Coal :

Computer Organization & Assembly Language.

Registers :

Data Register - Hold data for an operation

Address Register - Hold address of an instruction.

Status Register - Keeps the current status of processor.

four General Data Registers (Address Registers are divided into segment)

↳ Pointer Registers

↳ Index Registers.

Status Register is called FLAGS Registers.

Intotal, 14 16 bits registers.

Data Registers : AX, BX, CX, DX.

four registers for general data manipulation.

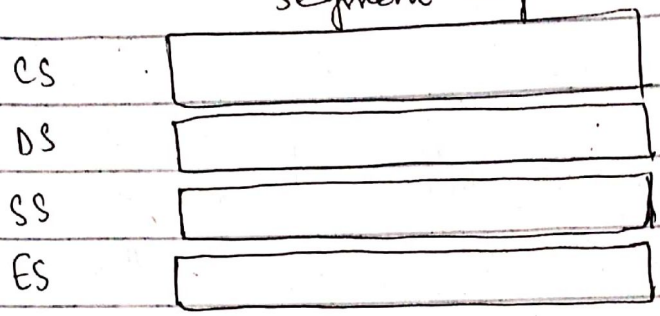
- Processor can operate on data stored in memory, the same instruction is faster if data are stored in registers.
- High bytes of AX is AH, low byte is AL.

1997
2023
26.

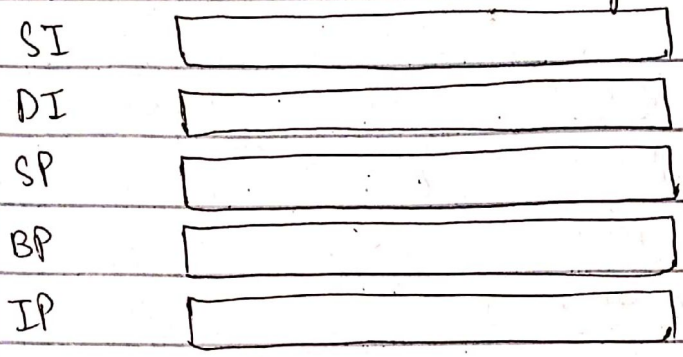
Data Registers.

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

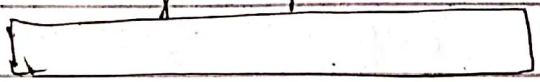
Segment Registers.



Pointer and Index Registers:



flags Register



- Input and Output requires the use of AL & AX.
- BX is also an address register.
- CX is count register. (loop counter).
  CX as repeat controls string operations,
  CL as count in instructions as shift & rotate bits.
- DX is used in multiplication & division. used in I/O operations.

Memory Segment:

Block of 2^{16} consecutive memory bytes.
Each segment is identified by segment number.
Segment Number is 16 bits, highest s.n is FFFFh
memory location is specified by giving an offset.

16 | 692706
16 | 43294 - 2
16 | 2705 - 14
16 | 169 - 1
10 - 9
A91E2

Memory location = Segment : offset.
known as logical address.

⇒ for example, A4FB : 4872h
means offset 4872h, segment A4FBh.

⇒ 20 bit Physical Address.

- first shifts the segment Address 4 bits to left.
- Adds offset.

```
  A 4 F B 0 h
+   4 8 7 2 h
-------------
  A 9 8 2 2 h
```

0111 — 7
1000 — 8
1001 — 9
1010 — 6
11100 — 11
0111
10011

— A4FBh = $(A * 16^3) + (4 * 16^2) + (F \times 16^1) + (B \times 16^0)$
$= (10 \times 4096) + (4 \times 256) + (15 \times 16) + (11 \times 1)$.
= 42135 (decimal)

— 4872h = $(4 * 16^3) + (8 * 16^2) + (7 * 16^1) + (2 * 16^0)$
= 18546 (decimal)

20-bit Physical Address = (Segment Address << 4) + Offset Address).
= (42135 << 4) + 18546.

- Shifting left by 4 bits is equal to multiply by 16.

42135 * 16 = 674160.

Physical Address = 674160 + 18546
= 692706 (decimal).

674160
18546
06

674160
+18546
825796

674160
+ 18546
692706

Example 3.1

Physical Address = 1256AH

segment : offset ? 1256h and 1240h are the segments.

let X be the offset in segment 1256h

Y be the offset in segment 1240h.

Physical Address = Segment + offset.

