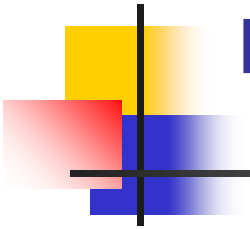
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN MẠNG VÀ CÁC HỆ THỐNG THÔNG TIN

CHƯƠNG 6

LẬP TRÌNH HỢP NGỮ



Nội dung

Giới thiệu về ngôn ngữ Assembly

Cách viết chương trình Assembler

Cấu trúc chương trình Assembler

Tập lệnh

Giới thiệu ngôn ngữ Assembly

- Là ngôn ngữ bậc thấp, dùng chính các câu lệnh trong tập lệnh của bộ xử lý
- Ưu điểm chương trình chạy nhanh và chiếm dung lượng bộ nhớ nhỏ
- Nhược điểm chương trình dài, khó kiểm soát lỗi, khó bảo trì, chương trình hợp ngữ chỉ chạy trên hệ thống máy có kiến trúc và tập lệnh tương ứng

Lý do nghiên cứu Assembly

- **Đó là cách tốt nhất để học phần cứng máy tính và hệ điều hành.**
- **Vì các tiện ích của nó.**
- **Có thể nhúng các chương trình con viết bằng ASM vào trong các chương trình viết bằng ngôn ngữ cấp cao.**

Assembler

- **Một chương trình viết bằng ngôn ngữ Assembly muốn máy tính thực hiện được ta phải chuyển thành ngôn ngữ máy.**
- Chương trình dùng để dịch 1 file viết bằng Assembly ra ngôn ngữ máy , gọi là Assembler.
- **Có các chương trình dịch:**

MASM TASM EMU8086

Lệnh máy

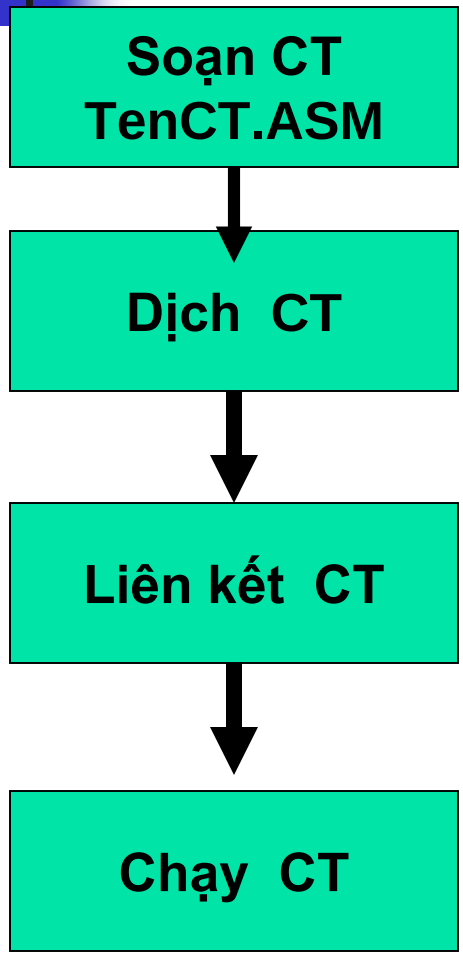
- Là 1 chuỗi nhị phân có ý nghĩa đặc biệt – nó ra lệnh cho CPU thực hiện tác vụ.
- Tác vụ đó có thể là :
  di chuyển 1 số từ vị trí nhớ này sang vị trí nhớ khác.
  Cộng 2 số hay so sánh 2 số.

0 0 0 0 0 1 0 0	Add a number to the AL register
1 0 0 0 0 1 0 1	Add a number to a variable
1 0 1 0 0 0 1 1	Move the AX reg to another reg

Lệnh máy

- Tập lệnh máy được định nghĩa trước, khi CPU được sản xuất và nó đặc trưng cho kiểu CPU .
- Ex : B5 05 là 1 lệnh máy viết dạng số hex, dài 2 byte.
- Byte đầu B5 gọi là Opcode
- Byte sau 05 gọi là toán hạng Operand
  - Ý nghĩa của lệnh B5 05 : chép giá trị 5 vào reg AL

Cách dịch chương trình Assembly



Dùng 1 phần mềm soạn thảo VB bất kỳ để soan CT Assembly như : NotePad, NC, màn hình C, Pascal ... lưu CT có phần mở rộng là .ASM

