LẬP TRÌNH HỢP NGỮ

Nội dung chính

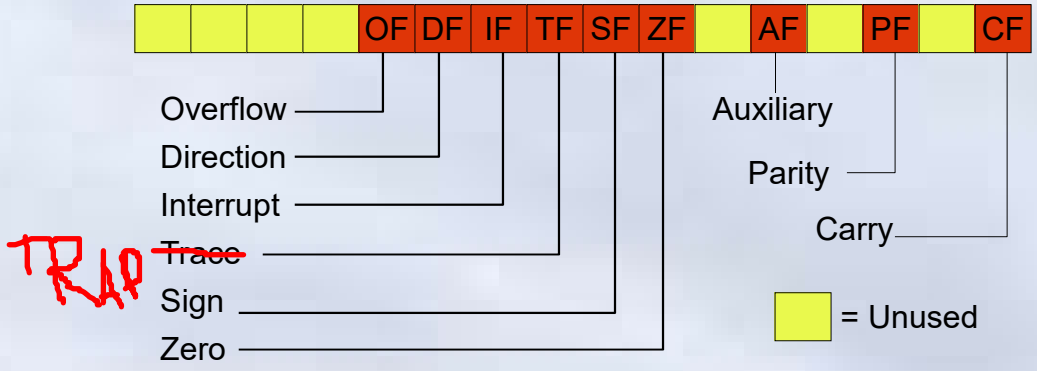
- Thành phần cơ bản của hợp ngữ
- Lập trình hợp ngữ căn bản
- Cấu trúc tập lệnh của x86
- Thủ tục và Macro
- Dữ liệu có cấu trúc
- Hợp ngữ và ngôn ngữ bậc cao

Nội dung chính

- **Thanh ghi trạng thái processor**
- Các lệnh sao chép dữ liệu
- Các lệnh số học
- Các lệnh thao tác với bit
- Các lệnh điều khiển
- Các câu lệnh với chuỗi
- Các câu lệnh khác

Thanh ghi cờ

- Thanh ghi cờ cho biết kiểu hoạt động của Processor và thông tin trạng thái của một số lệnh
- Mỗi bit ứng với một trạng thái: Đặt (1)/ xóa (0)



Chương trình Debug

- Dùng gỡ rối chương trình
  - Cho phép thực hiện từng lệnh (*lệnh **t/p***), đoạn lệnh (*lệnh **g***) hay chương trình con (*lệnh **p***)
  - Có thể xem nội dung của các thanh ghi (*lệnh **r***)
    - Biểu diễn trạng thái thanh ghi cờ theo tên

```
C:\Documents and Settings\Pham Dang Hai>debug
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1548  ES=1548  SS=1548  CS=1548  IP=0100  NV UP EI PL NZ NA PO NC
1548:0100 0000          ADD     [BX+SI],AL                  DS:0000=CD
-
```

Nhiều chương trình gỡ rối hiệu quả: TD, TD32 ..

Chương trình Debug

Trạng thái thanh ghi cờ được biểu diễn theo tên

Flags	Set (1)	Clear(0)
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:** Non zero
SF	**NG:** Negative	**PL:** Plus
OF	**OV:** Overflow	**NV:** No overflow
DF	**DN:** Down	**UP:** Up
IF	**EI:** Enable Interrupt	**DI:** Disable Interrupt

Cờ trạng thái

Phản ánh kết quả của một phép toán

- **Cờ tràn (**Overflow Flag**)**
  - Đặt (**OV**) khi kết quả số học vượt quá khả năng biểu diễn của số có dấu, ngược lại bị xóa (**NV**)
- **Cờ zero (**Zero Flag**)**
  - Đặt (**ZR**) nếu kết quả bằng 0
  - Thường dùng để kiểm tra 2 giá trị bằng nhau hay kiểm tra một bit đặc biệt của thanh ghi (AND..)