1256AH = 1256h + X — (1).

1256AH = 1240h + Y — (2).

for equ(1). 1256Ah = 1256 : 000Ah

X = 1256Ah - 1256h

= Ah.

```
  1256Ah
  1256 0h
  ------
  0000Ah
```

for equ(2). 1256Ah = 1240 : 016Ah

Y = 1256Ah - 1240h

= 16Ah.

```
  1256Ah
  1240 0h
  -------
  0016Ah
```

Example 3.2

Physical Address = 80FD2h.

Segment = ? offset = BFD2h.

Physical Address = Segment x 10h + offset.

Segment x 10h = Physical Address - offset.

= 80FD2h - BFD2h

= 74000h.

Segment = 74000h / 10 = 7400h.

```
  80FD2h
- BFD2h
  ------
  74000h
```

```
  0000
- 1100
  ----
  0100
```

(4)

Sequential Processing:

Completion of 1 instruction will be in 4 cycles.

- Fetch
- Decode
- Execute
- Output

Q. How many instructions will be executed in 1 sec?

$f = 1\,Khz$

$= \frac{1000}{4} = 250$ instruction/second.

Q. frequencies:

$T = \frac{1}{f}$

$= \frac{1}{100\,000} = 0.000001$

$= 1 \times 10^{-6} = 1\,\mu sec.$

formulas:

→ Signed : $\{-(2)^{n-1} \rightarrow +(2)^{n-1} - 1\}$

→ Unsigned: $\{0 \rightarrow 2^n - 1\}$

- flip flops : 1 bit
- Registers : 8 bit

Range:

8-bit : Unsigned → 0-255

Signed → -128 → +127

4 bit : Unsigned → 0-15

AX = | 15 | | | | | | | | 0 |

AH AL.

Index & Pointer Register : 16 bits.
(SI, DI, IP, SP, BP).

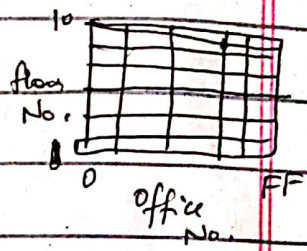
format :

{ } MOV AL, SS.

label opcode dest, source

Segment Address → floor No.

Offset Address → Office No.



Associators :

- DS → SI, DI } offset Address.
- CS → IP } offset Address
- SS → SP

→ Limit of Segment : $64K = 2^{16}$.

→ Physical Address = Seg × 10 + offset.

Calculations :

MOV AL, 12

MOV BL, 14

SUB AL, BL.

AL = -2

2's complement.

```
00010010
00010100
--------
00000110

00000010
11111101
+       1
--------
11111110
F   E   <- Result.
```

NV — Overflow flag
Carry Flag
Sign Flag
Zero Flag
} Reset State by Default. (1).

AL, BL — 8 bits register — so MOV AL, BL.

MOV AL, 30 → Initialization of 30 in AL.

MOV AL, BX (X) Not possible : AL: 8 bit, BX: 16 bits.
operands don't match.

MOV AL, Bh → Both are 8 bits so valid

MOV Bl, [SI] → valid.

If SI = 0000
runs :
BL = 41.

1234:0002	61
1234:0001	45
1234:0000	41

MOV ah, 01 → Input.
INT 21h → Run.
SUB AL, 20 → Change.
MOV ah, 02 → print

a = 61
A = 41
} ASCII

Q. Sum (Word Type).

```
mov ax, 1234
mov ds, ax.
mov si, 0100
mov bx, 0000
```

ax

SI : [3020, 1015].

3020
1015
4035

```
mov ax,[SI]   ;   ax = 3020
add bx,ax     ;   bx = 3020.
~~mov ax,fs~~
inc SI.
mov ax,[si]   ;  ax = 1015
add bx,ax     ;  bx = 4035
```

Q- Print Name & Roll No:

```
e 0000 "String xy2$"
mov dx,0.
mov ah,09
int 21h
```

JMP: zero flag → 0 (Set)
→ 1 (Reset).