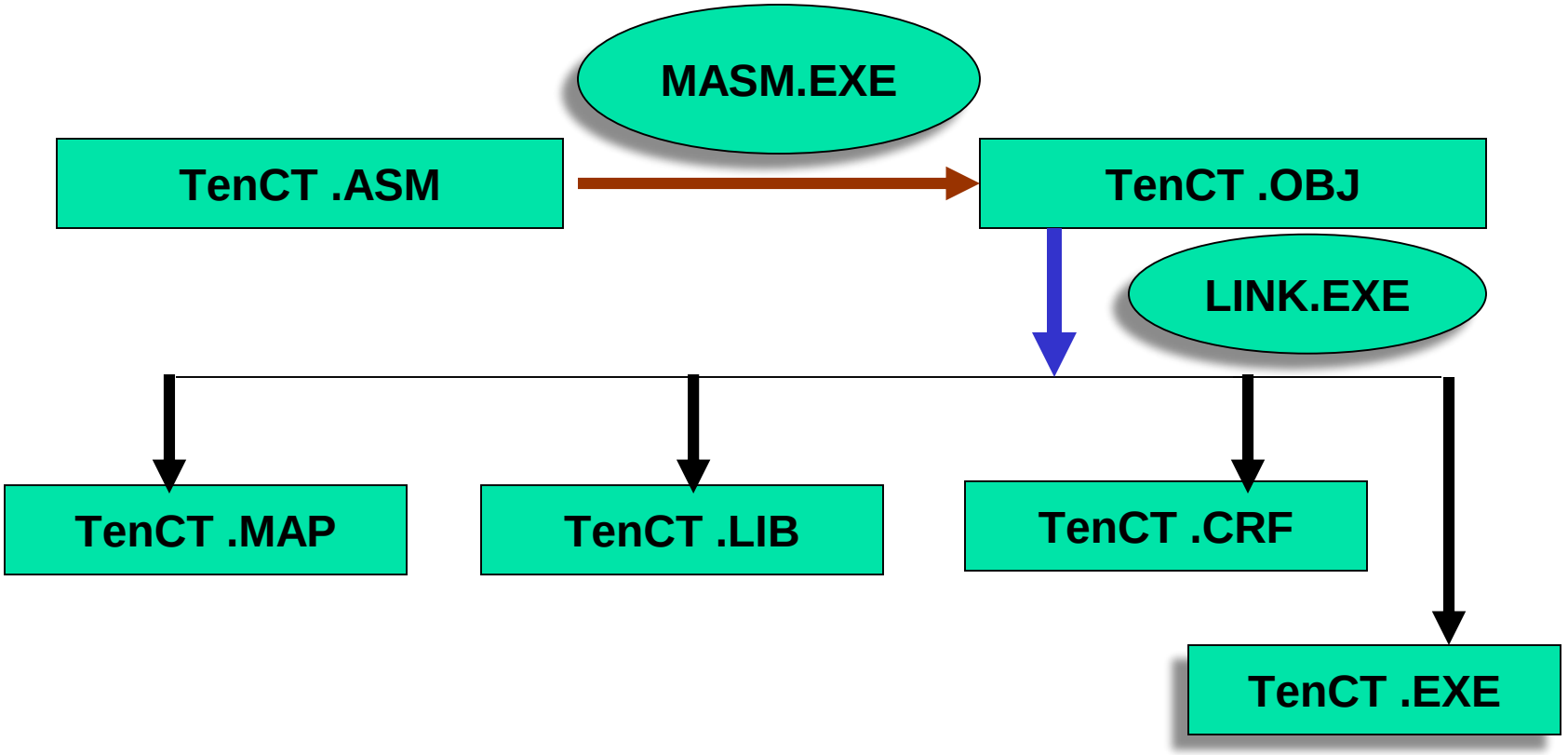
dùng MASM để dịch chương trình nguồn .ASM

- File Object.

