H], [5000H]	Direct

8086 Instructions: Data copy/transfer

IN Input the Port

Syntax: IN AL/AX (8/16-bit data from), 8/16-bit Port address

IN DX, Port address (16-bit)

- No flags get affected.

	Mnemonic	Operation
Valid	IN AL, 03H	Transfers 8-bit data to AL from port address 03H.
Valid	IN AX, DX	Reads 16-bit data from a 16-bit port address stored in DX to AX.
Valid	MOV DX, 0777H IN AX, DX	Reads 16-bit data from the port whose address is 0777H.
Invalid	MOV AX, 0777H	Cant read data directly from a 16-bit port address to AX

8086 Instructions: Data copy/transfer

OUT Output to the Port

Syntax: OUT 8/16-bit Port address, AL/AX (8/16-bit data from)

OUT DX, 16-bit data

- No flags get affected.
- Port address may be direct or specified in DX.
- Only content of AL/AX can be written on to the output port.
- Data to an odd addressed port is transferred on $D_8 – D_{15}$ while to an even addressed port is transferred on $D_0 – D_7$.

8086 Instructions: Data copy/transfer

	Mnemonic	Operation
Valid	OUT 03H, AL	Transfers 8-bit data to port address 03H from AL.
Valid	OUT DX, AX	Writes 16-bit data to a 16-bit port address stored in DX from AX.
Valid	MOV DX, 0777H IN DX, AX	Writes 16-bit data to the port whose address is 0777H.
Invalid	MOV 0777H, AX	Cant write data directly to a 16-bit port address from AX

XLAT Translate

Syntax: XLAT

- No flags get affected.
- Finds the codes using look-up table in case of code conversion problems.

8086 Instructions: Data copy/transfer

Mnemonic	Operation
MOV AX, SEG TABLE	TABLE segment address to AX
MOV DS, AX	Segment address from AX to DS
MOV AL, CODE	Code of the pressed KEY
MOV BX, OFFSET TABLE	Offset of the pressed KEY code in LUT
XLAT	Finds the equivalent code and stores in AL

LEA Load Effective Address

Syntax: LEA Register (16-bit), ADDRESS

- No flags get affected.

Mnemonic	Operation
LEA BX, ADR	EA/Offset of label ADR will be transferred to BX
LEA SI, ADR [BX]	EA of ADR + BX will be transferred to SI.

8086 Instructions: Data copy/transfer

LDS/LES Load Pointer to DS/ES

Syntax: LEA Register (16-bit), ADDRESS

- Loads the DS or ES register and the specified destination register in the instruction with content of memory location specified as source in the instruction.
- No flags get affected.

Mnemonic	Operation
LDS BX, 5000H	Loads BX with content of 5000H and 5001H and filled DS with the content of 5002H and 5003H
LES BX, 5000H	Loads BX with content of 5000H and 5001H and filled ES with the content of 5002H and 5003H

8086 Instructions: Data copy/transfer

LAHF Load AH from Lower Byte of Flag

Syntax: LAHF

- Used to observe the status of all the conditional code flags (except OF).
- No flags get affected.

SAHF Store AH to Lower Byte of Flag

Syntax: SAHF

- Used to sets/ resets all the conditional code flags (except OF) depending on the corresponding bit value in AH register.

PUSHF Push flags (16-bit) to stack

Syntax: PUSHF

POPF Pop stack content (16-bit) to flags

Syntax: POPF

8086 Instructions: Arithmetic and Logical

b) Arithmetic and Logical: Used to perform arithmetic and logical operations

- Corresponding flags get affected depending upon the result.

ADD Addition

Syntax: ADD destination operand, source operand

- All conditional code flags get affected depending upon the result.

	Destination operand	Source operand
Valid	Register	Register
Valid	Any register except segment registers	Immediate data
Valid	Register	Offset (with/without displacement)
Valid	Offset (with/without displacement)	Register
Valid	Offset (with/without displacement)	Immediate data

8086 Instructions: Arithmetic and Logical

	Mnemonic	Addressing Mode
Valid	ADD AX, 5000H	Immediate
Valid	ADD AX, BX	Register
Valid	ADD AX, [SI]	Indirect
Valid	ADD AX, [2000H]	Direct
Valid	ADD [2000H], AX	?
Invalid	ADD [AX], [BX]	Based Indexed
Invalid	ADD [2000H], [5000H]	Direct
Invalid	ADD DS, AX	Register
Invalid	ADD AX, DS	Register

8086 Instructions: Arithmetic and Logical

ADC Addition with carry

Syntax: ADC destination operand, source operand

- Same as ADD but also adds the carry (CF).
- All conditional code flags get affected depending upon the result.

INC Increment

Syntax: INC reg/memory

- Increments the content by 1.
- All conditional code flags get affected except carry.
- Immediate data is not allowed.

DEC Decrement

Syntax: DEC reg/memory

- Decrements the content by 1.
- All conditional code flags get affected except carry.
- Immediate data is not allowed.

8086 Instructions: Arithmetic and Logical

SUB Subtraction

Syntax: SUB destination operand, source operand

- Subtracts the source operand from the destination operation.
- All conditional code flags get affected depending upon the result.

	Destination operand	Source operand
Valid	Register	Register
Valid	Any register except segment registers	Immediate data
Valid	Register	Offset (with/without displacement)
Valid	Offset (with/without displacement)	Register
Valid	Offset (with/without displacement)	Immediate data

8086 Instructions: Arithmetic and Logical

	Mnemonic	Addressing Mode
Valid	SUB AX, 5000H	Immediate
Valid	SUB AX, BX	Register
Valid	SUB AX, [SI]	Indirect
Valid	SUB AX, [2000H]	Direct
Valid	SUB [2000H], AX	Register
Valid	SUB [5000H], 5000H	Immediate
Invalid	SUB [AX], [BX]	Based Indexed
Invalid	SUB [2000H], [5000H]	Direct
Invalid	SUB DS, AX	Register
Invalid	SUB AX, DS	Register

8086 Instructions: Arithmetic and Logical

SBB Subtraction with borrow

Syntax: SBB destination operand, source operand

- Same as SUB but also subtracts the borrow/carry (CF).
- All conditional code flags get affected depending upon the result.

CMP Compare

Syntax: CMP destination operand, source operand

- Compares the data present in the source and destination operands.
- Subtracts the source operand from the destination operand.
- Result not stored any where.
- All conditional code zero and carry flags get affected.
- The allowed source and destination operands are similar to ADD & SUB instructions.

8086 Instructions: Arithmetic and Logical

	Mnemonic	Addressing Mode	Comments
Valid	CMP AX, 5000H	Immediate	No result is available only the zero and carry flags get affected.
Valid	CMP AX, BX	Register	
Valid	CMP AX, [SI]	Indirect	
Valid	CMP AX, [2000H]	Direct	
Valid	CMP [2000H], AX	Register	
Valid	CMP [5000H], 5000H	Immediate	
Invalid	CMP [AX], [BX]	Based Indexed	
Invalid	CMP [2000H], [5000H]	Direct	
Invalid	CMP DS, AX	Register	
Invalid	CMP AX, DS	Register	

8086 Instructions: Arithmetic and Logical

AAA ASCII adjust after addition

Syntax: AAA

- Executed after an addition(ADD) instruction.
- Adds two ASCII coded operands and stores the result in AL register.
- Converts the resulting content of AL to unpacked decimal digits.
- First clears AH.
- If lower nibble of AL is between 0 – 9 and AF = 0, then AAA sets higher nibble of AL to 0.
- If lower nibble of AL is either between 0 – 9 and AF = 1 or is greater than 9, then AAA adds 6 to the lower nibble sets higher nibble and increments AH by 1.
- The AF & CF are set 1.
- Remaining flags unaffected.

