MODULE 2

INSTRUCTION SET & PROGRAMMING

2.1 Addressing mode

2.2 Instruction set

2.3 Assembler directives

MODULE 2

Definition :

- The manner in which an operand is given in an instruction.
- The different ways of specifying data to be operated by an instruction is known as addressing modes. This specifies that the given data is an immediate data or an address of data or data through register or register pair.

Example : ADD 25H to BL reg

Option 1 : ADD BL,25H

Option 2 : ADD BL,CL

Option 3 : ADD BL,[2000H]

Instruction, Register , Memory

ADDRESSING MODE

Data Addressing mode:

1. Immediate Addressing mode :
2. Register Addressing mode
3. Direct Addressing mode
4. Indirect Addressing mode
5. Implied Addressing mode

ADDRESSING MODE

1. Immediate Addressing mode :

- Operand is specified in the instruction itself.
- In this type of addressing mode the source operand is a 8 bit or 16 bit data.
- Destination operand can never be immediate data.
- Example: MOV CX, 2000H

  MOV CL, 0AH

  ADD CL, 45H

ADDRESSING MODE

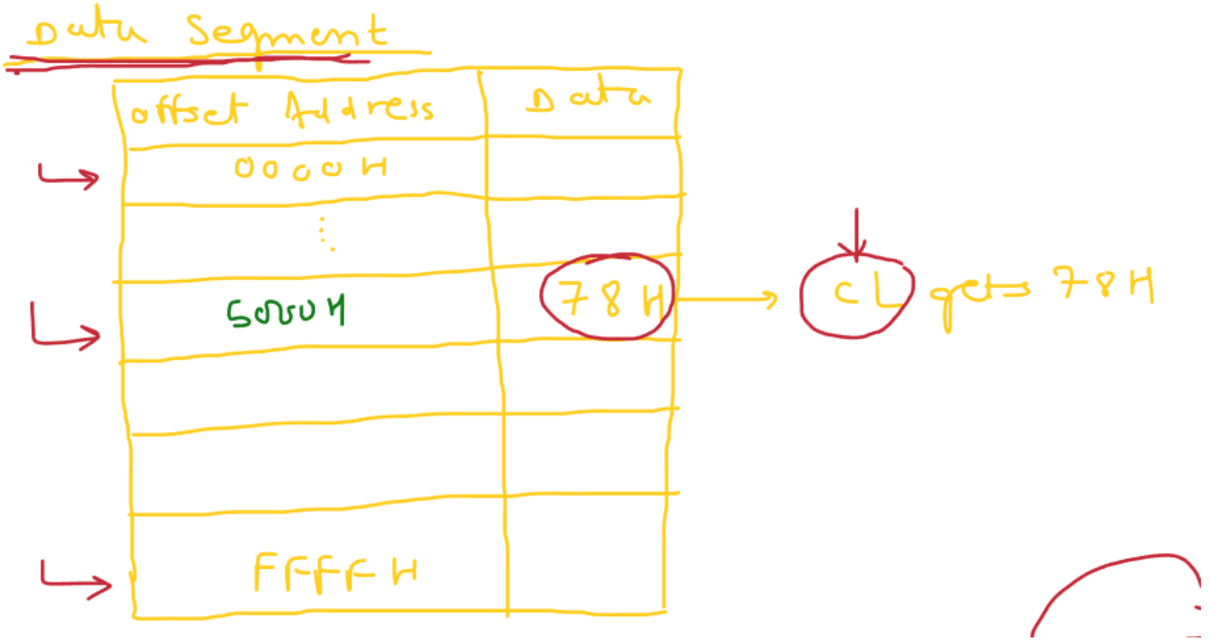
2. Register Addressing mode :

- In this type of addressing mode both the operands are registers i.e. operands are specified through register.
- Example: MOV CL, DL
  XOR AX, BX
  ADD AL, BL

ADDRESSING MODE

3. Direct mode :

- In this type of addressing mode the effective address of the operand is directly given in the instruction as displacement (Offset Address).
- Example 1: MOV CL, [2000H]
- Moves data from location 2000H in the data segment into CL.
- The physical address is calculated as DS*10H + 2000H (offset address)
- Assume DS=5000H
- PA=5000+2000=52000
- CL [52000H]
- Example 2 : MOV CX,[2000H]