Dùng LINK để liên kết Object tạo tập tin thực hiện .EXE

Gõ tên tập tin thực hiện .EXE từ dấu nhắc DOS để chạy

Dịch và nối kết chương trình



Khung chương trình hợp ngữ

```
.MODEL Small  ;kiểu bộ nhớ
.STACK  100    ; kích thước
.DATA
        ; khai báo biến và hằng
.CODE
   MAIN PROC
       ; khởi đầu cho DOS
        MOV  AX, @DATA
        MOV  DS,AX
             các lệnh chương trình chính
       MOV AH,4CH  ; thoát khỏi chương trình
       INT 21H
  MAIN ENDP
       các chương trình con (nếu có) để ở tại đây
END MAIN
```

Khai báo quy mô sử dụng bộ nhớ

Kiểu	Mô tả
TINY	Mã lệnh và dữ liệu gói gọn trong một đoạn
SMALL	Mã lệnh trong 1 đoạn. Dữ liệu trong 1 đoạn
MEDIUM	Mã lệnh nhiều hơn 1 đoạn. Dữ liệu trong 1 đoạn
COMPACT	Mã lệnh trong 1 đoạn. Dữ liệu trong nhiều hơn 1 đoạn
LARGE	Mã lệnh nhiều hơn 1 đoạn. Dữ liệu nhiều hơn 1 đoạn, không có mảng nào lớn hơn 64K
HUGE	Mã lệnh nhiều hơn 1 đoạn. Dữ liệu nhiều hơn 1 đoạn, các mảng có thể lớn hơn 64K

Khai báo đoạn ngăn xếp

- Khai báo vùng nhớ dùng làm ngăn xếp phục vụ cho chương trình
  - Stack kích thước

  Ví dụ:

  .Stack 100 ; stack 100B
- Nếu không khai báo thì mặc định là 1KB

Khai báo đoạn dữ liệu

- Dùng định nghĩa các biến và hằng
- Ví dụ

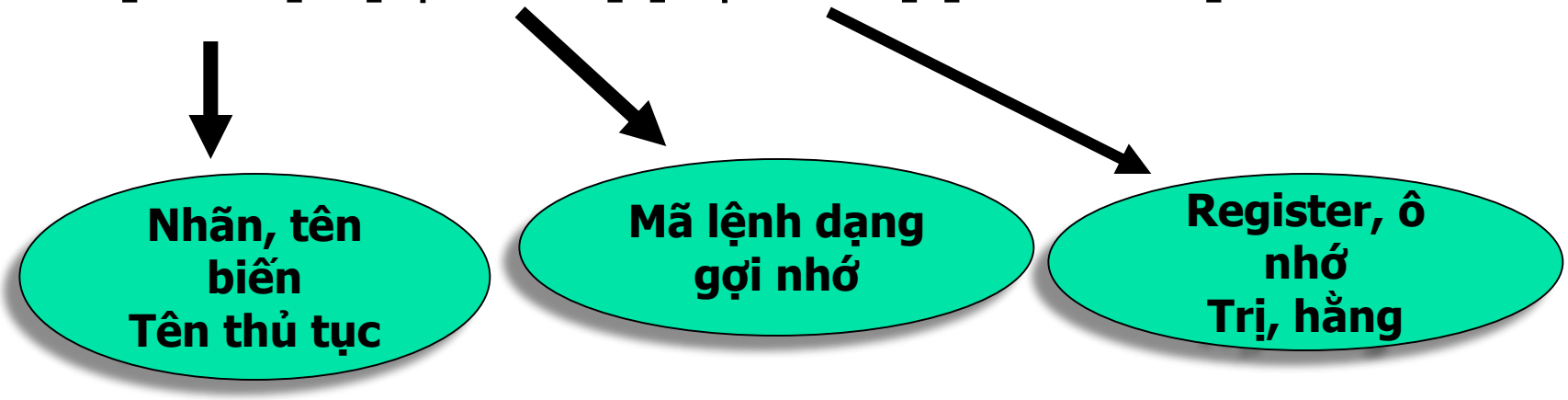
```
.DATA
        MSG   DB  'Hello!$'
        CR    DB  13
        LF    EQU 10
```

Khai báo đoạn mã

```
.Code
        Main Proc
           ; Các lệnh thân chương trình chính
            Call ten_chuong_trinh_con  ;goi ctc
         Main Endp
         Ten_chuong_trinh_con  Proc
            ; Các lệnh thân chương trình con
            RET
         Ten_chuong_trinh_con Endp
END MAIN
```

Dạng lệnh

- [name] [operator] [operand] [comment]



Ví dụ:

```
MOV CX , 0   ;gan zero
LAP:  MOV  CX, 4
LIST  DB  1,2,3,4
```

Mỗi dòng chỉ chứa 1 lệnh và mỗi lệnh phải nằm trên 1 dòng

INT 21H

- Lệnh INT <số hiệu ngắt> được dùng để gọi chương trình ngắt của DOS và BIOS.
- Muốn sử dụng hàm nào của INT 21h ta đặt function_number vào thanh ghi AH, sau đó gọi INT 21h

AH=1: nhập 1 ký tự từ bàn phím

AH=2: xuất 1 ký tự ra mà hình.

