Date:

COAL

Computer Organization and Assembly Language

Assembly Language

Phases
- → Fetch
- → Decode
- → Execution
- → Write/Back

Processor → 1 kh_z
1000 cycles s^{-1}
1 instruction ⟶ 4 cycles

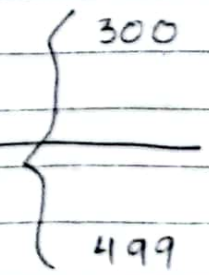
1 sec → 250 instructions

Processor designed on 1 Mhz
1 see → 250 × 1600
= 250000 instructions per second

PC (Program Counter) ← Register

At time T which instructions is executed

We have to check the value of PC at that specific time T.



Date: _____ Sun Mon Tue Wed Thu Fri Sat

Data

- unsigned
  - Range
  - $0 \rightarrow 2^n - 1$
- signed
  - Range
  - $-2^{n-1} \rightarrow 2^{n-1} - 1$

For 8 bit register

$0 \rightarrow 2^8 - 1$ $\quad -2^{8-1} \rightarrow 2^{8-1} - 1$

$0 \rightarrow 255$ $\quad -128 \rightarrow 127$

8 Registers are the combination of 8 Flip Flops

Number Systems

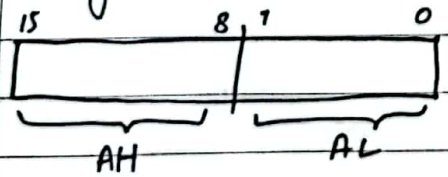
- Binary
- Decimal
- Hexadecimal
- Conversions
- Counting

Registers

Data Registers / General Purpose Registers

- AX (Accumulator Reg)
- BX (Base Reg)
- CX (Count Reg)
- DX (Data Reg)

They all are 16-bit registers



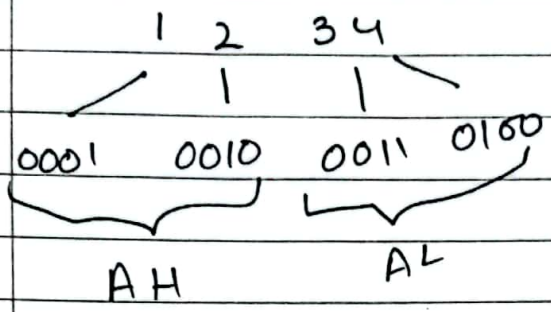
MOV AX, BX

↓

It copies BX to AX value

Date: Sun Mon Tue Wed Thu Fri Sat

MOV AX, 1234



Lecture #2:

Data Registers:

Segment Registers:

16-bit
- DS → Data segment Reg
- CS → Code " "
- SS → stack " "
- ES → Extra " "

Index and Pointer Register

16-bit
- SI → Source Index Reg
- DI → Destination index Reg
- IP → Instruction Pointer Reg
- SP → Stack Pointer Reg
- BP → Base Pointer Reg

To access a location in memory either for reading or writing a segment and index Register is required.

Date: Sun Mon Tue Wed Thu Fri Sat

Offset Address stores in Index ~~Pointer~~ Register

DS → SI, DI
CS → IP
Stack Reg → SP, BP

For 16 MBytes register, size of address will be 2^{16} = 16 bytes.

$2^4 = 16$

Address will be of 4

For 1K bytes = 1024 byte

$2^{10} = 1024$

$64K = 64 \times 1024$
$= 2^6 \times 2^{10}$
$= 2^{16}$

$1K = 2^{10}$
$1M = 2^{20}$
$1G = 2^{30}$

For 16 MB $= 16 \times 1024 \times 1024$
$= 2^4 \times 2^{10} \times 2^{10}$
$= 2^{24}$

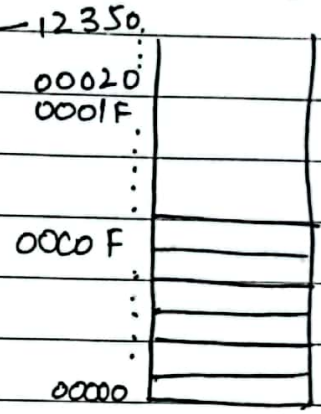
Address of very 1st location is all zeroes
" " " last " is all ones.

Date: Sun Mon Tue Wed Thu Fri Sat

For 1MB
$= 2^{20}$
20 bits
in hexa decimal
20/4 = 5 bits

lower 4 bits of binary or 1 bit in hexa decimal should be zero. So, that memory segment can start.

After every 10 hex or 16 decimal our segment can be start.