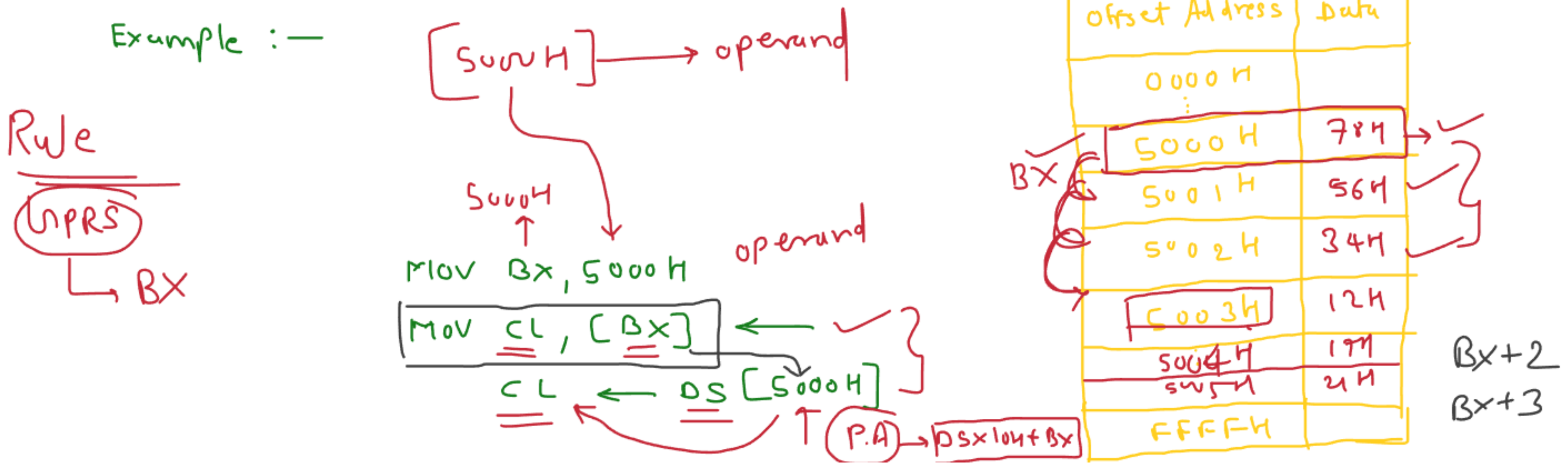
ADDRESSING MODE

4.Indirect Addressing mode:

I. Register Indirect Addressing mode:

1. Register Indirect Addressing Mode :

↳ Address of the operand is given using a register



Data Segment

Offset Address	Data
0000H	
5000H	78H
5001H	56H
5002H	34H
5003H	12H
5004H	19H
5005H	21H
FFFFH	

ADDRESSING MODE

4.Indirect Addressing mode:

II. Register Relative Addressing mode:

2. Register Relative Addressing mode :

↳ Here the address of the operand is given as sum of register & displacement.

Eg: MOV CL, [BX + 4] ✓

Relative → constant

Or

MOV CL, 04H [BX] ✓

ADDRESSING MODE

4.Indirect Addressing mode:

III. Base Indexed Addressing mode:

3. Base Indexed Addressing mode :

↳ only BX GPR's used for address along with offset address reg

SI DI

BX + SI ✓
BX + DI ✓
BP + SI ✓
BP + DI ✓

↳ BX, SI & DI will work for data segment & BP will work for stack segment.

↳ segment to operated by segment reg not by index reg.

Example :

mov CL, [BX + SI]; ✓ → Data segment → SI, DI

$PA = DS \times 10H + BX + SI$

mov CL, [BP + SI]; → Stack segment → SI

ADDRESSING MODE

4.Indirect Addressing mode:

IV. Base Relative plus Indexed Addressing mode:

4] Base relative Plus Indexed Addressing mode

Example : MOV CL, [BX + DI + 20] → move the byte address pointed by

↳ [Bx + DI + 20] → data segment

↳

MOV [BP + SI + 2000], CL → Stack segment

ADDRESSING MODE

4: Implied Addressing mode

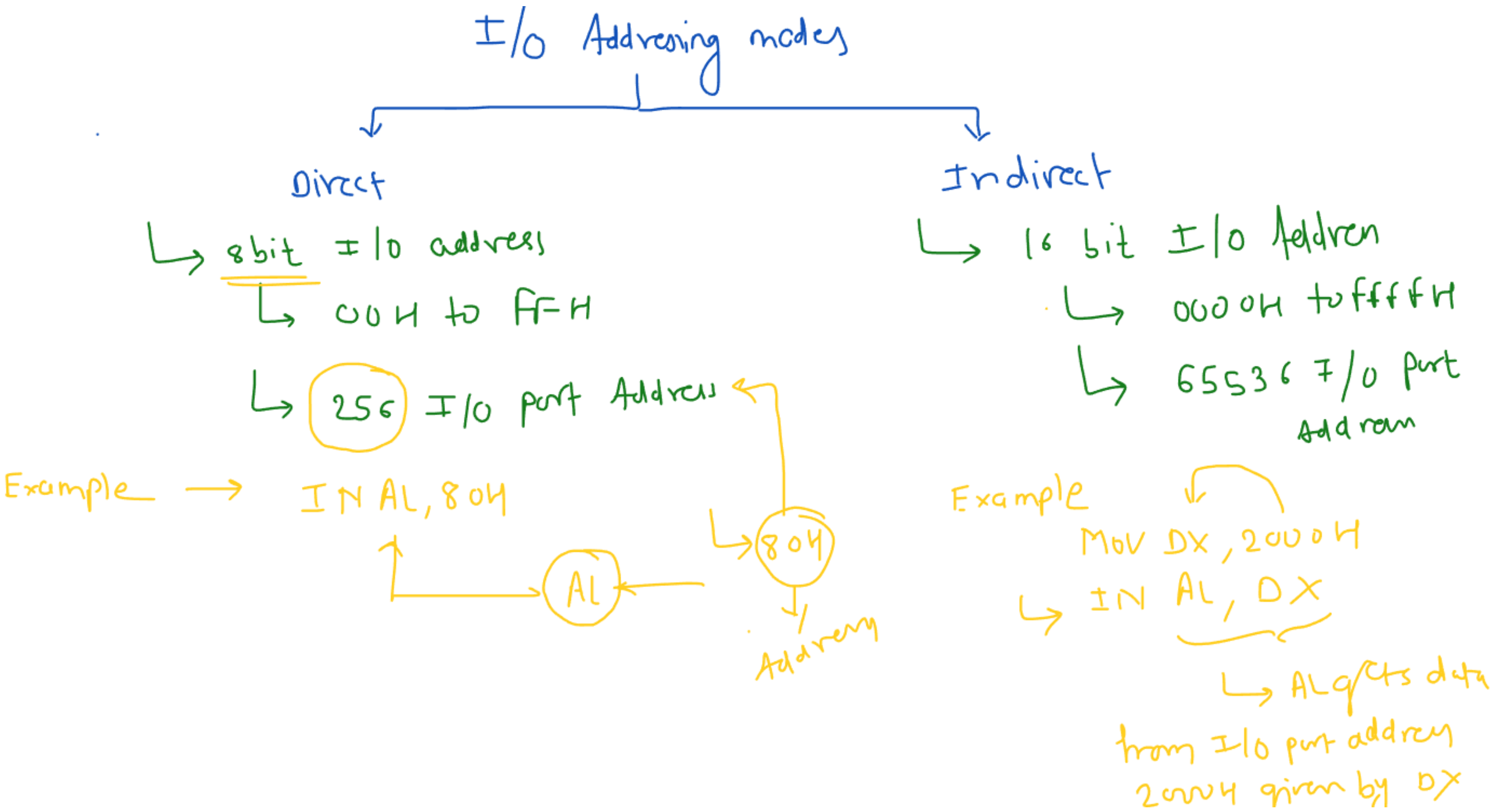
↳ operands are implied , not specified in instructions .

Example → STC → set the carry flag

CLD → Clears the direction flag .

ADDRESSING MODE

I/O Addressing mode:



ASSEMBLER DIRECTIVES

What is Assembler Directive ?