AH=9: xuất 1 chuỗi ký tự ra màn hình

AH=10: nhập 1 chuỗi ký tự từ bàn phím

AH=4CH: kết thúc chương trình .EXE

INT 21h

- **Hàm 1 : Nhập 1 ký tự**

  Input : AH = 1

  Output : AL = mã ASCCI của phím nhấn

  = 0 nếu 1 phím điều khiển được nhấn

- **Hàm 2 : Hiển thị 1 ký tự ra màn hình**

  Input : AH = 2

  DL = Mã ASCII của ký tự hiển thị hay ký tự điều khiển

Khai báo biến

- **Cú pháp : [tên biến] DB | DW |.... [trị khởi tạo]**
- **Là một tên ký hiệu dành riêng cho 1 vị trí trong bộ nhớ nơi lưu trữ dữ liệu.**
- **Offset của biến là khoảng cách từ đầu phân đoạn đến biến đó.**
- **VD : khai báo 1 danh sách aList ở địa chỉ 100 với nội dung sau :**

```
.data
aList  db “ABCD”
```

Khai báo biến

Từ gợi nhớ	Mô tả	Số byte	Thuộc tính
DB	Định nghĩa byte	1	Byte
DW	Từ	2	Word
DD	Từ kép	4	Doubleword
DQ	Từ tứ	8	Quardword
DT	10 bytes	10	tenbyte

Minh họa khai báo biến

- **Char db 'A'**
- **Num db 41h**
- **Mes db "Hello Word",'$'**
- **Array_1 db 10, 32, 41h, 00100101b**
- **Array_2 db 2,3,4,6,9**
- **Myvar db ? ; biến không khởi tạo**
- **Btable db 1,2,3,4,5**
  db 6,7,8,9,10

Toán tử DUP

Lặp lại 1 hay nhiều giá trị khởi tạo.

- VD :

  Bmem DB 50 Dup(?) ; khai báo vùng nhớ gồm 50 bytes.

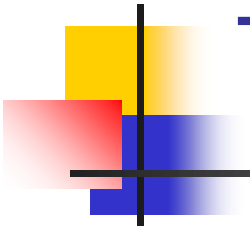
  db 4 dup (“ABC”) ;12 bytes “ABCABCABCABC”

  db 4096 dup (0) ; Vùng đệm 4096 bytes tất cả bằng 0

Khởi tạo biến

- Lưu ý : Khi khởi tạo trị là 1 số hex thì giá trị số luôn luôn bắt đầu bằng 1 ký số từ 0 đến 9. Nếu ký số bắt đầu là A.. F thì phải thêm số 0 ở đầu.
- VD :

```
Db  A6H   ; sai
Db  0A6h  ; đúng
```



Tập lệnh

Tham khảo PDF File

Chuyển ngôn ngữ cấp cao thành ngôn ngữ ASM

Giả sử A và B là 2 biến từ .

Chúng ta sẽ chuyển các mệnh đề sau trong ngôn ngữ cấp cao ra ngôn ngữ ASM .

Mệnh đề B=A

```
MOV     AX,A        ; đưa A vào AX
MOV     B,AX        ; đưa AX vào B
```

Mệnh đề A=5-A

```
MOV     AX,5        ; đưa 5 vào AX
SUB     AX,A        ; AX=5-A
MOV     A,AX        ; A=5-A
```

cách khác :

```
NEG     A           ;A=-A
ADD     A,5         ;A=5-A
```

Mệnh đề A=B-2*A

```
MOV     AX,B        ;Ax=B
SUB     AX,A        ;AX=B-A
SUB     AX,A        ;AX=B-2*A
MOV     A,AX        ;A=B-2*A
```