Segment NO → stored in DS
But DS is 16 bit and segment NO is 20 bits
so we ignore last 4 bits

= P

SA x 10h + 0A = Phy

16
9 10
12 11 h
B

110
100

16 | 16
1 - 0

Date: Sun Mon Tue Wed Thu Fri Sat

Label opcode operand

```
MOV  al, bl   ✓
Mov  bl, 20   ✓
Mov  20, bl   X
Mov  bl, [SI] ✓
MOV  bl, SI   X
```

DS = 1234
SI = 0200

```
MOV bL, 0
MOV al, [SI]
Add bL, al
INC SI;  SI = 0201
MOV al, [SI]
Add bL, al
INC SI
```

JUMP

Unconditional	Conditional
	JZ → Jump if zero flag set
	JNZ → " " reset
JA	above
AE	
B	Below
BE	
JG	
JGE	

zero Flag → set → when result is zero

Date:

Sun Mon Tue Wed Thu Fri Sat

JZ again

Jump if zero flag is set

```
MOV SI, 0200
MOV CX, 8
MOV bl, 0
again: MOV al, [SI]
       Add bl, al
       INC SI
       JMP again
```

```
MOV SI, 0200
MOV CX, 8
; MOV bl, 0
again: Add bl, [SI]
       INC SI
       DEC CX
       JNZ again
```

Jump

- unconditional: JMP
- conditional: JZ, JNZ, JC, JNC, JA, JAE, JB, JBE, JG, JGE

```
MOV AX, 1234
MOV DS, AX
```

Date: 26-09-23 Sun Mon Tue Wed Thu Fri Sat

COAL

We can't move data from memory to memory because of size
If al is destination than 8 bits will transfer or if ax is used than 16 bits will transfer

- Similarly in Add, Sub operations both operands cannot be memory address
  Add [SI], [DI] X
  Add [SI], al ✓

```
.model small
.stack 100h
.code
. main Proc
    MOV AX, @data
    MOV DS, AX
    MOV DX, offset string1
    MOV AH, 09
    INT 21h
    MOV AH, 4ch
    INT 21 H
main endp
```

use to terminate the program

```
.data
string1 db 'Good $'
end main
```

Premier Quality Paper Products
Premier

Date: 28-09-23: Sun Mon Tue Wed Thu Fri Sat

(Label) : opcode operands ; comment
destination, source

* In debugers — no label and comments

MOV al, bl — size should be same

MOV bl, [SI] — pointer

DS = 1234
SI = 0200
↳ it points to data in bytes store at that location

28-09-2023

```
.model small
.stack 100h
.code
 main Proc
    MOV  Ax, @data
    MOV  DS, AX
    MOV  DX, offset string1
    MOV  AH, 09
    INT  21H
    MOV  AH, 4ch
    INT  21H
 main endp
.data
 string1 db 'Good $'
end main
```

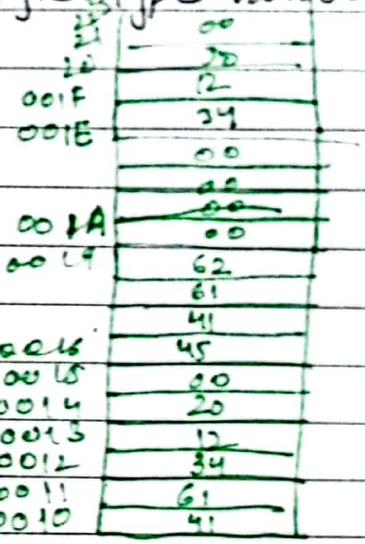
Date: 28-09-23

Sun Mon Tue Wed Thu Fri Sat

Starting offset of string should be in DX.

If you want to print string in next line then we 10,13 before string.
0Ah,0Dh

db (define byte) declare byte type variable
2 bytes — dw (define word)
dd (double word)
dq (quad word)



long 8
int 4
short 2

```
.data
num1  db  41h
num2  db  61h
num3  dw  1234h
num4  dw  20h
array1 db  45h, 41h, 61h, 62h
array2 db  4 dup(00)
array3 dw  1234h, 50h, 200h
```

↓

6 memory locations will be allocated for array 3.

string1 db 'Good Morning'

Date: 03-10-23

Sun Mon Tue Wed Thu Fri Sat

LOGIC INSTRUCTIONS

Arithmetic operations change the whole number completely

Example

```
  ① 1111 1111
+           1
-----------------
  0000 00 00
```

If we want to change a specific bit, we use logic instructions

AND → used to RESET certain bits keeping remaining bits unchanged.

Example: Let BL = 0100 0111 → Want to RESET bit 2

```
  1111 1011
    ↓    ↓
    F    B  → logical operand
```

Keep the bit zero in logical operand that you want to reset and remaining bits will be 1.

AND BL, 0FBh

```
  0100 0111
  1111 1011
-----------------
BL = 0100 0011
```

If logical operand is 0F0h than which bits of BL will be changed

As the logical operand is 1111 0000 so first four bits will be changed.