Cờ trạng thái

- **Cờ dấu (**Sign Flag**)**
  - Đặt (**NG**) nếu kết quả phép toán số học là số âm (bit cao nhất bằng 1)
- **Cờ chẵn lẻ (**Parity Flag**)**
  - Được đặt (**PE**) Nếu ***byte thấp*** của kết quả có tổng bit là chẵn, ngược lại bị xóa (**PO**)

```
Mov AX, 0909h;      PF = ?
Add AX, 0004;       PF = PO
Add AX, 0001;       PF = PO
Add AX, 0400;       PF =PO
Add AX, 0001;       PF = PE
```


Cờ trạng thái

- **Cờ nhớ** (Carry Flag)
  - Được dùng cho nhiều mcj đích khác nhau
  - Thường đánh dấu tràn số với số không dấu
    - Cờ **OF** dùng cho số có dấu
  - Bị ảnh hưởng bởi các phép dịch và quay
  - Lệnh: **STC** / **CLC** / **CMC**: Complement Carry
- **Cờ nhớ phụ** (Auxiliary Flag)
  - Dùng trong các thao tác liên quan với số BCD (Binary Coded Decimal)

Thay đổi giá trị cờ

LAHF /SAHF

- **LAHF**: Load AH from FLAG
  - Không ảnh hưởng tới thanh ghi cờ
- **SAHF**: Store AH into FLAG
  - Thay đổi giá trị của các cờ CF, PF, AF, ZF, SF
- Giới hạn
  - Không thay đổi được cờ Overflow

Cờ điều khiển

- Dùng điều khiển hoạt động của CPU
  - Cờ hướng, Cờ bẫy, Cờ ngắt
- **Cờ hướng** (*DF: Direction Flag*),
  - Dùng trong lệnh xử lý chuỗi của CPU
    - Dùng 2 thanh ghi SI và DI xác định địa chỉ chuỗi
  - Điều khiển hướng mà chuỗi được xử lý
    - **DF = 0 (UP):** Chuỗi được xử lý từ trái sang phải
      - Sau khi xử lý xong, giá trị các thanh ghi SI, DI được tăng lên
    - **DF = 1 (DOWN):** Chuỗi được xử lý từ phải sang trang
  - Có thể thay đổi được bởi các câu lệnh
    - **CLD**: Clear direction, **STD**: Set Direction

Cờ điều khiển

- **Cờ ngắt** (*IF: Interrupt Flag*)
  - Điều khiển khả năng đáp ứng sự kiện bên ngoài
    - Một số chuỗi lệnh không cho phép bị ngắt
  - Có thể thay đổi trực tiếp trạng thái cờ
    STI: Set Interrupt, **CLI:** Clear Interrupt
- **Cờ bẫy** (TF: Trap Flag)
  - Cho phép (*khi cờ bẫy được đặt*) các chương trình gỡ rối có thể thực hiện theo từng lệnh
    - CPU thục hiện 1 lệnh và trả điều khiển cho Debugers
  - Không thể thay đổi trực tiếp giá trị cờ được
    - Thay đổi gián tiếp qua stack và thanh ghi trung gian
      - Ví dụ: PUSHF; POP AX; OR AX,….: PUSH AX; POPF

Nội dung chính

- Thanh ghi trạng thái processor
- **Các lệnh sao chép dữ liệu**
- Các lệnh số học
- Các lệnh thao tác với bit
- Các lệnh điều khiển
- Các câu lệnh với chuỗi
- Các câu lệnh khác

Toán tử **OFFSET /PTR**

- **OFFSET**
  - Khoảng cách kể từ đầu vùng nhớ của một nhãn
  - Giá trị trả về là một con trỏ
- **PTR**
  - Thay đổi kiểu ban đầu của một nhãn dữ liệu
  - BYTE PTR, WORD PTR, DWORD PTR
- **TYPE**
  - Trả về kiểu ban đầu của một nhãn

Kiểu toán hạng

- Tên thanh ghi, hằng số, tham chiếu tới bộ nhớ
- Tham chiếu tới bộ nhớ
  - Sử dụng nhãn dữ liệu

```
.Data  Val DB 10
.Code
   Mov AL, Val       ; AL ← 10
   Mov AL, [Val]     ; AL ← 10
```

  - Thanh ghi chứa vị trí/địa chỉ dữ liệu
    - Mov AX, [BX]

Câu lệnh **MOV**

MOV Dest, Source

- Ràng buộc giữa các toán hạng
  - MOV DS, Val ; Chấp nhận nếu Val DW ?
  - MOV ES, 100 ; Chuyển hằng số vào Thg đoạn
  - MOV Count, Val ; 2 toán hạng cùng là địa chỉ
  - Mov DS, ES ; 2 toán hạng cùng là Thg đoạn
- CS, EIP không thể là đích
- Chú ý tương thích về kích thước dữ liệu
  - MOV Count, BX ;← Chấp nhận nếu **Count DW ?**
  - MOV Count, BL ;← Lỗi nếu **Count DW ?**
  - MOV AX, BL ;← Operand types do not match