- Assembler directive is a statement/reserved-key-word to give direction to the assembler to perform task of the assembly process.
- They indicate how an operand or a section of the program is to be processed by the assembler
- It control the organization of the program and provide necessary information to the assembler to understand the assembly language programs to generate necessary machine codes
- An assembler supports directives to define data, to organize segments to control procedure, to define macros.

Assembler Directives

I. .MODEL : Defines memory model for the program
   I. SMALL : Separate segments for code and data fits in one 64KB
   II. MEDIUM : Multiple code greater than 64KB and Single data segment fits in one 64KB
   III. LARGE : Multiple code and Multiple data segments greater than 64KB

II. .DATA : To inform/ask the assembler to start/initialize the Data segment

III. .CODE : To inform/ask the assembler to start/initialize the Code segment

IV. .STACK : To inform/ask the assembler to start/initialize the Stack segment

V. DB :
   a. Define Byte
   b. Define a byte type (8-bit) variable
   c. Reserves specific amount of memory locations to each variable
   d. Example:
   e. LIST DB 7FH, 42H, 35H

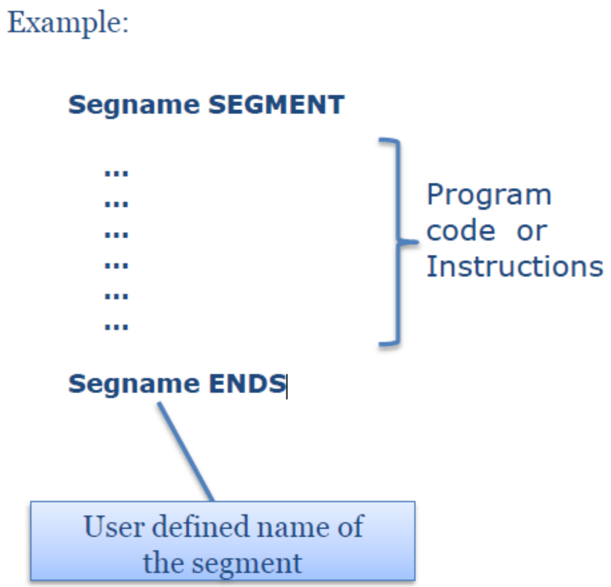
VI. DW :
   a. Define Word
   b. Define a word type (16-bit) variable
   c. Reserves two consecutive memory locations to each variable
   d. Example:
   e. ALIST DW 6512H, 7251H, 4DE2H
   f. Six consecutive memory locations are reserved for the variable ALIST and each 16-bit data specified in the statement is stored in two consecutive memory location.

ASSEMBLER DIRECTIVES

VII SEGMENT , ENDS

SEGMENT : Used to indicate the beginning of a code/ data/ stack/ extra segment

ENDS : Used to indicate the end of a code/



Example:

```
Segname SEGMENT
    ...
    ...
    ...
    ...
    ...
    ...
Segname ENDS
```

Program code or Instructions

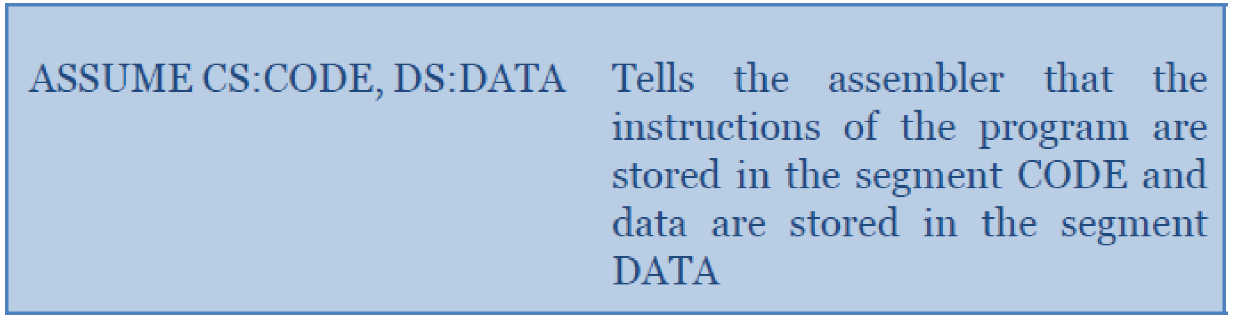
User defined name of the segment

ASSEMBLER DIRECTIVES

VII ASSUME

ASSUME : Informs the assembler the name of the program/ data segment that should be used for a specific segment.

