LỜI NÓI ĐẦU

Công việc lập trình trong lĩnh vực công nghệ thông tin có thể chia làm hai dạng: lập trình hệ thống (system programming) và lập trình ứng dụng (application development). Cuốn sách Lập trình Hệ thống và điều khiển thiết bị dành cho đối tượng thứ nhất : những người quan tâm tìm hiểu về lập trình hệ thống. Tuy nhiên, nó là tài liệu tham khảo có ích cho những người đang học lập trình nói chung.

Lập trình hệ thống là một công việc thú vị. Người lập trình sử dụng ngôn ngữ bậc thấp như hợp ngữ (assembly language) để lập trình trực tiếp với phần cứng và điều khiển các thiết bị ngoại vi bằng các chương trình hợp ngữ cấp thấp chạy trên bộ vi xử lý với cấu hình đòi hỏi thấp. Việc lập trình này dựa trên chính các dịch vụ hỗ trợ bởi các nhà sản xuất phần cứng hoặc các dịch vụ ở mức hệ điều hành hay dưới hệ điều hành như BIOS.

Nội dung của cuốn giáo trình Lập trình hệ thống và điều khiển thiết bị được chia thành 4 chương. Mỗi chương bao gồm các nội dung cơ bản, tóm tắt chương, các câu hỏi và bài tập cho mỗi chương.

Chương 1: trình bày về vấn đề liên quan đến bộ vi xử lý 8088 : kiến trúc, chức năng các thành phần và tập lệnh. Ngoài ra, chương 11 trong các ngắt được sử dụng phổ biến trong lập trình hệ thống- ngắt 21h của hệ điều hành DOS cũng được giới thiệu trong chương này.

Chương 2: trình bày về các vấn đề liên quan đến lập trình hợp ngữ: cách thức viết và thực hiện một chương trình, cách thức cài đặt các cấu trúc lập trình trong hợp ngữ và các vấn đề liên quan đến chương trình con và macro, cách viết các chương trình tổ chức trên nhiều file, cách liên kết chương trình hợp ngữ với ngôn ngữ bậc cao và chương trình con ngắt.

Chương 3: giới thiệu về công cụ gỡ rối debug, chương trình mô phỏng Emu 8086. Sau cùng, chúng tôi giới thiệu một công cụ để phát triển các ứng dụng hợp ngữ trên môi trưoowfng Windows 32 bít gần đây đó là RadASM .

Chương 4: Trình bày về lập trình phối ghép và điều khiển thiết bị ngoại. Các thiết bị ngoại vi điển hình như : lập trình điều khiển modem, bàn phím, màn hình và ổ đĩa. Ở mỗi thiết bị đều có những phần giới thiệu về thiết bị, các dịch vụ hỗ trợ thiết bị ở mức BIOS và hệ điều hành đi kèm với các chương trình ví dụ minh họa.

Do thời gian có hạn và kinh nghiệm còn hạn chế, cuốn giáo trình sẽ không tránh khỏi các sai sót. Tác giả biên soạn rất mong nhận được ý kiến đóng góp từ các độc giả.

Mọi ý kiến góp ý xin gửi về email : pcuongcntt@yahoo.com

Xin chân thành cảm ơn!

Hà Nội, tháng 11/2006

Tác giả

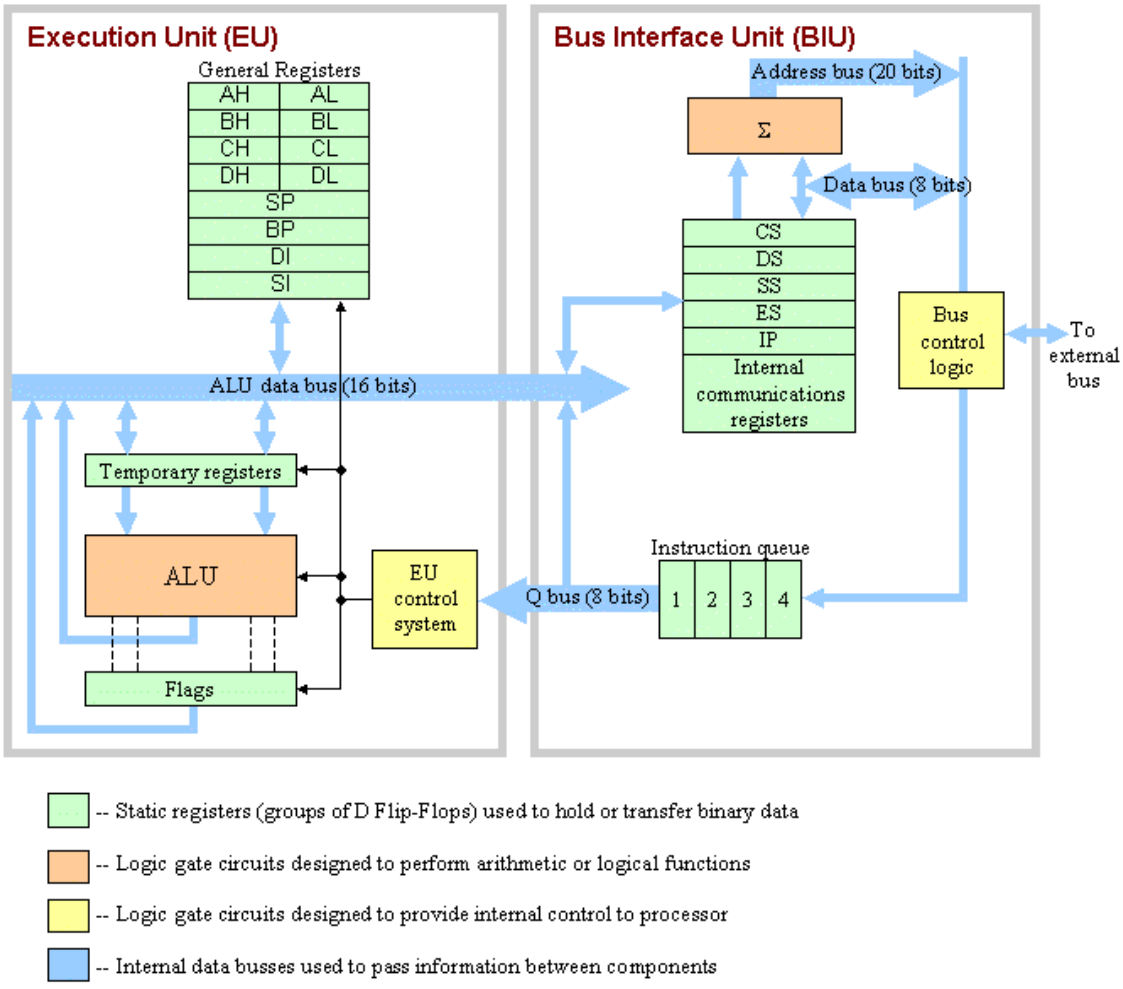
CHƯƠNG 1: GIỚI THIỆU

Trước khi tìm hiểu về Lập trình hệ thống, ta hãy tìm hiểu về cấu trúc của bộ vi xử lý 8088, tập lệnh và các chức năng của ngắt 21h, ngắt của DOS.

1.1 CẤU TRÚC BỘ VI XỬ LÝ

Bộ vi xử lý Intel 8088 là một bộ vi xử lý khá đơn giản, cấu trúc bên trong được trình bày dễ hiểu và phù hợp cho những người mới tìm hiểu về bộ xử lý. Hơn nữa, các bộ xử lý về sau (80x86) được phát triển dựa trên sự kế thừa từ bộ xử lý Intel 8088. Chính vì vậy, các chương trình được viết trên bộ xử lý 8088 có thể chạy được trên các bộ vi xử lý tiên tiến sau này. Phần này trình bày kiến trúc bên trong của bộ Vi xử lý 8088

1.1.1 Sơ đồ kiến trúc bộ Vi xử lý 8088



Hình 1.1: Kiến trúc bên trong của bộ Vi xử lý 8088

Bộ vi xử lý 8088 được chia làm 2 khối chính: Khối giao diện bus (BIU) và khối thực hiện lệnh (EU).

Các thành phần bên trong của CPU giao tiếp với nhau thông qua các bus trong. Giữa khối giao diện bus và khối thực hiện lệnh được liên hệ với nhau thông qua hàng đợi dữ liệu và hệ thống bus trong.

1.1.2 Chức năng các thành phần

1. Thành phần điều khiển Bus (Bus Control Logic)

Điều khiển các loại tín hiệu trên các bus bao gồm: các tín hiệu trên bus địa chỉ (20 bit), các tín hiệu trên bus dữ liệu (8 bit) và các tín hiệu trên bus điều khiển. Ngoài ra, thành phần này còn làm nhiệm vụ hỗ trợ giao tiếp giữa hệ thống bus trong và bus ngoài. Hệ thống bus ngoài là hệ thống bus kết nối giữa các thành phần của hệ vi xử lý với nhau: CPU, Bộ nhớ trong và Thiết bị vào/ra.

2. Hàng đợi lệnh (Prefetch Queue)

Chứa mã lệnh chờ được xử lý. Hàng đợi lệnh có kích thước 4 byte đối với 8088 và 6 byte đối với 8086. Sở dĩ có điều này là vì hàng đợi lệnh phải có kích thước có thể chứa được ít nhất một lệnh có độ dài bất kỳ (dài nhất) của bộ vi xử lý. Mà tập lệnh của 8086 chứa các lệnh có độ dài từ 1-6 byte.

Hàng đợi lệnh làm việc theo cơ chế FIFO (First In First Out), nghĩa là lệnh nào được đưa vào hàng đợi lệnh trước sẽ được xử lý trước

3. Khối điều khiển (Control Unit)

Khối điều khiển có hai chức năng chính: giải mã lệnh và tạo xung điều khiển . Đầu vào của khối điều khiển là mã lệnh được đọc từ hàng đợi lệnh và đầu ra là các xung điều khiển gửi đến các bộ phận khác nhau bên trong bộ vi xử lý. Quá trình này được thực hiện nhờ hai mạch giải mã lệnh và mạch tạo xung.

4. Khối số học và logic (Arithmetic Logic Unit)

Khối số học và logic có chức năng thực hiện các phép tính toán như phép cộng, trừ… hay các phép logic như AND, OR, NOT. Đầu vào ALU là hai thanh ghi tạm thời chứa dữ liệu của cho phép tính được lấy từ bus dữ liệu. Kết quả đầu ra của ALU được đưa trở lại bus dữ liệu và phản ánh vào thanh ghi cờ (flag register).

5. Các thanh ghi đoạn (Segment registers)

Ta hãy thử xem đoạn chương trình được viết bằng ngôn ngữ C sau:

```
int Cong(int a, int b)
{
   Return (a+b);
}
void main()
{
    int x=3; int  y=4;
    printf("Tong: %d", Cong(x,y));
```

```
}
```

Trong chương trình trên có 2 phần: phần khai báo và phần lệnh của chương trình. Trong phần lệnh có thể có lời gọi chương trình con.

Như vậy để thực hiện được một chương trình (dạng .EXE) thì người ta cần ít nhất 3 đoạn bộ nhớ (segment). Đoạn dành chứa dữ liệu được khai báo, đoạn chứa mã chương trình, đoạn ngăn xếp phục vụ cho các lời gọi chương trình con. Mỗi đoạn có kích thước 64KB. Khi chương trình được thực hiện, mỗi đoạn bộ nhớ này được trỏ bởi các thanh ghi đoạn. Đó là:

- Thanh ghi đoạn mã CS (Code Segment): trỏ đến đoạn bộ nhớ chứa mã của chương trình.
- Thanh ghi đoạn dữ liệu DS (Data Segment): trỏ đến đoạn bộ nhớ chứa các khai báo của chương trình.
- Thanh ghi đoạn ngăn xếp SS (Stack Segment): trỏ đến đoạn bộ nhớ dành cho stack.
- Ngoài ra, trong nhiều trường hợp người ta sử dụng thêm một đoạn dữ liệu phụ dùng trong trường hợp các dữ liệu cần khai báo vượt quá kích thước cho phép của 1 đoạn (các khai báo mảng, file…). Khi đó thanh ghi đoạn dữ liệu phụ ES (Extra Segment) sẽ trỏ đến đoạn này

6. Các thanh ghi con trỏ và chỉ số (pointers and index registers)

Các thanh ghi con trỏ và chỉ số là các thanh ghi 16 bit. Chúng thường được lưu địa chỉ lệch (offset) và kết hợp với thanh ghi đoạn tương ứng tạo thành cặp thanh ghi chứa địa chỉ xác định của mã lệnh, mục dữ liệu, hoặc mục dữ liệu lưu trong stack. Nhờ vào cặp thanh ghi này, người ta có thể tính đia chỉ vật lý cụ thể theo công thức sau:

*Địa chỉ vật lý = địa chỉ đoạn * 16 + địa chỉ lệch*

Dưới đây là các thanh ghi con trỏ và chỉ số:

- Thanh ghi con trỏ lệnh IP (Instruction Pointer): trỏ vào lệnh kế tiếp sẽ được thực hiện nằm trong đoạn mã do con trỏ CS trỏ tới. Địa chỉ đầy đủ của một lệnh được đặt trong cặp CS:IP.
- Thanh ghi con trỏ cơ sở BP (Base Pointer): trỏ vào một mục dữ liệu nằm trong đoạn ngăn xếp SS. Địa chỉ đầy đủ của mục dữ liệu là SS:BP.
- Thanh ghi con trỏ ngăn xếp SP (Stack Pointer): trỏ vào đỉnh hiện thời ngăn xếp nằm trong đoạn ngăn xếp SS. Địa chỉ đầy đủ của đỉnh ngăn xếp là SS:SP.
- Thanh ghi chỉ số nguồn SI (Source Index): trỏ vào một mục dữ liệu trong đoạn DS. Địa chỉ đầy đủ của mục dữ liệu là DS:SI.
- Thanh ghi chỉ số đích DI (Destination Index): trỏ vào một mục dữ liệu trong đoạn DS. Địa chỉ đầy đủ của mục dữ liệu là DS:DI.