8086 Instructions: Arithmetic and Logical

Mnemonic (AAA)	Mnemonic (AAS)	Mnemonic (AAM)

MOV AL, 07H
MOV BL, 06H
ADD AL, BL
AAA
	MOV AL, 07H
	MOV BL, 06H
	SUB AL, BL
	AAS
		MOV AL, 07H
		MOV BL, 06H
		MUL BL
		AAM

Mnemonic (AAD)

AAD
MOV AL, 07H
MOV BL, 06H
DIV BL

8086 Instructions: Arithmetic and Logical

AAS ASCII adjust after subtraction

Syntax: AAS

- Executed after a subtraction(SUB) instruction.
- Converts the resulting content of AL to unpacked decimal digits.
- First clears AH.
- If lower nibble of AL is between 0 – 9 and AF = 0, then AAA sets higher nibble of AL to 0.
- If lower nibble of AL is either between 0 – 9 and AF = 1 or is greater than 9, then AAA subtracts 6 from AL decrements AH by 1.
- The AF & CF are set 1.
- Remaining flags unaffected.

AAM ASCII adjust after Multiplication

Syntax: AAM

8086 Instructions: Arithmetic and Logical

- Converts the product in AL into unpacked BCD format.
- Higher nibble of multiplication operands must be zero.
- First clears AH.
- Replaces content of AH by tens of the decimal multiplication and AL by the ones of multiplication.

AAM ASCII adjust after Multiplication

Syntax: AAM

AAD ASCII adjust before Division

Syntax: AAD

- Converts content in AX into the equivalent binary number in AL.
- PF, SF, and ZF are modified while, OF, CF, and AF are undefined.

8086 Instructions: Arithmetic and Logical

MUL Unsigned byte or word Multiplication

Syntax: MUL reg/memory

- Multiplies an unsigned byte or word by AL or AX register, respectively.
- Upper byte of 8-bit multiplication is stored in AH and lower byte is stored in AL.
- Upper word of 16-bit multiplication is stored in DX and lower word is stored in AX.
- All the conditional code flags get affected depending upon the result.
- PF, SF, and ZF are modified while, OF, CF, and AF are undefined.
- If the most significant byte or word of the results is zero, then CF and OF both will be set.

8086 Instructions: Arithmetic and Logical

	Mnemonic	Addressing Mode
Valid	MUL BH	Register
Valid	MUL CX	Register
Valid	MUL WORD PTR [SI]	Indirect
Not valid	MUL 25H	Immediate

IMUL Signed byte or word Multiplication

Syntax: IMUL reg/memory

- Multiplies an signed byte or word by signed byte in AL or signed Word in AX register, respectively.
- Upper byte of 8-bit multiplication is stored in AH and lower byte is stored in AL.
- Upper word of 16-bit multiplication is stored in DX and lower word is stored in AX.

8086 Instructions: Arithmetic and Logical

- AF, PF, SF, and ZF are undefined.
- If the most significant byte or word of the results is nonzero, then CF and OF both will be set.
- The unsigned higher bits of the result are filled with sign bit and CF, AF are cleared.

	Mnemonic	Addressing Mode
Valid	IMUL BH	Register
Valid	IMUL CX	Register
Valid	IMUL [SI]	Indirect
Not valid	MUL 25H	Immediate

8086 Instructions: Arithmetic and Logical

DIV Unsigned division

Syntax: DIV reg/memory

- Divides an unsigned word or double word by a 16-bit or 8-bit operand.
- The dividend must be in AX for 16-bit and in (DX) (AX) for 32-bit.
- The divisor can be a register or memory but not an immediate data.
- 16-bit division result: Quotient stored in AL and Reminder stored in AH.
- 32-bit division result: Quotient stored in AX and Reminder stored in DX.
- If the quotient is too big to fit in AL (16-bit division) or in AX (for 32-bit division), them divided by 0 interrupt will be generated.
- Doesn't affect any flag.

IDIV Signed divison

Syntax: IDIV reg/memory

- Divides an unsigned word or double word by a 16-bit or 8-bit operand.
- The dividend must be in AX for 16-bit and in (DX) (AX) for 32-bit.
- The divisor can be a register or memory but not an immediate data.
- 16-bit division result: Quotient stored in AL and Reminder stored in AH.
- 32-bit division result: Quotient stored in AX and Reminder stored in DX.
- If the quotient is too big to fit in AL (16-bit division) or in AX (for 32-bit division), them divided by 0 interrupt will be generated.
- Doesn't affect any flag.

8086 Instructions: Arithmetic and Logical

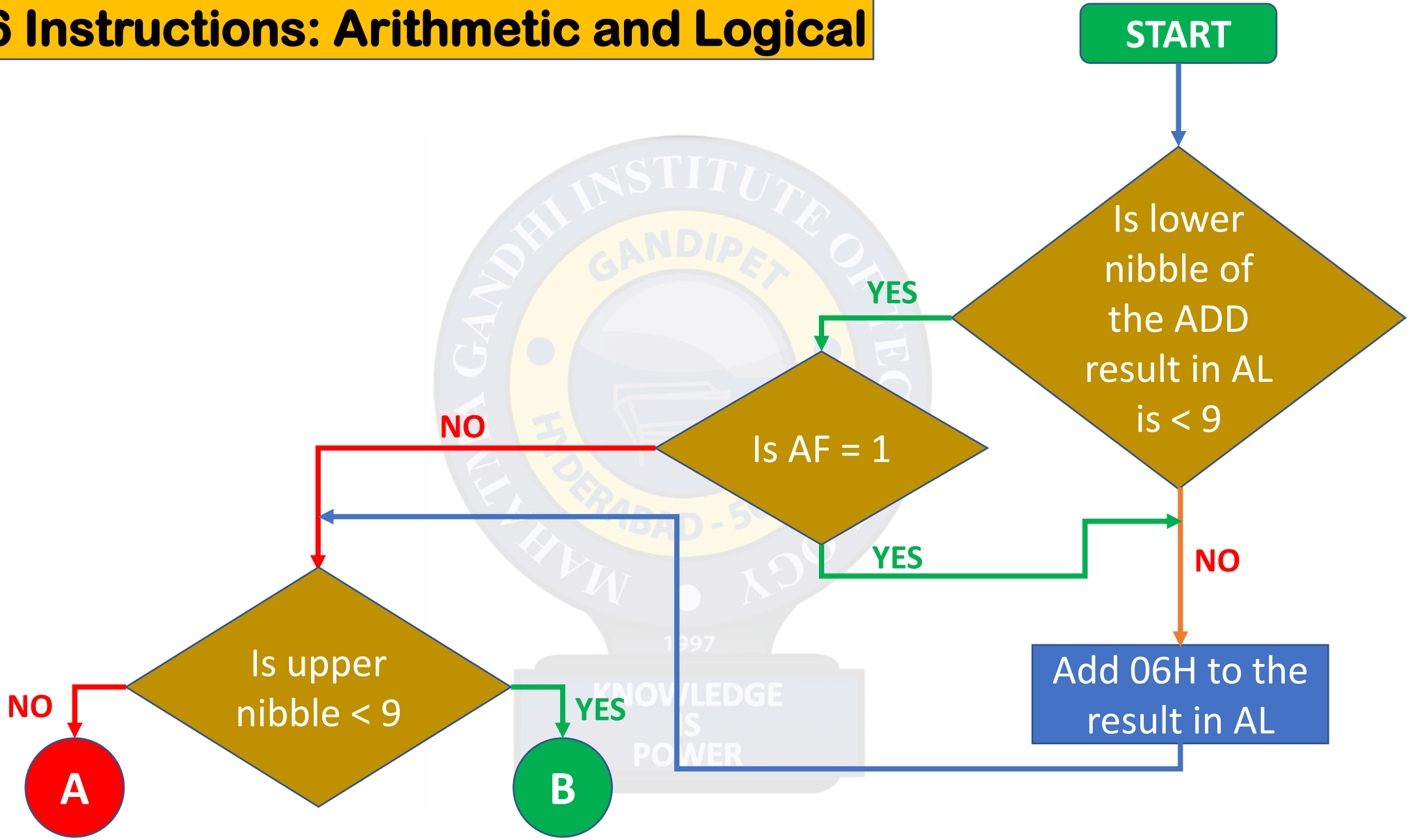
	Mnemonic	Mnemonic	Addressing Mode
Valid	DIV BH	IDIV BH	Register
Valid	DIV CX	IDIV CX	Register
Valid	DIV [SI]	IDIV [SI]	Indirect
Not valid	DIV 25H	IDIV 25H	Immediate