Example:



ASSUME CS:CODE, DS:DATA	Tells the assembler that the instructions of the program are stored in the segment CODE and data are stored in the segment DATA

ASSEMBLER DIRECTIVES

VIII ORG : ORG (Origin) is used to assign the starting address (Effective address) for a program/ data in the segment

ORG 1000H : Informs the assembler that the statements following ORG 1000H should be stored in memory starting with effective address 1000H

IX END : END is used to terminate a program; statements after END will be ignored

X EVEN : Informs the assembler to store program/ data in the segment starting from an even address

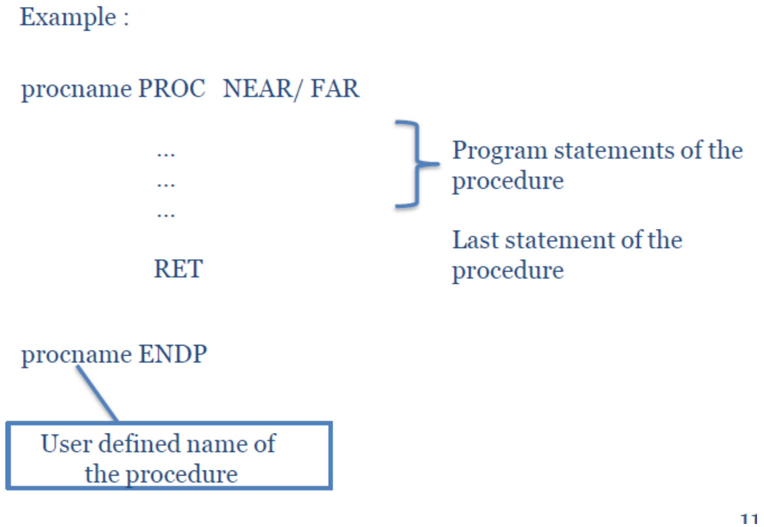
XI PROC : Indicates the beginning of a procedure

XII ENDP : End of procedure

XIII FAR : Intersegment call

XIV NEAR :Intrasegment call

XV AT : Instruct the assembler to start the segment at t



Example :

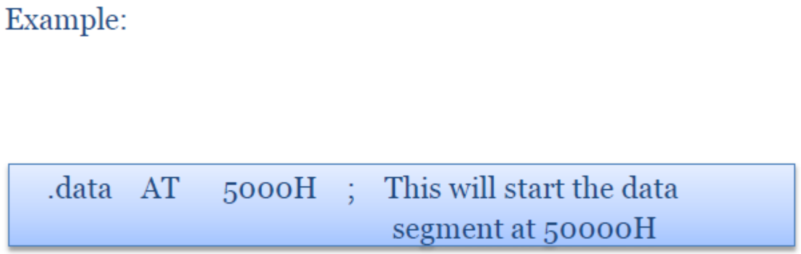
```
procname PROC   NEAR/ FAR

            ...
            ...
            ...

            RET

procname ENDP
```

