CHƯƠNG 3: LẬP TRÌNH HỢP NGỮ

1. Các tập tin .EXE và .COM

DOS chỉ có thể thi hành được các tập tin dạng .COM và .EXE. Tập tin .COM thường dùng để xây dựng cho các chương trình nhỏ còn .EXE dùng cho các chương trình lớn.

1.1. Tập tin .COM

- Tập tin .COM chỉ có một đoạn nên kích thước tối đa của một tập tin loại này là 64 KB.
- Tập tin .COM được nạp vào bộ nhớ và thực thi nhanh hơn tập tin .EXE nhưng chỉ áp dụng được cho các chương trình nhỏ.
- Chỉ có thể gọi các chương trình con dạng near.

Khi thực hiện tập tin .COM, DOS định vị bộ nhớ và tạo vùng nhớ dài 256 byte ở vị trí 0000h, vùng này gọi là PSP (Program Segment Prefix), nó sẽ chứa các thông tin cần thiết cho DOS. Sau đó, các mã lệnh trong tập tin sẽ được nạp vào sau PSP ở vị trí 100h và đưa giá trị 0 vào stack. Như vậy, kích thước tối đa thực sự của tập tin .COM là 64 KB – 256 byte PSP – 2 byte stack.

Tất cả các thanh ghi đoạn đều chỉ đến PSP và thanh ghi con trỏ lệnh IP chỉ đến 100h, thanh ghi SP có giá trị 0FFFEh.

1.2. Tập tin .EXE

- Nằm trong nhiều đoạn khác nhau, kích thước thông thường lớn hơn 64 KB.
- Có thể gọi được các chương trình con dạng near hay far.
- Tập tin .EXE chứa một header ở đầu tập tin để chứa các thông tin điều khiển cho tập tin.

2. Khung của một chương trình hợp ngữ

Khung của một chương trình hợp ngữ có dạng như sau:

```
TITLE       Chương trình hợp ngữ
.MODEL      Kiểu kích thước bộ nhớ     ; Khai báo quy mô sử
                                       ; dụng bộ nhớ
.STACK      Kích thước                 ; Khai báo dung lượng
                                       ; đoạn stack
.DATA                                  ; Khai báo đoạn dữ  liệu
msg   DB    'Hello$'
.CODE                                  ; Khai báo đoạn mã
main  PROC
…
CALL        Subname                    ; Gọi chương trình con
…
main  ENDP

Subname     PROC                       ; Định nghĩa chương
                                       ; trình con
…
```


```
RET
Subname     ENDP
END   main
```

❖ **Quy mô sử dụng bộ nhớ:**

Bảng 3.1:

Loại	Mô tả
Tiny	Mã lệnh và dữ liệu nằm trong một đoạn
Small	Mã lệnh trong một đoạn, dữ liệu trong một đoạn
Medium	Mã lệnh không nằm trong một đoạn, dữ liệu trong một đoạn
Compact	Mã lệnh trong một đoạn, dữ liệu không nằm trong một đoạn
Large	Mã lệnh không nằm trong một đoạn, dữ liệu không nằm trong một đoạn và không có mảng nào lớn hơn 64KB
Huge	Mã lệnh không nằm trong một đoạn, dữ liệu không nằm trong một đoạn và các mảng có thể lớn hơn 64KB

Thông thường, các ứng dụng đơn giản chỉ đòi hỏi mã chương trình không quá 64 KB và dữ liệu cũng không lớn hơn 64 KB nên ta sử dụng ở dạng Small:

```
.MODEL      SMALL
```

❖ **Khai báo kích thước stack:**

Khai báo stack dùng để dành ra một vùng nhớ dùng làm stack (chủ yếu phục vụ cho chương trình con), thông thường ta chọn khoảng 256 byte là đủ để sử dụng (nếu không khai báo thì chương trình dịch tự động cho kích thước stack là 1 KB):

```
.STACK      256
```

❖ **Khai báo đoạn dữ liệu:**

Đoạn dữ liệu dùng để chứa các biến và hằng sử dụng trong chương trình.

❖ **Khai báo đoạn mã:**

Đoạn mã dùng chứa các mã lệnh của chương trình. Đoạn mã bắt đầu bằng một chương trình chính và có thể có các lệnh gọi chương trình con (CALL).

Một chương trình chính hay chương trình con bắt đầu bằng lệnh PROC và kết thúc bằng lệnh ENDP (đây là các lệnh giả của chương trình dịch). Trong chương trình con, ta sử dụng thêm lệnh RET để trả về địa chỉ lệnh trước khi gọi chương trình con.

3. Cú pháp của các lệnh trong chương trình hợp ngữ

Một dòng lệnh trong chương trình hợp ngữ gồm có các trường (field) sau (không nhất thiết phải đầy đủ tất cả các trường):

Tên	Lệnh	Toán hạng	Chú thích
A:	MOV	AH,10h	; Đưa giá trị 10h vào thanh ghi AH
Main	PROC		

Trường tên chứa nhãn, tên biến hay tên thủ tục. Các tên nhãn có thể chứa tối đa 31 ký tự, không chứa ký tự trắng (space) và không được bắt đầu bằng số (A: hay Main:). Các nhãn được kết thúc bằng dấu '**:**'.

Trường lệnh chứa các lệnh sẽ thực hiện. Các lệnh này có thể là các lệnh thật (MOV) hay các lệnh giả (PROC). Các lệnh thật sẽ được dịch ra mã máy.

Trường toán hạng chứa các toán hạng cần thiết cho lệnh (AH,10h).

Trường chú thích phải được bắt đầu bằng dấu '**;**'. Trường này chỉ dùng cho người lập trình để ghi các lời giải thích cho chương trình. Chương trình dịch sẽ bỏ qua các lệnh nằm phía sau dấu **;**.

3.1. Khai báo dữ liệu

Khi khai báo dữ liệu trong chương trình, nếu sử dụng số nhị phân, ta phải dùng thêm chữ **B** ở cuối, nếu sử dụng số thập lục phân thì phải dùng chữ **H** ở cuối. ***Chú ý rằng đối với số thập lục phân, nếu bắt đầu bằng chữ A..F thì phải thêm vào số 0 ở phía trước.***

Ví dụ:

```
1011b       ; Số nhị phân
1111        ; Số thập phân
1011h       ; Số thập lục phân
```

3.2. Khai báo biến

Khai báo biến nhằm để chương trình dịch cung cấp một địa chỉ xác định trong bộ nhớ. Ta dùng các lệnh giả sau để định nghĩa các biến ứng với các kiểu dữ liệu khác nhau: DB (define byte), DW (define word) và DD (define double word).

VD:

```
A1      DB      1       ; Định nghĩa biến A1 dài 1 byte (chương
                        ; trình dịch sẽ dùng 1 byte trong bộ nhớ để
                        ; lưu trữ A1), trị ban đầu A1 = 1
A2      DB      ?       ; Biến A2 kiểu byte, không có giá trị gán
                        ; ban đầu
A3      DB      'A'     ; Biến kiểu ký tự
A4      DW      1       ; Định nghĩa biến A4 dài 2 byte, giá trị ban
                        ; đầu A4 = 1, ta cũng có thể dùng dấu ? để
                        ; xác định biến không cần khởi tạo giá trị ban đầu
A5      DD      1       ; Biến A5 dài 4 byte
A6      DB      1,2,3   ; Định nghĩa biến mảng (array) gồm có 3
                        ; phần tử, mỗi phần tử dài 1 byte (nghĩa là
                        ; sẽ dùng 3 byte lưu trữ) với các giá trị ban
                        ; đầu của các phần tử lần lượt là 1,2,3
A7      DB      10      DUP(0)
                        ; Khai báo biến mảng gồm 10 phần tử, mỗi
                        ; phần tử có chiều dài 1 byte với giá trị gán
                        ; ban đầu là 0
```


```
A8    DB    10    DUP(?)
                  ; Khai báo biến mảng gồm 10 phần tử, mỗi
                  ; phần tử có chiều dài 1 byte, không cần
                  ; gán giá trị ban đầu
```

Ngoài ra ta có thể dùng các toán tử DUP lồng vào nhau khi khai báo biến mảng. Giả sử ta cần khai báo mảng A9 có các giá trị gán ban đầu 1,2,3,1,1,3,2,2,1,1,3,2,2. Ta có thể thực hiện như sau:

```
      A9    DB    1,2,3,1,1,3,2,2,1,1,3,2,2
Hay:  A9    DB    1,2,3,2 DUP(1,1,3,2,2)
Hay:  A9    DB    1,2,3,2 DUP(2 DUP(1),3,2 DUP(2))
```

Đối với các biến có nhiều hơn 1 byte, byte thấp sẽ chứa ở ô nhớ có địa chỉ thấp và byte cao sẽ chứa ở ô nhớ có địa chỉ cao.

