x86’s instruction sets

Instruction Set Classification

- Transfer
  - Move
- Arithmetic
  - Add / Subtract
  - Mul / Div, etc.
- Control
  - Jump
  - Call / Return, etc.

Data transfer : Move

- MOV Dest, Src
  - `MOV reg, reg` `reg <- reg`
  - `MOV reg, mem` `reg <- mem`
  - `MOV mem, reg` `mem <- reg`
  - `MOV reg, imm` `reg <- imm`
  - `MOV mem, imm` `mem <- imm`

- There is no move mem<-mem instruction.

Move limitation

- Both operand must be in the same size.
- There is no instruction to put immediate value directly to segment register. Have to use accumulator (AX) to accomplish this.
- To put immediate value directly to memory, we have to specify its size. (Byte/Word PTR)

Move *(MOV)* Example

- `MOV AX,100h`
- `MOV BX,AX`
- `MOV DX,BX`

- `MOV AX,1234h`
- `MOV DX,5678h`
- `MOV AL,DL`
- `MOV BH,DH`

MOV Example

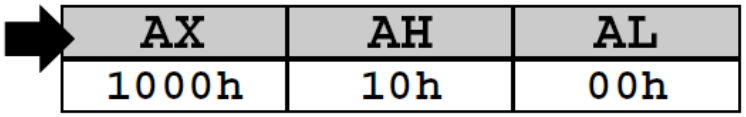
- `MOV AX,1000h`
- `MOV [100h],AX`
- `MOV BX,[100h]`

- `MOV BYTE PTR [200h],10h`
- `MOV WORD PTR [300h],10h`

- `MOV AX,2300h`
- `MOV DS,AX`

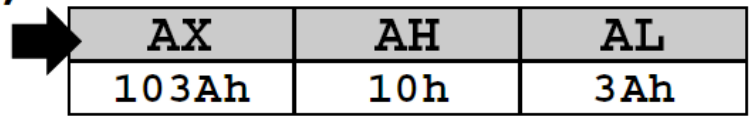
MOV : 16 / 8 Bit register

```
MOV  AX,1000h
```



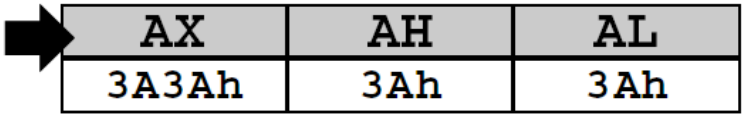
AX	AH	AL
1000h	10h	00h

```
MOV  AL,3Ah
```



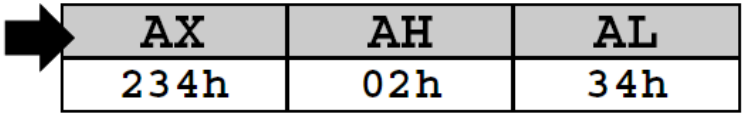
AX	AH	AL
103Ah	10h	3Ah

```
MOV  AH,AL
```



AX	AH	AL
3A3Ah	3Ah	3Ah

```
MOV  AX,234h
```

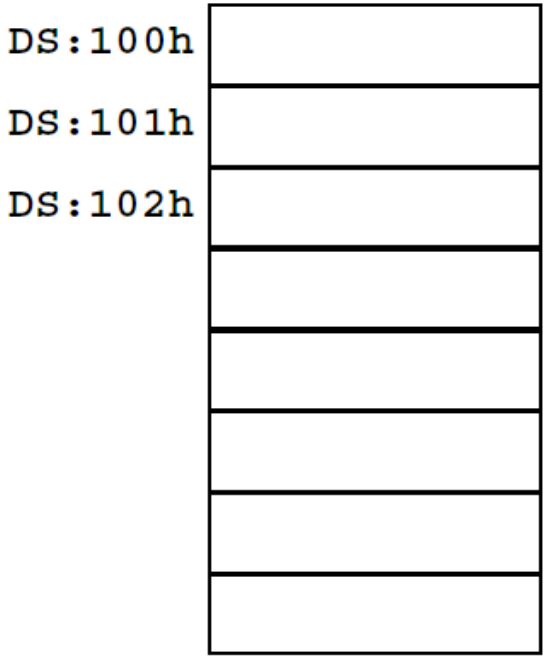


AX	AH	AL
234h	02h	34h

- To move value between registers, their size must be the same.