7. Các thanh ghi đa năng (Multi-purposed registers)

Bộ xử lý 8088 có 4 thanh ghi đa năng 16 bit đó là: AX, BX, CX và DX. Các thanh ghi này cũng có thể được tách ra thành 2 nửa gồm 8 bít cao (nửa cao) gồm bít thứ 8 đến bít thứ 15 và 8 bít thấp (nửa thấp) gồm các bít thứ 0 đến 7. Các nửa thanh ghi này có thể được sử dụng một cách độc lập để chứa các dữ liệu 8 bít. Đó là các nửa thanh ghi: AH và AL, BH và BL, CH và CL, và DH và DL. Trong đó AH, BH, CH, DH la các nửa cao còn AL,BL, CL, DL là các nửa thấp.

Ngoài chức năng "đa năng", mỗi thanh ghi 16 bít thường được sử dụng trong các tác vụ đặc biệt, giống như tên của chúng:

- AX (Accumulator) thanh chứa: các kết quả của các phép toán thường được lưu vào thanh ghi này. Ngoài ra, AX còn là toán hạng ẩn cho 1 số phép toán như nhân (AX là thừa số) hoặc chia (AX là số bị chia).
- BX (Base) thanh ghi cơ sở: thường được dùng để chứa các địa chỉ cơ sở.
- CX (Count) bộ đếm: CX thường dung để chứa số lần lặp trong trường hợp dùng lệnh LOOP. Ngoài ra, CL còn chứa số lần dịch chuyển, quay trái, quay phải của các toán hạng.
- DX (Data) thanh ghi dữ liệu: DX thường được chứa địa chỉ offset của xâu kí tự khi có các thao tác nhập vào xâu hoặc in xâu. DX (cùng với AX) còn tham gia chứa kết quả của phép nhân các số 16 bit hoặc làm số bị chia cho phép chia các số 16 bit. Ngoài ra, DX còn dùng để chứa địa chỉ của các cổng vào/ra trong trường hợp thực hiện các lệnh IN hoặc OUT.

8. Thanh ghi cờ (flag register)

Thanh ghi cờ là thanh ghi lưu trữ trạng thái của CPU tại mỗi thời điểm. Thanh ghi cờ có 16 bít, trong đó có 7 bít dự trữ cho tương lai (CPU 8088 chưa dùng đến các bít này). Còn lại 9 bít và mỗi bít tương ứng là một cờ. Kết hợp các lệnh nhảy có điều kiện (conditional jump) với các cờ này, người lập trình dễ dàng hơn

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	**x**	**x**	**x**	**OF**	**DF**	**IF**	**TF**	**SF**	**ZF**	**x**	**AF**	**x**	**PF**	**x**	**CF**

Hình 1.2: Cấu trúc của thanh ghi cờ của CPU 8088.

Các bit được đánh dấu x là các cờ chưa được dùng đến.

- *Cờ CF* (Carry Flag): cờ nhớ CF=1 khi có nhớ hoặc trừ có mượn từ bít có trọng số cao nhất (Most Significant Bit). Ngoài ra, cờ CF=1 trong trường hợp khi thao tác với file hoặc thư mục gây ra lỗi như các lỗi tạo, xóa file và thư mục.
- *Cờ PF* (Parity Flag): cờ chẵn lẻ PF=1 khi tổng số các bít bằng 1 trong kết quả của phép tính là một số chẵn.
- *Cờ AF* (Auxiliary Carry Flag): cờ nhớ phụ AF =1 khi có nhớ từ bít thứ 4 sang bít thứ 5 hoặc có mượn từ bít 5 sang bít thứ 4 trong biểu diễn BCD của 1 số.
- *Cờ ZF* (Zero Flag): cờ Zero ZF=1 khi kết quả tính toán bằng 0.
- *Cờ SF* (Sign Flag): cờ dấu SF=1 khi kết quả tính toán là một số âm.
- *Cờ TF* (Trap Flag): cờ bẫy TF=1 khi CPU đang làm việc ở chế độ chạy từng lệnh. Chế độ này được sử dụng cần thiết khi tìm lỗi (defect) và gỡ lỗi (debug) chương trình.

- *Cờ IF* (Interrupt enable Flag): cờ cho phép ngắt IF=1, cho phép tác động đến yêu cầu ngắt che được (maskable interrupts).
- *Cờ DF* (Direction Flag): cờ hướng DF=1 khi CPU xử lý chuỗi kí tự theo thứ tự từ phải sang trái.
- *Cờ OF* (Overflow Flag): cờ tràn OF=1 khi kết quả là một số bù hai vượt ra ngoài giới hạn biểu diễn dành cho nó.

9. Hệ thống bus trong (Internal bus system)

Hệ thống bus bên trong của CPU 8088 bao gồm 3 loại:

- Bus *dữ* liệu: 16 bít, cho phép di chuyển 2 byte dữ liệu tại một thời điểm
- Bus địa chỉ: 20 bít, có thể địa chỉ hóa được 2^{20} bytes và vì thế không gian địa chỉ nhớ của CPU 8088 là 1MB.
- Bus điều khiển: truyền tải các tín hiệu điều khiển như RD, WR …

1.2 MỘT SỐ CHỨC NĂNG CỦA NGẮT 21H

Phần này trình bày các hàm thông dụng của ngắt 21h. Đó là các hàm thao tác vào/ra đối với kí tự, chuỗi ký tự, file, thư mục, kết thúc chương trình và trả lại quyền điều khiển cho Hệ điều hành DOS.

Hàm 01: đọc 1 kí tự (có hiện) từ bàn phím

Input: AH=01

Output: AL= mã ASCII của ký tự

AL=0 nếu gõ vào phím chức năng.

Hàm 02: hiện 1 kí tự lên màn hình

Input: AH=02

DL= mã ASCII của ký tự cần hiển thị

Output:

Hàm 08: đọc 1 kí tự (không hiện) từ bàn phím

Input: AH=08

Output: AL= mã ASCII của ký tự

AL=0 nếu gõ vào phím chức năng.

Hàm 09: hiện xâu kí tự kết thúc bởi '$' lên màn hình

Input: AH = 09

DX = địa chỉ offset của xâu kí tự

Hàm 0Ah: đọc xâu kí tự từ bàn phím

Input: AH = 09

DX = địa chỉ offset của vùng đệm chứa xâu kí tự

Output: DX = địa chỉ offset của xâu kí tự

Hàm 39h: tạo thư mục

Input: AH = 39h

DX = địa chỉ offset của tên thư mục

Output:

- Nếu thành công, thư mục được tạo ra
- Nếu không thành công, CF=1 và AX= mã lỗi.

Hàm 3Ah: xóa thư mục

Input: AH = 3Ah

DX = địa chỉ offset của tên thư mục

Output:

- Nếu thành công, thư mục được xóa
- Nếu không thành công, CF=1 và AX=mã lỗi.

Hàm 3Ch: tạo file

Input: AH = 3Ch

DX = địa chỉ offset của tên file

CX = thuộc tính file

Output:

- Nếu thành công, file được tạo ra, CF=0 và AX= thẻ file (file handle)
- Nếu không thành công, CF=1 và AX= mã lỗi.

Thuộc tính file được định nghĩa như sau:

00h: file bình thường (plain old file)

01h: file chỉ đọc (Read Only)

02h: file ẩn (Hidden from searches)

04h: file hệ thống (system)

08h: thuộc tính cho nhãn đĩa.

10h: thuộc tính cho thư mục con.

Hàm 3Dh: mở file

Input: AH = 3Dh

AL = mode

Output:

- Nếu thành công, file được tạo ra, CF=0 và AX= thẻ file (file handle)
- Nếu không thành công, CF=1 và AX= mã lỗi.

Hàm 3Eh: đóng file

Input: AH = 3Eh

BX = thẻ file

Output:

- Nếu thành công, file được đóng lại và CF=0
- Nếu không thành công, CF=1 và AX= mã lỗi.

Hàm 3Fh: đọc từ file

Input: AH = 3Fh

DS:DX = địa chỉ offset của vùng đệm

CX = số byte cần đọc

BX = thẻ file

Output:

- Nếu thành công, CF=0 và AX= số byte đã đọc được
- Nếu không thành công, CF=1 và AX= mã lỗi.

Hàm 40h: ghi vào file

Input: AH = 40h

DS:DX = địa chỉ offset của vùng đệm

CX = số byte cần ghi

BX = thẻ file

Output:

- Nếu thành công, file được ghi và CF=0.
- Nếu không thành công, CF=1 và AX= mã lỗi.

Hàm 41h: xóa file

Input: AH = 41h

DX = địa chỉ offset của tên file

Output:

- Nếu thành công, file bị xóa
- Nếu không thành công, CF=1 và AX=mã lỗi.

Hàm 4Ch: kết thúc chương trình

Input: AH = 4Ch

Output:

Kết thúc chương trình, trả lại quyền điều khiển cho hệ điều hành.

1.3 GIỚI THIỆU VỀ TẬP LỆNH CỦA 8088

Phần này giới thiệu về một số lệnh thông dụng của bộ vi xử lý 8088. Để tiện dụng cho người học lập trình, các lệnh được chia thành các nhóm lệnh.

1.3.1 Nhóm lệnh di chuyển dữ liệu

1. *Lệnh:* MOV

Chức năng: chuyển giá trị từ toán hạng nguồn vào toán hạng đích

Cú pháp:

MOV	**Dst,src**	**Ví dụ**
	Reg1,reg2	Mov AX,BX

Reg, data	Mov AH,9Fh
Mem,reg	Mov [BX],AL
Reg,mem	Mov CL,[3456h]
Mem,data	Mov PTR [BX], FFh

Chú ý: Data chỉ nằm ở phía toán hạng nguồn

Hai toán hạng dst và src không thể đồng thời là hai ô nhớ.

2. *Lệnh:* PUSH

Chức năng: chuyển giá trị của toán hạng nguồn vào đỉnh ngăn xếp

Cú pháp:

PUSH	**Src**	**Ví dụ**
	Reg16	push AX
	Mem16	push x
	Segreg	push DS

Chú ý: Toán hạng nguồn luôn có kích thước 16 bít

Toán hạng nguồn không thể là data (hằng số)

3. *Lệnh:* POP

Chức năng: Lấy giá trị của đỉnh ngăn xếp đưa vào toán hạng đích

Cú pháp:

POP	**Dst**	**Ví dụ**
	Reg16	Pop AX
	Mem16	Pop x
	Segreg	Pop DS

Chú ý: Toán hạng nguồn luôn có kích thước 16 bít

Toán hạng đích không thể là data (hằng số)

4. *Lệnh:* PUSHF

Chức năng: chuyển giá trị của thanh ghi cờ vào đỉnh ngăn xếp

Cú pháp:

PUSHF

5. *Lệnh:* POPF

Chức năng: lấy giá trị đỉnh ngăn xếp lưu vào thanh ghi cờ.

Cú pháp:

POPF

Chú ý: hai lệnh PUSHF và POPF được hệ thống tự động gọi khi chương trình có lệnh gọi ngắt hoặc gọi chương trình con.

6. *Lệnh:* XCHG

Chức năng: Hoán vị giá trị giữa toán hạng nguồn và đích

Cú pháp:

XCHG	**Dst,src**	**Ví dụ**
	Reg,Reg	XCHG AX,BX
	Reg,Mem	XCHG AL,[BX]
	Mem,Reg	XCHG [BX],AH

7. *Lệnh:* IN

Chức năng: Đọc giá trị từ 1 cổng vào thanh ghi AL hoặc AX.

Cú pháp:

IN	**AL, địa chỉ cổng (8 bít)**	VD: IN AL,2Eh
IN	**AX, địa chỉ cổng (16 bít)**	VD: IN AX,2EBEh

8. *Lệnh:* OUT

Chức năng: Chuyển giá trị 1 byte hoặc 1 từ từ thanh ghi AL hoặc AX ra cổng.

Cú pháp:

OUT	**địa chỉ cổng (8 bít), AL**	VD: OUT 2Eh,AL
IN	**địa chỉ cổng (16 bít),AX**	VD: OUT 2EBEh,AX

1.3.2 Nhóm các lệnh tính toán số học

Phần này giới thiệu về các lệnh lien quan đến tính toán số học như các lệnh: cộng, trừ, nhân, chia, so sánh. Đồng thời, cũng giải thích sự tác động của các lệnh này lên các bit của thanh ghi cờ.

1. *Lệnh:* ADD

Chức năng: cộng toán hạng nguồn và toán hạng đích, lưu kết quả vào toán hạng đích.

Cú pháp:

ADD	**Dst,src**	**Ví dụ**
	Reg1,reg2	Add AX,BX

Reg, data	Add AH,19h
Mem,reg	Add [BX],AL
Reg,mem	Add CL,[3456h]
Mem,data	Add [BX], 1Fh

Chú ý:

- Không cộng trực tiếp 2 biến ô nhớ với nhau
- Toán hạng đích không thể là hằng số
- Kết quả có thể tác động đến các cờ: OF, SF, ZF, AF, PF, CF.

