MICRO PROCESSOR AND INTERFACING TECHNIQUES

DIGIAT LAB ASSIGNMENT - 1

NAME: Meenakshi. V

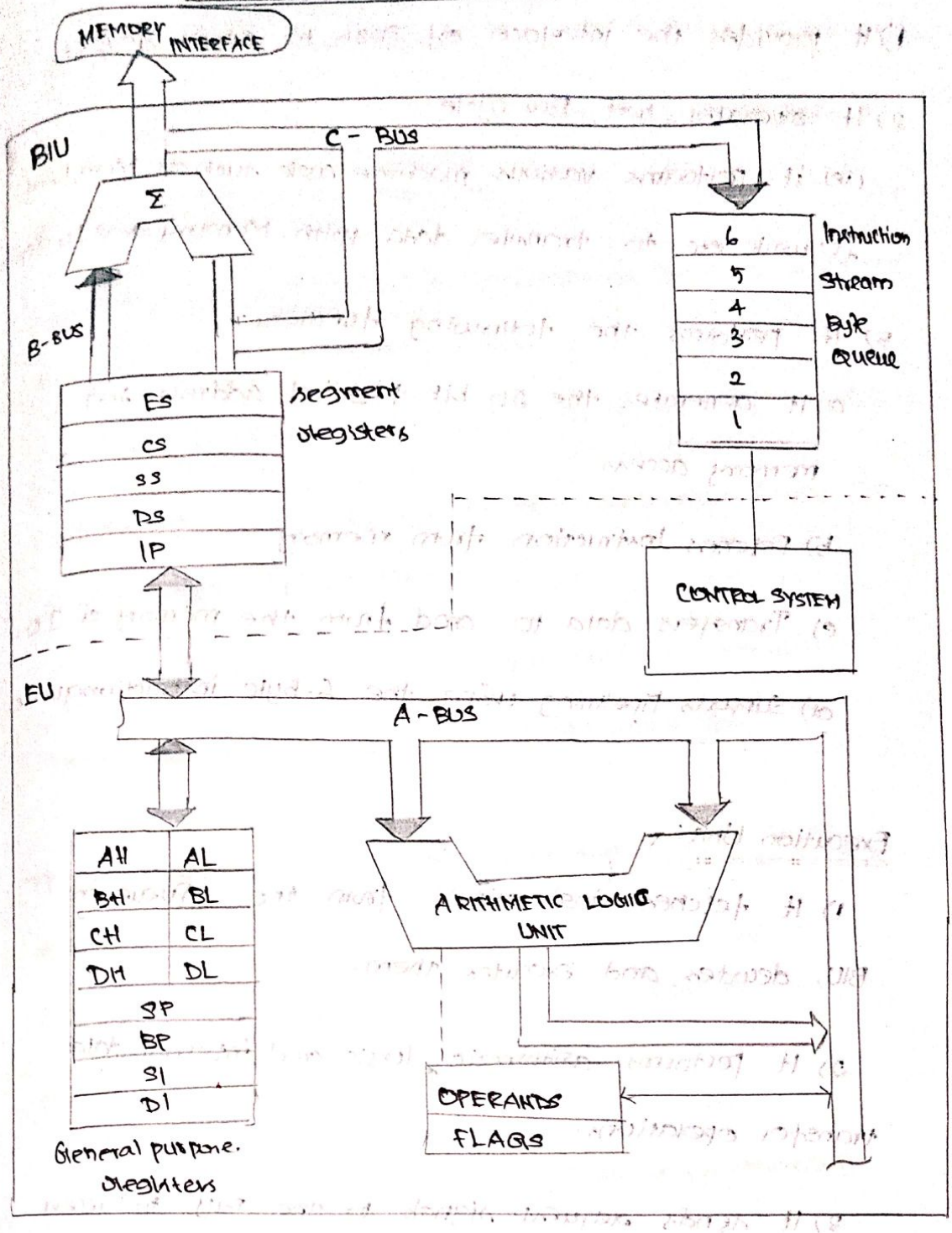
REG.NO: RMIDD019

CODE: MPL CSI2006

SLOT: L53+L54

FACULTY: NARESH. K.

INTERNAL BLOCK DIAGRAM - 8086



ARCHITECTURE OF 8086

As 8086 does 2-stage pipelining, its architecture is divided into two units.