MOV : Memory

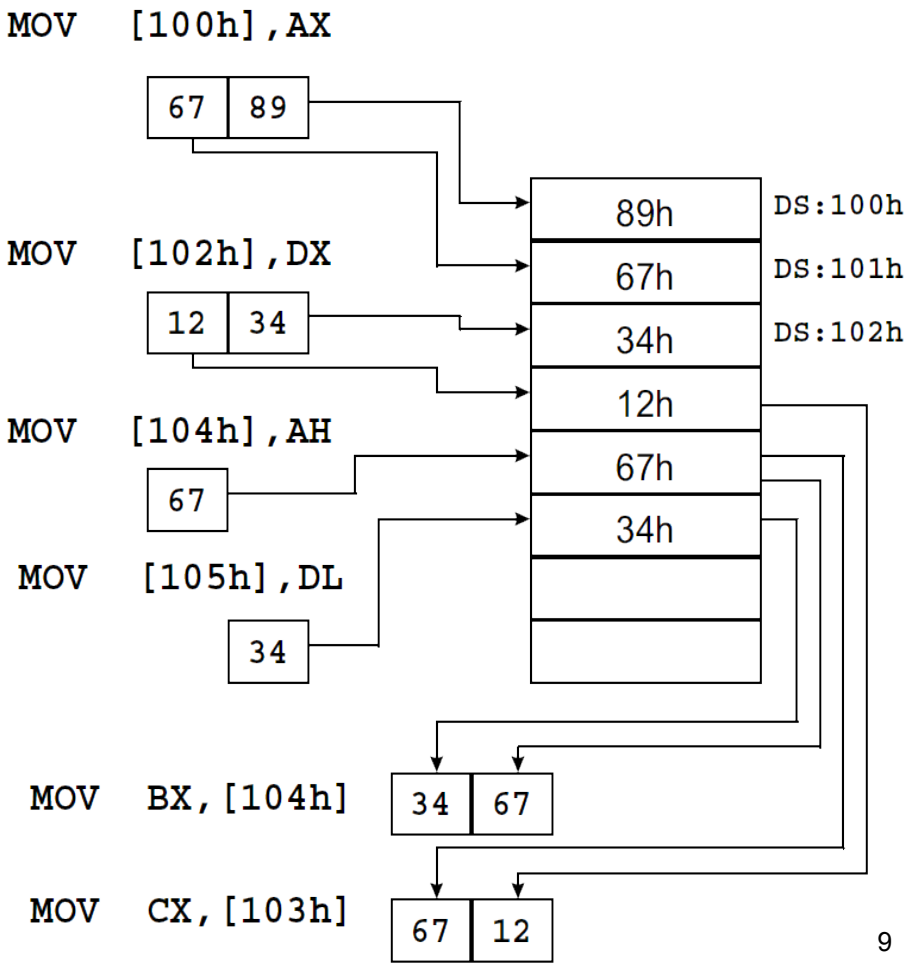
```
MOV   AX,6789h
MOV   DX,1234h
MOV   [100h],AX
MOV   [102h],DX
MOV   [104h],AH
MOV   [105h],DL
MOV   BX,[104h]
MOV   CX,[103h]
MOV   [106h],CL
```



- Given only offset where to put value, it will be automatically select DS as the segment register.