DAA Decimal adjust accumulator

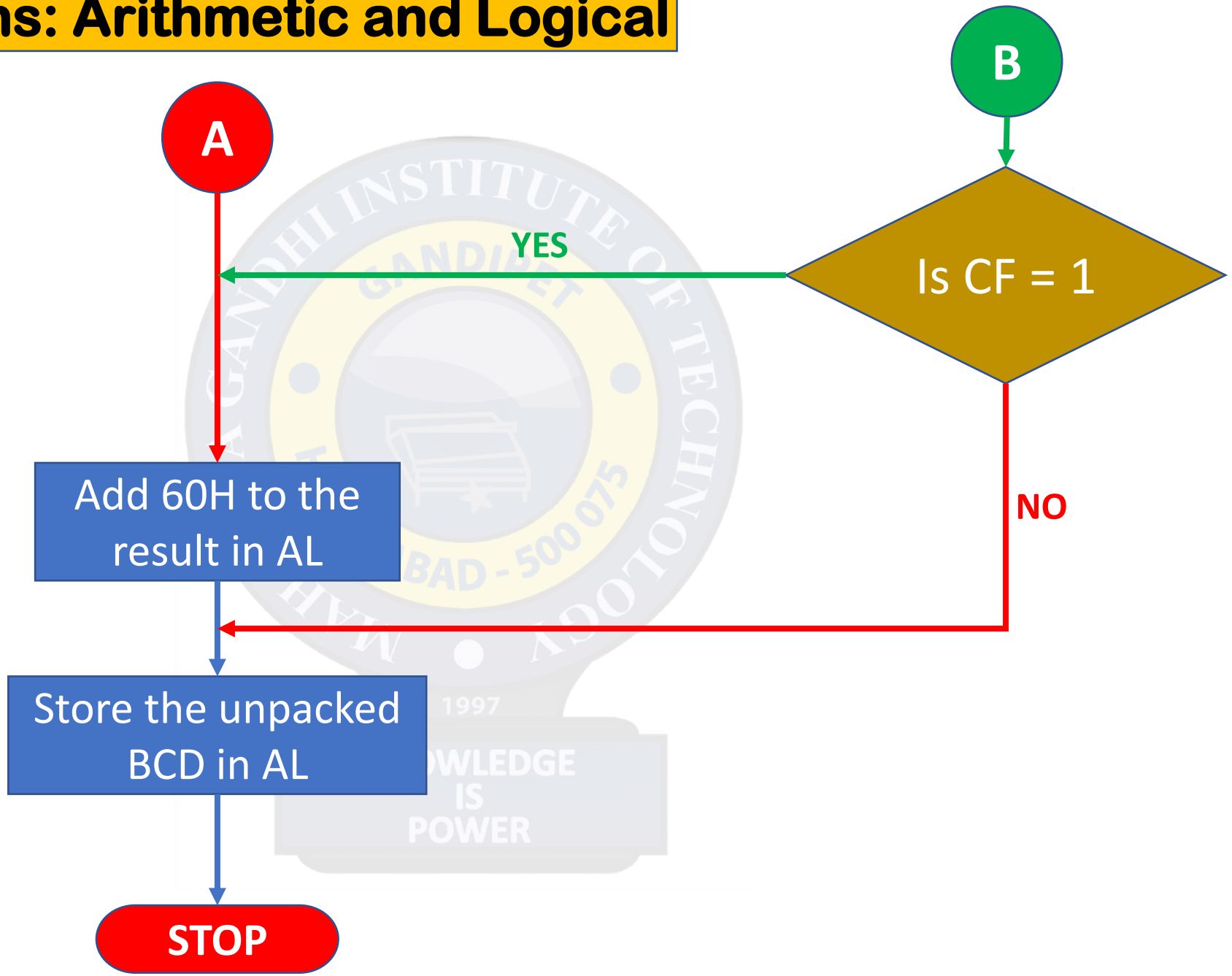
Syntax: DAA

- Executed after an addition(ADD) instruction.
- Converts the result of the addition of two packed BCD numbers to a valid BCD number.
- Result will present only in AL.