2. *Lệnh:* INC

Chức năng: Tăng giá trị của toán hạng đích lên 1.

Cú pháp:

INC	**Dst**	**Ví dụ**
	Reg	Inc CX
	Mem	Inc x

Chú ý:

- Toán hạng đích không thể là hằng số
- Kết quả có thể tác động đến các cờ: OF, SF, ZF, AF, PF, CF.

3. *Lệnh:* SUB

Chức năng: Trừ toán hạng đích cho toán hạng nguồn, lưu kết quả vào toán hạng đích.

Cú pháp:

SUB	**Dst,src**	**Ví dụ**
	Reg1,reg2	Sub AX,BX
	Reg, data	Sub AH,19h
	Mem,reg	Sub [BX],AL
	Reg,mem	Sub CL,[3456h]
	Mem,data	Sub [BX], 1Fh

Chú ý:

- Không trừ trực tiếp 2 biến ô nhớ với nhau
- Toán hạng đích không thể là hằng số
- Kết quả có thể tác động đến các cờ: OF, SF, ZF, AF, PF, CF.

4. *Lệnh:* DEC

Chức năng: Giảm giá trị của toán hạng đích đi 1.

Cú pháp:

DEC	**Dst**	**Ví dụ**
	Reg	DEC CX
	Mem	DEC [BX]

Chú ý:

- Toán hạng đích không thể là hằng số
- Kết quả có thể tác động đến các cờ: OF, SF, ZF, AF, PF, CF.

5. *Lệnh:* MUL

Chức năng: Nhân nội dung của toán hạng AX hoặc AL với nội dung của toán hạng nguồn. Giá trị của hai toán hạng đều là dạng không dấu. Kết quả sẽ được cất như sau:

- Nếu là phép nhân hai toán hạng 8 bít thì kết quả sẽ được đặt trong thanh ghi AX.
- Nếu là phép nhân hai toán hạng 16 bít thì kết quả sẽ được đặt trong thanh nghi DX:AX.

Cú pháp:

MUL	**Src**	**Ví dụ**
	Reg	MUL CL
	Mem	MUL [BX]

Chú ý:

- Toán hạng nguồn không thể là hằng số
- Kết quả có thể tác động đến các cờ: OF, ZF,CF.

6. *Lệnh:* DIV

Chức năng: Chia giá trị của thanh ghi AX hoặc DX:AX cho nội dung của toán hạng nguồn. Giá trị của hai toán hạng đều là dạng không dấu. Kết quả sẽ được cất như sau:

- Nếu số bị chia là toán hạng 16 bít thì phần thương sẽ được đặt trong thanh ghi AL và phần dư sẽ được đặt trong thanh ghi AH.
- Nếu số bị chia là toán hạng 32 bít thì phần thương sẽ được đặt trong thanh ghi AX và phần dư sẽ được đặt trong thanh ghi DX.

Cú pháp:

DIV	**Src**	**Ví dụ**
	Reg	DIV CL
	Mem	DIV [BX]

Chú ý:

- Toán hạng nguồn không thể là hằng số
- Kết quả có thể tác động đến các cờ: OF, CF.

7. *Lệnh:* CMP

Chức năng: So sánh giá trị của toán hạng đích và toán hạng nguồn. Nội dung của hai toán hạng đều không thay đổi sau lệnh này. Thực chất, lệnh này thực hiện bằng cách lấy toán hạng đích trừ đi toán hạng nguồn. Kết quả phản ánh lên thanh ghi cờ mà không được lưu lại.

Cú pháp:

CMP	**Dst,src**	**Ví dụ**
	Reg1,reg2	Cmp AX,BX
	Reg, data	Cmp AH,9Fh
	Mem,reg	Cmp [BX],AL
	Reg,mem	Cmp CL,[3456h]
	Mem,data	Cmp [BX], FFh

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Kết quả có thể tác động đến các cờ: OF, SF,ZF,AF,PF,CF.

1.3.3 Nhóm các lệnh thao tác bít

Phần này giới thiệu về các lệnh liên quan đến các lệnh logic, các lệnh dịch chuyển bít, các lệnh quay vòng các bit.

1. *Lệnh:* NOT

- *Chức năng:* Đảo giá trị từng bít một của toán hạng đích.

Cú pháp:

NOT	**Dst**	**Ví dụ**
	Reg	NOT CL
	Mem	NOT [BX]

Chú ý:

- Toán hạng đích không thể là hằng số

2. *Lệnh:* AND

Chức năng: Thực hiện phép VÀ logic giữa hai toán hạng. Kết quả đặt ở trong toán hạng đích.

Cú pháp:

AND	**Dst,src**	**Ví dụ**
	Reg1,reg2	And AX,BX
	Reg, data	And AH,9Fh
	Mem,reg	And [BX],AL

	Reg,mem	And CL,[3456h]
	Mem,data	And [BX], FFh

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Kết quả có thể tác động đến các cờ: SF,ZF, PF.

3. *Lệnh:* OR

Chức năng: Thực hiện phép HOẶC logic giữa hai toán hạng. Kết quả đặt ở trong toán hạng đích.

Cú pháp:

OR	**Dst,src**	**Ví dụ**
	Reg1,reg2	And AX,BX
	Reg, data	And AH,9Fh
	Mem,reg	And [BX],AL
	Reg,mem	And CL,[3456h]
	Mem,data	And [BX], FFh

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Kết quả có thể tác động đến các cờ: SF,ZF, PF.

4. *Lệnh:* XOR

Chức năng: Thực hiện phép EXCLUSIVE OR logic giữa hai toán hạng (các bít của kết quả có giá trị là 1 nếu hai bít tương ứng của 2 toán hạng là khác nhau). Kết quả đặt ở trong toán hạng đích.

Cú pháp:

XOR	**Dst,src**	**Ví dụ**
	Reg1,reg2	Xor AX,BX
	Reg, data	Xor AH,9Fh
	Mem,reg	Xor [BX],AL
	Reg,mem	Xor CL,[3456h]
	Mem,data	Xor [BX], FFh

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Kết quả có thể tác động đến các cờ: SF,ZF, PF.

5. *Lệnh:* TEST

Chức năng: So sánh nội dung của hai toán hạng bằng cách thực hiện lệnh AND giữa hai toán hạng mà không lưu lại kết quả. Kết quả tác động đến thanh ghi cờ.

Cú pháp:

TEST	**Dst,src**	**Ví dụ**
	Reg1,reg2	Test AX,BX
	Reg, data	Test AH,9Fh
	Mem,reg	Test [BX],AL
	Reg,mem	Test CL,[3456h]
	Mem,data	Test [BX], FFh

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Kết quả có thể tác động đến các cờ: SF,ZF, PF.

6. *Lệnh:* SHL/SAL

Chức năng: Dịch trái các bít của toán hạng đích đi COUNT lần. Trong đó CL=COUNT.

Cú pháp:

SHL/SAL	**Dst,COUNT**	**Ví dụ**
	Reg	SHL AL,CL
	Mem	SHL [BX],CL

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Khi Count=1 thì có thể đặt 1 trực tiếp vào toán hạng, SHL/SAL Dst,1.
- Kết quả có thể tác động đến các cờ: OF,SF,ZF, PF,CF.

7. *Lệnh:* SHR

Chức năng: Dịch phải các bít của toán hạng đích đi COUNT lần. Trong đó CL=COUNT.

Cú pháp:

SHR	**Dst,COUNT**	**Ví dụ**
	Reg	SHR AL,CL
	Mem	SHR [BX],CL

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Khi Count=1 thì có thể đặt 1 trực tiếp vào toán hạng, SHR Dst,1.
- Kết quả có thể tác động đến các cờ: OF,SF,ZF, PF,CF.

8. *Lệnh:* ROR

Chức năng: Quay vòng phải các bít của toán hạng đích đi COUNT lần. Trong đó CL=COUNT. Trong mỗi lần quay, giá trị bít thấp nhất vừa chuyển vào thanh ghi cờ CF đồng thời chuyển vào bít cao nhất.

Cú pháp:

ROR	**Dst,COUNT**	**Ví dụ**
	Reg	ROR AL,CL
	Mem	ROR [BX],CL

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Khi Count=1 thì có thể đặt 1 trực tiếp vào toán hạng, ROR Dst,1.
- Kết quả có thể tác động đến các cờ: OF, CF.

9. *Lệnh:* ROL

Chức năng: Quay vòng trái các bít của toán hạng đích đi COUNT lần. Trong đó CL=COUNT. Trong mỗi lần quay, giá trị bít cao nhất vừa chuyển vào thanh ghi cờ CF đồng thời chuyển vào bít thấp nhất.

Cú pháp:

ROL	**Dst,COUNT**	**Ví dụ**
	Reg	ROL AL,CL
	Mem	ROL [BX],CL

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Khi Count=1 thì có thể đặt 1 trực tiếp vào toán hạng, ROL Dst,1.
- Kết quả có thể tác động đến các cờ: OF, CF.

1.3.4 Nhóm các lệnh làm việc với xâu kí tự

1. *Lệnh:* MOVSB(hay MOVSW)

Chức năng: Chuyển một xâu kí tự theo từng byte (hay theo từng từ) từ một vùng nhớ nguồn sang vùng nhớ đích. Trong đó DS:SI trỏ đến xâu kí tự nguồn và ES:DI trỏ đến xâu kí tự đích. Sau mỗi lần chuyển 1 byte (hoặc 1 từ) thì giá trị của SI và DI tự động tăng lên 1 (hoặc 2) nếu cờ hướng DF=0, hoặc giảm đi 1 (hoặc 2) nếu cờ hướng DF=1.

Cú pháp:

MOVSB Hoặc

MOVSW

2 *Lệnh:* CMPSB (CMPSW)

Chức năng: So sánh hai xâu kí tự theo từng byte (hay theo từng từ) nằm ở hai vùng nhớ. Trong đó, DS:SI và ES:DI trỏ đến hai xâu kí tự. Sau mỗi lần so sánh từng byte (hoặc từng từ) thì

giá trị của SI và DI tự động tăng lên 1 (hoặc 2) nếu cờ hướng DF=0, hoặc giảm đi 1 (hoặc 2) nếu cờ hướng DF=1.

Cú pháp:

CMPSB Hoặc

CMPSW

- Kết quả có thể tác động đến các cờ: OF, SF,ZF,AF,PF, CF.

3 *Lệnh:* LODSB (LODSW)

Chức năng: Chuyển nội dung theo từng byte (hay theo từng từ) của vùng nhớ trỏ bởi DS:SI vào thanh ghi AL (hoặc AX). Sau mỗi lần chuyển từng byte (hoặc từng từ) thì giá trị của SI tự động tăng lên 1 (hoặc 2) nếu cờ hướng DF=0, hoặc giảm đi 1 (hoặc 2) nếu cờ hướng DF=1.

Cú pháp:

LODSB Hoặc

LODSW

4. *Lệnh:* STOSB (STOSW)

Chức năng: Chuyển nội dung theo từng byte (hay theo từng từ) của thanh ghi AL (hoặc AX) vùng nhớ trỏ bởi ES:DI. Sau mỗi lần chuyển từng byte (hoặc từng từ) thì giá trị của DI tự động tăng lên 1 (hoặc 2) nếu cờ hướng DF=0, hoặc giảm đi 1 (hoặc 2) nếu cờ hướng DF=1.

Cú pháp:

STOSB Hoặc

STOSW

1.3.5 Nhóm các lệnh nhảy

Nhóm các lệnh nhảy bao gồm 4 nhóm nhỏ: các lệnh nhảy không điều kiện, các lệnh nhảy có điều kiện, các lệnh lặp và các lệnh gọi ngắt mềm.

a. Các lệnh nhảy không điều kiện

1. Lệnh CALL

Chức năng: Gọi chương trình con

Cú pháp:

CALL Địa chỉ

Nhãn

Tên chương trình con

Reg

Mem

2. Lệnh RET

Chức năng: Quay trở về chương trình đã gọi chương trình con

Cú pháp:

RET

3. Lệnh JMP

Chức năng: Lệnh nhảy không điều kiện

Cú pháp:

JMP Địa chỉ

Nhãn

Tên chương trình con

Reg

Mem

Chú ý: Bước nhảy của lệnh nhảy này nằm trong một đoạn 64KB.

b. Các lệnh nhảy có điều kiện

Chức năng: Lệnh nhảy không điều kiện

Cú pháp:

Lệnh	**Toán hạng**	**Giải thích**
JE/JZ	Nhãn	Nhảy nếu ZF=1 hoặc 2 toán hạng của phép so sánh bằng nhau
JNE/JNZ	Nhãn	Nhảy nếu ZF=0 hoặc 2 toán hạng của phép so sánh khác nhau
JL/JNGE	Nhãn	Nhảy nếu toán hạng bên trái nhỏ hơn toán hạng bên phải của phép so sánh (CF=1)
JB/JNAE/JC	Nhãn	Nhảy nếu toán hạng bên trái nhỏ hơn toán hạng bên phải của phép so sánh (SF<>0)
JLE/JNG	Nhãn	Nhảy nếu toán hạng bên trái nhỏ hơn hoặc bằng toán hạng bên phải của phép so sánh (SF<>OF hoặc ZF=0)
JBE/JNA	Nhãn	Nhảy nếu toán hạng bên trái nhỏ hơn hoặc bằng toán hạng bên phải của phép so sánh (Cờ CF=1 hoặc SF=1)
JG/JNLE	Nhãn	Nhảy nếu toán hạng bên trái lớn hơn toán hạng bên phải của phép so sánh.
JA/JNBE	Nhãn	Nhảy nếu toán hạng bên trái lớn hơn hoặc bằng toán hạng bên phải của phép so sánh (Cờ CF=0 hoặc ZF=0)
JGE/JNL	Nhãn	Nhảy nếu toán hạng bên trái lớn hơn hoặc bằng toán hạng bên phải của phép so sánh (Cờ SF=OF).
JAE/JNB/JNC	Nhãn	Nhảy nếu toán hạng bên trái lớn hơn hoặc bằng toán hạng bên phải của phép so sánh (Cờ CF=0).
JP/JPE	Nhãn	Nhảy nếu cờ parity là chẵn (PF=1)

JNP/JPO	Nhãn	Nhảy nếu cờ parity là lẻ (PF=0)
JO	Nhãn	Nhảy nếu tràn (OF=1)
JNO	Nhãn	Nhảy nếu không tràn (OF=0)
JS	Nhãn	Nhảy nếu cờ dấu =1 (SF=1)
JNS	Nhãn	Nhảy nếu cờ dấu =0 (SF=0)
JCXZ	Nhãn	Nhảy nếu giá trị của thanh ghi CX =0.