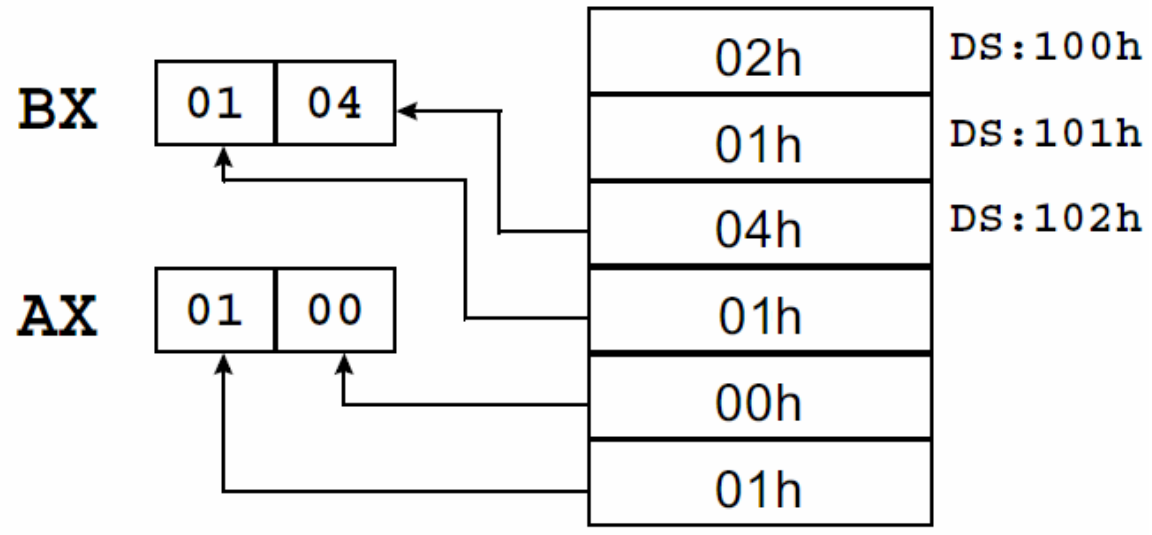
Byte ordering : Little endian

- Since, x86's byte ordering is little endian.
- Therefore, the LSB will be placed at lowest address and MSB will be placed at highest address.



Displacement

- We can use BX (Base) register to point a place of memory.
- Both register direct or displacement.
-




```
MOV   AX,102h
MOV   BX,100h
MOV   CX,4004h
MOV   DX,1201h
MOV   [BX],AX
MOV   [BX+2],CX
      [BX+3],DX
      [BX+4],BX
      BX,[102h]
      AX,[BX]
```