JMP → JZ
→ JNZ

Q- Declare 2 Arrays of 10 members in memory and copy their respective sum in array 3.

```
e 0000 41 42 43 44 45 46 47 48 49 50
e 0001 01 02 03 04 05 06 07 08 09 01
mov cx,10
0001 ← mov al,[si].
mov dl,al.
inc si
inc di
dec cx
jnz 0001
```

a 0020.

```
mov ax, e 0000
mov ds, ax.
mov si, 0000
mov di, 0020
mov bx, 6.
again: mov al, [si]
mov bl, [di]
add al, bl
mov [bx], al.
inc si
inc bx
inc di
dec cx.
jnz again
```

```
.model small
. stack 100h
. code
main proc
mov ax, @data
mov ds, ax
mov si, offset array1
mov di, offset array2
mov cx, 8
again: mov bl, [si]
mov [di], bl
inc si
inc di
dec cx
jnz again
mov ah, 4ch
int 21h
main endp
.data
array1 db 01, 02, 03 ... 08
array2 db 11, 12, 13 ... 18
end main
```

16 | 115360
16 | 7210 — 0
16 | 450 — 10(A
16 | 28 — 2
16 | 1 — 12(C

Q- Special functions :

AX - Arithmetic & data manipulation, Storing results of mathematical calculations.

BX - Addressing memory location, Access data in memory.

CX - Loop counter, Repetitions in program.

DX - Various Data operations, Relates to I/O operation. Store port addresses for input & output.

Q- Registers — Memory location.

Location	
Small, high-speed storage locations, located in CPU.	Part of computer's RAM, used to store larger data.
Speed	
fastest / Quick (AX, BX, CX, DX).	comparatively slower.
Number.	
limited & fixed.	limitless.
Size.	
Small in size. (8 bit, 16, 32, 64)	larger. (bytes, gb, ...).
Usage.	
manipulation	Storing Program.

Q- Physical Address = ? OA51 : CD90h.

Physical Address = (segment * 16) + offset.

OA51 = 3921 (decimal)

CD90 = 52.624 (decimal).

PA = (3921 * 16) + 52624.

= 115360 = 1C2A0

Q- Physical Address - 4A37Bh.

i) offset address if segment number is 40FFh

PA = (segment * 16) + offset.

PA - (segment * 16) = offset.

4A37Bh - (40FFh * 16) = offset.

offset =

ii)

Chapter # 04

DB - define bytes. (1).

DW - define Word (2).

DD - doubleword (Two) → Notsure

DQ - Quadword (4).

DT - Tenbytes (10).

→ offset address : 0200h.

b_array db 10h, 20h, 30h.

Symbol	Address	Contents
b_array	200h	10h
b_array+1	201h	20h
b_array+2	202h	30h.

offset : 0300h

w_array dw 1000, 40, 29887, 329

Symbol	Address	Contents
w_array	0300h	1000d.
w_array+2	0302h	0040d.
w_array+4	0304h	29887d
w_array+6	0306h	0329d.

→ high & low bytes of word.

word1 dw 1234h.

low bytes contains 34h, high contains 12h.

low → word1 , high → word1+1

word1	0300h	34h
word1+1	0301h	12h

→ legal combinations mov ax, ——
dest. source.

MOV:

Source	Destination: General Register	Segment Register	memory location	Constant
General Register	yes	yes	yes	no.
Segment Register	yes	no	yes	no.
memory location	yes	yes	no	no
constant	yes	no	yes	no.

XCHG:

xchg dest., source
ah, bl.

Source	Destination: General Register	memory location
General Register	yes	yes
memory location	yes	no.

⇒ Byte variables:

name db 'initial value → alpha db 4

decimal range of initial values that can be specified is -128 to 127 if signed. For unsigned, it ranges from 0 to 255.

⇒ word variables:

name dw 'initial value ⇒ wrd dw -2.

Signed Interpretation: -32768 to 32767

Unsigned Interpretation: 0 to 65535

⇒ Arrays:

for bytes: name db initial values ⇒ b_array db 10h, 20h

Offset Address = 0200h

b_array	200h	10h
b_array+1	201h	20h

for word name dw initial values ⇒ w_array dw 1000, 40, 329

w_array	0300h	1000d
w_array+2	0302h	0040d
w_array+4	0304h	0329d

⇒ low & high bytes:

word1 dw 1234h.

low byte = 34h, high byte = 12h.

low byte symbolic address WORD1, high byte address word1+1.

⇒ Character Strings:

letters db 'ABC' equivalent to letters db 41h, 42h, 43h

letters db 'abc' equivalent to letters db 61h, 62h, 63h.

OAH - line feed , ODH - carriage return.

- OAH - line feed (LF):

OAH in hex , 10 in decimal.

causes the text cursor to move down to next line, often without any horizontal movement (beginning of next line). Used to start a new line of text.