Date: Sun Mon Tue Wed Thu Fri Sat

Take a lower case character from user and change it in upper case using logic instruction.

Observe the bit pattern of both cases. Only 5th bit is changed

a = 0 1 1 0 0 0 0 1
A = 0 1 0 0 0 0 0 1

b = 0 1 1 0 0 0 1 0
B = 0 1 0 0 0 0 1 0

```
      7 6 5 4 3 2 1 0
b:    0 1 1 0 0 0 1 0
DF:   1 1 0 1 1 1 1 1
      ---------------
B:    0 1 0 0 0 0 1 0
```

AND AL, DFh

Take character from user if it is in lower case convert it into upper case otherwise display it as it is →

```
MOV AH, 01
INT 21H
AND AL, DFh
MOV DL, AL
MOV AH, 02
INT 21H
```

∴ OR → used to set specific bit

The bit you want to set is 1 and remaining will be zero

Upper case to lower case → OR used

Date:

Sun Mon Tue Wed Thu Fri Sat

XOR → compliment / toggle

Keep the bit → 1 you want to toggle and remaining bits will be 0

XOR AL, 02H →

```
  0 0 1 0   0 0 0 1
  0 0 0 0   0 0 1 0
-------------------
  0 0 1 0   0 0 1 1
```

Change the case of character → XOR used

Convert the case of string

```
Main Proc
    MOV AX, @data
    MOV DS, AX
    MOV SI, offset string
    MOV CX, 5
    MOV al, 20H
again: XOR [SI], al
    INC SI
    INC al
    DEC CX
    JNZ again

    MOV Ah, 09
    INT 21h
```

```
.data
string db "Hello $"
```

```
MOV bl, [SI]
XOR bl, 20H
MOV [SI], bl
```

Date: Sun Mon Tue Wed Thu Fri Sat

```
main Proc
    MOV  AX, @ data
    MOV  DS, AX
    MOV  DX, offset string
    MOV  AH, 09
    INT  21 H
    MOV. CX, 5
    MOV  SI, offset string
again: MOV  bl, [SI]
    XOR  bl, 20H
    MOV  [SI], bl
    INC  SI
    DEC  CX
    JNZ  again

    MOV  DX, offset string
    MOV  AH, 09
    INT  21h
    MOV  AH, 4ch
    INT  21h
' main endp
.data
    string db "Hello $"
```

0000 0001

Date: 05-10-23

1 2 4 A
0001 0010 0100 1010

Sun Mon Tue Wed Thu Fri Sat

→ Compare instruction works like subtract instruction
→ Flags update on the basis of result.

```
CMP AL, BL
JE  GO
=
=
=
=
SUB AL, BL
JMP exit
```
→ if AL ≠ BL

```
GO:  ==
     ==
exit: ==
```
→ if AL = BL

JE
JNE

unsigned:
- JB
- JBE
- JA
- JAE

treat as signed:
- JG
- JGE
- JLE
- JL

Label should be used before second block and unconditional jump is used at the end of first block.

FF as signed
↓
−1 as signed

```
11111111   −1
00000000
      +1
00000001  → +1
```
so

9

```
CMP al, '9' 39h
JBE Num
```

JMP exit

Num:

Date: Sun Mon Tue Wed Thu Fri Sat

Shift Instructions

- unsigned
  - Logic Shift
    - Left → SHL
    - Right → SHR
- signed
  - Arithmetic Shift
    - Left → SAL
    - Right → SAR

SHR BL, 1

1 0 0 1 0 0 1 0 → goes to carry flag

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ → Right shift

0 1 0 0 1 0 0 1

↓

vacated bit is filled with zero

MOV cl, 3

SHL BX, cl

0 0 1 E

\+ 1

0 0 1 F

1

0 0 2 0

16 | 16

1 - 0

2 | 41

2 | 20 - 1

2 | 10 - 0

2 | 5 - 0

2 | 2 - 1

1 - 0

001

Date: 10-10-23

Sun Mon Tue Wed Thu Fri Sat

Rotate Instructions

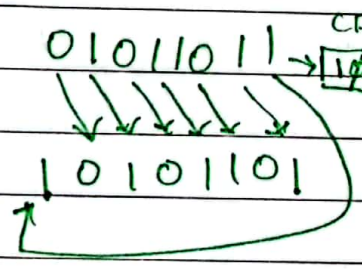
Rotate without carry

- Right → ROR
- Left → ROL

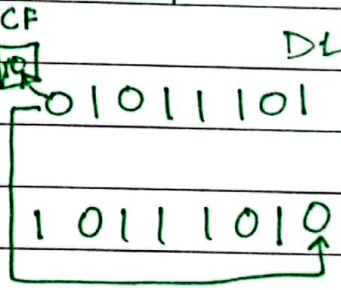
Rotate with carry

- Right → RCR
- Left → RCL

ROR AL, 1



ROL AL, 1

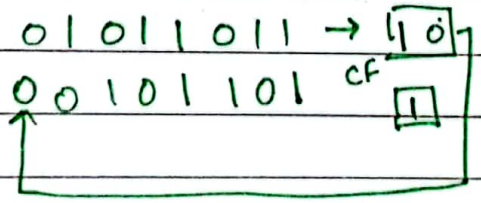


0000 0001 · 1

DL = 1 AND DL, 01

ADD DL, 30h

RCR AL, 1 · CF



ROR, ROL → Same bit goes to carry flag and vacated place

RCR, RCL → Bit goes to carry flag and carry flag bit move to the vacated place.