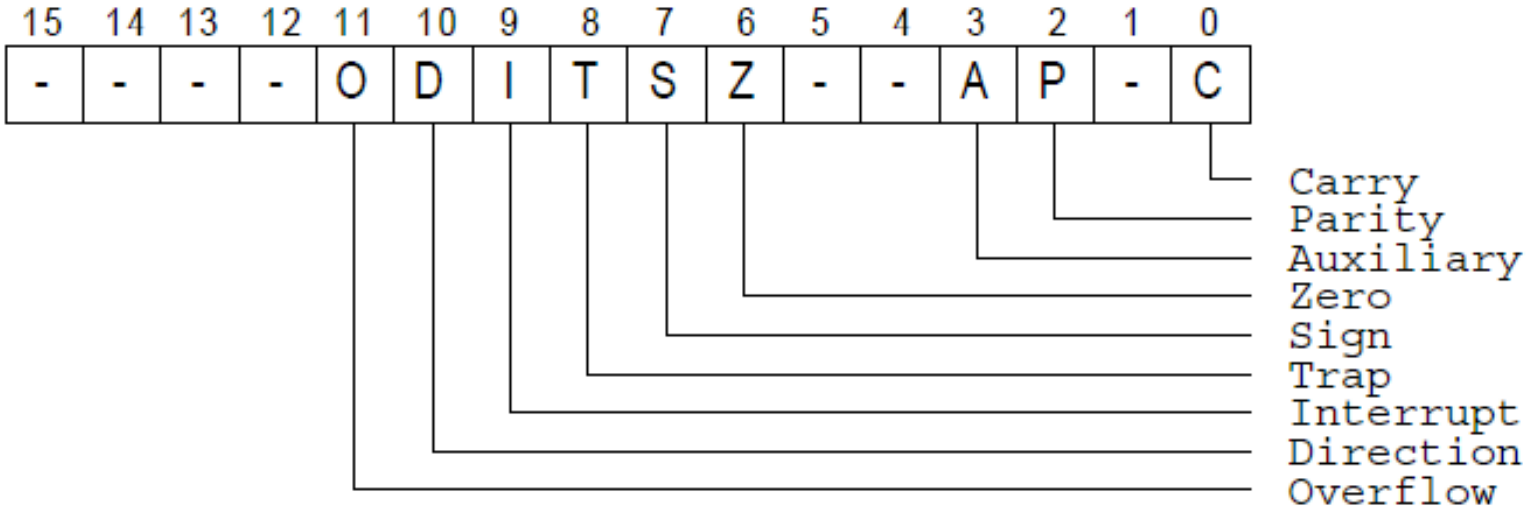
What is the result of …

- MOV [100h] , 10h
- Address 100 = 10h
- What about address 101?
- Word or Byte?
  - `MOV WORD PTR [100h], 10h`
  - `MOV BYTE PTR [100h], 10h`
- What about MOV [100h], AX ?

Status Register (Flag)

Flag

- 8086 has 16 bit flag to indicate the status of final arithmetic result.



Zero Flag

- The zero flag will be set (1) whenever the result is zero.

EX

```
MOV   AL,10h      Z=?
ADD   AL,E0h      Z=1     AL=0
ADD   AL,20h      Z=0     AL=20h
SUB   AL,10h      Z=0     AL=10h
SUB   AL,10h      Z=1     AL=0
```

Parity flag

- The parity flag will be set whenever the number of bit “1” are even.

EX

```
MOV   AL,14h      P=?
ADD   AL,20h      P=0     AL=34h
ADD   AL,10h      P=1     AL=44h
SUB   AL,8h       P=1     AL=3Ch
SUB   AL,10h      P=0     AL=2Ch
```

Carry flag

- Carry flag will be set whenever there is a carry or borrow (only with unsigned operation).

EX

```
MOV   AL,77h      C=?
ADD   AL,50h      C=0     AL=C7h
ADD   AL,50h      C=1     AL=17h
SUB   AL,A0h      C=1     AL=77h
ADD   AL,27h      C=0     AL=9Eh
```

Overflow flag

- Overflow flag will be set whenever the result is overflow (with signed operation).

EX

```
MOV   AL,77h      O=?
ADD   AL,50h      O=1     AL=C7h
ADD   AL,50h      O=0     AL=17h
SUB   AL,A0h      O=0     AL=77h
ADD   AL,27h      O=1     AL=9Eh
```

Sign flag

- Sign flag will be set whenever the result is negative with signed operation.

EX

```
MOV   AL,77h      S=?
ADD   AL,50h      S=1     AL=C7h
ADD   AL,50h      S=0     AL=17h
SUB   AL,A0h      S=0     AL=77h
ADD   AL,27h      S=1     AL=9Eh
```

More flag

- Auxiliary flag will be set when the result of BCD operation need to be adjusted.
- Direction flag is used to specify direction (increment/decrement index register) in string operation.
- Trap flag is used to interrupt CPU after each operation.
- Interrupt flag is used to enable/disable hardware interrupt.

Flag set/reset instructions

- Carry flag STC / CLC
- Direction flag STD / CLD
- Interrupt flag STI / CLI

Arithmetic instructions

Increment - Decrement

- INC / DEC
  - INC register DEC register
  - INC memory DEC memory
- EX.
  - INC AX
  - DEC BL
  - How can we increment a byte of memory?
    - INC ??? [100]

Add

- ADD reg, imm
- ADD reg, mem
- ADD reg, reg
- ADD mem, imm
- ADD mem, reg

ADC reg, imm

ADC reg, mem

ADC reg, reg

ADC mem, imm

ADC mem, reg

EX. ADD

```
MOV AL, 10h
ADD AL, 20h          ;AL =   30h
MOV BX, 200h         ;BX = 0200h
MOV WORD PTR [BX], 10h
ADD WORD PTR [BX], 70h
MOV AH, 89h          ;AX = 8930h
ADD AX, 9876h        ;AX = 21A6h
ADC BX, 01h          ;BX = 0202h ?
```