Cấu trúc rẽ nhánh

IF condition is true **THEN**
execute true branch statements
END IF

Hoặc

IF condition is true THEN
execute true branch statements
ELSE
execute false branch statements
END_IF

Ví dụ 1: Thay thế giá trị trên AX bằng giá trị tuyệt đối của nó

- **Thuật toán:**

```
IF AX<0
        THEN
                replace AX by - AX
END-IF
```

- **Mã hoá:**

```
;if AX<0
                CMP     AX,0
                JNL     END_IF          ; no , exit
;then
                NEG     AX              ; yes , change sign
END_IF :
```

Ví dụ 2 : giả sử AL và BL chứa ASCII code của 1 ký tự .Hãy xuất ra màn hình ký tự trước (theo thứ tự ký tự)

- **Thuật toán**

```
IF  AL<= BL  THEN
        display AL
ELSE
        display character in BL
END_IF
```

- **Mã hoá :**

```
        MOV     AH,2            ; chuẩn bị xuất ký tự
;if     AL<=BL
        CMP     AL,BL           ; AL<=BL?
        JNBE    ELSE_           ; no, display character in BL
;then
        MOV     DL,AL
        JMP     DISPLAY
ELSE_:
        MOV     DL,BL
DISPLAY:
        INT     21H
END_IF :
```

Rẽ nhánh nhiều hướng

- **Case là một cấu trúc rẽ nhánh nhiều hướng . Có thể dùng để test một thanh ghi hay , biến nào đó hay một biểu thức mà giá trị cụ thể nằm trong 1 vùng các giá trị.**
- **Cấu trúc của CASE như sau :**

```
CASE  expression
    value_1 : Statements_1
    value_2 : Statements_2
    .
    .
    value_n : Statements_n
```

Ví dụ

- **Nếu AX âm thì đặt -1 vào BX**
- **Nếu AX bằng 0 thì đặt 0 vào BX**
- **Nếu AX dương thì đặt 1 vào BX**

- **Thuật toán :**

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

Cài đặt

```
; case  AX
                    CMP     AX,0            ; test AX
                    JL      NEGATIVE        ; AX<0
                    JE      ZERO            ; AX=0
                    JG      POSITIVE        ; AX>0
    NEGATIVE:
                    MOV     BX,-1
                    JMP     END_CASE
    ZERO:
                    MOV     BX,0
                    JMP     END_CASE
    POSITIVE:
                    MOV     BX,1
                    JMP     END_CASE
    END_CASE :
```

Rẽ nhánh với một tổ hợp các điều kiện

- Đôi khi tình trạng rẽ nhánh trong các lệnh IF , CASE cần một tổ hợp các điều kiện dưới dạng :

  Condition_1 AND Condition_2

  Condition_1 OR Condition_2

Ví dụ 1: Đọc một ký tự và nếu nó là ký tự hoa thì in nó ra màn hình

- **Thuật toán :**

Read a character (into AL)

IF ('A'<= character) AND (charater <= 'Z') THEN

display character

END_IF

Cài đặt

```
; read a character
        MOV     AH,1
        INT     21H             ; character in AL
;  IF ( 'A'<= character ) AND ( charater <= 'Z')
        CMP     AL,'A'          ; char >='A'?
        JNGE    END_IF          ; no, exit
        CMP     AL,'Z           ; char <='Z'?
        JNLE    END_IF          ; no exit
; then display it
        MOV     DL,AL
        MOV     AH,2
        INT     21H
   END_IF :
```

Ví dụ 2: Đọc một ký tự , nếu ký tự đó là 'Y' hoặc 'y' thì in nó lên màn hình , ngược lại thì kết thúc chương trình .

- Thuật toán

```
Read a charcter ( into AL)
IF ( character ='Y') OR ( character='y') THEN
        dispplay it
ELSE
        terminate the program
END_IF
```

Cài đặt

```
; read a character
            MOV     AH,1
            INT     21H                 ; character in AL
;  IF (  character =‘y’ ) OR ( charater = ‘Y’)
            CMP     AL,’y’              ; char =‘y’?
            JE      THEN                ;yes , goto display it
            CMP     AL,’Y’              ; char =‘Y’?
            JE      THEN                ; yes , goto display it
            JMP     ELSE_               ;no , terminate
    THEN :
            MOV     DL,AL
            MOV     AH,2
            INT     21H
            JMP     END_IF
    ELSE_:
            MOV     AH,4CH
            INT     21h
    END_IF :
```

Cấu trúc lặp

- Một vòng lặp gồm nhiều lệnh được lặp lại , số lần lặp phụ thuộc điều kiện

Vòng **FOR**

- Lệnh LOOP có thể dùng để thực hiện vòng FOR .