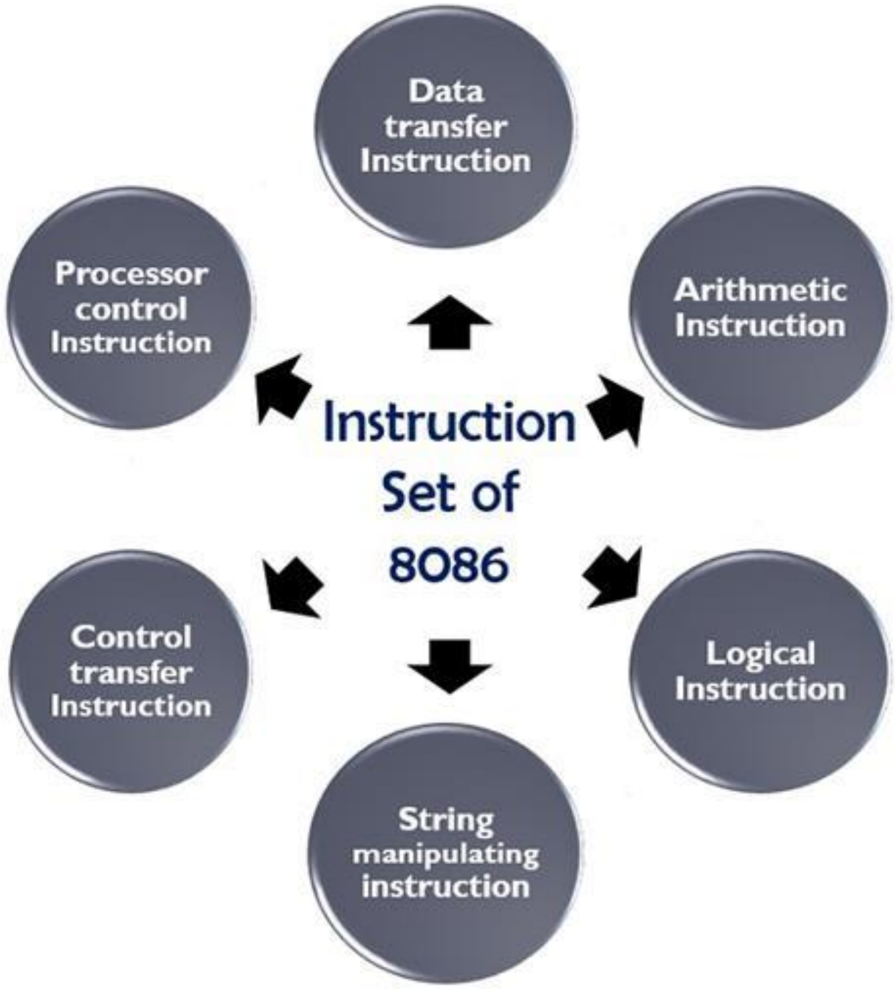
Program statements of the procedure
Last statement of the procedure
User defined name of the procedure
11



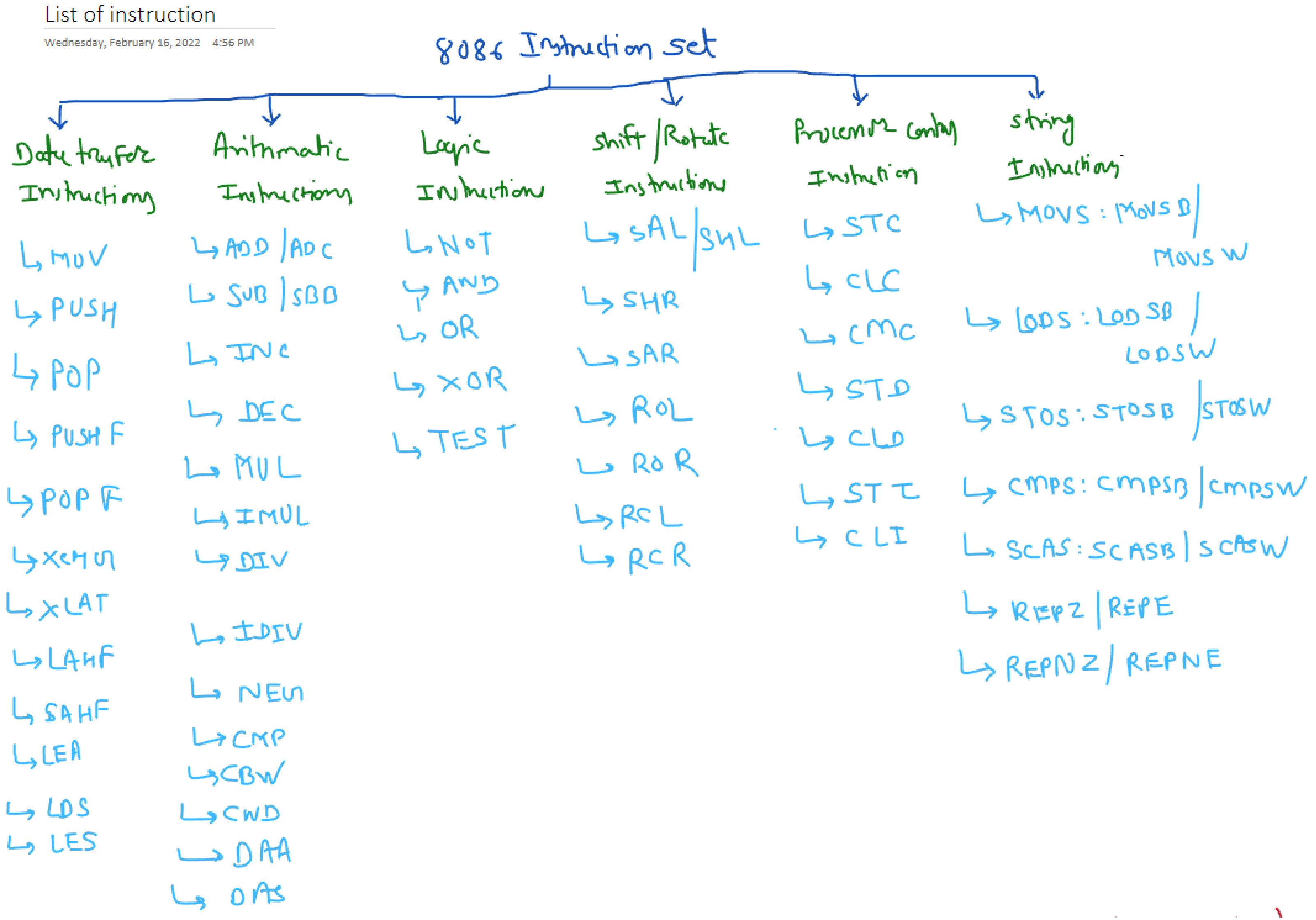
Example:

```
.data   AT    5000H   ;   This will start the data
                          segment at 50000H
```