Câu lệnh **MOVZX, MOVSX**

- Sao chép giá trị nhỏ sang giá trị lớn
  - Từ 386 (Yêu cầu **.386**)
  - Đích phải là một thanh ghi và phải lớn hơn nguồn
  - **MOVZX:** Các bit cao gán giá trị 0
  - **MOVSX:** Sao chép bit dấu sang phần cao
- Ví dụ

```
Count DB 90h
MOVZX AX, BX    ;Operand types do not match
MOVZX AX, Count     ;AX:0090h
MOVSX AX, Count     ;AX:FF90h
```

Câu lệnh **XCHG**

XCHG Op1, Op2

- Chuyển đổi giá trị giữa 2 toán hạng
  - Một toán hạng phải là thanh ghi
  - Lệnh không phân chia được trên một số hệ thống
  - Không thay đổi thanh ghi cờ

Câu lệnh **XLAT**

XLAT

- Thay giá trị của thanh ghi AL bởi giá trị tại byte thứ **AL** trong bảng có địa chỉ trong **BX**
  - AL = MEM [DS : BX+AL]
- Không ảnh hưởng thanh ghi cờ

```
Tab DW 1234h, 5678h, 9ABCh, 0DEF0h
MOV AL, 2                ; AL=2
MOV BX, OFFSET Tab;
XLAT                     ; AL = 78h
```

Câu lệnh **LEA**

LEA Reg, Mem

- Chuyển địa chỉ một vùng nhớ vào thanh ghi thông dụng 16bit(386: 16/32bit)
  - **Reg:** Thanh ghi thông dụng 16/32(386)bit
  - **Mem:** Địa chỉ một vùng nhớ
- Không ảnh hưởng thanh ghi cờ
- Ví dụ

```
Msg DB « Hello world $ »
MOV BX, OFFSET Msg;
LEA DX, Msg
LEA AX, [BX]          ;AX = BX =DX
```


Câu lệnh **LDS, LES, LFS, LGS, LSS**

LxS Reg_{16}, Mem_{32}

- Gán giá trị cho cặp thanh ghi đoạn và thông dụng
  - **Reg:** Thanh ghi thông dụng 16/32(386)bit
  - **Mem_{32}:** Địa chỉ một vùng nhớ, double word
- Kết quả
  - $Reg_{16} = [mem_{32}]$
  - $xS = [mem_{32}+2]$
- Ghi ghú
  - LFS, LGS, LSS: có từ 386
  - Không ảnh hưởng thanh ghi cờ
- Ví dụ: LES AX, [BX]

```
Cnt DB  12h,34h,56h,
        78h,9Ah,0BCh
LEA BX,Cnt
LES EAX, [BX]

EAX: 78563412h
ES :  0BC9Ah
```

Câu lệnh **PUSH, POP**

- **PUSH** Source / **POP** Destination
  - Source: Thanh ghi, hằng số, bộ nhớ
  - Destination: Bộ nhớ, thanh ghi (trừ CS)
- **PUSHF /POPF**
  - Cất giữ và khôi phục thanh ghi cờ
- **PUSHA /POPA**
  - Cất giữ và khôi phục tất cả các thanh ghi
  - **PUSHA:** AX, CX, DX, BX, SP, BP, SI, DI
  - **POPA:** DI, SI, BP, SP, BX, DX,CX, AX

Nội dung chính

- Thanh ghi trạng thái processor
- Các lệnh sao chép dữ liệu
- **Các lệnh số học**
- Các lệnh thao tác với bit
- Các lệnh điều khiển
- Các câu lệnh với chuỗi
- Các câu lệnh khác

Câu lệnh **INC /DEC**

INC Op và DEC Op

- **OP**: Thanh ghi thông dụng hoặc biến nhớ
  - INC Op → Op = Op +1
  - DEC Op→ Op = Op -1
- Không ảnh hưởng tới thanh ghi cờ

```
MOV AX, 0FFh      ; AX = 00FF
INC AX            ; AX = 0100, AF=AC
DEC AX            ; AX = 00FF
INC AL            ; AX = 0000
```