Chú ý: Các bước nhảy của các lệnh nhảy có điều kiện không vượt quá 128 byte.

c. Các lệnh lặp

Chức năng: Thực hiện vòng lặp cho đến khi điều kiện thỏa mãn.

Cú pháp:

Lệnh	**Toán hạng**	**Giải thích**
LOOP	Nhãn	Lặp khối lệnh từ Nhãn đến LOOP cho đến khi giá trị của CX=0. Sau mỗi lần thực hiện vòng lặp giá trị của CX tự động giảm đi 1.
LOOPZ/LOOPE	Nhãn	Lặp khối lệnh từ Nhãn đến LOOPZ hoặc LOOPE cho đến khi giá trị của CX=0 và cờ ZF=1. Sau mỗi lần thực hiện vòng lặp giá trị của CX tự động giảm đi 1.
LOOPNZ/LOOPNE	Nhãn	Lặp khối lệnh từ Nhãn đến LOOPZ hoặc LOOPE cho đến khi giá trị của CX<>0 và cờ ZF=0. Sau mỗi lần thực hiện vòng lặp giá trị của CX tự động giảm đi 1.

d. Các lệnh gọi ngắt mềm.

1. Lệnh: INT

Chức năng: Thực hiện ngắt mềm.

Cú pháp: **INT số hiệu ngắt (dạng hexa)**

Các cờ bị tác động: IF, TF

2. Lệnh: IRET

Chức năng: Trở về chương trình chính (chương trình gọi nó) sau khi thực hiện chương trình con phục vụ ngắt.

Cú pháp: **IRET**

Các cờ bị tác động: OF,SF,ZF,AF,PF,CF.

1.3.6 Các lệnh điều khiển khác

Phần này giới thiệu về một số lệnh điều khiển: thao tác với thanh ghi cờ, lệnh HLT và NOP.

1. Lệnh: CLC

Chức năng: Xóa giá trị cờ CF về 0 (CF=0).

Cú pháp: **CLC**

Các cờ bị tác động: CF.

2. Lệnh: CMC

Chức năng: Đảo giá trị hiện thời của cờ CF.

Cú pháp: **CMC**

Các cờ bị tác động: CF.

3. Lệnh: STC

Chức năng: Đặt cờ CF=1.

Cú pháp: **STC**

Các cờ bị tác động: CF.

4. Lệnh: CLD

Chức năng: Xoá giá trị cờ DF về 0 (DF=0).

Cú pháp: **CLD**

Các cờ bị tác động: DF.

5. Lệnh: STD

Chức năng: Đặt giá trị cờ DF bằng 1 (DF=1).

Cú pháp: **STD**

Các cờ bị tác động: DF.

6. Lệnh: CLI

Chức năng: Xoá giá trị cờ IF về 0 (IF=0). Cấm các ngắt cứng hoạt động, trừ các ngắt không che.

Cú pháp: **CLI**

Các cờ bị tác động: IF.

7. Lệnh: STI

Chức năng: Đặt giá trị cờ IF bằng 1 (IF=1). Cấm các ngắt cứng hoạt động.

Cú pháp: **STI**

Các cờ bị tác động: IF.

8. Lệnh: HLT

Chức năng: dừng máy.

Cú pháp: **HLT**

9 Lệnh: NOP

Chức năng: Không thực hiện gì

Cú pháp: **NOP**

Chú ý: Lệnh NOP rất có ý nghĩa khi CPU thực hiện các chu kỳ đợi và được xen vào một số chu kỳ lệnh trong quá trình thực hiện lệnh theo cơ chế pipeline.

1.4 TÓM TẮT

Chương này đã trang bị cho sinh viên những kiến thức cơ bản để chuẩn bị cho các phần kế tiếp: Lập trình bằng Hợp ngữ. Chương này có ba phần chính:

- Giới thiệu về cấu trúc bộ vi xử lý 8088. Đây là bộ vi xử lý khá đơn giản và dễ hiểu về mặt kiến trúc. Sơ đồ kiến trúc bao gồm hai khối chính: Khối giao diện BUS và khối thực hiện lệnh. Đồng thời, phần này cũng đề cập chi tiết chức năng của các thành phần bên trong bộ vi xử lý.
- Một số chức năng của ngắt 21h. Đây là một ngắt quan trọng nhất của hệ điều hành MS DOS. Ngắt 21h cung cấp nhiều các chức năng khác nhau cho các nhà lập trình hệ thống. Phần này đã giới thiệu 14 chức năng thông dụng của ngắt 21h. Từ các chức năng phục vụ vào ra đối với kí tự, xâu kí tự cho đến các chức năng phục vụ cho thao tác các file và thư mục.
- Tập lệnh của 8088 dạng hợp ngữ. Phần này trình bầy về một số lệnh thông dụng trong tập lệnh của 8088. Để tiện lợi cho người học lập trình ở các phần sau chúng tôi phân chia các lệnh ra thành các nhóm lệnh. Mỗi nhóm lệnh bao gồm một số lệnh thực hiện 1 số chức năng.

1.5 CÂU HỎI VÀ BÀI TẬP

Dưới đây là các câu hỏi dạng lựa chọn. Sinh viên sẽ lựa chọn **một và chỉ một** phương án trả lời **đúng nhất** cho mỗi câu hỏi

Câu 1: Khối giao diện bus (BIU) và khối thực hiện lệnh (EU) giao tiếp với nhau thông qua:

A. Hàng đợi lệnh

B. Hệ thống bus trong

C. Hệ thống bus trong và hàng đợi lệnh

D. Không có liên hệ gì với nhau.

Câu 2: Một trong những chức năng của thành phần điều khiển (CU) là:

A. Tạo xung điều khiển

B. Giải mã địa chỉ

C. Chứa các thanh ghi

D. Chứa các lệnh sắp được xử lý

Câu 3: Các nhóm thanh ghi (16 bít) nào dưới đây có thể được chia làm 2 nửa thanh ghi 8 bít độc lập với nhau:

A. AX,BX,DS

B. SP,IP,CX

C. ES, SS, DS

D. DX,AX,CX

Câu 4: Các cặp thanh ghi (16 bít) nào dưới đây được sử dụng để chứa địa chỉ đoạn và địa chỉ lệch của một mục dữ liệu trong ngăn xếp:

A. DS:DI

B. SS:BP

C. ES:SI

D. CS:IP

Câu 5: Các thanh ghi đoạn nào dưới đây trỏ vào các đoạn dữ liệu của cùng một chương trình .EXE:

A. DS,ES

B. DS,SS

C. SS, SP

D. DS

Câu 6: Lệnh nào dưới đây là sai cú pháp:

A. Mov AL,[BX+1]

B. Mov [BX+1], AL

C. Mov [AL],[BX+1]

D. Mov [CX], 3

Câu 7: Lệnh nào dưới đây là sai cú pháp:

A. Mul [BX+5]

B. Div [BX+5]

C. Mul AL,3

D. Mul DX

Câu 8: Lệnh nào dưới đây thực hiện việc xóa thanh ghi AX.

A. AND AX,AX

B. XOR AX,AX

C. NOT AX

D. MOV AX,[0000H]

Câu 9: Lệnh nào dưới đây thường được sử dụng trước các lệnh nhảy có điều kiện.

A. CMP

B. MOV

C. INT

D. RET

Câu 10 *: Xét đoạn chương trình dưới đây:

```
Mov AH,08
Int 21h
Mov DL,AL
Inc DL
Mov AH,02
Int 21h
```

Đoạn chương trình trên thực hiện công việc nào dưới đây:

A. Nhập một kí tự rồi in kí tự kế tiếp của kí tự đó ra màn hình
B. Nhập một kí tự rồi in kí tự đó ra màn hình
C. Nhập một xâu kí tự rồi in xâu đó ra màn hình
D. Nhập 1 kí tự không hiện lên kí tự đó.

1.6 TÀI LIỆU THAM KHẢO

1. Văn Thế Minh. Kỹ thuật Vi xử lý. Nhà XB Giáo dục 1997.
2. Đặng Thành Phu. Turbo Assembler và Ứng dụng. NXB Khoa học và Kỹ thuật 1998.
3. Nguyễn Minh San. Cẩm nang Lập trình hệ thống (bản dịch). NXB Tổng cục Thống kê.2001.

CHƯƠNG 2: LẬP TRÌNH BẰNG HỢP NGỮ

Hợp ngữ (Assembly language) là ngôn ngữ lập trình không thể thiếu trong lập trình hệ thống và điều khiển thiết bị. Với các ưu điểm như : dịch và thực hiện nhanh, tốn ít bộ nhớ, người dùng dễ kiểm soát lỗi…thì hợp ngữ có mặt trong nhiều chương trình hệ thống (như hệ điều hành chẳng hạn) hay các chương trình điều khiển. Trong chương này, chúng tôi giới thiệu các vấn đề về lập trình bằng hợp ngữ. Cách khai báo biến, hằng, khung chương trình, các cấu trúc lập trình, chương trình con, macro và kết nối hợp ngữ với các ngôn ngữ bậc cao và các chương trình con ngắt.

2.1 VIẾT VÀ THỰC HIỆN MỘT CHƯƠNG TRÌNH HỢP NGỮ

2.1.1 Cấu trúc lệnh và khai báo dữ liệu cho chương trình

Một chương trình bao gồm tập hợp các lệnh và các khai báo dữ liệu sử dụng trong chương trình nhằm mục đích giải quyết một vấn đề. Phần này trình bày về cấu trúc một dòng lệnh và các qui tắc khai báo biến, hằng trong một chương trình Hợp ngữ.

b. Cấu trúc dòng lệnh

Dưới Đây là một dòng lệnh đầy đủ của chương trình Hợp ngữ,. Trên thực tế, một dòng lệnh cần tối thiểu hai trường: trường mã lệnh và trường toán hạng. Các trường khác không bắt buộc cần phải đầy đủ.

Nhãn:	**Mã lệnh**	**Toán hạng**	**; Chú giải**

Ví dụ:

CongTiep: Add AL,[BX] ; cộng tiếp nội dung ô nhớ do thanh ghi BX ;trỏ tới vào AL

Các giải thích cho các trường:

Trường	Mô tả
Nhãn	Nhãn có thể là nhãn dung cho lệnh nhảy, tên thủ tục hoặc tên biến. Khi chạy chương trình, các nhãn này sẽ được chương trình dịch gán cho 1 địa chỉ ô nhớ xác định. Nhãn có thể chứa từ 1 đến 32 kí tự, không chứa dấu cách và phải bắt đầu bằng các kí tự a,b,c…,z. Nhãn được kết thúc bằng dấu hai chấm (:)
Mã lệnh	Chứa lệnh thật (Opcode) hoặc giả lệnh (Pseudo-Opcode). Với các lệnh thật thì trường này chứa mã lệnh gợi nhớ (thường là dạng viết ngắn hoặc đầy đủ của một động từ trong tiếng Anh). Trong quá trình chạy chương trình, mã lệnh thật sẽ được chương trình dịch dịch ra mã máy. Đối với các giả lệnh, thì chương trình dịch không dịch chúng ra mã máy.
Toán hạng	Đối với các lệnh thật thì trường này chứa các toán hạng cho lệnh đó. Lệnh thật của 8088 có thể có 0,1 hoặc 2 toán hạng. Toán hạng trong các giả lệnh của 8088 thường chứa các thông tin khác nhau như xác định mô hình bộ nhớ

	sử dụng, kích thước ngăn xếp…
Chú giải	Trường chú giải được bắt đầu bằng dấu chấm phẩy (;) để ghi những lời giải thích các lệnh của chương trình nhằm giúp cho người đọc chương trình một cách dễ hiểu hơn. Khi thực hiện chương trình, phần giải thích (sau dấu ;) sẽ bị bỏ qua. Ngoài ra, người ta cũng dùng trường này để ghi chu giải cho cả một đoạn chương trình, một chương trình con hay thâm chí là lời giới thiệu của bản quyền (copyright message) của người lập trình . Ví dụ: ; This program writen by X ; Last modified: 23/10/2006 ; This program is to delete a file …..

c. *Khai báo biến*

Các biến có thể được khai báo là ba kiểu dữ liệu khác nhau là: biến kiểu byte, biến kiểu từ (2 byte) và biến kiểu từ kép (4 byte).