  LOOP destination_label

- Số đếm cho vòng lặp là thanh ghi CX mà ban đầu nó được gán 1 giá trị nào đó . Khi lệnh LOOP được thực hiện CX sẽ tự động giảm đi 1 . Nếu CX chưa bằng 0 thì vòng lặp được thực hiện tiếp tục . Nếu CX=0 lệnh sau lệnh LOOP được thực hiện
- Lưu ý rằng vòng FOR cũng như lệnh LOOP thực hiện ít nhất là 1 lần. Do đó nếu ban đầu CX=0 thì vòng lặp sẽ làm cho CX=FFFH, tức là thực hiện lặp đến 65535 lần

Ví dụ : Dùng vòng lặp in ra 1 hàng 80 dấu '*'

```
        MOV   CX,80          ; CX chứa số lần lặp
        MOV   AH,2           ; hàm xuất ký tự
        MOV   DL,'*'         ; DL chưá ký tự '*'
TOP:
        INT   21h            ; in dấu '*'
LOOP    TOP                  ; lặp 80 lần
```

Vòng WHILE

- Vòng WHILE phụ thuộc vào 1 điều kiện .Nếu điều kiện đúng thì thực hiện vòng WHILE . Vì vậy nếu điều kiện sai thì vòng WHILE không thực hiện gì cả

Ví dụ : Viết đoạn mã để đếm số ký tự được nhập vào trên cùng một hàng .

```
                MOV   DX,0          ; DX để đếm số ký tự
                MOV   AH,1          ; hàm đọc 1 ký tự
                INT   21h           ; đọc ký tự vào AL
WHILE_:
                CMP   AL,0DH        ; có phải là ký tự CR?
                JE    END_WHILE     ; đúng , thoát
                INC   DX            ; tăng DX lên 1
                INT   21h           ; đọc ký tự vào AL
                JMP   WHILE_        ; lặp
END_WHILE:
```

6.1. Các nhóm lệnh cơ bản

6.1.1. Lệnh truyền dữ liệu (Data Transfer Instructions)

•**Mục đích:** Di chuyển dữ liệu giữa các thanh ghi, thanh ghi và bộ nhớ, hoặc thanh ghi và cổng I/O.

•**Các lệnh phổ biến:**

MOV (Move): Sao chép dữ liệu từ nguồn đến đích.

•Cú pháp: MOV đích, nguồn

•Ví dụ:

```
MOV AX, BX  ;chuyển nội dung BX vào AX
MOV CL, 10h ;chuyển giá trị 10h vào CL
MOV [BX], AL ;chuyển AL vào ô nhớ trỏ bởi BX
```

PUSH (Push onto Stack): Đẩy dữ liệu vào đỉnh của ngăn xếp (stack).

•Cú pháp: PUSH nguồn

•Ví dụ: PUSH AX

POP (Pop from Stack): Lấy dữ liệu từ đỉnh ngăn xếp ra.

•Cú pháp: POP đích

•Ví dụ: POP BX

XCHG (Exchange): Hoán đổi nội dung của hai toán hạng.

•Cú pháp: XCHG toán_hạng1, toán_hạng2

•Ví dụ: XCHG AX, BX

IN/OUT (Input/Output): Đọc dữ liệu từ cổng I/O hoặc ghi dữ liệu ra cổng I/O.

•Cú pháp: IN AL, cổng hoặc OUT cổng, AL

•Ví dụ: IN AL, 60h ;đọc từ cổng 60h vào AL.

6.1. Các nhóm lệnh cơ bản

6.1.2. Lệnh số học

- **Mục đích:** Thực hiện các phép toán số học cơ bản.
- **Các lệnh phổ biến:**
  - **ADD (Add):** Cộng hai toán hạng.

    ADD AX, BX ;AX = AX + BX
  - **SUB (Subtract):** Trừ hai toán hạng.