Câu lệnh **ADD /SUB**

- Dạng lệnh
  - **ADD** Dest, Souce → Dest = Dest + Source
  - **ADC** Dest, Souce → Dest = Dest + Source +CF
  - **SUB** Dest, Souce → Dest = Dest - Source
  - **SBB** Dest, Souce → Dest = Dest - (Source+CF)
- Ghi chú
  - **Dest:** Thanh ghi thông dụng, biến nhớ
  - **Source**: Th. ghi thông dụng, hằng, bộ nhớ
  - Các toán hạng cùng kích thước 8,16,32 bit
  - Có thể ảnh hưởng tới : CF, PF, AF, ZF, SF, OF

Câu lệnh **MUL/ IMUL**

MUL Op/ **IMUL** Op

- **MUL**: Nhân không dấu
- **IMUL**: Nhân có dấu, dấu được mở rộng ra thanh ghi chứa phần cao của kết quả
- **OP**: Toán hạng, có thể là thanh ghi, biến nhớ 8/16/32 bit
  - Op 8 bit, nhân với AL, lưu kết quả trong AX
  - Op 16 bit, nhân với AX, lưu kết quả trong DX:AX
    - DX chứa phần cao, AX chứa phần thấp
  - Op 32 bit, nhân với EAX, lưu KQ trong EDX:EAX

Câu lệnh **MUL/ IMUL**

- Tác động tới cờ: CF, OF
  - **MUL:** CF/OF = 0: Nếu nửa trên của KQ bằng 0
  - **IMUL:** CF/OF = 0: Nếu bit dấu phần cao giống phần thấp. Các trường hợp khác CF/OF=1
- Ví dụ
  - MOV AL, 0FFh; AL = -1
  - MOV BL, 1
  - MUL BL ; AX = 00FF, CF/OF=0
  - IMUL BL ; AX = FFFF, CF/OF=0
  - MUL AH ; AX = FE01, CF/OF=1

Câu lệnh **DIV**

DIV Op

- **Phép** chia không dấu
- **OP**: Toán hạng, có thể là thanh ghi, biến nhớ 8/16/32 bit
  - 8 bit: số bị chia AX, thương AL, phần dư AH
  - 16 bit: số bị chia DX:AX, thương AX, phần dư DX
  - 32 bit số bị chia EDX:EAX, thương EAX, phần dư EDX
- Chú ý: nếu kết quả vượt quá phạm vi của AL/ AX ⇒ lỗi (Ví dụ: AX=FFFF, BL=10; DIV BL)

Mở rộng dấu

- **CBW** (*convert byte to word*)
  - Mở rộng dấu của AL cho AH
- **CWD** (convert word to doubleword)
  - Mở rộng dấu của AX cho DX
- **CWDE** (convert word to doubleword)
  - Mở rộng dấu của AX cho EAX
- **CDQ** (convert doubleword to quadword)
  - Mở rộng dấu của EAX cho EDX
- Ví dụ:

```
MOV AX,80h;     AL = 80
CBW         ;   AX = FF80
CWD         ;   AX = FF80, DX = FFFF
```

Câu lệnh **IDIV**

IDIV Op

- Phép chia có dấu, các tham số tương tự **DIV**
- Cần mở rộng dấu trước khi chia
- Ví dụ: Thực hiện -48/5

```
MOV AX, 00D0h ;AL = -48 /208
MOV BL, 5     ;
IDIV BL       ;AL = 51, AH = 3, tương tự DIV
MOV AX, 00D0  ;
CBW           ;AX = FFD0
IDIV BL       ;AL = F7 (-9), AH = FD (-3)
```


Câu lệnh đảo dấu **NEG**

NEG Op

- Trả về mã bù 2 của toán hạng
  - Op = 0 - Op
- OP là thanh ghi thông dụng, biến nhớ
- Ghi chú
  - Nếu Op = 0 → Op không đổi, CF = NC
  - Nếu Op = -2^{n-1} → Op không đổi: OF = OV

Nội dung chính

- Thanh ghi trạng thái processor
- Các lệnh sao chép dữ liệu
- Các lệnh số học
- **Các lệnh thao tác với bit**
- Các lệnh điều khiển
- Các câu lệnh với chuỗi
- Các câu lệnh khác