- ODH - Carriage Return (CR):

ODH in hex, 13 in decimal

causes the text cursor do move to the beginning of current line. Used to return the leftmost position on current line.

Example:

msg db 'Hello', OAH, ODH, '$'

Output : Hello
$

msg db 'Hello', OAH, '$'

Output : Hello
$

msg db 'Hello', ODH, '$'

Output : $Hello.

⇒ ADD, SUB, INC, DEC.

add or subtract are used for content of registers.

ADD / SUB destination, source.

legal combination

Source operands	Destination operands	
	General Register	Memory location
General registers	Yes	yes.
memory location	yes	no
constant	yes	yes

Byte2 byte1
↳AL

Illegal: ADD Byte1, Byte2.

Solution: MOV AL, Byte2 ADD Byte1, AL.

INC destination , DEC destination

adds 1 decreases 1.

⇒ NEG.

to negate the contents of destination. Does this by replacing the contents by two's complement.

NEG destination NEG BX.

1000
1001
1010
1011
1100
0010 1101
1101 1110
+ 1
1111 → FFFE.

Example: 0002 FFFE

Examples: BX (before) BX (After).

Statement	Codes.	
B = A	MOV AX, A	MOV B, AX.
A = 5 - A	MOV AX, 5	SUB AX, A MOV A, AX.
	or NEG A ; A = -A	ADD A, 5.
A = B - 2 x A	MOV AX, B	SUB AX, A SUB AX, A MOV A, AX.
= B - 2A.		

B A
AX

⇒ Memory Models.

• model memory_mode.

memory mode can be small, medium, compact, large.

⇒ Syntax.

```
•model small
•stack 100h
•data
=
•code
main proc
=
main endp
=
end main
```

⇒ INT interrupt number ⇒ INT 21H.

INT 16H — BIOS routine, INT 21H — DOS routine.

function number	Routine.
1 (read into AL)	single key input
2 (DL)	single character output
9 (DX). string with $.	character string output.

ASCII code (Hex)	Symbol	function
7	Bel	beep (sound)
8	bs	backspace
9	ht	tab
A	lf	new line (line feed)
D	cr	carriage return (start of line).

Example:

Read a character from keyboard and display it at the beginning of the next line.

```
.model small
.stack 100h
.code
main proc
; display prompt
    MOV AH, 02
    MOV DL, '?'      } LEA DX, MSG
    INT 21H          } MOV AH, 9
                     } INT 21H
; input
    MOV AH, 01
    INT 21H
    MOV BL, AL
; new line
    MOV AH, 02
    MOV DL, 0DH
    INT 21H
    MOV DL, 0AH
    INT 21H
; display character
    MOV DL, BL
    INT 21H
; return to DOS
    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN.
```

Example:

Print string program.

```
.model small
.stack 100h
.data
  msg db 'hello!$'
.code
MAIN PROC
  MOV AX, @data ; initialize
  MOV DS, AX
  ; display
  LEA DX, MSG
  MOV AH, 09
  INT 21H
  ; return to DOS
  MOV AH, 4CH
  INT 21H
MAIN ENDP
END MAIN.
```

Output:

hello!

Example: Case conversion

Program (lower to Uppercase)

{ lowercase starts from 61h & uppercase starts from 41h so subtract 20h }

```
.model small
.stack 100h
.data
    CR EQU 0DH
    LF EQU 0AH
msg1 db 'enter a lowcase letter: $'
msg2 db 0DH,0AH,'In uppercase: $'
char db ?,'$'
.code
MAIN PROC
; initialize
    MOV AX,@data
    MOV DS,AX
; print user prompt
    LEA DX,msg1
    MOV AH,09
    INT 21H
; input & convert
    MOV AH,01
    INT 21H
    SUB AL,20H
    MOV CHAR,AL
; display on next line
    LEA DX,msg2
    MOV AH,09
    INT 21H
; DOS exit
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN.
```

Task 1:

lowercase to uppercase. input from user.

```
mov ah,01
int 21h
sub al,20h
mov dl,al
mov ah,02
int 21h.
```

Task 2:

Uppercase to lower. input from user.

```
mov ah,01
int 21h
add al,20h
mov dl,al
mov ah,02
int 21h.
```

chapter# 05

The Processor Status and The FLAGS Register

- The status flags are located in bits 0, 2, 4, 6, 7, 11.
- Control flags are located in bits 8, 9, 10.
- Other bits have no significance.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

⇒ The Status flags:

Reflect the result of an operation.