1) BUS INTERFACE UNIT (BIU)

2) EXECUTION UNIT (EU)

Bus Interface Unit (BIU)

1) It provides the interface of 8086 to other devices.

2) It operates with Bus cycle

(ie) it performs various machine cycle such as Mem, Read, IO write etc to transfer data with Memory and Io device

3) It performs the following functions:

a) It generates the 20 bit physical address for memory access

b) Fetches Instruction from memory

c) Transfers data to and from the memory & IO.

d) Supports Pipelining using the 6-byte instruction queue.

Execution Unit:

1) It fetches instructions from the Queue in BIU, decodes and executes them.

2) It performs arithmetic, logic and internal data transfer operations.

3) It sends request signals to the BIU to access the external module.

4) It operates with T-states (clock cycles).

Working principle of 8086: (Flow)

1) BIU outputs the content of instruction pointer onto the address Bus. This will cause selected byte or word to be read into BIU.

2) Instruction pointer (IP) is incremented by 1 to prepare for the next instruction fetch.

3) Once fetched inside BIU, it is passed to the Queue. (6 instructions)

4) Execution Unit (EU) draws this instruction from the queue and begins execution.

5) While EU is executing the current instruction, the BIU proceeds to fetch new instructions.

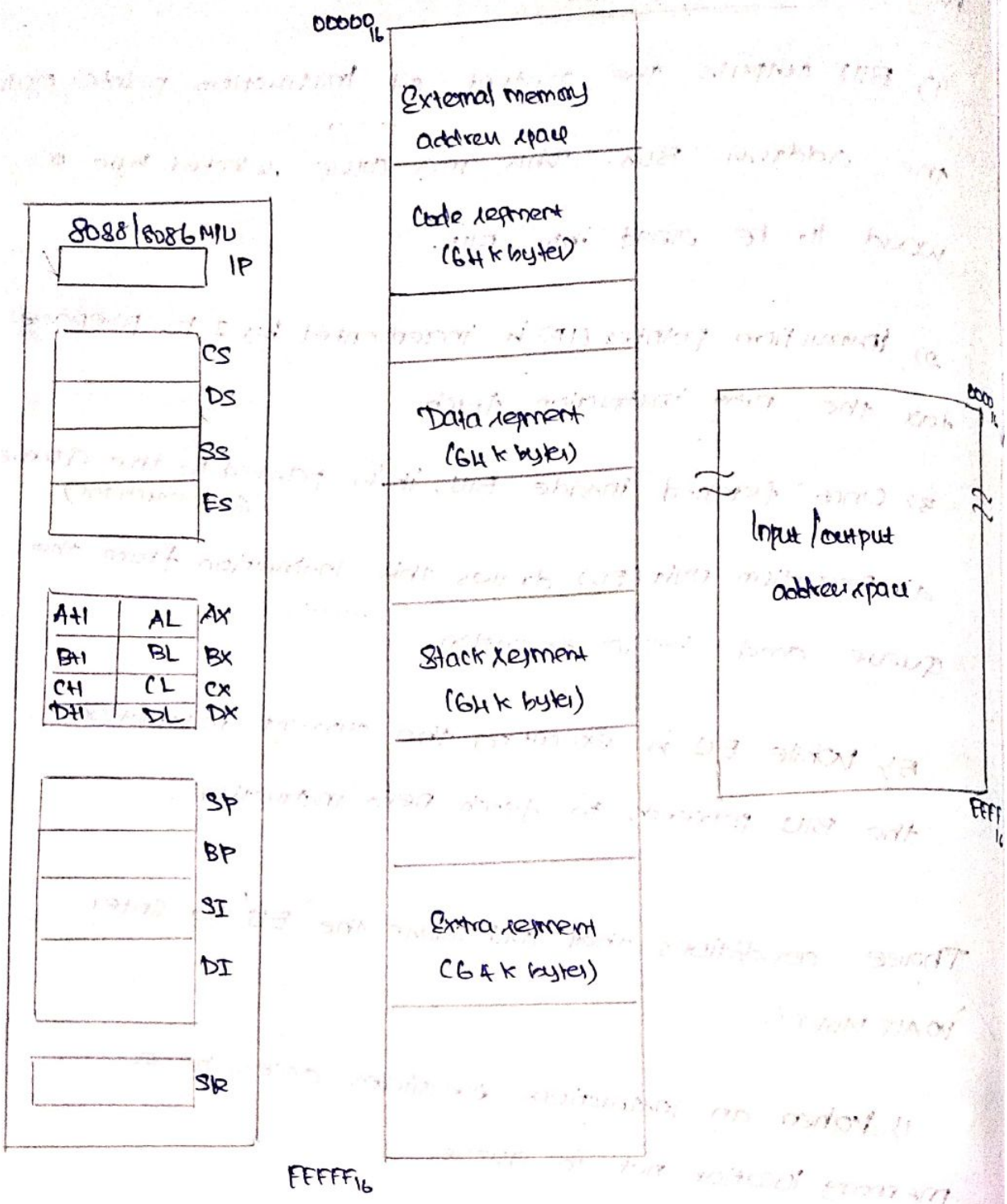
Three conditions that will cause the 'EU' to enter 'WAIT MODE':