VD:

```
A10   DW    1234h
```

Biến A10 giả sử bắt đầu lưu tại địa chỉ 1000h thì ô nhớ 1000h chứa giá trị 34h còn ô nhớ 1001h chứa giá trị 12h.

Đối với biến kiểu chuỗi (string), thực chất là một mảng các ký tự, ta có thể khai báo như sau:

```
      A11   DB    'ABCD'
Hay   A11   DB    65h,66h,67h,68h
```

Sau lệnh khai báo này thì ô nhớ 1000h (giả sử biến A11 lưu trữ tại địa chỉ 1000h) chứa 'A', 1001h chứa 'B', 1002h chứa 'C' và 1003h chứa 'D'.

3.3. Khai báo hằng

Các hằng khai báo trong chương trình hợp ngữ bằng lệnh giả EQU để chương trình dễ hiểu hơn. Hằng có thể ở dạng số, ký tự hay chuỗi.

VD:

```
A12   EQU   10
A13   EQU   'AAA'
```

Sau khi sử dụng khai báo này, nếu ta dùng lệnh:

```
MOV  AH,A12
```

thì AH = 10h

```
A14   DB    'B',A13
```

thì khai báo chuỗi A14 với giá trị gán ban đầu là 'BAAA'.

4. Các toán tử trong hợp ngữ

❖ **Toán tử số học:**

Bảng 3.2:

Toán tử	Cú pháp	Mô tả
+	+bt	Số dương
-	-bt	Số âm
*	bt1*bt2	Nhân
/	bt1/bt2	Chia
mod	bt1 mod bt2	Lấy phần dư
+	bt1 + bt2	Cộng
-	bt1 – bt2	Trừ
shl	bt shl n	Dịch trái n bit
shr	bt shr n	Dịch phải n bit

Trong đó bt, bt1, bt2 là các biểu thức hằng, n là số nguyên.

```
VD:  MOV  AH,(8+1)*7/3          ; AH ← 21
     MOV  AH, 00010001b shr 2   ; AH ← 0000 0100b
     MOV  AH,00010001b shl 2    ; AH ← 0100 0100b
     MOV  AH,100 mod 3          ; AH ← 1
```

❖ **Toán tử logic:**

Bao gồm các toán tử AND, OR, NOT, XOR

```
VD:  MOV  AH,10 OR 4 AND 2      ; AH = 10
     MOV  AH, 0F0h AND 7Fh      ; AH = 70h
```

❖ **Toán tử quan hệ:**

Các toán tử quan hệ so sánh 2 biểu thức, cho giá trị true (-1) nếu điều kiện thoả và false (0) nếu không thoả.

Bảng 3.3:

Toán tử	Cú pháp	Mô tả
EQ	bt1 EQ bt2	Bằng
NE	bt1 NE bt2	Không bằng
LT	bt1 LT bt2	Nhỏ hơn
LE	bt1 LE bt2	Nhỏ hơn hay bằng
GT	bt1 GT bt2	Lớn hơn
GE	bt1 GE bt2	Lớn hơn hay bằng

❖ **Các toán tử cung cấp thông tin:**

➢ **Toán tử SEG:**

SEG bt

Toán tử SEG xác định địa chỉ đoạn của biểu thức ***bt***. ***bt*** có thể là biến, nhãn, hay các toán hạng bộ nhớ.

➢ **Toán tử OFFSET:**

OFFSET bt

Toán tử OFFSET xác định địa chỉ offset của biểu thức ***bt***. ***bt*** có thể là biến, nhãn, hay các toán hạng bộ nhớ.

```
VD:   MOV  AX,SEG A          ; Nạp địa chỉ đoạn và địa chỉ offset
      MOV  DS,AX             ; của biến A vào cặp thanh ghi
      MOV  AX,OFFSET A       ; DS:AX
```

➢ **Toán tử chỉ số [ ]: (index operator)**

Toán tử chỉ số thường dùng với toán hạng trực tiếp và gián tiếp.

➢ **Toán tử (:) (segment override operator)**

Segment:bt

Toán tử **:** quy định cách tính địa chỉ đối với segment được chỉ. ***Segment*** là các thanh ghi đoạn CS, DS, ES, SS.

Chú ý rằng khi sử dụng toán tử : kết hợp với toán tử [ ] thì ***segment:*** phải đặt ngoài toán tử [ ].

VD: Cách viết [CS:BX] là sai, ta phải viết CS:[BX]

➢ **Toán tử TYPE:**

TYPE bt

Trả về giá trị biểu thị dạng của biểu thức ***bt***.

- Nếu ***bt*** là biến thì sẽ trả về 1 nếu biến có kiểu byte, 2 nếu biến có kiểu word, 4 nếu biến có kiểu double word.
- Nếu ***bt*** là nhãn thì trả về 0FFFFh nếu bt là near và 0FFFEh nếu bt là far.
- Nếu ***bt*** là hằng thì trả về 0.

➢ **Toán tử LENGTH:**

LENGTH bt

Trả về số các đơn vị cấp cho biến ***bt***

➢ **Toán tử SIZE:**

SIZE bt

Trả về tổng số các byte cung cấp cho biến ***bt***

```
VD:   A      DD    100    DUP(?)
      MOV  AX,LENGTH A     ; AX = 100
      MOV  AX,SIZE A       ; AX = 400
```


❖ **Các toán tử thuộc tính:**

➢ **Toán tử PTR:**

Loai PTR bt

Toán tử này cho phép thay đổi dạng của biểu thức ***bt***.

- Nếu ***bt*** là biến hay toán hạng bộ nhớ thì ***Loai*** là byte, word hay dword.
- Nếu ***bt*** là nhãn thì ***Loai*** là near hay far.

```
VD:  A     DW   100   DUP(?)
     B     DD   ?
     MOV  AH,BYTE PTR A        ; Đưa byte đầu tiên trong mảng A
                               ; vào thanh ghi AH
     MOV  AX,WORD PTR B        ; Đưa 2 byte thấp trong biến B
                               ; vào thanh ghi AX
```

➢ **Toán tử HIGH, LOW:**

HIGH bt

LOW bt

Cho giá trị của byte cao và thấp của biểu thức ***bt***, ***bt*** phải là một hằng.

```
VD:  A     EQU  1234h
     MOV  AH,HIGH A            ; AH ← 12h
     MOV  AH,LOW A             ; AH ← 34h
```

5. Các cách định địa chỉ trong hợp ngữ

❖ **Toán hạng trực tiếp:**

Toán hạng trực tiếp là một biểu thức hằng xác định. Các hằng số có thể ở dạng thập phân (có dấu và không dấu), nhị phân, thập lục phân, các hằng số định nghĩa bằng lệnh EQU, …

```
VD:  MOV  AH,10
     MOV  AH,1010b
     MOV  AH,0Ah
     MOV  AH,A12
     MOV  AX,OFFSET msg
     MOV  AX,SEG msg
```

❖ **Toán hạng thanh ghi:**

Các thanh ghi có thể sử dụng trong phép định địa chỉ thanh ghi là AH, BH, CH, DH, AL, BL, CL, DL, AX, BX, CX, DX, SP, BP, SI, DI, CS, DS, ES, SS.