Câu lệnh logic bit AND, OR, XOR, NOT

- Dạng lệnh
  - **AND** Dest, Souce→ Dest = Dest AND Source
  - **OR** Dest, Souce→ Dest = Dest OR Source
  - **XOR** Dest, Souce → Dest = Dest XOR Source
  - **NOT** Dest → Dest = NOT Dest
- Ghi chú
  - **Dest:** Thanh ghi thông dụng, biến nhớ
  - **Source**: Th. ghi thông dụng, hằng, bộ nhớ
  - Các toán hạng cùng kích thước 8,16,32 bit
  - Ảnh hưởng tới : PF, AF, ZF, SF, CF=NC, OF=NV

Ví dụ

- Chuyển chữ thường sang chữ hoa

```
MOV AL, 'a'        ;
AND AL, 0DFh       ; ~32 = 11011111b
```

- Chuyển chữ số thập phân sang ký tự ASCII

```
MOV AL, 6
OR AL,30h          ; 48 = 0011 0000
```

- Nhảy tới một nhãn nếu một số nguyên là lẻ

```
MOV AX, VAL        ;  VAL DW ?
AND AX, 1          ;  Kiểm tra bit 0
JNZ Label          ; Nếu bit 0 là 1 thì nhảy
```


Ví dụ→Mã hóa một chuỗi

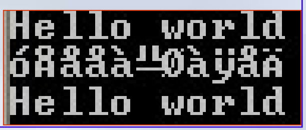
```
.model tiny        ;
.386               ;Processor 32bit
  KEY EQU  234
.data
  Buf DB "Hello world",10,13,'$'
  Len =($-Buf) - 3
.code
   .startup
    MOV AH, 9
    LEA DX,Buf
    INT 21h

    MOV CX, LEN           ;Mã hóa
    XOR SI, SI
L:  XOR Buf[SI], KEY
    INC SI
    LOOP L1
```

```
    MOV AH, 9
    LEA DX,Buf
    INT 21h

    MOV CX, LEN           ; Giải mã
    XOR SI, SI
L2:XOR Buf[SI], KEY
    INC SI
    LOOP L2

    MOV AH, 9
    LEA DX,Buf
    INT 21h
   .exit
end
```



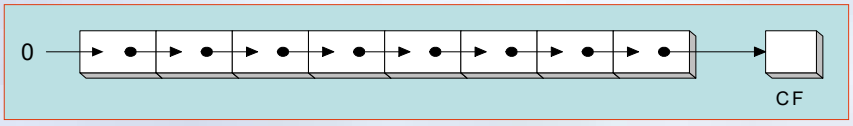
Kiểm tra giá trị một bit: TEST, BT,…

TEST Dest, Source

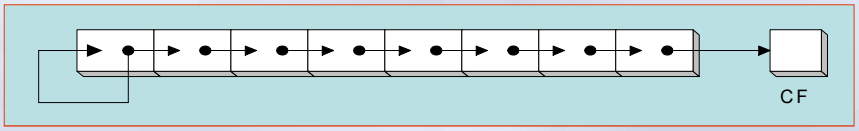
- Dest: Thanh ghi thông dụng, địa chỉ ô nhớ
- Source: Th.ghi thông dụng, biến nhớ, hằng
  - Dest, Source không cùng là địa chỉ nhớ
- Ảnh hưởng: PF, AF, ZF, SF, CF=NC, OF=NV
  - Hoạt động như lệnh **AND Dest,Source** nhưng không khi lại kết quả, chỉ ảnh hưởng tới các cờ
  - Ví dụ: TEST DX, 9; Nếu môt trong 2 bit bit 0, 3 khác 0, cờ zero bị xóa (ZF=NZ) do KQ khác 0
- Xem thêm: BT (*Bit test*), từ 386
  - VD: BT AX, 9 ; Copy bit 9 của thanh ghi AX vào cờ Carry

Phép dịch bit logic và số học

- Dịch bit logic điền vào các bit mới giá trị 0



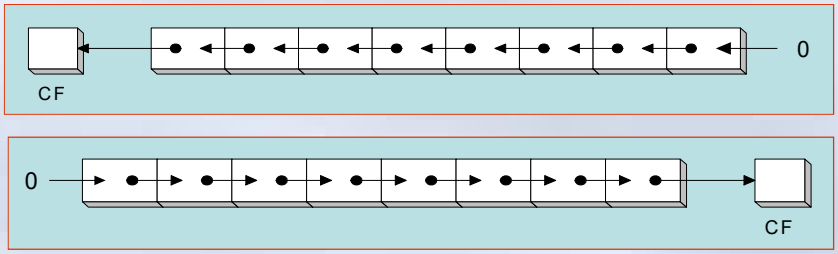
- Dịch bit số học điền vào các bit mới được tạo giá trị bit dấu của số ban đầu