Biến kiểu byte:

Dạng khai báo:	**Tên biến**	**DB**	**?**
Ví dụ:	X	DB	? ;không có giá trị khởi đầu
	Y	DB	4 ;giá trị khởi đầu là 4

Biến kiểu từ:

Dạng khai báo:	**Tên biến**	**DW**	**?**
Ví dụ:	X	DW	? ;không có giá trị khởi đầu
	Y	DW	4 ; giá trị khởi đầu là 4.

Biến kiểu từ kép:

Dạng khai báo:	**Tên biến**	**DD**	**?**
Ví dụ:	X	DD	? ;không có giá trị khởi đầu
	Y	DD	4Fh ; giá trị khởi đầu là 4Fh

Khai báo biến kiểu mảng

Mảng là một dãy liên tiếp các phần tử có cùng kiểu byte hoặc từ.

Ví dụ:

```
A1 DB 1,2,3,4,5
```

Là khai báo một biến mảng tên là A1, gồm 5 byte trong bộ nhớ. Nội dung ô nhớ [A1] có giá trị là 1, [A1+1] có giá trị là 2, [A1+2] có giá trị là 3…

Nếu muốn khai báo một mảng không cần khởi tạo giá trị ban đầu, ta khai báo như sau:

A2 DB 100 DUP(?)

khai báo một mảng có 100 phần tử và các phần tử là kiểu byte.

Khai báo sau là một mảng 100 phần tử kiểu byte và tất cả các phần tử được khởi tạo giá trị 0.

A3 DB 100 DUP(0)

Khai báo biến kiểu xâu kí tự

Xâu kí tự là một mảng mà mỗi phần tử là một kí tự hay mã ASCII của kí tự. Xâu kí tự được kết thúc bởi kí tự '$' (có mã ASCII là 24h).

Ví dụ:

Xau1 DB 'Chào các bạn','$';

XuongDong DB 13,10,'$';xâu chứa các kí tự xuống dòng và về đầu dòng

Xau2 DB 36h,40h,'a','b', '$' ; xâu chứa cả mã ASCII và kí tự.

d. *Khai báo hằng*

Hằng có thể là kiểu số hoặc kiểu kí tự. Cú pháp khai báo hằng như sau:

Tên hằng EQU Giá trị của hằng

Ví dụ:

CR EQU 0Dh ;

Có thể sử dụng hằng để khai báo:

Ví dụ:

Chao EQU 'Chào bạn'

LoiChao DB Chao,'$'

2.1.2 Khung của chương trình Hợp ngữ

Để thực hiện một chương trình dạng mã máy, hệ điều hành cấp phát một số vùng nhớ dành cho chương trình để chứa các mã lệnh, dữ liệu, và ngăn xếp. Phần này trình bày về các giả lệnh điều khiển đoạn dùng cho chương trình, khung của chương trình dạng .COM, khung của chương trình dạng .EXE, và cuối cùng là một số chương trình ví dụ đơn giản.

a. Các giả lệnh điều khiển đoạn (segment directives)

1. Lệnh: MODEL

Chức năng: Khai báo mô hình bộ nhớ cho một module Assembler. Lệnh này thường được đặt sau giả lệnh về khai báo loại CPU (CPU family) và được đặt trước tất cả các giả lệnh điều khiển đoạn khác.

Cú pháp: **.MODEL <Kiểu kích thước bộ nhớ>**

Trong đó kiểu kích thước bộ nhớ là một trong các kiểu sau:

Kiểu kích thước	Mô tả

Tiny (hẹp)	Mã lệnh và dữ liệu được gói vào cùng một đoạn. Kiểu này thường được dùng trong chương trình. COM
Small (nhỏ)	Mã lệnh được gói vào trong một đoạn. Dữ liệu nằm trong một đoạn khác
Medium (trung bình)	Mã lệnh được gói vào trong một đoạn. Dữ liệu không gói gọn trong một đoạn.
Compact (nén)	Mã lệnh không gói vào trong một đoạn. Dữ liệu không gói gọn trong một đoạn.
Large (lớn)	Mã lệnh không gói vào trong một đoạn. Dữ liệu không gói gọn trong một đoạn. Không có mảng dữ liệu được khai báo nào lớn hơn 64KB
Huge (rất lớn)	Mã lệnh không gói vào trong một đoạn. Dữ liệu không gói gọn trong một đoạn. Các mảng dữ liệu được khai báo có thể lớn hơn 64KB.

Ví dụ: .MODEL Small

2. Lệnh: STACK

Chức năng: Khai báo kích thước đoạn ngăn xếp dùng trong chương trình. Đoạn ngăn xếp là một vùng nhớ để lưu các trạng thái hoạt động của chương trình khi có chương trình con.

Cú pháp: **.STACK <Kích thước ngăn xếp>**

Trong đó kích thước ngăn xếp là số byte dành cho ngăn xếp. Nếu không khai báo kích thước ngăn xếp thì chương trình sẽ tự động gán cho giá trị là 1024 (1 KB). Số này là khá lớn, thông thường khoảng 256 byte hay 100h là đủ.

Ví dụ: .STACK 100h

3. Lệnh: DATA

Chức năng: Khai báo đoạn dữ liệu cho chương trình. Đoạn dữ liệu chứa toàn bộ các khai báo hằng, biến của chương trình.

Cú pháp: **.DATA**

Ví dụ: .DATA

```
        CR EQU 0Dh
        LF  EQU 0Ah
   LoiChao   DB 'Chào các bạn','$'
```

4. Lệnh: CODE

Chức năng: Khai báo đoạn mã lệnh chương trình. Đoạn mã chứa các dòng lệnh của chương trình.

Cú pháp: **.CODE**

Ví dụ: .CODE

```
Mov AH,09
Mov Dx, Offset LoiChao
....
```

b. Khung của chương trình Hợp ngữ để dịch ra dạng .EXE

Dưới đây là khung của một chương trình hợp ngữ mà sau khi được dịch (compiled) và hợp dịch (linked) thì sẽ thành một file thực hiện được dạng .EXE.

Sau khi sử dụng các giả lệnh điều khiển đoạn để khai báo mô hình bộ nhớ, kích thước ngăn xếp, đoạn dữ liệu và bắt đầu đoạn mã lệnh. Tất nhiên người lập trình có thể thay đổi mô hình bộ nhớ (có thể không phải là *Small*) hay kích thước ngăn xếp (một số khác, không phải là 100h) cho phù hợp với mục đích viết chương trình.

Ta dùng nhãn *Start* và *End Start* để đánh dấu điểm bắt đầu và kết thúc đoạn mã lệnh dùng cho chương trình (tất nhiên người lập trình có thể dùng tên nhãn khác để đánh dấu, không bắt buộc phải dùng nhãn *Start*)

```
.MODEL Small
.STACK 100h
.DATA
  ; Các khai báo hằng, biến ở đây
.CODE
Start:
        Mov AX,@Data
        Mov DS,AX ; cho DS trỏ đến đoạn Data

     ; Các lệnh của chương trình chính được viết ở đây
        Mov AH,4Ch ; Trở về và trả quyền điều khiển cho DOS
        Int 21h
End Start
; Các chương trình con (nếu có) sẽ được viết ở đây.
```

Hai lệnh:

```
Mov AX,@Data
Mov DS,AX
```

Làm nhiệm vụ cho con trỏ DS trỏ tới đoạn chứa dữ liệu khai báo (Data). Hằng số @Data là tên của đoạn dữ liệu (thực chất hằng số này mang giá trị là địa chỉ của đoạn bộ nhớ cấp phát cho chương trình trong quá trình chạy chương trình). Mà DS không làm việc trực tiếp với hằng số (không thể chuyển giá trị hằng số trực tiếp vào các thanh ghi đoạn), nên thanh ghi AX là biến trung gian để đưa giá trị @Data vào DS.

Hai lệnh cuối của chương trình:

```
Mov AH,4Ch
Int 21h
```

Làm nhiệm vụ kết thúc chương trình .EXE và trả lại quyền điều khiển cho hệ điều hành DOS. Nhắc lại rằng không giống như các hệ điều hành Windows 9x, 2K,XP là các hệ điều hành

đa nhiệm. Hệ điều hành DOS là hệ điều hành đơn nhiệm (Single-task). Nghĩa là, tại một thời điểm, chỉ có một chương trình chiếm quyền điều khiển và tài nguyên của hệ thống.

Ví dụ về một chương trình dạng .EXE đơn giản, chương trình Hello World. Chương trình thực hiện việc in ra màn hình một lời chào "Hello World"

```
; Chương trình này in ra màn hình lời chào Hello World
.MODEL Small
.STACK 100h
.DATA
    Msg db 'Hello World','$'
.CODE
Start:
        Mov AX,@Data
        Mov DS,AX  ; cho DS trỏ đến đoạn Data
        Mov AH,09h ; Hàm 09, in ra 1 xâu kí tự
        Mov DX,Offset Msg ; Dx chứa địa chỉ offset của xâu
        Int 21h ; thực hiện chức năng in xâu
        Mov AH,4Ch ; Trở về và trả quyền điều khiển cho DOS
        Int 21h
End Start
```

c. Khung của chương trình Hợp ngữ để dịch ra dạng .COM

Chương trình .COM ngắn gọn và đơn giản hơn so với các chương trình .EXE. Tất cả các đoạn ngăn xếp, dữ liệu, đoạn mã được gộp vào cùng một đoạn là đoạn mã. Nghĩa là, chương trình .COM được gói gọn trong một đoạn (việc dịch và thực hiện đối với chương trình .COM sẽ nhanh hơn các chương trình .EXE). Với các ứng dụng nhỏ mà mã lệnh và dữ liệu không vượt quá 64KB, ta có thể ghép luôn các đoạn ngăn xếp, dữ liệu và mã lệnh vào cùng với đoạn mã để tạo ra file dạng COM.

Để dịch được ra file dạng .COM, chương trình nguồn phải tuân thủ theo khung dưới đây

```
.MODEL Tiny
.CODE
        Org 100h
        Jmp Start
     ; Các khai báo hằng, biến ở đây
Start:
     ; Các lệnh của chương trình chính được viết ở đây
         Int 20h
     ; Các chương trình con (nếu có) sẽ được viết ở đây.
End Start
```

Khai báo mô hình kích thước sử dụng bộ nhớ luôn là Tiny. Ngoài ra, trong khung này không có lời khai báo đoạn ngăn xếp và đoạn dữ liệu. Lệnh đầu tiên trong đoạn mã là giả lệnh ORG 100h, dùng để gán địa chỉ bắt đầu cho chương trình tại 100h trong đoạn mã. Vùng dung lượng 256 byte đầu tiên được sử dụng cho đoạn mào đầu chương trình (Prefix Segment Program).

Kế tiếp, người ta dung lệnh JMP để nhảy qua phần bộ nhớ được dùng cho khai báo.

Ví dụ về một chương trình .COM đơn giản, chương trình Hello World.

```
.MODEL Tiny
.CODE
        Org 100h
        Jmp Start
         Msg db 'Hello World', '$'
Start:
        Mov AH,09h ; Hàm 09, in ra 1 xâu kí tự
        Mov DX,Offset Msg ; Dx chứa địa chỉ offset của xâu
        Int 21h ; thực hiện chức năng in xâu
    Int 20h ; kết thúc chương trình, trở về DOS
End Start
```

2.1.3 Tạo, dịch, hợp dịch và thực hiện chương trình Hợp ngữ

Phần này trình bày về các bước để tạo, cách dịch, hợp dịch và thực hiện một chương trình hợp ngữ. Dưới đây là các bước phải được thực hiện tuần tự. Nghĩa là, bước thứ i không thể được thực hiện nếu bước trước nó (bước i-1) chưa được thực hiện thành công.

Bước 1: Soạn chương trình nguồn

Dùng bất kỳ trình soạn thảo nào như Nodepad, Turbo Pascal Editor, Turbo C Editor…để tạo ra file văn bản chương trình. File chương trình phải có phần mở rộng là .ASM.

Bước 2: Dịch

Dùng một trong các chương trình dịch: MASM (Macro Assembler) hoặc TASM (Turbo Assembler) để dịch file .ASM thành file .OBJ. Nếu bước này có lỗi thì ta phải quay lại bước .