❖ **Toán hạng bộ nhớ:**

➢ **Trực tiếp:**

Toán hạng này xác định dữ liệu lưu trong bộ nhớ tại một địa chỉ xác định khi dịch, địa chỉ này là một biểu thức hằng (có thể kết hợp với toán tử chỉ số [ ] hay toán tử +, -, :). Thanh ghi đoạn mặc định là thanh ghi DS nhưng ta có thể dùng toán tử : để chỉ thanh ghi đoạn khác.

VD:
```
A       DW    1000h
B       DB    100   DUP(0)
MOV  AX,A              ; Chuyển nội dung của biến A vào
MOV  AX,[A]            ; thanh ghi AX
MOV  AH,B              ; Truy xuất phần tử đầu tiên của
MOV  AH,B[0]           ; mảng B
MOV  AH,B + 1          ; Truy xuất phần tử thứ hai của
MOV  AH,B[1]           ; mảng B
MOV  AH,B  + 5         ; Truy xuất phần tử thứ 6 của
MOV  AH,B[5]           ; mảng B
```

Chú ý rằng lệnh MOV AX,[1000h] sẽ chuyển giá trị 1000h vào thanh ghi AX. Nếu muốn chuyển nội dung tại ô nhớ 1000h vào thanh ghi AX thì phải dùng lệnh MOV AX,DS:[1000h] hay MOV AX,DS:1000h

- **Gián tiếp:**

Toán hạng bộ nhớ gián tiếp cho phép dùng các thanh ghi BX, BP, SI, DI để chỉ các giá trị trong bộ nhớ.

VD:
```
MOV  BX,2
MOV  SI,3
MOV  AH,B[BX]          ; Chuyển phần tử thứ 3 của mảng B
                       ; vào thanh ghi AH
MOV  AH,B[BX+1]        ; Chuyển phần tử thứ 4 của mảng B
MOV  AH,B[BX]+1        ; vào thanh ghi AH (BX + 1 = 3)
MOV  AH,B[BX+SI]       ; Chuyển phần tử thứ 6 của mảng B
MOV  AH,B[BX][SI]      ; vào thanh ghi AH
MOV  AH,[B+BX+SI]      ; BX + SI = 5
MOV  AH,[B][BX][SI]
MOV  AH,B[BX+SI+5]     ; Chuyển phần tử thứ 11 của mảng B
MOV  AH,B[BX][SI]+5    ; vào thanh ghi AH
MOV  AH,[B+BX+SI+5]    ; BX + SI + 5 = 10
```

6. Tạo và thực thi chương trình hợp ngữ

Ta có thể tạo và thực thi một chương trình hợp ngữ trên một máy PC theo các bước sau:

- Dùng một chương trình soạn thảo văn bản ***không định dạng*** (như NC) tạo một tập tin chứa chương trình hợp ngữ (gán phần mở rộng của tập tin này là .ASM, giả sử là TEMP.ASM).
- - Dùng chương trình TASM.EXE (Turbo Assembler) để dịch ra mã máy dạng .OBJ: **TASM TEMP**
- Sau khi dịch xong, ta sẽ được file TEMP.OBJ chứa các mã máy của chương trình. Để chuyển thành file thực thi, ta dùng chương trình TLINK.EXE để chuyển thành tập tin .EXE: **TLINK TEMP**
- - Nếu tập tin thực thi ở dạng .COM thì ta dùng thêm chương trình EXE2BIN.EXE: **EXE2BIN TEMP TEMP.COM**

7. Tập lệnh hợp ngữ

7.1. Nhóm lệnh chuyển dữ liệu

7.1.1. Nhóm lệnh chuyển dữ liệu đa dụng

❖ Lệnh **MOV dst,src**: chuyển nội dung toán hạng src vào toán hạng dst. Toán hạng nguồn src có thể là thanh ghi (reg), bộ nhớ (mem) hay giá trị tức thời (immed); toán hạng đích dst có thể là reg hay mem.

Lệnh MOV có thể có các trường hợp sau:

Reg8 ← reg8	MOV AL,AH
Reg16 ← reg16	MOV AX,BX
Mem8 ← reg8	MOV [BX],AL
Reg8 ← mem8	MOV AL,[BX]
Mem16 ← reg16	MOV [BX],AX
Reg16 ← mem16	MOV AX,[BX]
Reg8 ← immed8	MOV AL,04h
Mem8 ← immed8	MOV mem[BX],01h
Reg16 ← immed16	MOV AL,0F104h
Mem16 ← immed16	MOV mem[BX],0101h
SegReg ← reg16	MOV DS,AX
SegReg ← mem16	MOV DS,mem
Reg16 ← segreg	MOV AX,DS
Mem16 ← segreg	MOV [BX],DS

- Lệnh MOV không ảnh hưởng đến các cờ.
- Không thể chuyển trực tiếp dữ liệu giữa hai ô nhớ mà phải thông qua một thanh ghi

```
MOV  AX,mem1
MOV  mem2,AX
```

- Không thể chuyển giá trị trực tiếp vào thanh ghi đoạn

```
MOV  AX,1010h
MOV  DS,AX
```

- Không thể chuyển trực tiếp giữa 2 thanh ghi đoạn
- Không thể dùng thanh ghi CS làm toán hạng đích

❖ Lệnh **XCHG dst,src**: (Exchange) hoán chuyển nội dung 2 toán hạng. Toán hạng chỉ có thể là reg hay mem.

- Lệnh XCHG không ảnh hưởng đến các cờ
- Không thể dùng cho các thanh ghi đoạn

❖ Lệnh **PUSH src**: cất nội dung một thanh ghi vào stack. Toán hạng là reg16

❖ Lệnh **POP dst**: lấy dữ liệu 16 bit từ stack đưa vào toán hạng dst.

Ta có thể dùng nhiều lệnh PUSH để cất dữ liệu vào stack nhưng khi dùng lệnh POP để lấy dữ liệu ra thì phải dùng theo thứ tự ngược lại.

```
PUSH    AX
PUSH    BX
```


```
PUSH    CX
…
POP     CX
POP     BX
POP     AX
```

- Lệnh **XLAT [src]**: chuyển nội dung của ô nhớ 8 bit vào thanh ghi AL. Địa chỉ ô nhớ xác định bằng cặp thanh ghi DS:BX (nếu không chỉ ra src) hay src, địa chỉ offset chứa trong thanh ghi AL.

Lệnh XLAT tương đương với các lệnh:

```
MOV  AH,0
MOV  SI,AX
MOV  AL,[BX+SI]
```

7.1.2. Nhóm lệnh chuyển địa chỉ

- Lệnh **LEA reg16,mem16**: (Load Effective Address) chuyển địa chỉ offset của toán hạng bộ nhớ vào thanh ghi reg16.

Lệnh này sẽ tương đương với **MOV reg16, OFFSET mem16**

- Lệnh **LDS reg16,mem32**: (Load pointer using DS) chuyển nội dung bộ nhớ toán hạng mem32 vào cặp thanh ghi DS:reg16.