1) When an instructions requires access to a memory location not in queue.

2) Jump instructions executed [Branching instructions]

3) Slow to execute.

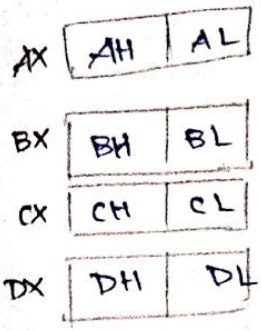
Eg. AAM → AASCI Adjust Multiplication.
↳ requires 83 clock cycles to complete



SOFTWARE MODEL OF 8086 MPU

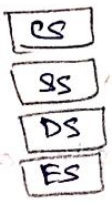
8086 Registers

General purpose



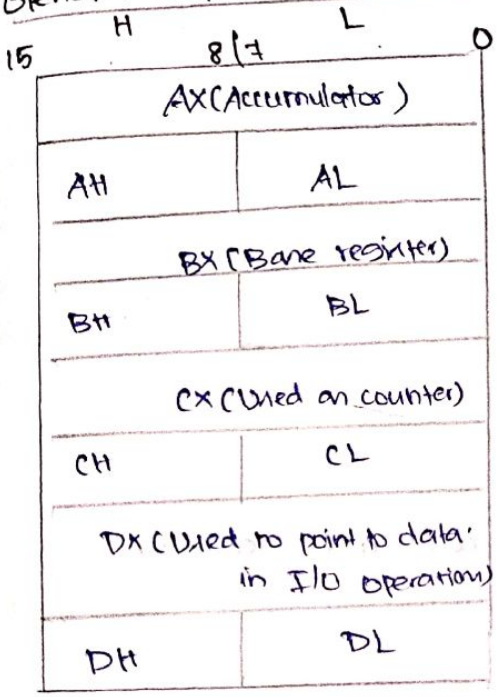
Index

Segment



Status and control

General Purpose registers



AX - the Accumulator
BX - the Base register
CX - the count register
DX - the data register

* Normally used for storing temporary results.
* Each of the registers is 16 bits wide (AX, BX, CX, DX)
* Can be accessed in either 16 or 8 bits AX, AH, AL.

AX:

* Accumulator register → preferred register to use in arithmetic, logic and data transfer instructions because it generates the shortest Machine language code
* Must be used in Multiplication and division Operation.
* Must also be used in I/O operation.

BX

* Base register
* Also serves as an address register

CX

Count register

Used as a loop counter

Used in shift and rotate operations

DX

Data registers

Used in multiplication and division

Also used in I/O operation.

Pointer and Index Registers

SP	Stack pointer
BP	Base pointer
SI	Source Index
DI	Destination Index
IP	Instruction pointer

* All 16 bits wide, H/H bytes are not accessible
* Used as memory pointers

Eg: MOV AH, [SI]

→ Move the byte stored in memory location whose address is contained in register SI to register AH.

IP is not under direct control of the programmer.

FLAG REGISTERS

Flags H								Flags L							
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

OF → overflow flag
DF → direction flag
IF → Interrupt enable flag } Control flags
TF → Trap flag
SF → sign flag
ZF → zero flag
AF → Auxiliary carry flag
PF → Parity Flag
CF → Carry Flag.

6 are ~~conditional~~ status flags
3 are control flags.