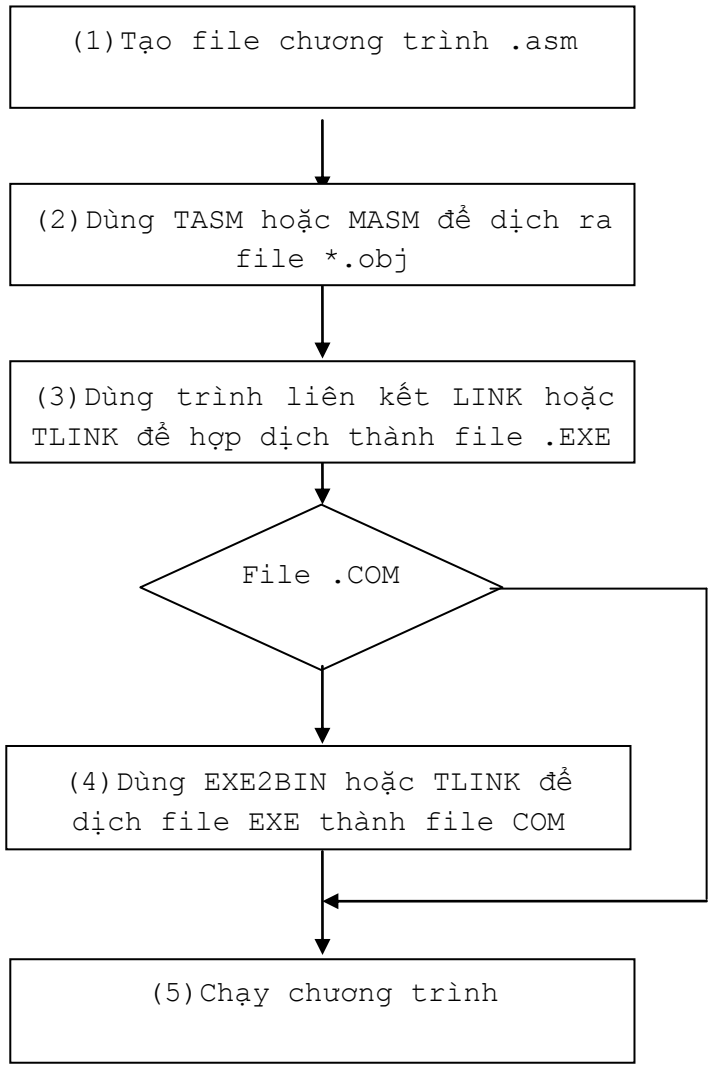
Bước 3: Hợp dịch

Dùng chương trình LINK hoặc TLINK để liên kết một hay nhiều file OBJ lại để thành một file .EXE, file có thể chạy được. Đối với chương trình có cấu trúc là file EXE thì bỏ qua bước

Bước 4: Tạo file .COM

Dùng chương trình EXE2BIN để dịch .EXE thành file .COM.

Bước 5. Thực hiện chương trình vừa tạo.



2.2 CÁC CẤU TRÚC LẬP TRÌNH CƠ BẢN TRONG CHƯƠNG TRÌNH HỢP NGỮ

Phần này trình bày về các cấu trúc lập trình cơ bản được sử dụng trong việc lập trình nói chung và lập trình hợp ngữ nói riêng. Các cấu trúc này thường được sử dụng để điều khiển một lệnh hoặc một khối lệnh. Đó là:

- Cấu trúc tuần tự
- Cấu trúc điều kiện IF-THEN
- Cấu trúc điều kiện rẽ nhánh IF-THEN-ELSE
- Cấu trúc CASE
- Cấu trúc lặp xác định FOR-DO
- Cấu trúc lặp WHILE-DO
- Cấu trúc lặp REPEAT-UNTIL

2.2.1 Cấu trúc tuần tự

Cấu trúc tuần tự có mặt hầu hết tất cả các ngôn ngữ lập trình. Đây là cấu trúc thông dụng, đơn giản nhất, trong đó các lệnh được sắp xếp kế tiếp nhau hết lệnh này đến lệnh khác. Trong quá trình thực hiện chương trình các lệnh tuần tự được xử lý theo thứ tự của chúng. Bắt đầu từ lệnh đầu tiên cho đến khi gặp lệnh cuối cùng của cấu trúc thì công việc cũng được hoàn tất.

Cấu trúc có dạng như sau:

```
lệnh 1
lệnh 2
lệnh 3
….
lệnh n
```

Ví dụ:

Cần tính tổng nội dung các ô nhớ có địa chỉ FFFAh, FFFBh, FFFCh, FFFDh rồi lưu kết quả vào thanh ghi AX.

Để thực hiện việc này ta có thể sử dụng đoạn chương trình sau:

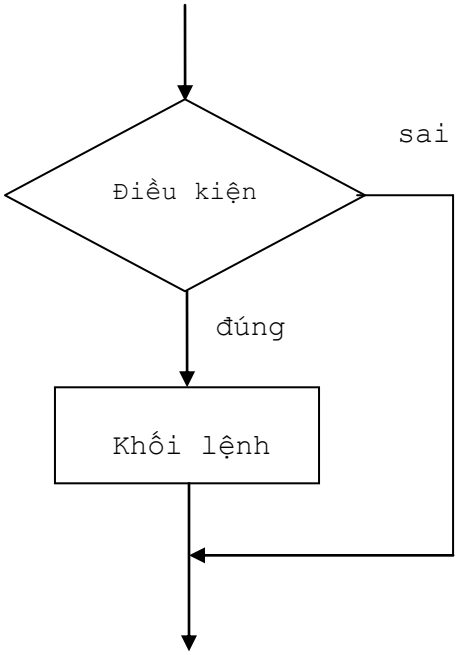
```
Mov BX,FFFAh ; BX trỏ đến FFFAh
Xor AX,AX        ; Tổng =0
Add AL,[BX] ;cộng nội dung ô nhớ có địa chỉ FFFAh vào AX
Inc BX     ;  BX trỏ đến FFFBh
Add AL,[BX]; cộng nội dung ô nhớ có địa chỉ FFFBh vào AX
Inc BX     ;  BX trỏ đến FFFCh
Add AL,[BX]; cộng nội dung ô nhớ có địa chỉ FFFCh vào AX
Inc BX     ;  BX trỏ đến FFFDh
Add AL,[BX]; cộng nội dung ô nhớ có địa chỉ FFFDh vào AX

Xong:         ; ra khỏi cấu trúc
```

2.2.2 Cấu trúc IF… THEN

Đây là cấu trúc điều kiện (conditional statement) mà khối lệnh được thực hiện nếu nó thỏa mãn điều kiện.

Cú pháp: IF <điều kiện> THEN <Khối lệnh>

Cấu trúc này có thể được minh họa bằng sơ đồ khối sau đây:



Khối lệnh có thể gồm 1 hoặc nhiều lệnh. Trong hợp ngữ, để cài đặt cấu trúc IF…THEN người ta thường sử dụng lệnh CMP (so sánh) và đi kèm theo sau là một lệnh nhảy có điều kiện.

Ví dụ:

Viết đoạn chương trình kiểm tra nếu thanh ghi AX>BX thì sẽ tính hiệu AX-BX và lưu kết quả vào thanh ghi AX.

Dưới đây là đoạn chương trình thực hiện công việc đó:

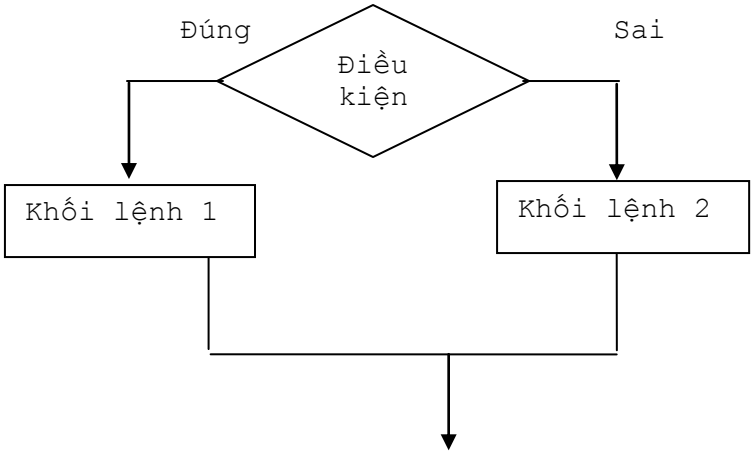
```
        Cmp AX,BX    ; so sánh AX và BX
        Jb Ketthuc   ; nếu AX<BX nhảy đến nhãn Ketthuc
        Sub AX,BX    ;  AX=AX-BX
 Ketthuc:
```

2.2.3 Cấu trúc IF… THEN…ELSE

Đây là dạng phân nhánh của cấu trúc có điều kiện.

Cú pháp: IF <điều kiện> THEN <Khối lệnh 1> ELSE <Khối lệnh 2>

Nếu điều kiện được thỏa mãn thì khối lệnh thứ nhất được thực hiện. Ngược lại, nếu điều kiện là sai thì khối lệnh thứ hai sẽ được thực hiện. Trong mọi trường hợp, một và chỉ một trong hai khối lệnh được thực hiện.



Hình: cấu trúc IF…THEN…ELSE

Trong cài đặt của cấu trúc IF…THEN…ELSE dạng hợp ngữ, thông thường người ta sử dụng một lệnh CMP, một lệnh nhảy có điều kiện và một lệnh nhảy không điều kiện.

Ví dụ: Cho hai số được lưu vào thanh ghi AX và BX, tìm số lớn nhất và lưu kết quả vào thanh ghi DX.

Dưới đây là đoạn chương trình thực hiện công việc đó.

```
        Cmp AX,BX    ; so sánh AX và BX
        JG AXLonHon   ; nếu AX>BX nhảy đến nhãn AXLonhon
        Mov DX,BX    ;  BX>=AX nên DX=AX
       Jmp Ketthuc   ; nhảy đến nhãn Ketthuc
 AXLonHon:
      Mov DX,AX    ;AX>BX nên DX=AX
 Ketthuc:
```

2.2.4 Cấu trúc CASE

Cấu trúc CASE là cấu trúc lựa chọn để thực hiện một khối lệnh giữa nhiều khối lệnh khác.

Cú pháp:

CASE <biểu thức>

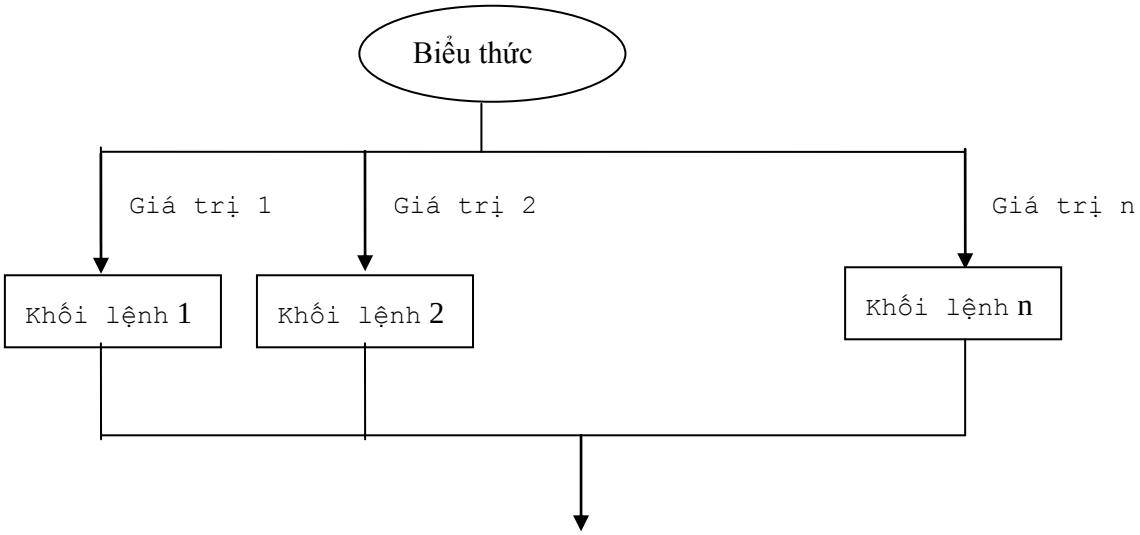
Giá trị 1: Khối lệnh 1

Giá trị 2: Khối lệnh 2

……

Giá trị n: Khối lệnh n

END CASE



Để thực hiện cấu trúc CASE trong chương trình Hợp ngữ, ta phai kết hợp các lệnh CMP, lệnh nhảy có điều kiện, và lệnh nhảy không điều kiện.

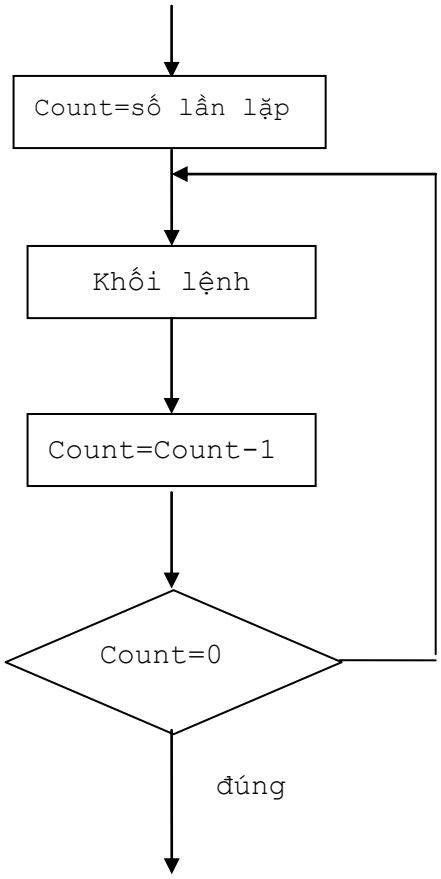
Ví dụ:

```
Am db 'Nhỏ hơn 0','$'
Duong db 'Lớn hơn 0','$'
Khong db 'Bằng không','$'
…
        Sub AX,BX
        Sub AX,CX
        Cmp AX,0
        Je Khong0
        Jb Am0
        Ja Duong0
Khong0:
        Mov AH,9
        Mov DX,offset Khong
        Int 21h
        Jmp Ketthuc
Am0:
        Mov AH,9
        Mov DX,offset Am0
        Int 21h
        Jmp Ketthuc
Duong0:
        Mov AH,9
        Mov DX,offset Duong0
        Int 21h
Ketthuc:
```

Tính hiệu AX-BX-CX và thông báo kết quả là âm, dương hay bằng không.