Lệnh LDS AX,mem tương đương với:

```
MOV  AX,mem
MOV  BX,mem+2
MOV  DS,BX
```

- Lệnh **LES reg16,mem32**: (Load pointer using ES) giống như lệnh LDS nhưng dùng cho thanh ghi ES

7.1.3. Nhóm lệnh chuyển cờ hiệu

- Lệnh **LAHF**: (Load AH from flag) chuyển các cờ SF, ZF, AF, PF và CF vào các bit 7,6,4,2 và 0 của thanh ghi AH (3 bit còn lại không đổi)

- Lệnh **SAHF**: (Store AH into flag) chuyển các bit 7,6,4,2 và 0 của thanh ghi AH vào các cờ SF, ZF, AF, PF và CF.

- Lệnh **PUSHF**: chuyển thanh ghi cờ vào stack

- Lệnh **POPF**: lấy dữ liệu từ stack chuyển vào thanh ghi cờ

7.1.4. Nhóm lệnh chuyển dữ liệu qua cổng

Mỗi I/O port giao tiếp với CPU sẽ có một địa chỉ 16 bit cho nó. CPU gởi hay nhận dữ liệu từ cổng bằng cách chỉ đến địa chỉ cổng đó. Tuỳ theo chức năng mà cổng có thể: chỉ đọc dữ liệu (input port), chỉ ghi dữ liệu (output port) hay có thể đọc và ghi dữ liệu (input/output port).

❖ Lệnh **IN**: đọc dữ liệu từ cổng và đưa vào thanh ghi AL

```
IN      AL,port8
IN      AL,DX
```

Nếu địa chỉ port chỉ có 8 bit thì có thể đưa giá trị trực tiếp vào, nếu là 16 bit thì phải thông qua thanh ghi AX.

❖ Lệnh **OUT**: ghi dữ liệu trong thanh ghi AL ra cổng

```
OUT   port8,AL
OUT   DX,AL
```

```
VD:   MOV  AL,3
      OUT  61h,AL          ; Gởi giá trị 03h ra cổng 61h
      MOV  AL,1
      MOV  DX,03F8h        ; Xuất ra cổng máy in
      OUT  DX,AL
      MOV  DX,03F8h
      IN   AL,DX           ; Đọc dữ liệu từ cổng máy in
```

7.2. Nhóm lệnh chuyển điều khiển

7.2.1. Lệnh nhảy không điều kiện JMP

JMP label
JMP reg/mem

Lệnh JMP dùng để chuyển điều khiển chương trình từ vị trí này sang vị trí khác (thay đổi nội dung cặp thanh ghi CS:IP).

7.2.2. Lệnh nhảy có điều kiện

Lệnh nhảy có điều kiện chỉ sử dụng cho các nhãn nằm trong khoảng từ –127 đến 128 byte so với vị trí của lệnh.

❖ Lệnh **JA label**: (Jump if Above)
Nếu CF = 0 và ZF = 0 thì JMP label

❖ Lệnh **JAE label**: (Jump if Above or Equal)
Nếu CF = 0 thì JMP label

❖ Lệnh **JB label**: (Jump if Below)
Nếu CF = 1 thì JMP label

❖ Lệnh **JBE label**: (Jump if Below or Equal)
Nếu CF = 1 hoặc ZF = 1 thì JMP label

❖ Lệnh **JNA label**: (Jump if Not Above)
Giống lệnh JBE

❖ Lệnh **JNAE label**: (Jump if Not Above or Equal)
Giống lệnh JB

❖ Lệnh **JNB label**: (Jump if Not Below)
Giống lệnh JAE

❖ Lệnh **JNBE label**: (Jump if Not Below or Equal)
Giống lệnh JA

❖ Lệnh **JG label**: (Jump if Greater)
Nếu SF = OF và ZF = 0 thì JMP label

❖ Lệnh **JGE label**: (Jump if Greater or Equal)
Nếu SF = OF thì JMP label

❖ Lệnh **JL label**: (Jump if Less)
Nếu SF <> OF thì JMP label

❖ Lệnh **JLE label**: (Jump if Less or Equal)
Nếu CF <> OF hoặc ZF = 1 thì JMP label

❖ Lệnh **JNG label**: (Jump if Not Greater)
Giống lệnh JLE

❖ Lệnh **JNGE label**: (Jump if Not Greater or Equal)
Giống lệnh JL

❖ Lệnh **JNL label**: (Jump if Not Less)
Giống lệnh JGE
❖ Lệnh **JNLE label**: (Jump if Not Less or Equal)
Giống lệnh JG

❖ Lệnh **JC label**: (Jump if Carry)
Giống lệnh JB

❖ Lệnh **JNC label**: (Jump if Not Carry)
Giống lệnh JNB

❖ Lệnh **JZ label**: (Jump if Zero)
Nếu ZF = 1 thì JMP label

❖ Lệnh **JE label**: (Jump if Equal)
Giống lệnh JZ

❖ Lệnh **JNZ label**: (Jump if Not Zero)
Nếu ZF = 0 thì JMP label

❖ Lệnh **JNE label**: (Jump if Equal)
Giống lệnh JNZ

❖ Lệnh **JS label**: (Jump on Sign)
Nếu SF = 1 thì JMP label

❖ Lệnh **JNS label**: (Jump if No Sign)
Nếu SF = 0 thì JMP label

❖ Lệnh **JO label**: (Jump on Overflow)
Nếu OF = 1 thì JMP label

❖ Lệnh **JNO label**: (Jump if No Overflow)
Nếu OF = 0 thì JMP label

❖ Lệnh **JP label**: (Jump on Parity)
Nếu PF = 1 thì JMP label

❖ Lệnh **JNP label**: (Jump if No Parity)
Nếu PF = 0 thì JMP label

❖ Lệnh **JCXZ label**: (Jump if CX Zero)
Nếu CX = 1 thì JMP label

7.2.3. Lệnh so sánh

CMP left(reg/mem), right(reg/mem/immed)

Lệnh CMP dùng để so sánh nội dung 2 toán hạng, kết quả chứa vào thanh ghi cờ và không làm thay đổi nội dung các toán hạng.

VD: Đoạn chương trình so sánh 2 số A và B: A >B thì nhảy đến label1, A = B thì nhảy đến label2, A < B thì nhảy đến label3.

```
MOV  AX,A
CMP  AX,B
JG   label1
JL   label2
JMP  label3
```

7.2.4. Các lệnh vòng lặp

❖ Lệnh **LOOP**:
LOOP label
Mô tả:
CX = CX – 1
Nếu CX <> 0 thì JMP label

❖ Lệnh **LOOPE**:
LOOPE label
Mô tả:
CX = CX – 1
Nếu (ZF = 1) và (CX <> 0) thì JMP label

❖ Lệnh **LOOPZ**:
Giống lệnh LOOPE

❖ Lệnh **LOOPNE**:
LOOPNE label
Mô tả:
CX = CX – 1
Nếu (ZF = 0) và (CX <> 0) thì JMP label

❖ Lệnh **LOOPNZ**:
Giống lệnh LOOPNE

7.2.5. Lệnh liên quan đến chương trình con

❖ Lệnh **CALL**:
Lệnh CALL dùng để gọi một chương trình con, có thể là near hay far.

```
CALL    label       ; Gọi chương trình con tại vị trí xác định
                    ; bởi nhãn label
CALL    reg/mem     ; Gọi chương trình con tại vị trí xác định
                    ; trong reg/mem
```

❖ Lệnh **RET**: (return)
RET [n]
RETN [n]
RETF [n]
Lệnh RET dùng để kết thúc chương trình con, điều khiển sẽ được đưa về địa chỉ trước khi gọi chương trình con. RETN để kết thúc chương trình con dạng near và RETF dùng để kết thúc chương trình con dạng far.

Trong trường hợp lệnh RET có hằng số n theo sau thì sẽ cộng với thanh ghi SP giá trị n (n phải là số chẵn). Lệnh này dùng để loại bỏ một số tham số chương trình con sử dụng ra khỏi stack.