8086 INSTRUCTION SET


Data transfer Instruction
Arithmetic Instruction
Logical Instruction
String manipulating instruction
Control transfer Instruction
Processor control Instruction
Instruction Set of 8086

8086 INSTRUCTION SET



List of instruction

Wednesday, February 16, 2022 4:56 PM

8086 Instruction set

Data trafer Instructions
- MOV
- PUSH
- POP
- PUSH F
- POP F
- XCHG
- XLAT
- LAHF
- SAHF
- LEA
- LDS
- LES

Arithmatic Instructions
- ADD / ADC
- SUB / SBB
- INC
- DEC
- MUL
- IMUL
- DIV
- IDIV
- NEG
- CMP
- CBW
- CWD
- DAA
- DAS

Logic Instruction
- NOT
- AND
- OR
- XOR
- TEST

Shift / Rotate Instruction
- SAL / SHL
- SHR
- SAR
- ROL
- ROR
- RCL
- RCR

Processor control Instruction
- STC
- CLC
- CMC
- STD
- CLD
- STI
- CLI

String Instructions
- MOVS : MOVSB / MOVSW
- LODS : LODSB / LODSW
- STOS : STOSB / STOSW
- CMPS : CMPSB / CMPSW
- SCAS : SCASB / SCASW
- REPZ / REPE
- REPNZ / REPNE

8086 INSTRUCTION SET

Data Transfer Instructions

OPCODE (MNEMONIC)	OPERAND	EXPLANATION	EXAMPLE
MOV	D, S	D = S	MOV AX, CX
PUSH	S	pushes S to the stack	PUSH DX
POP	D	pops the stack to D	POP AX
PUSHF	none	put flag registers into the stack	PUSHF
POPF	none	gets words from the stack to flag register	POPF
XCHG	D, S	exchanges contents of D and S	XCHG DX, AX
IN	D, S	copies a byte or word from S to D	IN AX, DX
OUT	D, S	copies a byte or word from S to D	OUT 05H, AL

8086 INSTRUCTION SET

Example :

MOV AL, CH ; AL ← CH

MOV BL, [SI] ; BL ← DS:[SI]

PUSH AX ; SS:[SP-1] ← AH & SS:[SP-2] ← AL

POP DX ; DL ← SS:[SP] & DH ← SS:[SP+1], [SP]=[SP-2]

IN AL, 80H ; AL ← [80H]

IN AX, 80H ; AL ← [80H] & AH ← [81H]

IN AX, DX ; AL ← [[DX]] & AH ← [[DX]+1]