2.2.5 Cấu trúc lặp FOR-DO

Đây là vòng lặp với số lần lặp đã biết trước. Cú pháp như sau:



FOR Count (=Số lần lặp) DO Khối lệnh

Khối lệnh sẽ được thực hiện Count lần. Để cài đặt cấu trúc này trong hợp ngữ người ta dùng thanh ghi CX để chứa Count và kết hợp với lệnh LOOP để duy trì vòng lặp. Mỗi lần lặp xong thì CX sẽ tự động giảm đi 1.

Ví dụ:

Tính tổng S=1+2+3+….+100

Lưu kết quả vào thanh ghi AX.

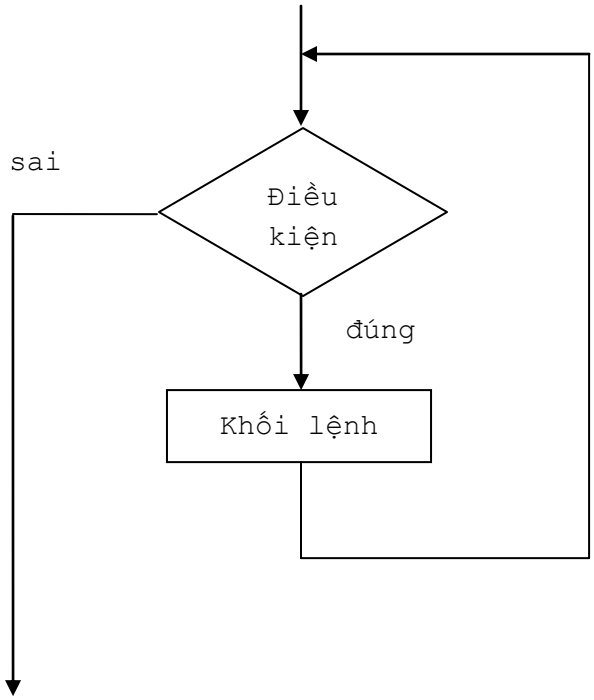
Mỗi lần thực hiện khối lệnh ta sẽ cộng vào AX một số. Lần thực hiện thứ i thì 100-i+1 sẽ được cộng vào AX. Như vậy số lần lặp sẽ là 100. Đoạn chương trình được viết như sau:

```
        Mov CX,100 ; khởi tạo số lần lặp
        Xor AX,AX  ; AX=0 để chứa tổng
Cong:
        Add AX,CX ; Cộng  CX vào AX
        Loop Cong ; Lặp cho đến khi CX=0
                  ; ra khỏi vòng lặp, AX chứa tổng
```

2.2.6 Cấu trúc lặp WHILE-DO

Cú pháp: WHILE điều kiện DO Khối lệnh

Trong cấu trúc lặp WHILE…DO điều kiện được kiểm tra trước khi thực hiện khối lệnh. Nếu điều kiện đúng thì khối lệnh được thực hiện, còn điều kiện sai thì vòng lặp sẽ dừng. Số lần thực hiện khối lệnh chưa được biết trước.



Để cài đặt cấu trúc này trong chương trình hợp ngữ, người ta thường dùng lệnh CMP để kiểm tra điều kiện và kết hợp mới một lệnh nhảy có điều kiện để thoát khỏi vòng lặp.

Ví dụ:

Nhập vào một số nguyên lớn hơn 0 và bé hơn 9 từ bàn phím. Kiểm tra xem có nhập đúng không?. Nếu nhập sai thì yêu cầu phải nhập lại.

Lời giải

Ta biết rằng số 0 có mã ASCII là 30h và số 9 có mã ASCII là 39h. Đoạn chương trình sẽ kiểm tra nếu kí tự gõ vào có mã ASCII bé hơn 30h hoặc lớn hơn 39h thì sẽ yêu cầu người dùng nhập lại kí tự khác.

Dưới đây là đoạn chương trình.

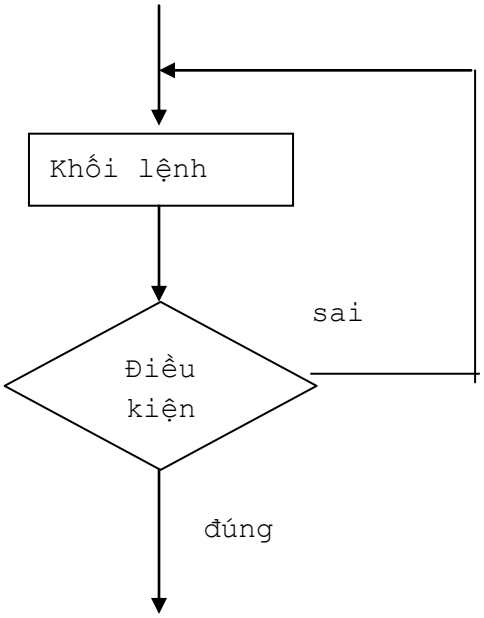
```
Mov AH,01 ; nhập vào 1 kí tự
Nhap:
      Int 21h
      Cmp AL,30h ; kí tự nhập vào <'0'
      Jb Nhap    ; nhập lại
      Cmp AL,39h ;  kí tự nhập vào >'9'
      Ja Nhap    ; nhập lại
```

```
Xong: Sub AL,30h ; Kí tự đã hợp lệ, đổi ra số
```

2.2.7 Cấu trúc lặp REPEAT-UNTIL

Cú pháp: REPEAT Khối lệnh UNTIL Điều kiện

Trong cấu trúc này, khối lệnh được thực hiện ít nhất một lần, sau đó mới kiểm tra điều kiện. Khối lệnh sẽ được lặp đi lặp lại cho đến khi điều kiện thỏa mãn.



Để cài đặt cấu trúc này trong hợp ngữ, người ta thường dùng một lệnh CMP đi kèm với với một lệnh nhảy có điều kiện.

Ví dụ:

Tìm n nhỏ nhất sao cho 1+2+3 +…+n >10000, lưu kết quả (số n) vào BX

Lời giải:

Ta cộng vào tổng (chứa trong AX) các số 1,2,… mỗi lần tính tổng ta đều so sánh tổng đó với 10000, ta cộng cho đến khi AX>10000 thì ta dừng, số hạng cuối cùng của tổng chính là số n cần tìm.

Đoạn chương trình được viết như sau:

```
      Mov CX,1  ; CX=1
      Xor AX,AX ; AX=Tong=0
Cong:
      Add AX,CX ; Cộng CX vào AX
      Cmp AX ,10000 ; Tổng đã > 10000 chưa?
      Ja Xong       ; đã lớn hơn, xong
      Inc CX   ; Tăng CX lên 1
      Jmp Cong  ; Tiếp tục cộng
Xong:
 Mov BX,CX ; lưu n vào BX
```

2.3 CHƯƠNG TRÌNH CON VÀ MACRO

2.3.1 Chương trình con: cơ chế làm việc và cấu trúc

a. Khái niệm:

Chương trình con có mặt hầu hết các ngôn ngữ lập trình. Chương trình con rất có ý nghĩa trong lập trình có cấu trúc. Nó làm cho chương trình trở lên sáng sủa, dễ bảo trì hơn. Bên cạnh đó, nó còn có một ý nghĩa khác: một chương trình con được viết *một* lần nhưng được sử dụng (gọi đến) *nhiều* lần.

Một cách định nghĩa đơn giản: chương trình con là một nhóm các lệnh được gộp lại phục vụ cho việc sử dụng nhiều lần thông qua tên và các tham số của chương trình con đó .

Ví dụ: Một bài toán yêu cầu ta tính tổng của 3 số hạng được nhập từ bàn phím. Thay vì phải viết 3 đoạn chương trình (khá giống nhau) để lần lượt nhập từng số thì ta viết 1 chương trình con nhập vào 1 số rồi gọi nó 3 lần.

b. Cơ chế làm việc của chương trình con

Giả sử có một chương trình (chính) đang thực hiện như sau:

Địa chỉ	Lệnh
1F00	Mov AX,1
1F02	Mov BX,2FFF
1F04	Mov CX,2
1F06	Call TinhTong
1F09	Sub AX,BX
1F0B	Add AX,CX
1F0D	SHR AX,1
1F0F	Jmp Done

Chương trình con TinhTong được lưu ở 1 vùng nhớ khác

Địa chỉ	Lệnh
FFD0	Mov DL,[BX]
FFD2	Add AL,[BX]
FFD4	Inc BX
FFD6	Add AL,[BX]
FFD8	Ret

Khi thực hiện đến lệnh Call TinhTong ở địa chỉ 1F06h, bộ xử lý sẽ thực hiện như sau:

- Lưu địa chỉ của lệnh kế tiếp 1F09h vào ngăn xếp.
- Nạp địa chỉ của lệnh đầu tiên của chương trình con FFD0h vào IP
- Các lệnh của chương trình con được thực hiện cho đến khi gặp lệnh RET, chương trình con kết thúc và trả lại quyền điều khiển cho chương trình chính.
- Địa chỉ 1F09h được lấy ra từ ngăn xếp rồi đặt vào IP. Lệnh tại địa chỉ 1F09h (Sub AX,BX) của chương trình chính sẽ được thực hiện.

c. Cấu trúc của chương trình con

Cấu trúc của một chương trình con có dạng như sau:

<Tên chương trình con> PROC NEAR/FAR

Các lệnh của chương trình con được viết ở đây

RET

<Tên chương trình con> ENDP

Giải thích:

- Lệnh điều khiển PROC được sử dụng để khởi động chương trình con. Nhãn đứng trước toán tử PROC là tên của thủ tục. Sau toán tử PROC có lệnh điều khiển NEAR hoặc FAR để báo cho lệnh RET lấy địa chỉ quay về chương trình của nó trong ngăn xếp.
- Nếu là NEAR thì chương trình con được gọi thì địa chỉ OFFSET (16 bít) được lấy từ ngăn xếp để gán cho thanh ghi IP. Nghĩa là, trong trường hợp này thì chương trình con và chương trình gọi nó ở trên cùng một đoạn (segment)
- Nếu là FAR thì chỉ lấy địa chỉ SEGMENT và OFFSET trong ngăn xếp được lấy ra để gán cho thanh ghi CS và IP. Nghĩa là, trong trường hợp này thì chương trình con và chương trình gọi nó ở hai đoạn khác nhau.

Để lệnh RET có đầy đủ thông tin để biết là phải nạp cả CS:IP hay chỉ nạp IP, có hai cách sau:

- Thay vì sử dụng RET ta sử dụng lệnh RETN (near return) hoặc RETF (far return).
- Dùng hai lệnh điều khiển PROC và ENDP. Lệnh ENDP sẽ đánh dấu kết thúc chương trình con. Nếu đi sau PROC là NEAR thì tất cả các lệnh RET trong chương trình con nằm trong PROC … ENDP đều là RETN nằm chung một đoạn với chương trình gọi đến chương trình con này. Nếu đi sau PROC là FAR thì tất cả các lệnh RET trong chương trình con nằm trong PROC … ENDP đều là RETF (nằm chung một đoạn với chương trình gọi đến chương trình con này).
- Nếu cả chương trình chính và các chương trình con có kích thước nhỏ (tất cả mã lệnh không vượt quá 64KB) thì ta nên sử dụng chúng như là các thủ tục NEAR. Vì như thế chương trình sẽ được thực hiện nhanh hơn.
- Khi sử dụng mô hình bộ nhớ, các giá trị NEAR là FAR cũng được gán ngầm định theo mô hình kích thước bộ nhớ. Chẳng hạn: nếu ta dùng .MODEL Tiny hoặc .MODEL Compact thì chương trình con sẽ được tự động xác định là NEAR. Còn nếu ta dùng

.MODEL Medium, .MODEL Large hoặc .MODEL Huge thì chương trình con sẽ được tự động xác định là FAR.

- Trong trường hợp ta không có khai báo NEAR hoặc FAR sau lệnh PROC thì ngầm định là NEAR.