8086 Programmer's Model

BIU registers (20 bit adder)

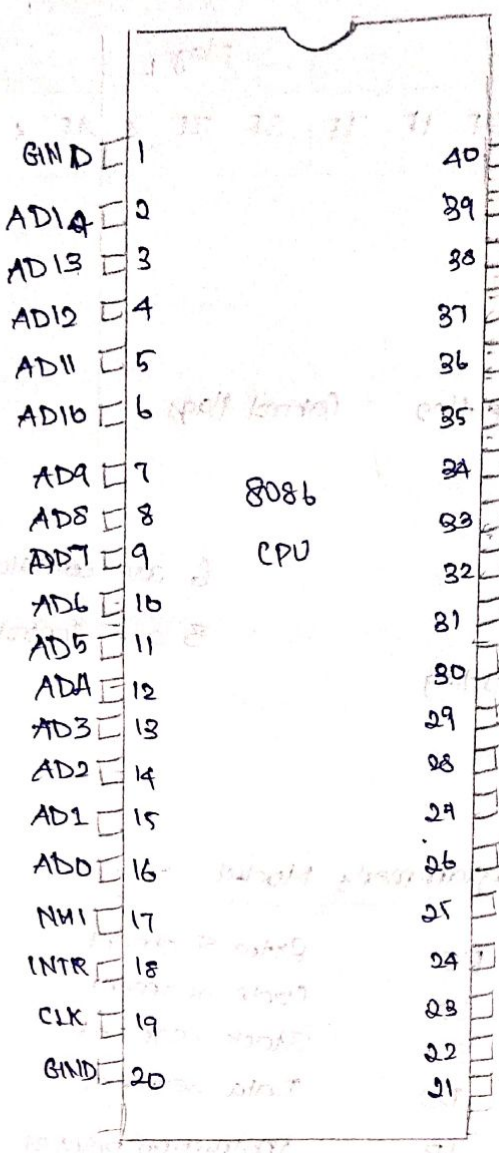
ES	Extra Segment
CS	Code segment
SS	Stack segment
DS	Data segment
IP	Instruction pointer

EU registers

AX	AH	AL	Accumulator
BX	BH	BL	Base register
CX	CH	CL	Count register
DX	DH	DL	Data register
SP			Stack pointer
BP			Base pointer
SI			Source Index register
DI			Destination Index Register
FLAGS			

INTEL 8086- PIN DIAGRAM

MAX MODE { MIN MODE }



* Ground: Pin 1 and Pin 20 (GND)

* Clock: Pin 19, Duty cycle (33%) (CLK)

* Powercycle: Pin 40, 5V ± 10%. (VCC)

* Reset: • Pin 21 (RESET)
  • Registers, segments, flags
  • CS: FFFFH
  IP: 0000H

* Address latch Enable: Pin 25 (ALE)
  When high, multiplexed address/data bus contains address information.

* Address/Data Bus: Pin 2 - Pin 16 and Pin 39
(AD0 - AD15)

contains address bits $A_{15} - A_0$ when ALE is 1.

contains data bits $D_{15} - D_0$ when ALE is 0.

* Interrupt: Non maskable interrupt (NMI) → Pin 17
Interrupt Request (INTR) → Pin 18
Interrupt Acknowledge ($\overline{INTA}$) → Pin 24

* Hold: Direct Memory Access purpose
Hold (HOLD) - pin 31
Hold acknowledge (HLDA) - pin 30

* Address/status Bus: Pin 35 - Pin 38
(A16 - A19 and S3 - S6)
Address bits A19 - A16
status bits S6 - S3

S6 - logic 0
S5 - Indicates condition of IF flag bits
S_4, S_3 - Indicates which segment is accessed during current bus cycle

S4	S3	Function
0	0	Extra segment
0	1	Stack segment
1	0	code or no segment
1	1	Data segment.

* Bus High Enable/S7 : Pin 34 ($\overline{BHE}$ / S7)

Enables most significant data bits D15 - D8 during read (or) write operation

S7 - always 1.

BHE#, A0 :

0,0 : Whole word (16 bits)

0,1 : High byte to/from odd address

1,0 : Low byte to/from

1,1 : No selection.

* Min/Max mode : Pin 33 (MN/$\overline{MX}$)

Maximum mode : 0V

Minimum mode : +5V

Minimum mode Pins → Pin 24 to Pin 31.

* Read signal : Pin 32 ($\overline{RD}$)

* Write signal : Pin 29 ($\overline{WR}$)

* Memory (or) (I/O): Pin 28 (M/$\overline{IO}$)

* Data transmit/receive: Pin 27 (DT/$\overline{R}$)

* Data Bus enable: Pin 26 ($\overline{DEN}$)

* Status signal : Pin 26 to Pin 28 ($\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$)

Inputs to 8288 to generate eliminated signals due to max mode.