OUT 80H, AL ; [80H] ← AL

OUT 80H, AX ; [80H] ← AL & [81H] ← AH

OUT DX, AX ; [[DX]] ← AL & [[DX]+1] ← AH

8086 INSTRUCTION SET

Data Transfer Instructions

OPCODE	OPERAND	EXPLANATION	EXAMPLE
XLAT	none	AL=DS:[BX+AL]	XLAT
LAHF	none	loads AH with the lower byte of the flag register	LAHF
SAHF	none	stores AH register to lower byte of the flag register	SAHF
LEA	D, S	loads effective address of source to the destination	LEA BX, [SI+66H]
LDS	D, S	Loads destination as well as DS with 4-bytes pointed by source	LDS SI, [DI]
LES	D, S	Loads destination as well as ES with 4-bytes pointed by source	LES BX, [DI]

8086 INSTRUCTION SET

Arithmetic Instructions

OPCODE	OPERAND	EXPLANATION	EXAMPLE
ADD	D, S	D = D + S	ADD DH, CL
ADC	D, S	D = D + S + prev. carry	ADC AX, BX
SUB	D, S	D = D – S	SUB AX, BX
SBB	D, S	D = D – S – prev. carry	SBB DX, BX
CMP	D, S	compares D & S, affects the flag register only	CMP AL, DL
MUL	8-bit register	AX = AL * 8-bit reg.	MUL BH
MUL	16-bit register	DX:AX = AX * 16-bit reg.	MUL CX
IMUL	8 or 16 bit register	performs signed multiplication	IMUL CX

8086 INSTRUCTION SET

Arithmetic Instructions

OPCODE	OPERAND	EXPLANATION	EXAMPLE
DIV	8-bit register	AX = AX / 8-bit reg. ; AL = quotient ; AH = remainder	DIV BL
DIV	16-bit register	DX:AX / 16-bit reg. ; AX = quotient ; DX = remainder	DIV CX
IDIV	8 or 16 bit register	performs signed division	IDIV BL
INC	D	D = D + 1	INC AX
DEC	D	D = D – 1	DEC BL
CBW	none	converts byte in AL to a word in AX by sign extension	CBW
CWD	none	converts word in AX to a double word in DX:AX by sign extension	CWD

8086 INSTRUCTION SET

Arithmetic Instructions

OPCODE	OPERAND	EXPLANATION	EXAMPLE
NEG	D	D = 2's compliment of D	NEG AL
DAA	none	decimal adjust accumulator after addition	DAA
DAS	none	decimal adjust accumulator after subtraction	DAS
AAA	none	ASCII adjust accumulator after addition	AAA
AAS	none	ASCII adjust accumulator after subtraction	AAS
AAM	none	ASCII adjust accumulator after multiplication	AAM
AAD	none	ASCII adjust accumulator before division	AAD

8086 INSTRUCTION SET

Logic Instructions

OPCODE	OPERAND	EXPLANATION	EXAMPLE
AND	D, S	D = D AND S	AND AX, CX
OR	D, S	D = D OR S	OR AX, BX
NOT	D	D = NOT of D	NOT AL
XOR	D, S	D = D XOR S	XOR AL, BL
TEST	D, S	performs bit-wise AND operation and affects the flag register only	TEST AX, 8000H
SHR	D, 1 or CL	shifts each bit in D to the right by 1 or CL times and 0 is stored at MSB position	SHR AL, 01H
SHL/SAL	D, 1 or CL	shifts each bit in D to the left by 1 or CL times and 0 is stored at LSB position	SHL AX, CL

8086 INSTRUCTION SET

Logic Instructions

OPCODE	OPERAND	EXPLANATION	EXAMPLE
SAR	D, 1 or CL	shifts each bit in D to the right by 1 or CL times and copies current MSB bit to lower MSBs	SAR AL, 01H
ROR	D, 1 or CL	rotates all bits in D to the right by 1 or CL times	ROR BL, CL
ROL	D, 1 or CL	rotates all bits in D to the left by 1 or CL times	ROL BX, 01H
RCR	D, 1 or CL	rotates all bits in D to the right along with carry flag by 1 or CL times	RCR BL, CL
RCL	D, 1 or CL	rotates all bits in D to the left along with carry flag by 1 or CL times	RCL BX, 01H