8086 Instructions: Arithmetic and Logical



8086 Instructions: Arithmetic and Logical



DAS Decimal adjust after subtraction

Syntax: DAS

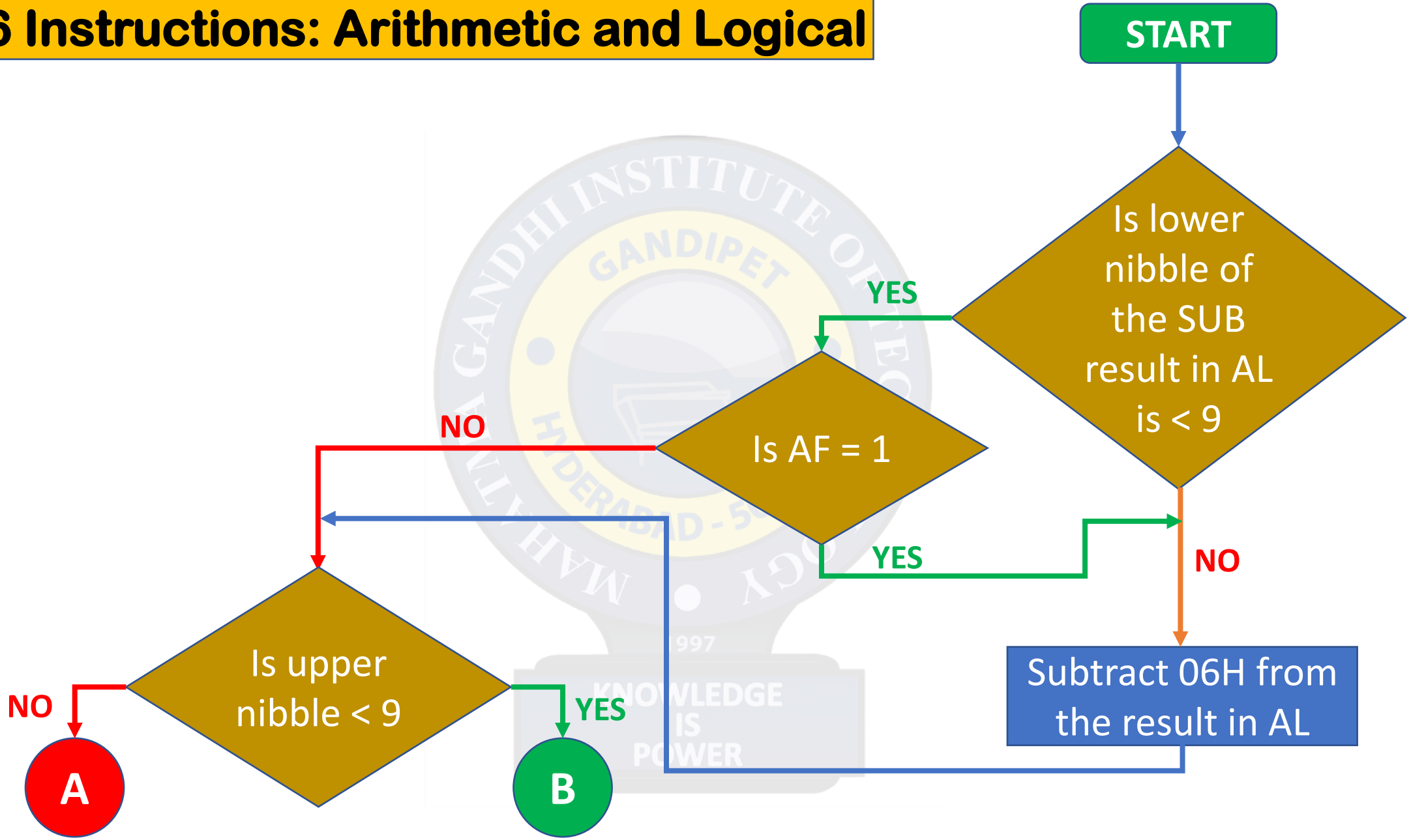
- Executed after the subtraction (SUB) instruction.
- Converts the result of the subtraction of two packed BCD numbers to a valid BCD number.
- Result will present only in AL.

NEG Negate

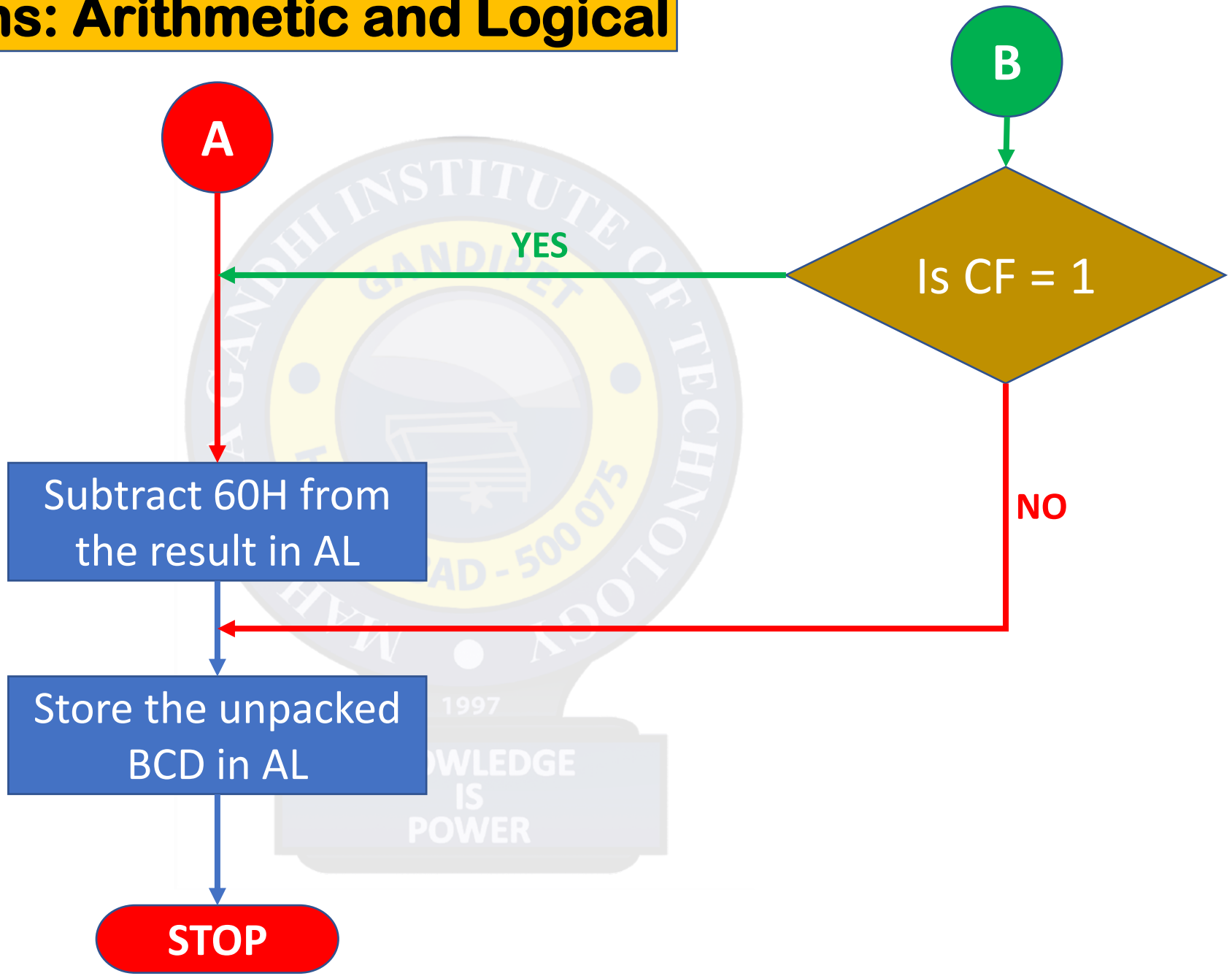
Syntax: NEG reg/memory

- Forms the 2's complement of the specified operand.
- The operand may be a reg/ memory location whose offset specified in the instruction.
- If OF = 1, the operation couldn't be completed successfully.
- All conditional code flags get affected.

8086 Instructions: Arithmetic and Logical



8086 Instructions: Arithmetic and Logical



8086 Instructions: Arithmetic and Logical

CBW Convert signed Byte to Word

Syntax: CBW

- Converts a signed byte in AL register into a signed word.
- Result stored in the AX register.
- The upper byte is filled with the SIGN (MSB of the 8-bit number in AL) bit.
- Doesn't affect any flag.

CWD Convert signed Word to Double Word

Syntax: CWD

- Copies the sign bit of AX and fill DX register with it.
- Doesn't affect any flag.

AND Logical AND

Syntax: AND destination operand, source operand

8086 Instructions: Arithmetic and Logical

- Performs bit-by-bit logical AND operation.
- Result stored in the destination operand.
- All the conditional code flags get affected.