S_2 S_1 S_0

000 - INTA

001 - read I/O port

010 - write IO port

011 - halt

100 - code access

101 - read memory

110 - write memory

111 - none - passive

* Lock output : Used to lock peripherals to the system.
Activated by using the Lock: prefix on any instructions.

Lock output ($\overline{LOCK}$) - pin 29.

DMA Request / Grant - Pin 30 & Pin 31

* Queue status: Pin 24 and Pin 25

Used by numeric co-processor (8087)

QS1 QS0

00 - Queue is idle
01 - First byte of opcode
10 - Queue is empty
11 - Subsequent byte of opcode

ADDRESSING MODES

The way in which the processor gets data from the user is called Addressing mode.

It can get data from different sources

registers

by instructions.

We can also refer addressing modes on the way in which the operand of an instruction is specified.

DIFFERENT ADDRESSING MODES:

8086 provides different addressing modes for Data, Program and stack memory.

ADDRESSING MODES FOR DATA MEMORY:

1) Implied Addressing Mode:

The operand is specified in the instruction itself.

The 8 bit or 16 bit data is a part of instruction.

Zero address instruction are formed with implied addressing mode.

Example:

CLC - The instructions clears the carry flag to zero

STC - sets the carry flag

CLD - clears the direction flag.

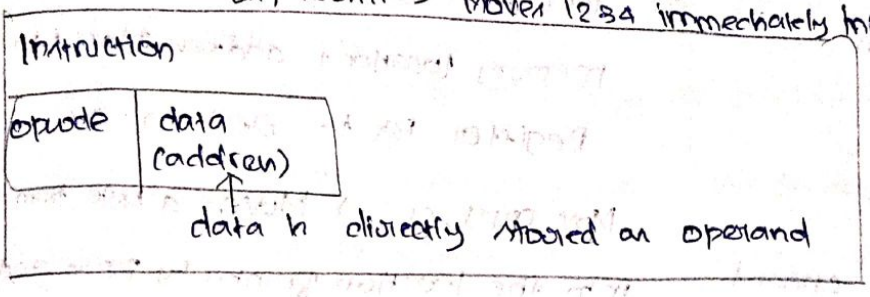
2) Immediate Addressing Mode:

In this mode, data is placed in addres field of instruction designed as one address instruction format.

Eg

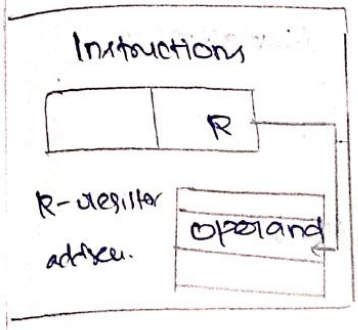
MOV AL, 35H → Moves data 35 to AL register

MOV BX, 1234H → Moves 1234 immediately into BX register.



3) Register Mode:

In this mode, operand is placed in 8-bit or 16 bit register. The register is specified in the instructions.



Eg:

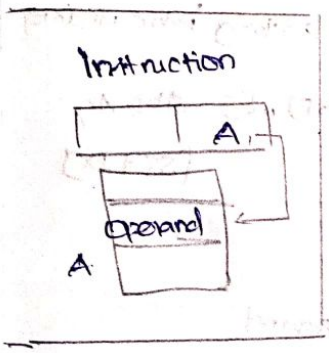
MOV AX, CX → Moves data of CX register into AX register.

MOV CL, DL → Moves data of DL into CL register

4) Direct Addressing mode: (Absolute)

Operands address is given in the instruction as an 8-bit.

In the mode the 16 bit address of data is a part of instruction.



Eg: ADD, AL, [0301] → Add the data of direct address 301 to the contents of AL

MOV CL, [4321H] : Moves data from location 4321H in the segment into CL.