7.3. Nhóm lệnh xử lý số học

7.3.1. Xử lý phép cộng

❖ Lệnh **ADD dst,src**:
dst ← dst + src
Toán hạng src có thể là reg, mem hay immed còn toán hạng dst là reg hay mem.

- Không thể cộng trực tiếp 2 thanh ghi đoạn
- Lệnh ADD ảnh hưởng đến các cờ sau:
  + Cờ CF: = 1 khi kết quả phép cộng có nhớ hay có mượn
  + Cờ AF: = 1 khi kết quả phép cộng có nhớ hay có mượn đối với 4 bit thấp
  + Cờ PF: = 1 khi kết quả phép cộng có tổng 8 bit thấp là một số chẵn.
  + Cờ ZF: = 1 khi kết quả phép cộng là 0.
  + Cờ SF: = 1 nếu kết quả phép cộng là một số âm
  + Cờ OF: = 1 nếu kết quả phép cộng bị sai dấu, nghĩa là vượt ra ngoài phạm vi lớn nhất hay nhỏ nhất mà số có dấu có thể chứa trong toán hạng dst.

❖ Lệnh **ADC dst, src**: (Add with Carry)

dst ← dst + src + CF

Lệnh ADC thường dùng để cộng các số lớn hơn 16 bit.

❖ Lệnh **INC dst**: (Increment)

dst ← dst + 1

Dst có thể là reg hay mem.

❖ Lệnh **AAA**: (ASCII Adjust for Addition)

Hiệu chỉnh kết quả phép cộng 2 số BCD dạng không nén (mỗi chữ số BCD lưu bằng 1 byte).

```
VD:   MOV  AX,9
      MOV  BX,3
      ADD  AL,BL          ; Kết quả là AX = 0Ch
      AAA                 ; AX = 0102h (AH = 1, AL = 2)
```

Lệnh AAA chỉ ảnh hưởng đến các cờ AF và CF, không ảnh hưởng đến các cờ còn lại.

❖ Lệnh **DAA**: (Decimal Adjust for Addition)

Hiệu chỉnh kết quả phép cộng 2 số BCD dạng nén (mỗi chữ số BCD lưu bằng 4 bit, nghĩa là 1 byte biểu diễn được các số nguyên từ 0 đến 99).

```
VD:   MOV  AX,4338h
      ADD  AL,AH        ; AX ← 437Bh
      DAA               ; AX ← 4381h (43 + 38 = 81)
```

Lệnh DAA chỉ ảnh hưởng đến các cờ AF, CF, PF, SF, ZF và không ảnh hưởng đến thanh ghi AH.

7.3.2. Xử lý phép trừ

❖ Lệnh **SUB dst,src**:

dst ← dst - src

Toán hạng src có thể là reg, mem hay immed còn toán hạng dst chỉ có thể là reg hay mem.

- Không thể trừ trực tiếp thanh ghi đoạn
- Ảnh hưởng đến các cờ AF, CF, OF, PF, SF và ZF.

❖ Lệnh **SBB dst,src**:

dst ← dst – src – CF

Lệnh ADC thường dùng để trừ các số lớn hơn 16 bit.

❖ Lệnh **DEC dst**: (decrement)

dst ← dst – 1

dst là reg hay mem. Lệnh DEC ảnh hưởng đến các cờ AF, OF, PF, SF, ZF.

❖ Lệnh **NEG dst**:

dst ← - *dst*

dst là reg hay mem.

Lệnh NEG ảnh hưởng đến các cờ:

CF = 1 nếu nội dung kết quả là số khác 0.

SF = 1 nếu nội dung kết quả là số âm khác 0.

PF = 1 nếu tổng 8 bit thấp là một số chẵn.

ZF = 1 nếu nội dung kết quả là 0.

OF = 1 nếu nội dung toán hạng dst là 80h (dạng byte) hay 8000h (dạng word).

VD: Nếu muốn thực hiện phép toán 100 – AH, ta không thể cùng lệnh:

```
SUB   100,AH
```

mà phải dùng lệnh:

```
SUB   AH,100
NEG   AH
```

❖ Lệnh **AAS**: (Ascii Adjust for Substract)

Hiệu chỉnh kết quả phép trừ 2 số BCD dạng không nén (mỗi chữ số BCD lưu bằng 1 byte). Lệnh AAS chỉ ảnh hưởng cờ AF và CF.

❖ Lệnh **DAS**: (Decimal Adjust for Substract)

Hiệu chỉnh kết quả phép trừ 2 số BCD dạng nén (mỗi chữ số BCD lưu bằng 4 bit). Lệnh AAS chỉ ảnh hưởng cờ AF và CF.

7.3.3. Xử lý phép nhân

❖ Lệnh **MUL src**:

Nếu src là reg hay mem 8 bit: AX ← AL*src

Nếu src là reg hay mem 16 bit: DX:AX ← AX*src

Lệnh MUL chỉ ảnh hưởng đến cờ CF và OF.

❖ Lệnh **IMUL src**:

Giống như lệnh MUL nhưng kết quả là số có dấu.

❖ Lệnh **AAM**: (Ascii Adjust for Multiple)

Hiệu chỉnh kết quả phép nhân 2 số BCD dạng không nén, lệnh AAM thực hiện chia AL cho 10, lưu phần thương vào AL và phần dư vào AH. Lệnh AAM ảnh hưởng đến các cờ PF, SF và ZF.

7.3.4. Xử lý phép chia

❖ Lệnh **DIV src**:

Nếu src là reg/mem 8 bit: AL ← AX DIV src và AH ← AX MOD src

Nếu src là reg/mem 16 bit: AX ← DX:AX DIV src và DX ← DX:AX MOD src

Lệnh DIV không ảnh hưởng đến các cờ nhưng xảy ra tràn trong các trường hợp sau:

- Chia cho 0

- Thương lớn hơn 256 đối với dạng 8 bit.
- Thương lớn hơn 65536 đối với dạng 16 bit.

❖ Lệnh **IDIV src**:

Giống như lệnh DIV nhưng kết quả là số có dấu. Các trường hợp tràn:

- Chia cho 0
- Thương nằm ngoài khoảng (-128,127) đối với dạng 8 bit.
- Thương nằm ngoài khoảng (-32767,32768) đối với dạng 16 bit.

❖ Lệnh **AAD**: (Ascii Adjust for Division)

Hiệu chỉnh kết quả phép chia 2 số BCD dạng không nén. Lưu ý rằng lệnh AAD phải được thực hiện trước lệnh chia. Sau khi thực hiện chia thì phải hiệu chỉnh lại dạng BCD bằng cách dùng lệnh AAM.

❖ Lệnh **CBW**: (Convert Byte to Word)

Nếu AL < 80h thì AH = 0, ngược lại AH = 0FFh

Lệnh CBW dùng để chuyển số nhị phân có dấu 8 bit thành số nhị phân có dấu 16 bit.

❖ Lệnh **CWD**: (Convert Word to Double word)

Nếu AX < 8000h thì DX = 0, ngược lại DX = 0FFFFh

Lệnh CWD dùng để chuyển số nhị phân có dấu 16 bit thành số nhị phân có dấu 32 bit chứa trong DX:AX.

7.3.5. Dịch chuyển và quay

❖ Lệnh **SHL**: (Shift Logical Left)

SHL dst,1
SHL dst,CL

Dịch trái 1 bit hay CL bit.

CF ← dst7 ← dst6 … ← dst0 ← 0

❖ Lệnh **SHR**: (Shift Logical Right)

SHR dst,1
SHR dst,CL

Dịch phải 1 bit hay CL bit.