Các lệnh SHL/SHR

SHL/SHR Dest, Count

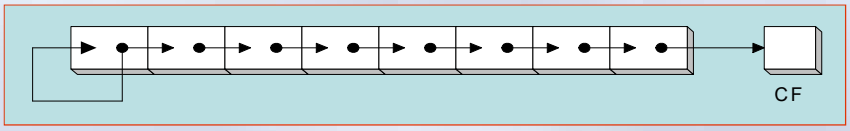
- Dịch logic Dest sang trái (SHL) hoặc sang phải (SHR) Count bit.
  - Các bit mới được điền giá trị 0
  - **Dest:** thanh ghi thông dụng hoặc biến nhớ
  - **Count:** Hằng số hoặc thanh ghi CL



Các lệnh SAL/SAR

SAL/SAR Dest, Count

- Dịch số học Dest sang trái (SHL) hoặc sang phải (SHR) Count bit.
- **SAL** tương tự SHL: Các bit mới mang g/trị 0
- **SAR** duy trì dấu của toán hạng

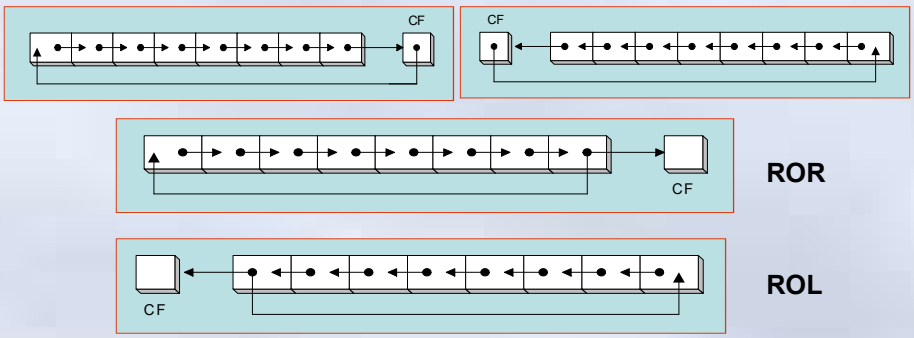


Ví dụ

```
MOV AL,6Ah      ;AL = 0110 1010
SHR AL,1        ;AL = 0011 0101; CF=NC
SHL AL,3        ;AL =1010 1000;  CF =CY
MOV AL,8Ch      ;AL = 1000 1100
SAR AL,1        ;AL = 1100 0110; CF=NC
SAR AL,3        ;AL = 1111 1000; CF=CY
```


Các lệnh RCR /RCL/ ROR /ROL

- Cú pháp tương tự các lệnh dịch bit
  - Dest: Thanh ghi thông dụng, biến nhớ
  - Count: Hằng số hoặc thanh ghi CL
- **RCR/RCL**: Quay trái, phải qua bit ờ Carry



Ví dụ

```
CLC             ; CF = NC
MOV AL, 4Ah     ;AL = 0100 1010;CF = NC
ROR AL, 2       ;AL = 1001 0010;CF = CY
ROL AL, 3       ;AL = 1001 0100; CF = NC
STC             ;CF = CY
MOV AL, 4Ch     ;AL = 0100 1100; CF=CY
RCR AL, 2       ;AL = 0101 0011 CF = NC
RCL AL, 3       ;AL = 1001 1001; CF = NC
```


Ví dụ → Nhân nhị phân

- SHL thực hiện nhân hiệu quả với lũy thừa của 2
- Ví dụ 123*36

```
    01111011      123
  ×  00100100      36
  ----------------------
    01111011      123 SHL 2
+ 01111011        123 SHL 5
  ----------------------
  0001000101001100  4428
```

EAX * 36
= EAX * (32 + 4)
= (EAX * 32)+(EAX * 4)

```
MOV EAX,123
MOV EDX, EAX
SHL EAX,5; mul by 2^5
SHL EDX,2; mult by 2^2
ADD EAX, EDX
```

Ví dụ → Hiện thị sô nhị phân 16bit

```
.DATA
   Buf DB 16 DUP (0),'$'
.code
        MOV CX, 16
        MOV SI,OFFSET Buf
L1:     SHL AX,1
        MOV BYTE PTR [SI],'0'
        JNC L2
        MOV BYTE PTR [SI],'1'
L2:     INC SI
        LOOP L1
```