    SUB CX, DX ;CX = CX - DX
  - **MUL (Unsigned Multiply):** Nhân không dấu. Kết quả thường lớn hơn, lưu vào các thanh ghi mở rộng (ví dụ: AX = AL * BH, hoặc DX:AX = AX * BX).

    MUL BL
  - **DIV (Unsigned Divide):** Chia không dấu. Kết quả chia và phần dư lưu vào các thanh ghi (ví dụ: AX = AX / BL, AH = AX % BL).

    DIV BL ;AL=AX/BL, AH=AX%BL

6.1. Các nhóm lệnh cơ bản

6.1.2. Lệnh số học (tiếp)

- **INC (Increment):** Tăng toán hạng lên 1.

  INC CX

- **DEC (Decrement):** Giảm toán hạng đi 1.

  DEC DX

- **NEG (Negate):** Đổi dấu toán hạng (phép bù 2).

  NEG AX

6.1. Các nhóm lệnh cơ bản

6.1.3. Lệnh logic và bit

- **Mục đích:** Thực hiện các phép toán logic và thao tác bit.
- **Các lệnh phổ biến:**
  - **AND (Logical AND):** Phép AND từng bit.
    AND AX, 00FFh
  - **OR (Logical OR):** Phép OR từng bit.
    OR BX, 1000h
  - **NOT (Logical NOT):** Đảo tất cả các bit (phép bù 1).
    NOT CX
  - **XOR (Logical XOR):** Phép XOR từng bit.
    XOR AX, AX (thường dùng để xóa nội dung thanh ghi về 0).

6.1. Các nhóm lệnh cơ bản

6.1.3. Lệnh logic và bit (tiếp)

- **TEST (Test Bits):** Thực hiện phép AND nhưng không lưu kết quả, chỉ cập nhật cờ trạng thái. Dùng để kiểm tra bit.

```
TEST AL, 01h ;kiểm tra bit 0 của AL
```

- **SHL (Shift Left):** Dịch trái các bit, chèn 0 vào bên phải. Bit ngoài cùng bên trái vào cờ Carry.
- **SHR (Shift Right):** Dịch phải các bit, chèn 0 vào bên trái. Bit ngoài cùng bên phải vào cờ Carry.
- **SAL (Shift Arithmetic Left):** Giống SHL.
- **SAR (Shift Arithmetic Right):** Dịch phải có bảo toàn dấu (bit dấu được sao chép).
- **ROL (Rotate Left):** Xoay trái các bit (bit ngoài cùng bên trái quay về bên phải).
- **ROR (Rotate Right):** Xoay phải các bit (bit ngoài cùng bên phải quay về bên trái).

6.1. Các nhóm lệnh cơ bản

6.1.4. Lệnh điều khiển luồng

- **Mục đích:** Thay đổi trình tự thực hiện các lệnh.
- **Các lệnh phổ biến:**
  - **JMP (Jump):** Nhảy không điều kiện đến một vị trí khác trong chương trình.

    JMP LABEL_DAU
  - **CALL (Call Procedure):** Gọi một chương trình con. Lưu địa chỉ trả về vào stack.

    CALL Ten_Chuong_Con
  - **RET (Return from Procedure):** Trở về từ chương trình con. Lấy địa chỉ trả về từ stack.

    RET
  - **CMP (Compare):** So sánh hai toán hạng bằng cách thực hiện phép trừ nhưng không lưu kết quả, chỉ cập nhật cờ trạng thái (ZF, SF, OF, CF).

    CMP AX, BX

6.1. Các nhóm lệnh cơ bản

6.1.4. Lệnh điều khiển luồng (tiếp)

- **Các lệnh nhảy có điều kiện (Conditional Jumps):** Nhảy dựa trên trạng thái của các cờ sau lệnh CMP hoặc các phép toán khác.
  - JE / JZ (Jump if Equal / Jump if Zero): Nhảy nếu bằng (ZF=1).
  - JNE / JNZ (Jump if Not Equal / Jump if Not Zero): Nhảy nếu không bằng (ZF=0).
  - JG (Jump if Greater): Nhảy nếu lớn hơn (có dấu).
  - JL (Jump if Less): Nhảy nếu nhỏ hơn (có dấu).
  - JGE (Jump if Greater or Equal): Nhảy nếu lớn hơn hoặc bằng (có dấu).
  - JLE (Jump if Less or Equal): Nhảy nếu nhỏ hơn hoặc bằng (có dấu).
  - JA (Jump if Above): Nhảy nếu lớn hơn (không dấu).
  - JB (Jump if Below): Nhảy nếu nhỏ hơn (không dấu).
  - Và nhiều lệnh Jxx khác...
- **LOOP (Loop):** Giảm thanh ghi CX đi 1, nếu CX khác 0 thì nhảy đến nhãn.