Example: SUB AX, AX is executed, Zero flag becomes 1.

• carry flag (CF) • Parity flag (PF) ....

Status flags

Bit	Name	Symbol
0	Carry flag	CF
2	Parity flag	PF
4	Auxiliary Carry flag	AF
6	Zero flag	ZF
7	Sign flag	SF
11	Overflow flag	OF

- Carry flag: CF=1 if there is carry out from the most significant bit (msb) on addition or there is a borrow into msb on subtraction. Otherwise, it is 0. Effected by SHL and RO instructions.
- Parity flag: PF=1 if low byte of result has an even number of one bits. It is 0 if low byte has odd parity. for example: if result of word addition is FFFEh, then low byte contains 7 one bits, so PF=0.

⇒ Control flags.

Bit	Name	Symbol.
8	Trap flag	TF
9	Interrupt flag	IF
10	Direction flag	DF

Continued Status flag:

- Auxiliary carry flag (AF): AF = 1 if there is a carry out from bit 3 on addition, or a borrow into bit 3 on subtraction. Used in BCD.
- Zero flag (ZF): ZF = 1 for zero result and ZF = 0 for nonzero result.
- Sign flag (SF): SF = 1 if msb is 1, means result is negative. SF = 0 if msb is 0.
- Overflow flag (OF): OF = 1 if signed overflow occured, otherwise it is 0.

⇒ Overflow:

Range of signed numbers:

16-bit word: -32768 to 32767

8-bit byte: -128 to 127.

Range of unsigned numbers:

Word : 0 to 65535

Byte : 0 to 255

Four Possible outcomes after arthemetic operations:

- No overflow - Signed overflow only - Unsigned overflow only - Both signed and unsigned overflow.

Example:

AX : FFFFh, BX : 0001h, ADD AX, BX.

```
  1111  1111  1111  1111
+ 0000  0000  0000  0001
-------------------------
1 0000  0000  0000  0000  ⇒ 10000h
```

10000h = 65536 (Out of range). 1 is carried out of msb, AX stores 0000h (which is wrong) so unsigned overflow occured.

Example:

AX = BX = 7FFFh , ADD AX, BX

0111 1111 1111 1111

+ 0111 1111 1111 1111

1111 1111 1111 1110 = FFFEh.

32767 = 7FFFh ; 7fffh + 7fffh = 32767 + 32767 = 65534 (out of range). Stored answer is -2. So signed overflow occured.

⇒ Unsigned Overflow: On addition, when carry out of msb. which means the correct answer is larger than the biggest unsigned ~~overflow~~ number. FFFFh for word, ffh for byte. On subtraction, it occurs when there is borrow into msb. This means answer is smaller than 0.

⇒ Signed overflow: On addition of same sign, it occurs when sum has a different sign.

Example: Add 7FFFh & 7FFFh (+ 0111 1111 1111 1111) got FFFEh (- 1111 1111 1111 1110) a negative result.

Subtracting of different sign numbers is like addition with same sign (A - (-B)) = A + B. So, it occurs when result has different sign than expected.

Example S.1 ADD AX, BX , AX - FFFFh , BX = FFFFh.

FFFF h	SF (Sign Flag) = 1 · msb is 1
FFFF h	PF = 0 (7) odd numbers.
(1)FFFE h	ZF = 0 result is non zero.
1 1111 1111 1111 1110	OF = 1 carry out of msb.
	OF = 0 (sign of stored is same as addition).

PF [odd - 0
even - 1

Example: ADD AL, BL AL contains 80h, BL contains 80h.

```
  1000 0000
+ 1000 0000
 10000 0000
```

SF = 0
PF = 1
ZF = 1 (result is 0).
CF = 1
OF = 1

100h., AL stores 00h (byte).

Example: SUB AX, BX AX - 8000h, BX - 0001h.

```
  1000 0000 0000 0000
- 0000 0000 0000 0001
  1111 1111 1111 1111
```

SF = 0
PF = 1.
ZF = 0
CF = 0
OF = 1

7FFFh.

Example: INC AL AL contains FFh.

```
  1111 1111
+ 0000 0001
 10000 0000
```

SF = 0
PF = 1
ZF = 1
CF = 0
OF = 0

100h → Stores 00h.