Subtract

- SUB reg, imm
- SUB reg, mem
- SUB reg, reg
- SUB mem, imm
- SUB mem, reg

SBB reg, imm

SBB reg, mem

SBB reg, reg

SBB mem, imm

SBB mem, reg

Ex. SUB

```
MOV AL, 10h
ADD AL, 20h          ;AL =   30h
MOV BX, 200h         ;BX = 0200h
MOV WORD PTR [BX], 10h
SUB WORD PTR [BX], 70h
MOV AH, 89h          ;AX = 8930h
SBB AX, 0001h        ;AX = 892Eh ?
SBB AX, 0001h        ;AX = 892Dh
```

Compare

- CMP reg, imm
- CMP reg, mem
- CMP reg, reg
- CMP mem, reg

- There is no “CMP mem, imm”

Ex. CMP

```
MOV CX, 10h
CMP CX, 20h   ;Z=0,S=1,C=1,O=0
MOV BX, 40h
CMP BX, 40h   ;Z=1,S=0,C=0,O=0
MOV AX, 30h
CMP AX, 20h   ;Z=0,S=0,C=0,O=0
```

Negation

- Compute 2’complement.
- Carry flag always set.

- Usage
  - NEG reg
  - NEG mem

Ex. NEG

```
MOV CX, 10h
NEG CX                  ; CX = 0FFF0h
MOV AX,0FFFFH
NEG AX                  ; AX = 1
MOV BX,1H
NEG BX                  ; BX = 0FFFFh
```

Multiplication

- IMUL (Integer multiplication) unsigned multiplication
- MUL (Multiplication) signed multiplication.
  - MUL reg IMUL reg
  - MUL mem IMUL mem
- Always perform with accumulator.
- Effected flag are only over and carry flag.

8 bit multiplication

- AL is multiplicand
- AX keep the result

- MOV AL,10h ; AL = 10h
- MOV CL,13h ; CL = 13h
- IMUL CL ; AX = 0130h

16 bit multiplication

- AX is multiplicand
- DX:AX keep the result

```
MOV AX,0100h    ; AX = 0100h
MOV BX,1234h    ; BX = 1234h
IMUL BX         ; DX = 0012h
                ; AX = 3400h
```

Division

- IDIV (Integer division) unsigned division.
- DIV (Division) signed division.
  - DIV reg IDIV reg
  - DIV mem IDIV mem
- Always perform with accumulator.
- Effected flag are only over and carry flag.

8 bit division

- AL is dividend
- AL keep the result
- AH keep the remainder

- MOV AX, 0017h
- MOV BX, 0010h
- DIV BL ; AX = 0701

16 bit multiplication

- DX:AX dividend.
- AX keep the result, DX keep the remainder.

```
MOV AX,4022h          ;
MOV DX,0000h          ;
MOV CX,1000h          ;
DIV CX                ;AX = 0004
                      ;DX = 0022
```

Conversion

- Byte to Word : CBW
  - Signed convert AL -> AX
- Word to Double word : CWD
  - Signed convert AX -> DX:AX

Ex. Conversion

```
MOV AL,22h
CBW                 ; AX=0022h
MOV AL,F0h
CBW                 ; AX=FFF0h
MOV AX, 3422h
CWD                 ; DX=0000h
                    ; AX=3422h
```

Instruction	Flag affected				
	Z-flag	C-flag	S-flag	O-flag	A-flag
ADD	*yes*	*yes*	*yes*	*yes*	*yes*
ADC	*yes*	*yes*	*yes*	*yes*	*yes*
SUB	*yes*	*yes*	*yes*	*yes*	*yes*
SBB	*yes*	*yes*	*yes*	*yes*	*yes*
INC	*yes*	***no***	*yes*	*yes*	*yes*
DEC	*yes*	***no***	*yes*	*yes*	*yes*
NEG	*yes*	*yes*	*yes*	*yes*	*yes*
CMP	*yes*	*yes*	*yes*	*yes*	*yes*
MUL	*no*	*yes*	*no*	*yes*	*no*
IMUL	*no*	*yes*	*no*	*yes*	*no*
DIV	*no*	*no*	*no*	*no*	*no*
IDIV	*no*	*no*	*no*	*no*	*no*
CBW	*no*	*no*	*no*	*no*	*no*
CWD	*no*	*no*	*no*	*no*	*no*