  LOOP Label_Lap

6.2. Các chế độ địa chỉ

6.2.1. Khái niệm

•**Chế độ địa chỉ (Addressing Mode):** Là cách CPU xác định vị trí của toán hạng (dữ liệu) mà một lệnh cần thao tác.

•CPU 8086/8088 sử dụng kết hợp thanh ghi đoạn và địa chỉ offset để truy cập bộ nhớ.

- **Địa chỉ vật lý = (Địa chỉ đoạn * 10h) + Địa chỉ offset**

6.2. Các chế độ địa chỉ

6.2.2. Các chế độ địa chỉ phổ biến

a. Chế độ thanh ghi (Register Addressing)

- **Mô tả:** Toán hạng là một thanh ghi của CPU.
- **Ưu điểm:** Nhanh nhất vì dữ liệu nằm ngay trong CPU.
- **Ví dụ:**

```
MOV AX, BX
ADD CX, DX
```

b. Chế độ tức thời (Immediate Addressing)

- **Mô tả:** Toán hạng là một giá trị hằng số được cung cấp trực tiếp trong lệnh.
- **Ví dụ:**

```
MOV AL, 0FFh,
ADD BX, 1234h
```

c. Chế độ trực tiếp (Direct Addressing)

- **Mô tả:** Toán hạng là nội dung của một ô nhớ có địa chỉ offset được chỉ rõ trong lệnh
- **Ví dụ:**

```
MOV AL, [2000h] ;đọc nội dung ô nhớ có offset 2000h trong đoạn dữ liệu hiện hành
```

6.2. Các chế độ địa chỉ

d. Chế độ gián tiếp thanh ghi (Register Indirect Addressing)

•**Mô tả:** Địa chỉ offset của toán hạng được lưu trữ trong một thanh ghi (thường là BX, BP, SI, DI).

•**Ưu điểm:** Linh hoạt, có thể duyệt qua các vị trí bộ nhớ.

•**Ví dụ:**

```
MOV AL, [BX] ;đọc nội dung ô nhớ mà BX trỏ tới)
```

e. Chế độ cơ sở (Base Addressing)

•**Mô tả:** Địa chỉ offset được tính bằng cách cộng nội dung của thanh ghi cơ sở (BX hoặc BP) với một độ dời (displacement) có dấu.

•**Thường dùng:** Truy cập các trường trong cấu trúc dữ liệu hoặc các biến cục bộ trong stack.

•**Ví dụ:**

```
MOV AL, [BX + 5] ;đọc dữ liệu từ offset BX + 5
MOV AL, [BP + DI]
```

6.2. Các chế độ địa chỉ

f. Chế độ chỉ số (Index Addressing)

•**Mô tả:** Địa chỉ offset được tính bằng cách cộng nội dung của thanh ghi chỉ số (SI hoặc DI) với một độ dời có dấu.

•**Thường dùng:** Duyệt mảng.

•**Ví dụ:**

```
MOV AL, [SI + 10] ;đọc dữ liệu từ offset SI + 10
MOV AL, [DI + BX]
```

g. Chế độ cơ sở có chỉ số (Base-Indexed Addressing)

•**Mô tả:** Địa chỉ offset được tính bằng tổng của thanh ghi cơ sở (BX hoặc BP)
và thanh ghi chỉ số (SI hoặc DI).

•**Thường dùng:** Truy cập các phần tử trong mảng 2 chiều hoặc mảng cấu trúc.

•**Ví dụ:**

```
MOV AL, [BX + SI]
```

6.2. Các chế độ địa chỉ

h. Chế độ cơ sở có chỉ số và độ dời

- **Mô tả:** Địa chỉ offset được tính bằng tổng của thanh ghi cơ sở (BX hoặc BP), thanh ghi chỉ số (SI hoặc DI) và một độ dời có dấu.
- **Ví dụ:**

  MOV AL, [BX + SI + 20]