Example: MOV AX, -5 None of the flags are affected by MOV.

```
  0000 0000 0000 0101
  1111 1111 1111 1010
+                  1
  1111 1111 1111 1011
```

→ FFFBh.

```
 .0100
  0101
  1010
+    1
 .1011
```

Example: NEG AX , AX = 8000h.

```
  1000 0000 0000 0000
  0111 1111 1111 1111
+                  1
 1000 0000 0000 0000
```

→ 8000h.

SF = 1, PF = 1, ZF = 0 OF = 1,
CF = 1 (because for NG, CF is always 1 unless result is 0).

Status Flag	Set (1) symbol	Set (0) Symbol.
CF	CY (carry)	NC (no carry)
PF	PE (even parity)	PO (odd parity)
AF	AC (auxiliary carry)	NA (no auxiliary carry)
ZF	ZR (zero)	NZ (nonzero).
SF	NG (negative).	PL (Plus).
OF	OV (overflow)	NV (no overflow).

chapter #06 Flow Control Instructions

Conditional Jumps.

Program: Display 256 IBM characters.

```
.model small  .stack 100h  .code
MAIN PROC  MOV AH,02  MOV CX,256  MOV DL,0.
PRINT_LOOP: INT 21H  INC DL  DEC CX  JNZ PRINT_loop
MOV AH,4CH  INT 21H  MAIN ENDP  END MAIN
```

⇒ Conditional loops:

J*** destination.

⇒ Compare Instruction:

CMP destination, source.

(Result is not stored but flags are effected).

Example:

CMP AX, BX AX = 7FFFh, BX = 0001h.

JG Below.

CMP: 7FFFh − 0001h = 7FFEh So, AX > BX so jg satisfied. ZF = SF = OF = 0.

⇒ Signed Jumps:

Symbol	conditions
JG/JNLE	ZF=0, SF=OF
JGE/JNL	SF=OF
JL/JNGE	SF<>OF
JLE/JNG	ZF=1, SF<>OF

⇒ Unsigned Jumps:

JA/JNBE (above)	CF=0, ZF=0
JAE/JNB (equal, below)	CF=0
JB/JNAE	CF=1
JBE/JNA	CF=1, or ZF=1.

~~② Single Flag~~

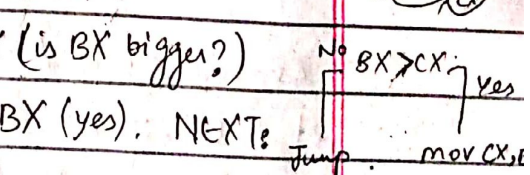
⇒ Single Flag Jumps.

Symbol	conditions
JE/JZ (zero)	ZF=1
JNE/JNZ	ZF=0.
JC (carry)	CF=1
JNC	CF=0
JO (overflow)	OF=1
JNO	OF=0.
JS (sign -)	SF=1
JNS	SF=0
JP/JPE (parity even)	PF=1
JNP/JPO(	PF=0.

Example: (Signed).

```
CMP AX, BX      AX=7FFFh      7FFFh > 8000h is signed. (doesn't jump)
JA Below.       BX=8000h.     7FFFh < 8000h is unsigned. (will jump)
```

Example: AX and BX contains signed numbers. write some code to put the biggest one in CX.




```
MOV CX, AX   CMP BX, CX (is BX bigger?)
JLE NEXT (no)  MOV CX, BX (yes).  NEXT:
```

- JMP destination.
- If condition is true
  Then
  ==
  end if

Example: Replace the number in AX by its absolute value

Pseudocode:

```
If AX < 0:
  Then
    replace AX by -AX
End_if
```

Code:

```
CMP AX, 0
JNL END_IF
NEG AX
end_if:
```

- If condition is true
  then
  ==
  else
  ==
  end_if

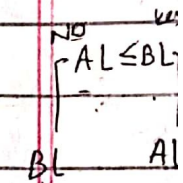
Example: Suppose AL and BL contain extended ASCII characters. Display the one that comes first in character sequence.

Pseudocode:

```
If AL ≤ BL:
  Then
    display character in AL
  else
    Display character in BL
End_if
```

Code:

```
MOV AH, 2
CMP AL, BL ; if
JNBE else
MOV DL, AL ; then
JMP DISPLAY
else: MOV DL, BL ; else
DISPLAY:
    INT 21H
END_IF.
```



Example: If AX contains negative number, put -1 in BX; if AX contains 0, put 0 in BX; if AX contains + number, put 1 in BX.