The physical address is calculated as $DS*10_H + 4321$

Assume DS = 5000H, ∴ PA = 50000+4321 = 54321

CL = [54321 H]

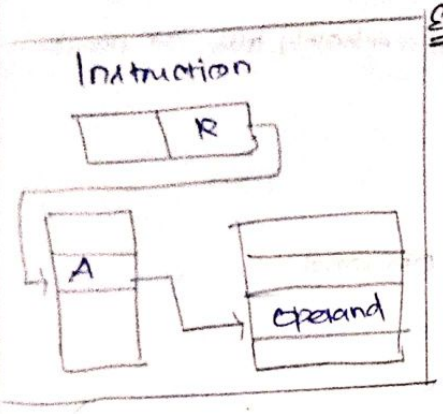
5) Indirect Addressing Modes.

i) Register Indirect Addressing modes.

The data (operand) will be placed in the memory location.

The address of memory will be placed in register.

The register will be part of instruction.

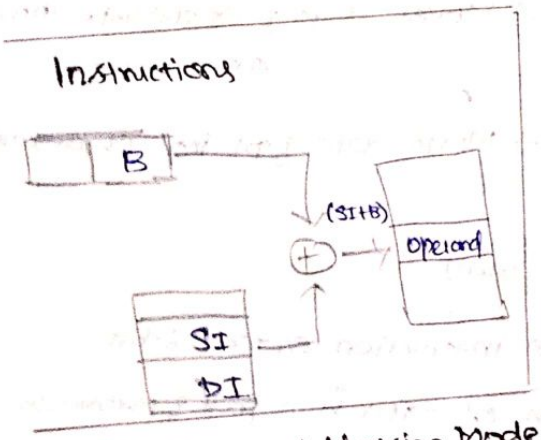


Eg MOV AX, [BX] → Moves the contents of memory location's address placed in Register BX to register AX

MOV [BP], CL → Moves a byte from CL into the location pointed by BP in stack segment

ii) Indexed Addressing Mode:

Operands address is the sum of the content in index register SI or DI, and an 8-bit or) 16-bit displacement.

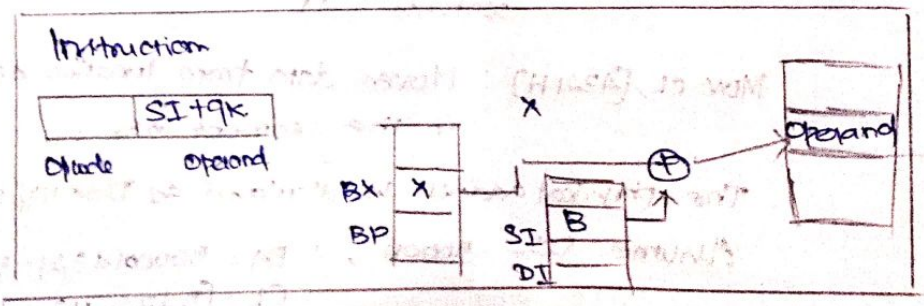


Eg

MOV AX, [SI+05]

iii) Base Addressing Mode

Effective address is obtained by adding base register value to index register SI or DI, (BX) or (BP). Eg. ADD AX, [SI + BX]



DATA TRANSFER INSTRUCTIONS

General-purpose byte or word transfer instruction:

MNEMONIC	DESCRIPTION
MOV	Copy byte or word from specified source to specified destination
PUSH	Copy specified word to top of stack.
POP	Copy word from top of stack to specified location.
PUSHA	(80186/80188 only) Copy all registers to stack.
POPA	(80186/80188 only) Copy words from stack to all registers.
XCHNG	Exchange bytes or exchange words.
XLAT	Translate a byte in AL using a table in memory

Simple input and output port transfer instructions:

IN	Copy a byte or word from specified port to accumulator.
OUT	Copy a byte or word from accumulator to specified port.

Special address transfer instructions:

LEA	Load effective address of operand into specified register
LDS	Load DS register and other specified register from memory.
LES	Load ES register and other specified register from memory.

Flag transfer instruction:

LAHF	Load (copy to) AH with the low byte of the flag register.
SAHF	Store (copy) AH register to low byte of flag register.
PUSHF	Copy flag register to top of stack.
POPF	Copy word at top of stack to flag register.

ARITHMETIC INSTRUCTIONS

Addition instructions:

ADD — Add specified byte or specified word to word.

ADC — Add byte + byte + carry flag or word + word + carry flag.

INC — Increment specified byte or specified word by 1.

AAA — ASCII adjust after addition.

DAA — Decimal (BCD) adjust after addition.

Subtract instructions:

SUB — Subtract byte from byte or word from word.

SBB — Subtract byte and carry flag from byte or word and carry flag word.