	Destination operand	Source operand
Valid	Register	Register
Valid	Any register except segment registers	Immediate data
Valid	Register	Offset (with/without displacement)
Valid	Offset (with/without displacement)	Register
Valid	Offset (with/without displacement)	Immediate data

8086 Instructions: Arithmetic and Logical

	Mnemonic	Addressing Mode
Valid	AND AX, 5000H	Immediate
Valid	AND AX, BX	Register
Valid	AND AX, [SI]	Indirect
Valid	AND AX, [2000H]	Direct
Valid	AND [2000H], AX	Register
Invalid	AND [AX], [BX]	Based Indexed
Invalid	AND [2000H], [5000H]	Direct
Invalid	AND DS, AX	Register
Invalid	AND AX, DS	Register

8086 Instructions: Arithmetic and Logical

OR Logical OR

Syntax: OR destination operand, source operand

- Performs bit-by-bit logical OR operation.

XOR Logical Exclusive OR

Syntax: XOR destination operand, source operand

- Performs bit-by-bit logical XOR operation.

NOT Logical invert

Syntax: NOT destination operand (reg/memory)

- Inverts the content of the reg/memory, bit-by-bit.

8086 Instructions: Arithmetic and Logical

TEST Logical compare

Syntax: TEST destination operand, source operand

- Performs bit-by-bit logical AND operation between the two operands but the result not stored any where.
- OF, CF, SF, ZF, and PF flags get affected.

SHL/SAL Shift Logical/Arithmetic Left

Syntax: SHL/SAL destination operand (reg/memory), none/CL

- SHL/SAL: Perform bit-by-bit left shift operation and inserts ZEROS at the LSB positions.
- If no source operand is specified, default shift count is 1.
- The destination operand can not be immediate data.
- All conditional code flags get affected depending upon the result.

8086 Instructions: Arithmetic and Logical

	Mnemonic	Addressing Mode
Valid	TEST AX, 5000H	Immediate
Valid	TEST AX, BX	Register
Valid	TEST AX, [SI]	Indirect
Valid	TEST AX, [2000H]	Direct
Valid	TEST [2000H], AX	Register
Valid	TEST [5000H], 5000H	Immediate
Invalid	TEST [AX], [BX]	Based Indexed
Invalid	TEST [2000H], [5000H]	Direct
Invalid	TEST DS, AX	Register
Invalid	TEST AX, DS	Register

SHR Shift Logical Right

Syntax: SHR destination operand (reg/memory), none/CL

- SHR: Performs bit-by-bit right shift operation and inserts ZEROS at the LSB positions.
- If no source operand is specified, default shift count is 1.
- The destination operand can not be immediate data.
- All conditional code flags get affected depending upon the result.

SAR Shift Arithmetic Right

Syntax: SAR destination operand (reg/memory), none/CL

- SHR: Performs bit-by-bit right shift operation and inserts MSB of the operand at the shifted bit positions.
- Rest all are same as that of SAR.

8086 Instructions: Arithmetic and Logical

ROR Rotate Right without carry

Syntax: ROR destination operand (reg/memory), none/CL

- Rotates bit-by-bit right either by one or by the count specified in CL.
- Doesn't take carry flag status into consideration.
- The LSB is pushed to CF and MSB simultaneously.
- The destination operand can not be immediate data.
- PF, ZF, and SF are left unchanged.

ROL Rotate Left without carry

Syntax: ROL destination operand (reg/memory), none/CL

- Rotates bit-by-bit left either by one or by the count specified in CL.
- The MSB is pushed to CF and LSB simultaneously.
- Rest are all the same as ROR.

RCR Rotate Right through carry

Syntax: RCR destination operand (reg/memory), none/CL

- Rotates bit-by-bit right either by one or by the count specified in CL.
- For each rotation, CF is pushed into MSB and LSB is pushed into CF.
- The destination operand can not be immediate data.
- PF, ZF, and SF are left unchanged.

RCL Rotate Left through carry

Syntax: RCL destination operand (reg/memory), none/CL

- Rotates bit-by-bit left either by one or by the count specified in CL.
- CF is pushed into LSB and MSB is pushed to CF.
- Rest are all the same as RCR.

8086 Instructions: Arithmetic and Logical

Opcode	Mnemonic	Addressing Mode	Operation
SHL SAL SHR SCR ROL RCL ROR RCR	AX	Register	AX content shifted/rotated left/right by 1
	BL, CL	Register	BL content shifted/rotated left/right by count specified in CL
	DX, CL	Register	DX content shifted/rotated left/right by count specified in CL
	[2000H]	Direct	Content at offset 2000 shifted/rotated by 1
	[2000H], CL	Direct	Content at offset 2000 shifted/rotated by count specified in CL
	[BX]	Based Indexed	Content at offset in BX shifted/rotated by 1
	[SI], CL	Register Indirect	Content at offset in SI shifted/rotated by count specified in CL
	AX, 05H	**Invalid**	
	1234H, CL	**Invalid**	

8086 Instructions: String Manipulations

c) **String Manipulation Instructions**: Collection of characters/numbers/both stored in an array of memory locations is called a string.

- Byte string or Word string
- For byte string operations, the index registers are updated by 1 whereas, for word string operations, the index registers updated by 2.
- The counter in both the cases decremented by 1.
- Characters are represented by their ASCII equivalent in the memory
  - Start or end address of the string
  - Length of the string (usually stored in CX)
- DF decides the direction of accessing the string
- Source string is usually located using DS:SI register pair and the destination string is located using ES:DI register pair.

8086 Instructions: String Manipulations

REP Repeat instruction prefix

Syntax: REP String instruction

- Used as a prefix to other string manipulation instruction.
- Instruction prefixed by REP executed until CX becomes ZERO.
- REPE/REPZ: Repeat if equal/ repeat if ZF sets.
- REPNE/REPNZ: Repeat if not equal/ repeat if ZF resets.
- The conditional REP used only for CMPS and SCAS instructions.

MOVSB/MOVSW Move string byte or word

Syntax: REP MOVSB/MOVSW

- Moves string of byte/word from locations specified by DS:SI register pair to the locations pointed by ES:DI.

8086 Instructions: String Manipulations

Mnemonic	Operation
MOV AX, 5000H	Source string segment address is 5000H loaded to DS via AX
MOV DS, AX	
MOV AX, 6000H	Destination string segment address is 6000H loaded to ES via AX
MOV ES, AX	
MOV CX, 0FFH	Length of the string
MOV SI, 1000H	Source index address
MOV DI, 2000H	Destination Index address
CLD	Clear directional flag (auto-incrementing mode)
REP MOVSB	Move 0FFH string bytes from source to the destination
STD	Set directional flag (auto-decrementing mode)
REP MOVSW	Move 0FFH string words from source to the destination

8086 Instructions: String Manipulations

CMPS Compare string byte or word

Syntax: REP CMPSB/CMPSW

REPE/REPZ/REPNE/REPNZ CMPSB/CMPSW

- Compares two strings of byte/word.
- Length of the string is specified by CX.
- If both the strings are equal, ZF sets.
- Other flags affected similar to CMP instruction.
- REP is used and compares the strings until CX becomes ZERO.
- REPE/REPZ or REPNE/REPNZ compare the string bytes/words until the condition is false.
- Source string: DS:SI and the destination string: ES:DI.