Nội dung chính

- Thanh ghi trạng thái processor
- Các lệnh sao chép dữ liệu
- Các lệnh số học
- Các lệnh thao tác với bit
- **Các lệnh điều khiển**
- Các câu lệnh với chuỗi
- Các câu lệnh khác

Lệnh nhảy không điều kiện

JMP Target

- Nhảy đến một vị trí được xác định bởi Target
  - Target có thể là nhãn, thanh ghi 16 bít, biến nhớ
- **Target** là **Short_label**: Nhảy ngắn (Short jump)
  - Câu lệnh 2 byte. Một byte mã lệnh, 1 byte vị trí tương đối
  - Nhảy tới vị trí cách vị trí hiện thời -128→127byte
- **Target** là **Near_label**: Nhảy gần (near jump)
  - Câu lệnh 3 byte. Một byte mã lệnh, 2 byte vị trí tương đối
  - Nhảy tới vị trí trong cùng một đoạn lệnh,
- **Target** là **Far_label**: Nhảy xa (far jump)
  - Lệnh 5 byte, 1 byte mã lệnh, 4 byte vị trí
  - CS←Target_Segment; IP ←Target_Offset

Câu lệnh so sánh **CMP**

CMP Dest, Source

- Cho phép so sánh 2 toán hạng
  - Tương tự lệnh SUB, không thay đổi toán hạng Dest
  - Cập nhật cờ phù hợp với kết quả: Dest - Source
- Dest: Thanh ghi thông dụng, biến
- Source: Thanh ghi thông dụng, biến, hằng
- Thường dùng cờ Zero, Carry để so sánh
  - ZF = ZR: → Dest = Source (Dest-Source = 0)
  - CF= CY: → Dest < Source (*Số không dấu: trừ có nhớ*)
- So sánh số có dấu cần chú ý tới **cờ dấu** và **cờ tràn**

Câu lệnh so sánh **CMP**

Flags	Unsigned operand	Signed operand
ZF	**NZ:** Dest ≠ Source (Dest-Source ≠0)	
	ZR: Dest = Source (Dest-Source =0)	
CF	CY: Dest < Source	No meaning
	NC Dest ≥ Source	
SF	No meaning	Dest < Source –SF=PL và OF=OV hoặc –SF=NG và OF=NV
OF	No meaning	Dest >=Source –SF=PL và OF=NV hoặc –SF=NG và OF=OV

Xem thêm: **CMPXCHG Reg, Reg**

Lệnh không tách rời sử dụng trong điều độ tiến trình của HĐH

Lệnh nhảy có điều kiện

Jxxx short_label

- Nếu điều kiện thỏa mãn, nhảy tới nhãn đích, ngược lại thực hiện lệnh tiếp theo lệnh.
- Điều kiện
  - Cờ trạng thái, thanh ghi CX, kết quả so sánh
- Giới hạn phạm vi
  - Thuộc dạng short jump → 2 byte: mã lệnh + vị trí
    - Khoảng cách trong phạm vi -128 → +127
  - Với IA32: Cho phép nhảy với khoảng cách 2^{32}

Lệnh nhảy dựa trên cờ trạng thái

Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

Lệnh nhảy dựa trên phép so sánh bằng

Mnemonic	Description
JE	Jump if equal (*leftOp* = *rightOp*)
JNE	Jump if not equal (*leftOp* ≠ *rightOp*)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0

```
CMP AX, BX; So sánh AX, BX, thay đổi ZF, CF,..
JNE  lab    ; Jmp nếu AX ≠ BX
            ; ⇔ JNZ Lab; AX ≠ BX → ZF = NZ=0
```

Lệnh nhảy dựa trên so sánh không dấu

Mnemonic	Description
JA	Jump if above (if *leftOp* > *rightOp*)
JNBE	Jump if not below or equal (same as JA)
JAE	Jump if above or equal (if *leftOp* >= *rightOp*)
JNB	Jump if not below (same as JAE)
JB	Jump if below (if *leftOp* < *rightOp*)
JNAE	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if *leftOp* <= *rightOp*)
JNA	Jump if not above (same as JBE)

Nhảy tới môt nhãn nếu số **không dấu** AX lớn hơn BX

```
CMP AX,BX
JA  Larger
```