DEC — Decrement specified byte or specified word by 1.

NEG — (Negative) Negate - invert each bit of a specified byte or word and add 1 (from 2's complement)

CMP — Compare two specified bytes or 2 specified words.

AAS — ASCII adjust after subtraction.

DAS — Decimal (BCD) adjust after subtraction.

Multiplication instruction.

MUL — Multiply unsigned byte by byte or unsigned word by word

IMUL — Multiply signed byte by byte or signed word by word.

AAM — ASCII adjust after multiplication.

Division instructions:

DIV — Divide unsigned word by byte or unsigned double word by word.

IDIV — Divide signed word by byte or signed double word by word

AAD	ASCII adjust before division
CBW	Fill upper byte of word with copies of sign bit of lower byte.
CWD	Fill upper word of double word with sign bit of lower word.

BIT MANIPULATION INSTRUCTIONS

Logical Instructions

NOT	Invert each bit of a byte or word.
AND	AND each bit in a byte or word with the corresponding bit in another byte or word.
OR	OR each bit in a byte or word with the corresponding bit in another byte or word.
XOR	Exclusive OR each bit in a byte or word with the corresponding bit in another byte or word.
TEST	AND operands to update flags, but don't change operands.

Shift Instructions:

SHL/SAL	Shift bits of word or byte left, put zero(s) in LSB(s)
SHR	Shift bits of word or byte right, put zero(s) in MSB(s)
SAR	Shift bits of word or byte right, copy old MSB into new MSB

Rotate instructions:

ROL Rotate bits of byte or word left, MSB to LSB and to CF

ROR Rotate bits of byte or word right, LSB to MSB and to CF.

RCL Rotate bits of byte or word left, MSB to CF and CF to LSB.

RCR Rotate bits of byte or word right, LSB to CF and CF to MSB.

STRING INSTRUCTIONS

REP An instruction prefix, Repeat following instruction until CX=0.

REPE/REPZ An instruction prefix. Repeat instruction until CX=0 or zero flag ≠1.

REPNE/REPNZ An instruction prefix, Repeat until CX=0, ZF

MOVS/MOVSB/MOVSW Move byte or word from one string to oth

COMPS/COMPSB/COMPSW Compare two strings byte a two string wor

INS/INSB/INSW (80186/80188) Input strins byte or word from port.

OUTS/OUTSB/OUTSW Output string byte a word to port.

SCAS/SCASB/SCASW Scan a string, compare string byte with byt in AL or a string word with a word in AX.

LODS/LODSB/LODSW Load string byte into AL or string word into A

STOPS/STOSB/STOSW Store byte from AL or word from AX into string.

PROGRAM EXECUTION TRANSFER INSTRUCTIONS:

Unconditional transfer instruction:

CALL — call a procedure save a return address on stack.

RET — Return from procedure to calling program.

JMP — Go to specified address to get next instruction.

Conditional transfer instructions:

JC — jump if carry CF = 1

JE/JZ — jump if equal / jump if ZF = 1

JNC — Jump if no carry (CF = 0)

JNE/JNZ — jump if not equal / jump if not ZF = 0.

JNO — jump if no overflow flag = 0.

Iteration control instructions:

LOOP → loop through a sequence of instructions with CF = 0

LOOPE/LOOPZ — loop through a sequence of instructions while ZF = 1 & CF ≠ 0.

LOOPNE/LOOPNZ → while ZF = 0 & CX ≠ 0.

JCXZ → jump to specified address if CX = 0.

Interrupt instructions:

INT — Interrupt program instructions execution, call service procedure

INTO — Interrupt program execution if OF = 1

IRET — Return from interrupt service procedure to main program

PROCESSOR CONTROL INSTRUCTIONS:

Flag set / clear instructions:

STC — set CF to 1

CLC — set CF to 0

CMC — complement the state of CF

STD — set DF to 1 (decrement stack pointer)

CLD — clear DF to 0.

STI — set IF to 1 (enable INTR input)

CLI — set IF to 0 (disable INTR input)

External hardware synchronization instruction.

HLT — Halt (do nothing) until interuppt or reset

WAIT — Wait (do nothing) until signal on the TEST pin is low

ESC — Escape to external coprocessor such as 8087/89.

LOCK — prevents another processor from taking the bus while the adjacent instruction executes.

NOP — No operation instruction → No action except fetch and decode.