Example about flag with arithmetic

Instruction		Z-flag	C-flag	O-flag	S-flag	P-flag	หมายเหตุ
MOV	AX,7100h	?	?	?	?	?	
MOV	BX,4000h	?	?	?	?	?	
ADD	AX,BX	0	0	1	1	1	AX=0B100h
ADD	AX,7700h	0	1	0	0	1	AX=2800h
SUB	AX,2000h	0	0	0	0	0	AX=0800h
SUB	AX,1000h	0	1	0	1	0	AX=F800h
ADD	AX,0800h	1	1	0	0	1	AX=0000h

- เอกสารอ้างอิง
  - เอกสารประกอบการสอนวิชา 204221 องค์ประกอบคอมพิวเตอร์และภาษาแอสเซมบลี้ มหาวิทยาลัยเกษตรศาสตร์, 1994

Example about flag with arithmetic

Instruction		Z-flag	C-flag	O-flag	S-flag	P-flag	หมายเหตุ
MOV	AL,10	?	?	?	?	?	
ADD	AL,F0h	0	0	0	1	1	AL=0FAh
ADD	AL,6	1	1	0	0	1	AL=0
SUB	AL,5	0	1	0	1	0	AL=0FBh
INC	AL	0	1	0	1	1	AL=0FCh
ADD	AL,10	0	1	0	0	1	AL=6h
ADD	AL,FBh	0	1	0	0	0	AL=1h
DEC	AL	1	1	0	0	1	AL=0h
DEC	AL	0	1	0	1	1	AL=0FFh
INC	AL	1	1	0	0	1	AL=0

Instruction		Z-flag	C-flag	O-flag	S-flag	P-flag	หมายเหตุ
MOV	AL,120	?	?	?	?	?	
ADD	AL,15	0	0	1	1	1	AL=87h=-121
NEG	AL	0	1	0	0	0	AL=79h
SUB	AL,130	0	1	1	1	0	AL=0F7h

NEG -> Carry flag always 1, INC/DEC does not effect any flag

Jump and Loops

- Control structures
  - Selection
  - Repetition / Loop

- Jxx Label

คำสั่ง	ความหมาย	เงื่อนไข
คำสั่งกระโดดแบบไม่มีเงื่อนไข		
JMP	Jump	
เงื่อนไขตามแฟล็ก		
แฟล็กศูนย์		
JZ	Jump if Zero	ZF=1
JNZ	Jump if Not Zero	ZF=0
แฟล็กโอเวอร์โฟล์		
JO	Jump if Overflow	OF=1
JNO	J. if Not Overflow	OF=0
แฟล็กทด		
JC	Jump if Carry	CF=1
JNC	Jump if No Carry	CF=0
แฟล็กเครื่องหมาย		
JS	Jump if Sign	SF=1
JNS	Jump if No Sign	SF=0

คำสั่ง	ความหมาย	เงื่อนไข
เงื่อนไขตามแฟล็ก		
พาริตี้แฟล็ก		
JPO	Jump if Parity Odd	PF=0
JPE	Jump if Parity Even	PF=1
ตัวเลขแบบไม่คิดเครื่องหมาย		
JA	Jump if Above	(CF and ZF)=0
JB	Jump if Below	CF=1
JAE	J. if Above or Equal	CF=0
JBE	J. if Below or Equal	(CF or ZF)=1
ตัวเลขแบบคิดเครื่องหมาย		
JG	Jump if Greater	ZF=0 and SF=OF
JL	Jump if Less	SF<>OF
JGE	J. if Greater or Equal	SF=OF
JLE	J. if Less or Equal	(ZF=1) or (SF<>OF)
เงื่อนไขตามค่าในรีจิสเตอร์		
JCXZ	Jump if CX=0	CX=0