8086 Instructions: String Manipulations

Mnemonic	Operation
MOV AX, SEG1	Source string segment address SEG1 loaded to DS via AX
MOV DS, AX	
MOV AX, SEG2H	Destination string segment address SEG2 loaded to ES via AX
MOV ES, AX	
MOV CX, 77H	Length of the string
MOV SI, OFFSET1	Source index address
MOV DI, OFFSET2	Destination Index address
CLD	Clear directional flag (auto-incrementing mode)
REPE CMPSB	Compare string bytes repeatedly until both are not equal.
STD	Set directional flag (auto-decrementing mode)
REP CMPSW	Compare string words repeatedly until CX becomes ZERO

8086 Instructions: String Manipulations

SCAS Scan string byte or word

Syntax: REP SCASB/SCASW

REPE/REPZ/REPNE/REPNZ CMPSB/CMPSW

- Scans a string of bytes/words for an operand byte/word specified in AL/AX.
- The string is pointed by ES:DI register pair.
- CX specifies the length of the string.
- DF controls the direction of scanning.
- Execution stops when a match to AL/AX is found in the string and ZF sets.
- REPNE prefix is used with SCAS.
- The pointers and counters updated accordingly.

8086 Instructions: String Manipulations

Mnemonic	Operation
MOV AX, SEG	Destination string segment address SEG loaded to ES via AX
MOV ES, AX	
MOV CX, 77H	Length of the string
MOV DI, OFFSET	String offset address
MOV AX, WORD	Word to be scanned in the string
CLD	Clear directional flag (auto-incrementing mode)
REPE SCASB	Scans the string byte-by-byte until a match is found or CX = 0
STD	Set directional flag (auto-decrementing mode)
REPNE SCASW	Scans the string word-by-word until a match is found or CX=0

8086 Instructions: String Manipulations

LODS Load string byte or word

Syntax: LODS

- Loads AL/AX register by the content of string pointed by DS:SI.
- SI is updated according to DF.
- If a byte is transferred, SI is modified by 1 and if a word is transferred, SI is updated by 2.
- No flags get affected.

STOS Store string byte or word

Syntax: STOS

- Stores the string in AL/AX register to the location pointed by ES:DI.
- Rest all are the same as that of **LODS**.

8086 Instructions: String Manipulations

Mnemonic	Operation
MOV AX, SEG1	Source string segment address SEG1 loaded to DS via AX
MOV DS, AX	
MOV AX, SEG2H	Destination string segment address SEG2 loaded to ES via AX
MOV ES, AX	
MOV SI, OFFSET1	Source index address
MOV DI, OFFSET2	Destination Index address
LODSB/LODSW	Loads AL/AX with the strings from DS:SI location
STOSB/STOSW	Stores the string in AL/AX to the location pointed by ES:DI

8086 Instructions: Control Transfer/Branch

d) **Control Transfer/Branch Instructions**: Transfers the flow of execution to a new location specified in the instruction directly/indirectly.

- CS and IP are loaded with new location values to which the control has to be transferred.
- CS may or may not have the same content depending upon the addressing mode.
- Control Transfer
  - Unconditional control transfer/branching: execution control will be transferred to the specified location independent of any condition.
  - Conditional control transfer/branching: control transferred to the new location depending upon the result of the previous instruction. Otherwise, the execution continues normally.

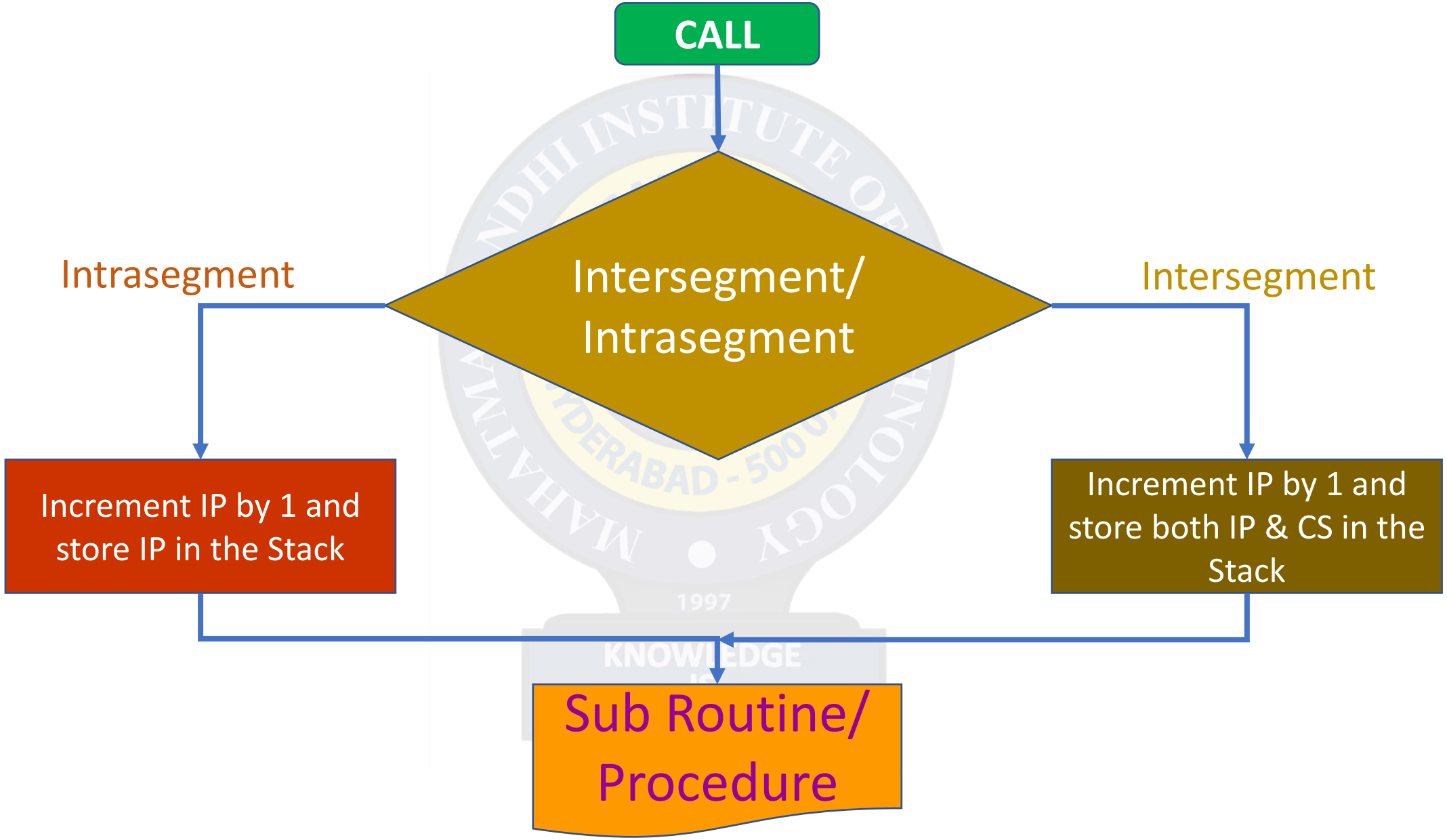
Unconditional Transfer/Branch Instructions

CALL Unconditional call

Syntax: CALL LABEL

- Calls a subroutine from the main program.
- LABEL can be specified directly or indirectly.
- CALL:
  - **NEAR CALL:** the subroutine present in the same segment within the displacement of ±32KB. This addressing mode is intrasegment.
  - **FAR CALL:** the subroutine present in any other segment than the current. This addressing mode is intersegment.