Pseudocode:

```
CASE AX
  < 0 : put -1 in BX
  = 0 : put 0 in BX
  > 0 : put 1 in BX
End_case.
```

Code:

```
CMP AX, 0
JL NEGATIVE
JE ZERO
JG POSITIVE
NEGATIVE:
    MOV BX, -1
    JMP. END_CASE
ZERO:
    MOV BX, 0 JMP END_CASE
POSITIVE:
    MOV BX, 1 JMP END_CASE
END_CASE:
```

Example: If AL contains 1 or 3, display "o"; if AL contains 2 or 4, display 'e'.

Pseudocode:

```
CASE AL
   1,3 : Display 'o'
   2,4 : Display 'e'
end-case
```

code:

```
CMP AL,1  JE ODD
CMP AL,3  JE ODD
CMP AL,2  JE EVEN
CMP AL,4  JE EVEN
JMP END_CASE
ODD:
   MOV DL,'o'  JMP DISPLAY
EVEN:
   MOV DL,'e'
DISPLAY:
   MOV AH,02  INT 21H
END_CASE:
```

Example: READ a character, if its an uppercase display it.

```
MOV AH,01  INT 21H  CMP AL,'A'  JNGE END_IF
CMP AL,'Z'  JNLE END_IF  MOV DL,AL  MOV AH,02
INT 21H  END_IF:
```

Q- Display 'o' for odd number & 'e' for even numbers.

```
MOV AH,0  MOV AL,?  TEST AL,1 (lsb)  JNZ EVEN
JMP ODD  EVEN: MOV DL,'e'  JMP DISPLAY
ODD: MOV DL,'o'  DISPLAY: MOV AH,2  INT 21H  END MAIN
```

Example: Read a character, if its 'y' or 'Y', display it; otherwise terminate the program.

```
MOV AH,1  INT 21H  CMP AL,'y'  JE Then  CMP AL,'Y'
JE Then  JMP ELSE_Then: MOV AH,02  MOV DL,AL  INT 21H
JMP END_IF  ELSE: MOV AH,4CH  INT 21H  END_IF:
```

Example: Write a count controlled loop to display a row of 80 stars.

```
MOV CX,80    MOV AH,02    MOV DL,'*'
TOP: INT 21H    LOOP TOP
```

JCXZ (Jmp if CX is Zero).

Example: Write some code to count the number of characters in input line.

```
MOV DX,0    MOV AH,01    INT 21H    WHILE: CMP AL,0DH
JE END_WHILE    INC DX    INT 21H    JMP while    end-while:
```

Example: Write some code to read characters until a blank is read.

```
MOV AH,01    REPEAT: INT 21H    CMP AL,' '    JNE REPEAT
```

Program: First & Last Capitals

```
.model small    .stack 100h    .data
prompt db "Type a line of Texts' 0DH,0AH,'$'.
NO_CAP_MSG DB 0DH,0AH,'No capitals$'
CAP_MSG DB 0DH,0AH,'first capital=' First db ']'
DB 'Last capital=' LAST db '@$'?
.code    MAIN PROC    mov ax,@data    lea dx,prompt
int21h    mov ah,01    int21h    while: cmp AL,0DH JE end-while
cmp AL,'A' JNGE end-if cmp AL,'Z' JNLE 'end-if
cmp AL,first JNL CHECK_LAST MOV FIRST,AL
Check_last: CMP AL,Last JNG END_IF MOV LAST,AL
END_If: INT 21H JMP while
END_while: mov AH,9 CMP FIRST,']' JNE CAPS LEA DX,NO_CAPSMSG
JMP display caps: LEA DX,caps_msg. DISPLAY: INT 21H MOV AH,4CH INT 21H
MAIN endp    END MAIN
```

chapter # 08.

"The Stack and Introduction to Procedures"

- Stack Segment – Temporary storage of data & addresses.
- LIFO (Last In First Out).
- Push & Pop Instructions
- Stack can be used to reverse a list of data.

→ To add new word, we use PUSH in stack.

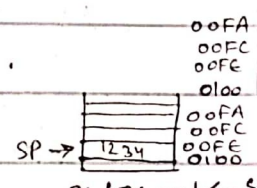
PUSH source. (source is 16-bit register or memory word).

PUSH AX.