0 → dst7 → dst6 … → dst0 → CF

❖ Lệnh **SAL**: giống SHL

❖ Lệnh **SAR**:

Giống như lệnh SHR nhưng giá trị bit dst7 không thay đổi, nghĩa là

dst7 → dst7 → dst6 … → dst0 → CF

❖ Lệnh **ROL**: (Rotate Left)

ROL dst,1
ROL dst,CL

Quay trái 1 bit hay CL bit.

CF ← dst7 ← dst6 … ← dst0 ← dst7

❖ Lệnh **ROR**: (Rotate Right)
ROR dst,1
ROR dst,CL
Quay phải 1 bit hay CL bit.
dst0 → dst7 → dst6 … → dst0 → CF

❖ Lệnh **RCL**: (Rotate though Carry Left)
RCL dst,1
RCL dst,CL
Quay trái 1 bit hay CL bit.
CF ← dst7 ← dst6 … ← dst0 ← CF

❖ Lệnh **RCR**: (Rotate though Carry Right)
RCR dst,1
RCR dst,CL
Quay phải 1 bit hay CL bit.
CF → dst7 → dst6 … → dst0 → CF

7.3.6. Các lệnh logic

❖ Lệnh **AND**:
AND dst,src
dst ← dst AND src
CF ← 0, OF ← 0
Src là reg, mem hay immed còn dst là reg, mem.

❖ Lệnh **OR**:
OR dst,src
dst ← dst OR src
CF ← 0, OF ← 0

❖ Lệnh **XOR**:
XOR dst,src
dst ← dst XOR src
CF ← 0, OF ← 0

❖ Lệnh **NOT**:
NOT dst
dst ← NOT dst
Lệnh NOT không ảnh hưởng đến các cờ.

❖ Lệnh **TEST**:
TEST dst,src
Lệnh TEST thực hiện phép toán AND 2 toán hạng nhưng chỉ ảnh hưởng đến các cờ và không ảnh hưởng đến toán tử.

7.4. Nhóm lệnh xử lý chuỗi

Bao gồm các lệnh sau:

- Lệnh **MOVS**: chuyển dữ liệu từ vùng nhớ này sang vùng nhớ khác.

+ MOVSB: chuyển 1 byte từ vị trí chỉ đến bởi SI đến vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 1, DI ← DI + 1 còn nếu DF = 1 thì SI ← SI - 1, DI ← DI - 1.

+ MOVSW: chuyển 1 word từ vị trí chỉ đến bởi SI đến vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 2, DI ← DI + 2 còn nếu DF ← 1 thì SI ← SI - 2, DI ← DI - 2.

- Lệnh **CMPS**: so sánh nội dung 2 vùng nhớ

+ CMPSB: so sánh 1 byte tại vị trí chỉ đến bởi SI và tại vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 1, DI ← DI + 1 còn nếu DF ← 1 thì SI ← SI - 1, DI ← DI - 1.

+ CMPSW: so sánh 1 word tại vị trí chỉ đến bởi SI và tại vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 2, DI ← DI + 2 còn nếu DF = 1 thì SI ← SI - 2, DI ← DI - 2.

- Lệnh **SCAS**: tìm một phần tử trong vùng nhớ, địa chỉ vùng nhớ xác định bằng cặp thanh ghi ES:DI, giá trị cần tìm đặt trong thanh ghi AL, nếu tìm thấy thì ZF = 1. Giá trị của DI và SI thay đổi giống như trên.

- Lệnh **LODS**: đưa một byte hay word có địa chỉ xác định bởi cặp thanh ghi DS:SI vào thanh ghi AL hay AX. Giá trị của DI và SI thay đổi giống như trên.

- Lệnh **STOS**: chuyển nội dung của AL hay AX vào vùng nhớ xác định bởi cặp thanh ghi ES:DI. Giá trị của DI và SI thay đổi giống như trên.

8. Các cấu trúc cơ bản trong lập trình hợp ngữ

8.1. Cấu trúc tuần tự

Cấu trúc tuần tự là cấu trúc đơn giản nhất. Trong cấu trúc tuần tự, các lệnh được sắp xếp tuần tự, lệnh này tiếp theo lệnh kia.

Lệnh 1
Lệnh 2
…
Lệnh n

VD: Cộng 2 giá trị của thanh ghi BX và CX, rồi nhân đôi kết quả, kết quả cuối cùng chứa trong AX

```
MOV  AX,BX
ADD  AX,CX      ; Cộng BX với CX
SHL  AX,1       ; Nhân đôi
```


8.2. Cấu trúc IF – THEN, IF – THEN – ELSE

IF Điều kiện THEN Công việc
IF Điều kiện THEN Công việc1 ELSE Công việc2
VD: Gán BX = |AX|

```
            CMP  AX,0        ; AX > 0?
            JNL  DUONG       ; AX dương
            NEG  AX          ; Nếu AX < 0 thì đảo dấu
DUONG:      MOV  BX,AX
NEXT:
```

VD: Gán CL giá trị bit dấu của AX

```
            CMP  AX,0        ; AX > 0?
            JNS  AM          ; AX âm
            MOV  CL,1        ; CL = 1 (AX dương)
            JMP  NEXT
        AM: MOV  CL,0        ; CL = 0 (AX âm)
        NEXT:
```

8.3. Cấu trúc CASE

CASE Biểu thức
Giá trị 1: Công việc 1
Giá trị 2: Công việc 2
…
Giá trị n: Công việc n
END

VD: Nếu AX > 0 thì BH = 0, nếu AX < 0 thì BH = 1. Ngược lại BH = 2

```
        CMP  AX,0
             JL   AM
        JE   KHONG
             JG   DUONG
DUONG:  MOV  BH,0
        JMP  NEXT
AM:     MOV  BH,1
        JMP  NEXT
KHONG:  MOV  BH,2
NEXT:
```

8.4. Cấu trúc FOR

FOR Số lần lặp DO Công việc

VD: Cho vùng nhớ M dài 200 bytes trong đoạn dữ liệu, chương trình đếm số chữ A trong vùng nhớ M như sau:

```
            MOV  CX,200          ; Đếm 200 bytes
            MOV  BX,OFFSET M     ; Lấy địa chỉ vùng nhớ
            XOR  AX,AX           ; AX = 0
```


```
NEXT:   CMP  BYTE PTR [BX],'A'; So sánh với chữ A
        JNZ  ChuA              ; Nếu không phải là chữ A thì tiếp
        INC  AX                ; tục, ngược lại thì tăng AX
ChuA:   INC  BX
        LOOP       NEXT
```

8.5. Cấu trúc lặp WHILE

WHILE Điều kiện DO Công việc

VD: Chương trình đọc vùng nhớ bắt đầu tại địa chỉ 1000h vào thanh ghi AH, đến khi gặp ký tự '$' thì thoát:

```
        MOV  BX,1000h
CONT:   CMP  AH,'$'
        JZ   NEXT
        MOV  AH,DS:[BX]
        JMP  CONT
NEXT:
```

8.6. Cấu trúc lặp REPEAT

REPEAT Công việc UNTIL Điều kiện

VD: Chương trình đọc vùng nhớ bắt đầu tại địa chỉ 1000h vào thanh ghi AH, đến khi gặp ký tự '$' thì thoát:

```
        MOV  BX,1000h
CONT:   MOV  AH,DS:[BX]
        CMP  AH,'$'
        JZ   NEXT
        JMP  CONT
NEXT:
```

9. Các ngắt của 8086

Bảng 3.4:

Vector ngắt	Công dụng
00h	CPU: tác động khi chia cho 0
01h	CPU: chương trình thực thi từng bước
02h	CPU: ngắt không che được
03h	CPU: tạo điểm dừng chochương trình
04h	CPU: tác động khi kết quả số học tràn
05h	Tác động khi nhấn Print Screen
06h - 07h	Dành riêng
08h	Tác động bởi nhịp đồng hồ (18.2 lần/s)
09h	Tác động khi có phím nhấn
0Ah	Dành riêng
0Bh - 0Ch	Tác động phần cứng liên lạc nối tiếp