Lệnh nhảy dựa trên so sánh có dấu

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* >= *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* <= *rightOp*)
JNG	Jump if not greater (same as JLE)

Nhảy tới môt nhãn nếu số **có dấu** AX lớn hơn BX

```
CMP AX,BX
JG  Larger
```

Lệnh lặp **LOOP**

LOOP Short_Label

- Thực hiện một vòng lặp liên tục
  - CX = CX - 1
  - Nếu CX ≠ 0 nhảy tới Short_Label
- Ví dụ

```
MOV CX, 10
MOV AX, 2
Start : ADD AX,AX
Loop Start                ;Kết thúc, AX = 0800
```


Lệnh lặp **LOOP**

- Số lần lặp phụ thuộc thanh ghi CX
  - Vấn đề: vòng lặp lồng nhau ?
- Phải ghi lại CX trước khi vào vòng lặp trong

```
MOV CX,100      ;
L1:             ; Vòng lặp ngoài
  PUSH CX       ;

  MOV CX, 20    ;
  L2:           ; Vòng lặp trong

    LOOP L2     ; Lặp lại vòng lặp trong
  POP CX        ; Khôi phục CX
  LOOP L1       ; Lặp lại vòng lặp ngoài
```

Lệnh lặp **LOOPE/LOOPZ/LOOPNE/LOOPNZ**

- **LOOPE/LOOPZ** Short_Label
  - CX = CX – 1 (ECX, RCX)
  - Nhảy tới Short_Label nếu CX>0 và ZF=ZR=1
- **LOOPNE/LOOPNZ** Short_Label
  - CX = CX - 1
  - Nhảy tới Short_Label nếu CX>0 và ZF=NZ=0
- **Ghi chú**
  - Câu lệnh 2 byte: Mã lệnh lặp và vị trí tương đối

Ví du: writeInt

```
.model tiny    ;
.386           ;Processor 32bit
.data
.code
   .startup
   Mov AX,23456     ; Số in ra
   Mov BX, 10       ;
   XOR CX, CX       ;Xóa CX
   Lap:
      XOR DX, DX    ;DX=0
      DIV BX        ;DX:AX/BX
      PUSH DX       ;Cất số dư
      INC CX
      CMP AX, 0     ;KQ=0 ?
      JNZ Lap
INKQ:
      POP AX  ; Số dư <10
      ADD AX,30h
      MOV AH,0Eh
      XOR BX,BX
      INT 10h
      LOOP INKQ
   .exit
end
```

Ví du: Đưa ra tổng của dãy số

```
.model tiny      ;
.386             ;Processor 32bit
.data
  Vec DW 34, -50, 24, 57,22, -69, 20
  Len =($-Vec)/ Type Vec
.code
   .startup
        XOR AX, AX
        Mov CX, Len
        Mov BX, Offset Vec
         Sum :
           Add AX, [BX]
           Add BX, Type Vec
           Loop Sum

   Mov BX, 10
   XOR CX, CX ;Xóa thanh ghi CX
Lap:
      XOR DX, DX;
      DIV BX
      PUSH DX
      INC CX
      CMP AX, 0
      JNZ Lap
  INKQ:
      POP AX
      ADD AX,30h
      MOV AH,0Eh
      XOR BX,BX
      INT 10h
      LOOP INKQ
   .exit
end
```


Ví du: Nhập số nguyên, lưu trong AX

```
.model tiny    ;
.386           ;Processor 32bit
.data
.code
   .startup
 PUSH 0;
_Do:
   Mov AH, 0     ;Nhập 1 ký tự
   Int 16h
   Cmp AL, 13    ;Là ký tự Enter
   JE _endDo     ;Ket thuc
   Cmp al,30h    ;  <0
   JL _Do
   Cmp al,39h    ;  >9
   JG _Do
   Mov AH, 0Eh   ;Hien ký tự
   Int 10h
   SUB AL, 30h ;Chu so
   Xor BX, BX
   Mov BL, Al
   POP AX
   Mov DX, 10
   MUL DX
   ADD AX, BX
   PUSH AX
   Jmp _Do
_endDo:
POP AX
   .exit
end
```

Bài tập

1. Hãy viết chương trình nhập một số nguyên N và dãy gồm N số và in ra tổng của chúng
2. Hãy viết chương trình nhập một dãy số cho tới khi gặp số 0 và sắp xếp theo thứ tự tăng dần