You weren't put on Earth to be remembered.
You were put here to prepare for eternity.

Date: Sun Mon Tue Wed Thu Fri Sat

Multiple Rotation

Ax: 4 3 A 7

Ax: 0100 0011 1010 0111

ROL Ax, 4

0011 1010 0111 0100

ROL Ax, 8 A743 ↓ 3 A74

ROL Ax, 12 7 4 3 A

```
CMP DL, 0
JE zero
CMP DL, 1
JE ONE
```

zero: ——

JMP exit

ONE:

CMP DL, 9
JBE bb

aa: Add DL, 37h
MOV AH, 02
INT 21H

bb: Add DL, 30h
MOV AH, 02
INT 21H

aa:
0 — 30h
1 — 31h
2 — 32h
|
9 — 39h
Add 30h

CMP DL, 9
JBE aa
A — 41h
|
F — 46H
Add 37

41
− A
37h

10
A 41

Date: 19-10-23

Sun Mon Tue Wed Thu Fri Sat

COAL

```
main Proc
    Mov  Ax, @data
    Mov  DS, Ax
    MOV  Dx , offset string1
    Call display
    Mov Dx, offset string2.
    Call display
```

```
display Proc
MOV AH, 09
INT 21H
RET
display endp
```

```
.data
string1  db 'Good Morning $"
string2  db 'Hope you will be fine $'.
```

When call instruction is executed IP update and sp is decremented by 2.

Date:

0000
-1

Sun Mon Tue Wed Thu Fri Sat

```
HEX proc
MOV CL, 4
ROL DL, CL
MOV BL, DL
AND BL, 0Fh
CMP BL, '9'
```

SP = 0100

```
0100  MOV  AX, 1234
0103  MOV  BX, 5678
0106  PUSH AX → SP = 00FE
0107  PUSH BX → SP = 00FC
0108  ADD  AX, 10   //
0109  ADD  BX, 10   //
010A  POP  BX → SP = 00FE
010B  POP  AX → SP = 0100
010C
```

→ If we PUSH in procedure then before RET POP all the values that you PUSH

SP se value → IP

Take a number in main make 2 proc (A HEX and binary)

Make a single function that prints HEX on first call and binary on 2nd call.

Date _____

COAL

Program to display Number in HEX.

```
MOV  BX, 12A4h
MOV  CX, 4
again : ROL  BX, 4
   MOV DL, BL
   AND DL, 0Fh
   CMP DL, 09
   JBE aa
   ADD DL, 37H
   MOV AH, 02
   INT 21H
   JMP exit
aa:  Add DL, 30H
   MOV AH, 02
   INT 21H
exit . DEC CX
   JNZ again
```

Date _____

```
.model small
.stack 100h
.code
main PROC
        MOV BX, 1234h
        MOV CX, 4
again:  MOV DL, 13
        AND DL, 0Fh

        Same code
exit:   ROR BX, 4
        DEC CX
        JNZ again
```

Date _____

reverse

Program to display binary using rotate

```
MOV BX, 124AH
MOV CX, 16
again: MOV DL, BL
       AND DL, 01H
       CMP DL, 09
       JBE, aa
       Add  DL, 37H
       MOV  AH, 02
       INT  21H
       JMP  exit
aa:    ADD  DL, 30H
       MOV  AH, 02
       INT  21H
exit:  ROR  BX, 1
       DEC  CX
       JNZ  again
```

SKANS
School of Accountancy

Date _____

one by one Store in BX

Binary Input → ~~Binary~~ Output

```
MOV BX,00H
MOV CX,16
again:MOV AH,01
INT 21H
SUB AL,30H
SHL BX,1
OR BL,AL

DEC CX
JNZ again
```

B=A
MOV AX,A
MOV B,AX

1. Input Binary → Output ✓
   ↓
   BX
2. Input Binary → output HEX ✓
3. HEX → BINARY ✓
4. HEX → HEX ✓

Date _____

Book

→ 4.9

4.12

lower case → upper case sub 20h

Q3, Q4, Q6, Q7(A,B), Q8, Q9

Pg 93, 94 (6,2)

Example 6.5

JG == JNLE

LEA = Load Effective Address

Don't use LEA