0Dh	Đĩa cứng
0Eh	Đĩa mềm
0Fh	Máy in
10h	BIOS: màn hình
11h	BIOS: xác định cấu hình máy tính
12h	BIOS: thông báo kích thước RAM
13h	BIOS: gọi các phục vụ đĩa cứng/mềm
14h	BIOS: giao tiếp nối tiếp
15h	BIOS: truy xuất cassette hay mở rộng ngắt
16h	BIOS: xuất / nhập bàn phím
17h	BIOS: máy in
18h	Xâm nhập ROM basic
19h	BIOS: khởi động máy tính
1Ah	BIOS: ngày / giờ hệ thống
1Bh	Lấy điều khiển từ ngắt bàn phím
1Ch	Lấy điều khiển từ ngắt đồng hồ (sau int 08h)
1Dh	Địa chỉ bảng tham số màn hình
1Eh	Địa chỉ bảng tham số đĩa
1Fh	Địa chỉ bộ mã ký tự
20h	DOS: kết thúc chương trình
21h	DOS: các chức năng DOS
22h	Địa chỉ cần chuyển khi kết thúc chương trình
23h	Địa chỉ cần chuyển khi gặp Ctrl – Break
24h	Địa chỉ cần chuyển khi gặp lỗi
25h	DOS: đọc đĩa cứng / mềm
26h	DOS: ghi đĩa cứng / mềm
27h	DOS: chấm dứt chương trình và thường trú
28h – 3Fh	Dành riêng cho DOS
40h	BIOS: các chức năng đĩa mềm
41h	Bảng thông số đĩa cứng thứ nhất
42h – 45h	Dành riêng
46h	Bảng thông số đĩa cứng thứ hai
47h – 49h	Định nghĩa do người sử dụng
4Ah	Giờ báo hiệu (chỉ trong AT)
4Bh – 67h	Định nghĩa do người sử dụng
68h – 6Fh	Không sử dụng
70h	Đồng hồ thời gian thực (chỉ trong AT)
71h – 7Fh	Dành riêng
80h – 85h	Dành riêng
86h – F0h	Sử dụng bởi chương trình thông dịch BASIC
F1h – FFh	Không sử dụng

9.1. Ngắt 21h

❖ **Hàm 01h**: nhập một ký tự từ bàn phím và hiện ký tự nhập ra màn hình. Nếu không có ký tự nhập, hàm 01h sẽ đợi cho đến khi nhập.

- Gọi: AH = 01h
- Trả về: AL chứa mã ASCII của ký tự nhập

```
MOV  AH,01h
INT  21h          ; AL chứa mã ASCII của ký tự nhập
```

❖ **Hàm 02h**: xuất một ký tự trong thanh ghi DL ra màn hình tại vị trí con trỏ hiện hành

- Gọi AH = 02h, DL = mã ASCII của ký tự
- Trả về: không có

```
MOV  AH,02h
MOV  DL,'A'
INT  21h
```

❖ **Hàm 08h**: giống hàm 01h nhưng không hiển thị ký tự ra màn hình

❖ **Hàm 09h**: xuất một chuỗi ký tự ra màn hình tại vị trí con trỏ hiện hành, địa chỉ chuỗi được chứa trong DS:DX và phải được kết thúc bằng ký tự $

- Gọi AH = 09h, DS:DX = địa chỉ chuỗi
- Trả về: không có

```
.DATA
Msg  DB   'Hello$'
…
MOV  AH,09h
LEA  DX,Msg
INT  21h
```

❖ **Hàm 0Ah**: nhập một chuỗi ký tự từ bàn phím (tối đa 255 ký tự), dùng phím ENTER kết thúc chuỗi

- Gọi AH = 0Ah, DS:DX = địa chỉ lưu chuỗi
- Trả về: không có

Chuỗi phải có dạng sau:

- Byte 0: Số byte tối đa cần đọc (kể cả ký tự Enter)
- Byte 1: số byte đã đọc
- Byte 2: lưu các ký tự đọc

```
.DATA
Msg  DB   101          ; Đọc tối đa 100 ký tự
     DB   ?
     DB   101  DUP(?)
     …
     MOV  AH,0Ah
     LEA  DX,Msg
     INT  21h
```

❖ **Hàm 4Ch**: kết thúc chương trình

```
MOV  AH,4Ch
INT  21h
```


9.2. Ngắt 10h

❖ **Xoá màn hình:**

- Gọi AX = 02h
- Trả về: không có

```
MOV  AX,02h
INT  10h
```

❖ **Chuyển toạ độ con trỏ:**

- Gọi AH = 02h, DH = dòng, DL = cột

```
MOV  AH,02h
MOV  DX,0F15h
INT  10h
```

10. Truyền tham số giữa các chương trình

Trong lập trình, một vấn đề ta cần quan tâm là truyền tham số giữa chương trình chính và chương trình con. Để thực hiện truyền tham số, ta có thể dùng các cách sau đây:

- Truyền tham số qua thanh ghi
- Truyền tham số qua ô nhớ (biến)
- Truyền tham số qua ô nhớ do thanh ghi chỉ đến
- Truyền tham số qua stack

10.1. Truyền tham số qua thanh ghi

Ta thực hiện truyền tham số qua thanh ghi bằng cách: một chương trình con sẽ đưa giá trị vào thanh ghi và chương trình con khác sẽ xử lý giá trị trên thanh ghi đó.

VD: Cộng giá trị tại 2 ô nhớ 1000h và 1001h, kết quả chứa trong 1002h (bye cao) và 1003h (byte thấp).

```
.MODEL    SMALL
.STACK    100h
.CODE
main  PROC
      MOV       AX,@DATA
      MOV       DS,AX
      MOV       BYTE PTR DS:[1000h],10h      ; Đưa giá trị vào
      MOV       BYTE PTR DS:[1001h],0FFh     ; các ô nhớ
      CALL      Read
      CALL      Sum
      Mov       AH,4Ch
      INT       21h
main  ENDP
Read  PROC                    ; Đọc dữ liệu vào thanh ghi AX
      MOV       AH,DS:[1000h]
      MOV       AL,DS:[1001h]
      RET
Read  ENDP                    ; Xử lý dữ liệu tại thanh ghi AX
```


```
Sum   PROC
      ADD       AH,AL
      JZ        next
      MOV       DS:[1003h],1
next: MOV       DS:[1002h],AH
RET
Sum   ENDP
END   main
```

10.2. Truyền tham số qua ô nhớ (biến)

Quá trình truyền tham số cũng giống như trên nhưng thay vì thực hiện thông qua thanh ghi, ta sẽ thực hiện thông qua các ô nhớ.

VD: Cộng giá trị tại 2 ô nhớ m1 và m2, kết quả chứa trong m3 (bye cao) và m4 (byte thấp).

```
.MODEL    SMALL
.STACK    100h
.DATA
      m1    db    ?
      m2    db    ?
      m3    db    ?
      m4    db    ?
.CODE
main  PROC
      MOV       AX,@data
      MOV       DS,AX
      MOV       m1,10h      ; Đưa giá trị vào
      MOV       m2,0FFh     ; các ô nhớ
      CALL      Sum
      MOV       AH,4Ch
      INT       21h
main  ENDP
Sum   PROC
      MOV       m4,0
      MOV       AH,m1
      ADD       AH,m2
      JNC       next
      MOV       m4,1
next: MOV       m3,AH
RET
Sum   ENDP
END   main
```

10.3. Truyền tham số qua ô nhớ do thanh ghi chỉ đến

Trong cách truyền tham số này, ta dùng các thanh ghi SI, DI, BX để chỉ địa chỉ offset của các tham số còn thanh ghi đoạn mặc định là DS.