8086 Instructions: Control Transfer/Branch



8086 Instructions: Control Transfer/Branch

	D_7 … D_0	D_7 … D_0	D_7 … D_0						
Direct Near	OPCODE	DISP. LB	DISP. HB						
	D_7 … D_0	D_7 … D_0	D_7	D_6	D_5	D_4	D_3	D_2 D_1 D_0	
Indirect Near	OPCODE	OPCODE	MOD		0	1	0	R/M	
	D_7 … D_0	D_7 … D_0 D_7 … D_0	D_7 … D_0 D_7 … D_0						
Direct Far	OPCODE	LB | HB	LB | HB						
		OFFSET	SEGMENT						
	D_7 … D_0	D_7 … D_0	D_7	D_6	D_5	D_4	D_3	D_2 D_1 D_0	
Indirect Far	OPCODE	OPCODE	MOD		0	1	0	R/M	

RET Return from the Procedure

Syntax: RET

- Must be executed at the end of the procedure.
- Stored values of CS, IP and flags in the stack are retrieved into the corresponding registers and the execution of the main program continues.
- In FAR procedure, the SP points to both CS and IP whereas, in NEAR procedure, SP points to only IP.
- **RET:**
  - Return within segment
  - Return within segment adding 16-bit immediate displacement to the SP contents
  - Return intersegment
  - Return intersegment adding 16-bit immediate displacement to the SP contents

8086 Instructions: Control Transfer/Branch

INT N Interrupt Type N

Syntax: INT **N**

- N points to one of the 256 interrupts defined in 8086 (00H – FFH).
- N is multiplied by 4 and this is taken as offset address into IP register.
- The segment address will be 0000H stored in CS.
- IF must be set to execute the instruction.

Example: INT 20H

offset = 20H * 4 = 80H

- The CS : IP = 0000H : 0080H

15 – 8	7 – 0
CS High	CS Low
SEGMENT	

15 – 8	7 – 0
IP High	IP Low
OFFSET	

. . .	
CS HIGH	0000:0083
CS LOW	0000:0082
IP HIGH	0000:0081
IP LOW	0000:0080
. . .	

8086 Instructions: Control Transfer/Branch

INTO Interrupt on overflow

Syntax: INTO

- Executed when OF is set.
- The CS and IP are taken from the address 0000H:0010H.
- This is similar to Type 4 interrupt.

Example: INT 04H

offset = 04H * 4 = 10H

- The CS : IP = 0000H : 0010H

8086 Instructions: Control Transfer/Branch

JMP Unconditional jump

Syntax: JMP LABEL

- Unconditionally transfers the control of execution to the specified address.
- 8-bit displacement (intrasegment, relative, short) or 16-bit displacement (intrasegment, relative, far) or CS : IP (intersegment, direct, far).
- No flags affected.

JMP	8-bit DISP				Intrasegment, relative, SHORT
JMP	16-bit DISP (LB)		16-bit DISP (UB)		Intrasegment, relative, FAR
JMP	IP (LB)	IP (HB)	CS (LB)	CS (HB)	Intersegment, relative, FAR

8086 Instructions: Control Transfer/Branch

IRET Return from ISR

Syntax: IRET

- Must be executed at the end of an ISR.
- Stored values of CS, IP and flags in the stack are retrieved into the corresponding registers and the execution of the main program continues.

LOOP Loop unconditionally

Syntax: LOOP LABEL

- Executes a part of the program from the label CX number of times.
- At each iteration, CX is decremented by 1.
- DECREMENT COUNTER and JUMP IF NOT ZERO.

8086 Instructions: Control Transfer/Branch

Label(s)	Mnemonic	Operation
	MOV CX, 0009H	CX = 09H (number of iterations) and BX is the data
	MOV BX, 0FF7H	
LABEL	MOV AX, 1234H	AX is filled with data
	OR BX, AX	Perform logical OR, store the result in BX
	AND DX, AX	Perform logical AND, store the result in DX
	LOOP LABEL	Repeat from LABEL for CX times
	MOV [SI], DX	Store the result at location pointed by SI
	MOV [DI], BX	Store the result at location pointed by DI

- Executes next instruction in the sequence when CX = 00H.
- If the initial value of CX = 0, LOOP will not be executed.
- No flags get affected.

8086 Instructions: Control Transfer/Branch

Conditional branch instructions:

- Execution jumps to the LABEL specified in the instruction only certain condition is TRUE.
- Conditions are status of conditional flags.
- No flags get affected.
- Only SHORT jumps are allowed (i.e., the displacement is -80H to 7FH).

Mnemonic	Operation
JZ/JE LABEL	Transfers execution to LABEL, if ZF = 1
JNZ/JNE LABEL	Transfers execution to LABEL, if ZF = 0
JS LABEL	Transfers execution to LABEL, if SF = 1
JNS LABEL	Transfers execution to LABEL, if SF = 0
JO LABEL	Transfers execution to LABEL, if OF = 1

8086 Instructions: Control Transfer/Branch

Mnemonic	Operation
JNO LABEL	Transfers execution to LABEL, if OF = 0
JP/JPE LABEL	Transfers execution to LABEL, if PF = 1
JNP LABEL	Transfers execution to LABEL, if PF = 0
JB/JANE/JC LABEL	Transfers execution to LABEL, if CF = 1
JNB/JAE/JNC LABEL	Transfers execution to LABEL, if CF = 0
JBE/JNA LABEL	Transfers execution to LABEL, if CF = 1 or ZF = 1
JNBE/JA LABEL	Transfers execution to LABEL, if CF = 0 or ZF = 0
JL/JNGE LABEL	Transfers execution to LABEL, if neither SF = 1 nor OF = 1
JNL/JGE LABEL	Transfers execution to LABEL, if neither SF = 0 nor OF = 0
JLE/JNC LABEL	Transfers execution to LABEL, if ZF = 1 or neither SF nor OF is 1
JNLE/JE LABEL	Transfers execution to LABEL, if ZF = 0 or at least any one of SF and OF is 1 (both SF and OF are not zero)

8086 Instructions: Control Transfer/Branch

Mnemonic	Operation
JCXZ LABEL	Transfers execution to LABEL, if CX = 0
LOOPZ/LOOPE LABEL	Loop through a sequence of instructions from LABEL while ZF = 1, and CX = 0
LOOPNZ/LOOPNE LABEL	Loop through a sequence of instructions from LABEL while ZF = 0, and CX = 0

8086 Instructions: Flag manipulation and Processor control

e) **Flag manipulation and Processor control**: Control the functioning of available hardware inside the processor.

- Remaining flags will be modifies using either POPF or SAHF instructions.
- Except carry flag no other status flags can be changed directly using flag manipulation instructions.

Mnemonic	Operation
CLC	Clear carry flag
CMC	Complement carry flag
STC	Set carry flag
CLD	Clear direction flag
STD	Set direction flag
CLI	Clear interrupt flag
STI	Set interrupt flag

8086 Instructions: Flag manipulation and Processor control

Processor control/Machine control: Control the processor and its execution directly.

Mnemonic	Operation
WAIT	Wait for test input pin goes LOW
HLT	Halt the processor
NOP	No operation
ESC	Escape to external device like numeric co-processor
LOCK	Bus lock instruction prefix

8086/88 Assembler Directives and Operators

8086 Assembler Directives & Operators

- ✓ An assembler is a program used to convert an assembly language program (ALP) into machine language.
- ✓ These machine language codes further converted to executable codes.
- ✓ Assembler can only find syntax errors but not the logical or other programming errors.
- ✓ Assembler directives provide the necessary guidance to the assembler to all these tasks.
- ✓ Help the assembler to correctly understand the instructions.
- ✓ An operator assists the assembler to assign a constant with label, initialize memory locations, or labels with constants.
- ✓ Operators perform certain arithmetic and logical tasks whereas, the directives helps in correct interpretation of program.