คำสั่ง	ชื่ออื่น	ความหมาย
JZ	JE	Equal
JNZ	JNE	Not Equal
JA	JNBE	Not Below or Equal
JB	JNAE	Not Above or Equal
JAE	JNB	Not Below
JBE	JNA	Not Above
JG	JNLE	Not Less or Equal
JL	JNGE	Not Greater or Equal
JGE	JNL	Not Less
JLE	JNG	Not Greater

If ah=10, if ah>=10

```
        cmp     ah,10           ;เปรียบเทียบ ah กับ 10
        jz      lab1            ;ถ้าเท่ากันให้กระโดดไปที่ lab1
        mov     bx,2
lab1:   add     cx,10
```

```
        cmp     ah,10           ;เปรียบเทียบ ah กับ 10
        jge     tenup           ;ถ้ามากกว่าหรือเท่ากับให้กระโดดไปที่ tenup
        add     dl,'0'
        jmp     endif           ;กระโดดไปที่ endif
tenup:
        add     dl,'A'
endif:
```

Get ‘Q’ and Print ASCII code.

```
getonechar:
        mov     ah,1            ; ใช้บริการหมายเลข 1 : อ่านอักขระ
        int     21h
        cmp     al,'Q'          ; เปรียบเทียบ al กับ 'Q'
        jne     getonechar      ; ถ้าไม่เท่ากันให้กระโดดไปที่ getonechar (กลับไปรับตัวอักษรใหม่)


        mov     ah,02           ; บริการหมายเลข 2 : พิมพ์อักขระ
        mov     dl,32           ; เริ่มที่ ช่องว่าง ASCII = 32 (' ')
printloop:
        cmp     dl,128          ; เปรียบเทียบ dl กับ 128 (ASCII สุดท้าย)
        ja      finish          ; ถ้ามากกว่ากระโดดไปที่ finish
        int     21h             ; พิมพ์อักขระที่มี ASCII = dl
        inc     dl              ; เพิ่ม dl
        jmp     printloop
finish:
```

Loop

- Base on CX (Counter register) to count the loop.
- Instructions :
  - LOOP ; Dec CX … 0
  - LOOPZ ;CX<>0 and Z=1
  - LOOPNZ ;CX<>0 and Z=0
  - JCXZ ; Jump if CX=0, used with LOOP to determine the CX before loop.

LOOP

```
        mov     cx,20                   ; ทำซ้ำ 20 ครั้ง
        mov     bl,1                    ; เริ่มจาก 1
        mov     dx,0                    ; กำหนดค่าเริ่มต้นให้กับผลรวม
addonenumber:
        add     dl,bl                   ; บวก 8 บิตล่าง
        adc     dh,0                    ; รวมตัวทด
        inc     bl                      ; ตัวถัดไป
        loop    addonenumber            ; ถ้า CX ลดลงแล้วไม่เท่ากับ 0  ทำซ้ำต่อไป
```

JCXZ

```
        initialization
        jcxz   endloop                  ; CX =0 ?
label1:
        actions
        loop   label1                   ; loop
endloop:
```

Example finding first character

```
        cmp     byte ptr [bx],' '    ; พิจารณาอักษรตัวแรก
        jnz     found                ; อักษรตัวแรกไม่ใช่ช่องว่าง
        mov     cx,100               ; ทำซ้ำ 100 ครั้ง
findnotspace:
        inc     bx                   ; BX ชี้ไปยังอักษรตัวถัดไป
        cmp     byte ptr [bx],' '    ; เปรียบเทียบ
        loopz   findnotspace         ; ทำซ้ำถ้ายังพบช่องว่างและยังไม่ครบข้อมูล
        jnz     found                ; ค้นเจออักษรตัวแรก

        ; not found                  ; ข้อความมีแต่ช่องว่าง
        …

found:
        ; found                      ; พบอักษรตัวแรก
        …
```

That’s all.