VD: Cộng giá trị tại 2 ô nhớ m1 và m2, kết quả chứa trong m3 (bye cao) và m4 (byte thấp).

```
.MODEL    SMALL
.STACK    100h
.DATA
        m1    db    ?
        m2    db    ?
        m3    db    ?
        m4    db    ?
.CODE
main    PROC
        MOV       AX,@data
        MOV       DS,AX
        LEA       SI,m1
        LEA       DI,m2
        LEA       BX,m3
        MOV       [SI],10h      ; Đưa giá trị vào
        MOV       [DI],0FFh     ; các ô nhớ
        CALL      Sum
        MOV       AH,4Ch
        INT       21h
main    ENDP
Sum     PROC
        MOV       AL,[SI]
        ADD       AL,[DI]
        JZ        next
        MOV       [BX+1],1
next:   MOV       [BX],AL
RET
Sum     ENDP
END     main
```

10.4. Truyền tham số qua stack

Trong phương pháp truyền tham số này, ta dùng stack làm nơi chứa các tham số cần truyền thông qua các tác vụ PUSH và POP.

VD: Cộng giá trị tại 2 ô nhớ m1 và m2, kết quả chứa trong m3 (byte cao) và m4 (byte thấp).

```
.MODEL    SMALL
.STACK    100h
.DATA
        m1    dw    ?
        m2    dw    ?
        m3    dw    ?
        m4    dw    ?
.CODE
main    PROC
        MOV       AX,@data
```


```
        MOV     DS,AX
        LEA     SI,m1
        LEA     DI,m2
        MOV     [SI],1234h          ; Đưa giá trị vào
        MOV     [DI],0FEDCh         ; các ô nhớ
        PUSH    m1                  ; Đưa vào stack
        PUSH    m2
        CALL    Sum
        POP     m3                  ; Lấy kết quả đưa vào stack
        POP     m4
        MOV     AH,4Ch
        INT     21h
main    ENDP
Sum     PROC
        POP     DX    ; Lưu lại địa chỉ trả về của lệnh CALL
        POP     AX    ; Lấy dữ liệu từ stack
        POP     BX
        ADD     AX,BX
        JNC     next
        PUSH    1
next:   PUSH    AX
        PUSH    DX    ; Trả lại địa chỉ trở về của lệnh CALL
RET
Sum     ENDP
END     main
```

11. Các ví dụ minh hoạ

11.1. In chuỗi ký tự ra màn hình

```
.MODEL  SMALL
.STACK  100h
.DATA
        msg     DB      'Hello$'
.CODE
main    PROC
        MOV     AX,@DATA        ; Khởi động thanh ghi DS
        MOV     DS,AX
        MOV     AX,02h          ; Xoá màn hình
        INT     10h
        MOV     AH,02h          ; Chuyển toạ độ con trỏ
        MOV     DX,0C15h        ; đến dòng 12 (0Ch) và cột 21 (15h)
        INT     10h
        LEA     DX,msg          ; Địa chỉ thông điệp
        MOV     AH,09h          ; In thông điệp ra màn hình
        INT     21h
        MOV     AH,4Ch          ; Kết thúc chương trình
        INT     21h
main    ENDP
```


```
END   main
```

11.2. In chuỗi ký tự ra màn hình tại toạ độ nhập vào

```
.MODEL      SMALL
.STACK      100h
.DATA
        msg     DB      'Hello$'
        msg1    DB      'Nhap vao toa do:$'
        Crlf    DB      0Dh,0Ah,'$'
        Td      DB      3
                DB      ?
                DB      3       DUP(?)
.CODE
main    PROC
        MOV     AX,@DATA
        MOV     DS,AX               ; Khởi động thanh ghi DS
        MOV     AX,02h
        INT     10h                 ; Xóa màn hình
        LEA     DX,msg1
        MOV     AH,09h              ; In thông điệp
        INT     21h
        CALL    Nhap                ; Nhập dòng
        MOV     CL,AL
        LEA     DX,Crlf             ; Xuống dòng
        MOV     AH,09h
        INT     21h
        CALL    Nhap                ; Nhập cột
        MOV     CH,AL
        MOV     AH,02h              ; Chuyển tọa độ con trỏ
        MOV     DX,CX
        INT     10h
        LEA     DX,msg
        MOV     AH,09h              ; In ra màn hình
        INT     21h
        MOV     AH,4Ch              ; Kết thúc chương trình
        INT     21h
main    ENDP
Nhap    PROC
        MOV     AH,0Ah              ; Nhập vào
        LEA     DX,Td
        INT     21h
        LEA     BX,Td               ; Lấy chữ số hàng chục
        MOV     AL,DS:[BX+2]
        SUB     AL,'0'              ; Chuyển từ dạng ký tự sang dạng số
        MOV     BL,10
        MUL     BL                  ; Nhân số hàng chục với 10
        PUSH    AX
        LEA     BX,Td               ; Lấy chữ số hàng đơn vị
```


```
        MOV  AL,DS:[BX+3]
        SUB  AL,'0'
        POP  BX
        ADD  AL,BL
        RET
Nhap    ENDP
END     main
```

11.3. Cộng 2 số nhị phân dài 5 byte

```
.MODEL      SMALL
.STACK      100h
.DATA
        m1      DB      00h,08h,10h,13h,24h,00h
        m2      DB      0FFh,0FCh,0FAh,0F0h,0F1h,00h;
        m3      DB      6       DUP(0)
.CODE
main    PROC
        MOV  AX,@DATA
        MOV  DS,AX              ; Khởi động thanh ghi DS
        LEA  SI,m1
        LEA  DI,m2
        LEA  BX,m3
        MOV  CX,6
        XOR  AL,AL
next:   MOV  AL,[SI]
        ADC  AL,[DI]
        MOV  [BX],AL
        INC  BX
        INC  SI
        INC  DI
        LOOP next
        MOV  AH,4Ch
        INT  21h
main    ENDP
END     main
```

11.4. Nhập một chuỗi ký tự và chuyển chữ thường thành chữ hoa

```
.MODEL      SMALL
.STACK      100h
.DATA
        m1      DB      81
                DB      ?
                DB      81      DUP(?)
        m2      DB      'Chuoi da doi:$'
.CODE
main    PROC
        MOV  AX,@DATA
```


```
        MOV  DS,AX          ; Khởi động thanh ghi DS
        MOV  ES,AX
        LEA  DX,m1
        MOV  AH,0Ah         ; Nhập chuỗi
        INT  21h
        LEA  SI,m1          ; Lấy địa chỉ chuỗi
        ADD  SI,2
        MOV  DI,SI          ; Chuỗi nguồn và đích trùng nhau
Next:   LODSB               ; Lấy ký tự
        CMP  AL,0Dh         ; Nếu là ký tự Enter thì kết thúc
        JE   quit
        CMP  AL,'a'         ; Nếu ký tự nhập không phải là ký tự
        JB   cont           ; thường từ 'a' đến 'z' thì bỏ qua
        CMP  AL,'z'
        JA   cont
        SUB  AL,20h         ; Chuyển ký tự thường thành ký tự hoa
        STOSB               ; Lưu ký tự vừa chuyển
        DEC  DI             ; Nếu là ký tự thường thì dùng lệnh STOSB
                            ; nên DI tăng lên 1, ta phải giảm DI
cont:   INC  DI             ; Tăng lên ký tự kế
        JMP  next
quit:   MOV  AL,'$'
        STOSB
        MOV  AX,02h         ; Xóa màn hình
        INT  10h
        LEA  DX,m2
        MOV  AH,09h
        INT  21h
        LEA  DX,m1+2
        MOV  AH,09h
        INT  21h
        MOV  AH,4Ch
        INT  21h
main    ENDP
END     main
```