8086 Assembler Directives & Operators

DB: Define Byte: used to define a byte or array of bytes.

- ✓ It directs the assembler to allocate specified number of bytes in the memory to a constant, variable or a string.
- ✓ Reserves memory bytes with ASCII codes of the string.

Example: OPR1 DB 54H

RANKS DB 01H, 02H, 03H, 04H, 05H

STRINGS DB 'HELLO WORLD..!'

DW: Define Word: used to define a word or array of words.

Example: OPR1 DW 2354H

RANKS DW 0123H, 0234H, 0345H, 0456H

WDATA DW 7 DUP (5555H)

8086 Assembler Directives & Operators

DQ: Define Quadword: used to reserve 4 words (8 bytes) of memory.

DT: Define Ten Bytes: used to reserve 10 bytes to the specified variable.

ASSUME: Assume Logical Segment Name: informs assembler the names of the logical segments to be assumed for different segments used in program.

- ✓ Each segment is given a name.

Example: ASSUME CS: CODE, DS: DATA, ES:EXTRA, SS: STACK

- ✓ The corresponding segment register loaded with the segment address.
- ✓ Assembler assumes the segment DATA as default data segment and CODE as default code segment.
- ✓ ASSUME must be at the starting of each program. Without which 'CODE/DATA EMITTED WITHOUT SEGMENT' will be issued by the assembler.

8086 Assembler Directives & Operators

END: End of Program: marks the end of an assembly language program.

- ✓ Assembler usually ignores the source lines after END statement.

ENDP: End of Procedure: indicates the end of subroutine/ procedure.

Example:

```
        CALL PROCUDURE1
        ⋮
PROCEDURE PROCEDURE1 NEAR
        ⋮
        ADD AX, BX
        RET
PROCEDURE1 ENDP
```

ENDS: End of Segment: marks the end of a logical segment.

Example:

```
DATA SEGMENT
      ⋮
DATA ENDS
```

EVEN: Align on Even Memory Address: if the current location is not even, EVEN updates the location counter to the next even address.

Example:

```
        EVEN
PROCEDURE  ROOTS
      ⋮
ROOTS  ENDP
```

EQU: Equate: assigns a label with a value or a symbol.

- ✓ Reduces the recurrence of numerical values or constants.

Example:

```
LABEL  EQU  1234H

ADDITION  EQU  ADD
```

EXTRN: External & PUBLIC: Public: informs the assembler that the names, procedures and labels declared after EXTRN have already been defined in some other module.

- ✓ These names, procedures, and labels in the original module must be declared public using PUBLIC.

Example:

```
MODULE1  SEGMENT
PUBLIC  FACTORIAL FAR
MODULE1  ENDS
MODULE2  SEGMENT
EXTRN  FACTORIAL FAR
MODULE2  ENDS
```

8086 Assembler Directives & Operators

GROUP: Group the related segments: forms the logical group of segments with similar purpose or type.

- ✓ The assembler conveys the linker/loader to form the code such that the group declared segments or operands must lie within a 64KB memory segment.
- ✓ All the grouped segments have same segment address.

Example: PROGRAM GROUP CODE, DATA, STACK
ASSUME CS: PROGRAM, DS: PROGRAM, SS: PROGRAM

LABEL: Label: used to assign a name to the current content of the location counter.

- ✓ It assigns the declared label with the current contents of the location counter.
- ✓ Type of label must be specified as NEAR, FAR, BYTE, or WORD.

8086 Assembler Directives & Operators

Example:

```
DATA SEGMENT
DATAS DB 50H DUP (?)
DATA-LAST LABEL BYTE FAR
DATA  ENDS
```

LENGTH: Byte length of a Label: used to refer to the length of a data array or string.

Example:

```
MOV CX, LENGTH ARRAY
```

LOCAL: labels, variables, constants or procedures declared LOCAL in a module are to be used only in that module.

- ✓ Each variable with LOCAL serves different purposes in different modules.

Example:

```
LOCAL A, B, DATA, ARRAY
```

NAME: Logical name of a Module: used to assign a name to the assembly language programming module.

8086 Assembler Directives & Operators

OFFSET: Offset of a Label: computes the 16-bit displacement of a label and replaces the string OFFSET LABEL by the computed displacement.

- ✓ Used with arrays, strings, labels and procedures to decide offset in their default segments.
- ✓ Commonly used in indirect, indexed, based indexed, and relative based indexed addressing modes.

Example:

```
CODE SEGMENT
MOV SI, OFFSET LIST
CODE ENDS
DATA SEGMENT
LIST DB 10H
DATA ENDS
```

8086 Assembler Directives & Operators

ORG: Origin: directs the assembler to start the memory allotment for the particular segment, block, or code from the declared address in the ORG.

- ✓ If ORG is not defined, the location counter points to 0000H.
- ✓ ORG 0200H means the offset starts from 0200H location in that segment.

PROC: Procedure: marks the start of a named procedure.

- ✓ Type is also specified as NEAR or FAR.

Example:

```
RESULT PROC NEAR
ROUTINE PROC FAR
```

PTR: Pointer: used to declare the type of a label, variable or memory.

- ✓ Prefixed by either BYTE or WORD.
- ✓ The specified label is treated as 8-bit quantity if it is prefixed by BYTE PTR.
- ✓ The specified label is treated as 16-bit quantity if it is prefixed by WORD PTR.

8086 Assembler Directives & Operators

Example:

```
MOV AL, BYTE PTR [SI]
INC BYTE PTR [BX]
INC WORD PTR [BX]
JMP NEAR PTR [BX]
JMP FAR PTR [BX]
```

SEG: Segment of a Label: used to decide the segment address of a label, variable, or procedure.

Example:

```
MOV AX, SEG ARRAY
MOV DS, AX
```

SEGMENT: Logical Segment: marks the starting of a logical segment.

Example:

```
EXE.CODE SEGMENT GLOBAL
EXE.CODE ENDS
```

8086 Assembler Directives & Operators

SHORT: size of the displacement is just a byte (-128 to +127).

Example: JMP SHORT LABEL

TYPE: the data type of the specified label.

- ✓ For word type variable the TYPE is 2, for byte 1, and for double word 4.

Example: STRING DW 'Hello World..!'
MOV AX, TYPE STRING Loads 2 into AX.

GLOBAL: labels, variables, constants and procedures declared GLOBAL may be used by other modules of the program.

Example: ROUTINE PROC GLOBAL

8086 Assembler Directives & Operators

+ & - : Operators used to add or subtract the displacement (8/16-bit).

Example:

```
MOV AL, [SI + 2]
MOV DX, [BX - 05]
MOV BX, [OFFSET LABEL + 10H]
```

FAR PTR: a label with FAR PTR is not available in the same segment.

Example:

```
JMP FAR PTR LABEL
CALL FAR PTR ROUTINE
```

NEAR PTR: a label with NEAR PTR is available in the same segment.

Example:

```
JMP NEAR PTR LABEL
CALL NEAR PTR ROUTINE
```

- The default pointer is NEAR PTR if none is specified with the label.

References

1. Advanced Microprocessors and Peripherals – A. K. Ray and K. M. Bhurchandani, TMH, 2nd Edition 2006.
2. Microprocessors and Interfacing – D. V. Hall, TMH, 2nd Edition 2006.
3. The Intel Microprocessors: Architecture, Programming, and Interfacing – Barry B. Brey, Pearson, 8th Edition 2009.

Thank You