Causes: • SP decreased by 2 • Copy of source content moved to address specified by SS:SP. Source is unchanged.

Example:

- Stack 100h.

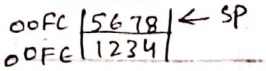


SP 0100
AX 1234
BX 5678

Push ax

SP 00FE

push bx



SP 00FC.

→ To remove top item, we use POP in stack.

POP destination

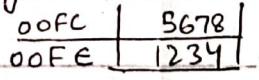
POP BX.

illegal: PUSH DL,
PUSH 2.

Causes: • content of SS:SP is moved to destination,

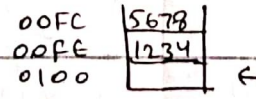
• SP is increased by 2.

pop bx



SP 00FE.

pop ax



SP 0100

read a sequence of characters and display them in reverse order on next line.

```
.model small
.stack 100h
.code
main proc.
mov ah, 02 ← prepare to display.
mov dl, '?' ← char to display
int 21h
xor cx, cx ; count=0.
mov ah, 01 ; prepare to read
int 21h ; read char
while:
    cmp al, 0DH
    JE end_while
    push ax
    inc cx
    int 21h
    jmp while
end_while:
    mov ah, 02
    mov dl, 0DH
    int 21h
    mov dl, 0AH
    int 21h
    jCXz exit
Top:
    pop dx.
    int 21h
    loop Top.
exit:
    mov ah, 4ch
    int 21h
main endp
end main.
```

Procedure Declaration

```
name PROC type
    ; body of proc
    RET
name endp.
```

CALL and RET

To invoke a procedure, call instruction is used.

It maybe direct or indirect

CALL name] Direct

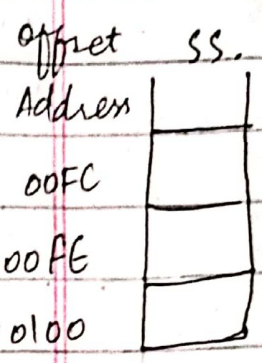
CALL address_expression] Indirect.

Causes:

- The return address to calling program is saved on stack. This is the offset of next instruction after call.

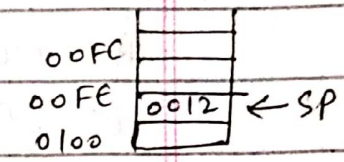
Segment : offset is CS:IP.

```
main proc.
   ≡
0010  call XY2
0012  MOV ah, 02
   ≡
XY2 proc.
0200  mov ah, 09
   ≡
0300  ret.
```



After call:

IP → 0200



RET pop_value.

Before Ret.

IP → ~~0012~~ 0300

After ret.

IP → 0012.

Example:

Find the product of two positive integers A & B by addition and bit shifting.

```
.model small
.stack 100h
.code
main proc
    call multiply
    mov ah, 4ch
    int 21h
main endp
~~multiply~~
end main
```

```
multiply proc
    push ax
    push bx
    xor dx, dx
repeat:
    test bx, 1    ; is B odd?
    jz end_if     ; no, even
    add dx, ax
end_if:
    shl ax, 1
    shr bx, 1
    jnz repeat
    pop bx
    pop ax
    ret
multiply endp
```

Exercise:

Q. AX = 1234h, BX = 5678h, CX = 9ABCh, SP = 100h.

Push AX

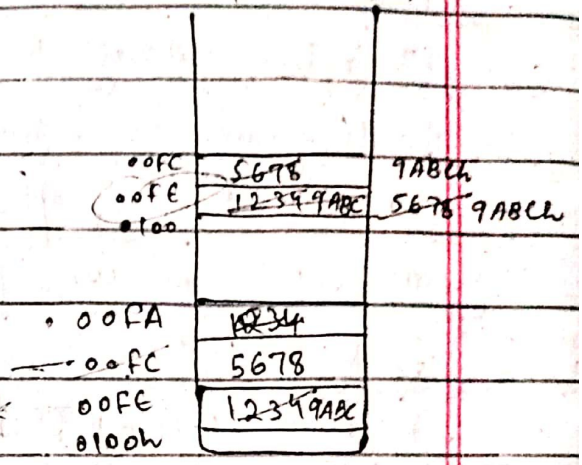
Push bx

Xchg Ax, cx

pop cx

pushw AX

pop bx.



After execution, contents: AX = 9ABCh, BX = 1234h, CX = 5678h, SP = 00FEh.