2.3.2 Truyền tham số

Đây là phần rất quan trọng khi chương trình được thiết kế thành các chương trình con. Chúng được "giao tiếp" với nhau một cách trong suốt, lô gic trong quá trình thực hiện các chức năng của mình để tăng thêm tính tái sử dụng- một tính chất quan trọng của chương trình con. Dữ liệu phải được trao đổi từ chương trình gọi và chương trình con được gọi. Trong các ngôn ngữ bậc cao khác, cấu trúc chương trình con cho phép người lập trình khai báo danh sách tham số. Tuy nhiên, như ta thấy cấu trúc của một chương trình con hợp ngữ ta không thấy đi kèm với một danh sách tham số. Dưới đây ta sẽ xem xét tất cả các vấn đề liên quan đến việc truyền tham số.

a. Truyền giá trị tham số từ chương trình gọi sang chương trình được gọi

- Truyền tham số thông qua các thanh ghi: đây là cách thức đơn giản và dễ thực hiện nhất, thường được sử dụng đối với các chương trình được viết thuần túy bằng hợp ngữ. Để thực hiện cách truyền tham số này ta chỉ cần đặt một giá trị nào đó vào thanh ghi ở chương trình gọi và sau đó chương trình con được gọi sẽ sử dụng giá trị ở thanh ghi này.
- Truyền tham số thông qua các biến toàn cục: các biến toàn cục được khai báo trong chương trình chính có tác dụng trong toàn bộ chương trình (cả chương trình chính và các chương trình con). Vì vậy ta có thể dùng nó để truyền giá trị giữa chương trình chính và các chương trình con. Cách này khá phổ biến khi ta viết chương trình thuần túy bằng hợp ngữ hoặc phát triển chương trình hỗn hợp bằng hợp ngữ và các ngôn ngữ bậc cao khác.
- Truyền tham số thông qua ngăn xếp: đây là phương pháp khá phức tạp. Tuy nhiên, cách này được sử dụng rất nhiều khi ta viết các module bằng hợp ngữ và các ngôn ngữ bậc cao khác rồi cho chúng liên kết (link) với nhau trong quá trình thực hiện. Cách này sẽ được đề cập chi tiết ở phần sau: *kết nối chương trình hợp ngữ với các chương trình ngôn ngữ bậc cao*.

b. Truyền giá trị tham số từ chương trình được gọi lên chương trình gọi

- Truyền tham số từ chương trình được gọi (chương trình con) lên chương trình gọi (chương trình chính) cũng theo ba cách: thông qua các thanh ghi, biến toàn cục và ngăn xếp. Trong trường hợp liên kết với ngôn ngữ bậc cao thì chương trình con được gọi (được viết bằng hợp ngữ) có thể chuyển giá trị lên chương trình gọi (được viết bằng ngôn ngữ bậc cao) bằng giá trị trả về (returned value). Để làm được điều này trong hợp ngữ thì giá trị trả về của chương trình con được gọi phải tuân thủ các qui cách sau:

  + Nếu giá trị trả về (tên hàm mang giá trị trả về) là 8 bít hoặc 16 bít thì giá trị đó phải được đặt vào thanh ghi AX của hàm trước khi quay về chương trình gọi nó.

  + Nếu giá trị trả về (tên hàm mang giá trị trả về) là 32 bít thì giá trị đó phải được đặt vào thanh ghi DX:AX của hàm trước khi quay về chương trình gọi nó.

- Lưu ý rằng số lượng các thanh ghi của máy tính là có hạn, nên ta không nên dùng quá nhiều thanh ghi cho việc chuyển giao các tham số.

c. Vấn đề bảo vệ các thanh ghi

Khác với lập trình với các ngôn ngữ bậc cao, khi lập trình hợp ngữ người lập trình hợp ngữ còn phải để ý đến việc bảo vệ các thanh ghi trong quá trình gọi các chương trình con. Ở các ngôn ngữ bậc cao, chương trình con không làm thay đổi giá trị của các biến của chương trình chính trừ khi ta chủ tâm làm việc đó. Trong các chương trình được viết bằng hợp ngũ thì ngược lại là rất hay xảy ra trường hợp là các giá trị của các biến trong chương trình chính được nạp vào các thanh ghi mà trong khi đó chương trình con cũng cần các thanh ghi này để thực hiện một công việc nào đó. Và như vậy thì chương trình con khi sử dụng thanh ghi có thể sẽ xóa giá trị trong thanh ghi mà chương trình chính đã đặt vào đó để sử về sau. Do vậy các giá trị đã được lưu vào trong thanh ghi cần phải được bảo vệ khi cần thiết. Có hai cách người ta hay dùng là:

- Sử dụng các lệnh PUSH và POP: Khi bắt đâu một chương trình con, ta nên tiến hành lưu các giá trị của các thanh ghi mà chương trình con sẽ dùng đến vào ngăn xếp nhờ lệnh PUSH và trước khi ra khỏi chương trình con ta phải phục hồi lại các giá trị của các thanh ghi đó từ ngăn xếp nhờ lệnh POP.
- Sử dụng theo một qui ước nhất quán (code convension): qui định sử dụng một số thanh ghi sử dụng cho chương trình chính và tất cả các chương trình con không được sử dụng đến các thanh ghi đó.

2.3.3 Chương trình gồm nhiều module

Đó là chương trình gồm nhiều file, thích hợp cho các chương trình lớn và phức tạp. Chúng được dịch một cách độc lập nhưng được hợp dịch (link) với nhau khi chạy. Sau đây là những ưu điểm của việc viết chương trình gồm nhiều file:

- Cho phép nhiều người lập trình cùng tham gia phát triển một chương trình lớn.
- Dễ dàng cho việc sửa lỗi, khi dịch module nào phát hiện ra lỗi thì chỉ cần sửa và dịch lại module đó.
- Mỗi module thường giải quyết một vấn đề ngắn gọn nên dễ tìm sai sót.

Để chia xẻ các biến toàn cục hoặc các chương trình con được sử dụng chung giữa các module người ta sử dụng các lệnh điều khiển PUBLIC, EXTRN và GLOBAL.

a. Lệnh điều khiển PUBLIC

Chức năng: Lệnh điều khiển PUBLIC chỉ cho chương trình dịch hợp ngữ biết nhãn nào nằm trong module này được phép sử dụng ở các module khác.

Cú pháp: PUBLIC tên nhãn

Khai báo nhãn

Trong đó tên nhãn có thể là:

- Tên chương trình con
- Tên biến
- Tên hằng (theo sau bởi lệnh EQU)

Ví dụ:

Nhãn là tên biến nhớ

```
.DATA
PUBLIC  gTong, gSoHang, gMang, gMangLength
gTong dd ?
gSoHang dw ?
gMangLength EQU 100
gMang db gMangLength DUP(?)
Nhãn là tên của chương trình con
.CODE
PUBLIC  gTinhTong, gTimMax
gTinhTong PROC NEAR
….
gTinhTong ENDP
    ;--------------------------------
gTimMax PROC FAR
….

gTimMax ENDP
   ;---------------------------------
```

Chú ý: chương trình dịch hợp ngữ không phân biệt chữ hoa hay thường trong các nhãn. Tất cả các chữ đều hiểu như chữ hoa. Nếu muốn có sự phân biệt đó thì:

- dùng tùy chọn /ml khi dịch cho tất cả mọi ký hiệu
- dùng tùy chon /mx khi dịch cho các nhãn được khai báo PUBLIC, EXTRN, hoặc GLOBAL.

b. Lệnh điều khiển EXTRN

Chức năng: Lệnh điều khiển EXTRN báo cho chương trình dịch hợp ngữ biết nhãn nào đã được khai báo PUBLIC ở các module khác được sử dụng trong module này. Nói cách khác các nhãn đã được PUBLIC ở các module khác sẽ được sử dụng trong module này mà không cần khai báo lại nếu chúng được khai báo EXTRN.

Cú pháp: EXTRN tên nhãn: kiểu

Trong đó kiểu có các dạng như sau:

Kiểu	Giải thích
ABS	Giá trị tuyệt đối, dùng để khai báo các nhãn được xác định bởi EQU hoặc =
BYTE	Giá trị nhãn là 1 byte
WORD	Giá trị nhãn là 2 byte
DWORD	Giá trị nhãn là 4 byte
FWORD	Giá trị nhãn là 6 byte

QWORD	Giá trị nhãn là 8 byte
TBYTE	Giá trị nhãn là 10 byte
DATAPTR	Con trỏ NEAR hoặc FAR phụ thuộc vào MODEL của bộ nhớ
NEAR	Chỉ chương trình con dạng khai báo NEAR
FAR	Chỉ chương trình con dạng khai báo FAR
PROC	Xác định nhãn là thủ tục; còn NEAR hoặc FAR phụ thuộc vào lệnh điều khiển .MODEL
UNKNOWN	Cho nhãn không biết kích cỡ

Các kiểu dữ liệu theo sau nhãn được khai báo EXTRN phải xác định đúng, nếu không sẽ gây ra sai sót.

Ví dụ:

Ở module 1 có chương trình con được khai báo như sau:

```
PUBLIC TinhTong
 TinhTong PROC FAR
……
   Ret
 TinhTong ENDP
```

Ở module 2 sử dụng chương trình con TinhTong được khai báo trong module 1.

```
.CODE
EXTRN TinhTong: FAR
…
Call TinhTong
```

Để sử dụng EXTRN để khai báo cho chương trình dịch hợp ngữ biết những nhãn nào đã được khai báo PUBLIC ở phần trước được sử dụng trong module này.

```
.DATA
EXTRN gTong:DWORD, gSoHang:WORD, gMang:BYTE,gMangLength:ABS
EXTRN TinhTong: NEAR, TimMax: FAR
 ….
Call TinhTong
….
Call TimMax
…
```

c. Lệnh điều khiển GLOBAL

Lệnh GLOBAL được hỗ trợ bởi chương trình dịch TASM (Turbo Assembler) của hẵng Borland. Lệnh điều khiển này còn có thể thay thế hai lệnh PUBLIC và EXTRN. Nếu ta khai báo GLOBAL cho các nhãn có kèm theo khai báo dạng nhãn thì GLOBAL trong trường hợp này sẽ

thay thế cho PUBLIC, còn khi khai báo nhãn đi sau GLOBAL mà chỉ xác định kiểu nhãn thì GLOBAL sẽ thay thế EXTRN.

Ví dụ:

```
.DATA
GLOBAL gSoHang:WORD, gMang:BYTE
Count DW ?
…
.CODE
GLOBAL TinhTong: NEAR, TimMax: FAR
TimMax PROC FAR
Call TinhTong
…
```

Các nhãn TinhTong, TimMax được khai báo do đó lệnh điều khiển GLOBAL đối với các nhãn này có ý nghĩa như PUBLIC, còn các nhãn gSoHang, gMang chỉ nêu kiểu mà không khai báo thì GLOBAL đối với chúng là EXTRN.

Một trường hợp vô cùng thuận lợi với việc sử dụng lệnh điều khiển GLOBAL là việc sử dụng GLOBAL trng file INCLUDE. Giả sử ta có một tập hợp các nhãn mà ta muốn sử dụng ở tất cả các module của chương trình gồm nhiều module. Ta có thể làm được như vậy nhờ việc gộp tất cả các nhãn vào file INCLUDE và sau đó đưa file này vào các module. Trong trường hợp này ta không thể sử dụng PUBLIC hoặc EXTRN vì EXTRN không thể làm việc được với các nhãn có xác định kích thước khai báo. Còn lệnh PUBLIC chỉ làm việc với các module trong đó các nhãn được khai báo mà không xác định kiểu. Do vậy, chỉ có GLOBAL là thỏ mãn cả hai điều kiện trên.

Ví dụ về một chương trình nằm trên hai file (hai module) khác nhau. Với:

- Module của chương trình chính là main.asm có chức năng xác định địa chỉ OFFSET của hai xâu kí tự, gọi chương trình con làm nhiệm vụ nối hai xâu kí tự đó lại và hiển thị xâu kết quả.
- Module chương trình con là KetNoi.ASM làm nhiệm vụ kết nối hai xâu và xếp vào vùng nhớ kết quả.

Dưới đây là chương trình chính:

```
.MODEL SMALL
.STACK 100h
.DATA
  Xau1 DB "Hello",0
  Xau2 DB "Mr Bin", 13,10,'$',0
  GLOBAL XauKQ:BYTE
  XauKQ DB 50 DUP (?)
.CODE
   EXTRN KetNoi:PROC
Start:
      Mov AX,@Data
      Mov DS,AX
      Mov AX,offset Xau1; AX chứa OFFSET của Xau1
```

```
            Mov BX,offset Xau2; BX chứa OFFSET của Xau2
            Call KetNoi  ; nối hai xâu
            Mov DX,Offset XauKQ ; In ra màn hình
            Mov AH,9
            Int 21h
            Mov AH,4Ch ; Trở về DOS
            Int 21h
     End Start
```

Module của chương trình con KetNoi

```
     .MODEL SMALL
     .STACK 100h
     .DATA
       GLOBAL XauKQ:BYTE
     .CODE
     PUBLIC KetNoi
     KetNoi PROC
     Cld
          Mov DI, SEG XauKQ ; ES:DI trỏ đến xâu kq
          Mov ES,DI
          Mov DI, OFFSET XauKQ
          Mov SI,AX; DS:SI trỏ đến Xau1
Lap1:
          Lodsb   ; lấy 1 kí tự đưa vào AL
          And AL,AL ; cho ZF=1
          Jz DoKetNoi
          Stosb  ; Lưu kí tự từ AL vào xâu
          Jmp Lap
DoKetNoi:
          Mov SI,BX; DS:SI; trỏ đến Xau2
Lap2:
          Lodsb  ; lấy kí tự đưa vào AL
          Stosb ; Cất kí tự vào XauKQ
          And AL,AL  ; cho ZF=1
          Jnz Lap2  ; giá trị khác 0 thì nhảy
          Ret   ; trở về chương trình chính
KetNoi ENDP
END
```

Để chạy hai module trên ta thực hiện theo các thao tác sau:

Dịch hai module một cách tách biệt

```
     TASM Main
     TASM KetNoi
```

Sau đó tiến hành hợp dịchTLINK Main+KetNoi ta sẽ có file Main.exe.

2.3.4 Liên kết thủ tục vào một thư viện

Trong quá trình lập trình, có thể một số thao tác sau ta cần thực hiện:

- Đưa một khối lệnh vào các nơi khác nhau hoặc các module nguồn
- Phân chia các nhãn gán (EQU) và MACRO giữa các phần khác nhau của một chương trình hoặc sử dụng lại chúng trong nhiều chương trình.
- Viết một chương trình dài, xong không muốn chia nhỏ ra nhiều module vì phải dịch từng module rồi liên kết chúng với nhau song chương trình quá to không thể chứa trong một file.

Để giải quyết các vấn đề trên, chương trình dịch của hợp ngữ có lệnh INCLUDE.

Giả sử ta muốn tạo một file INCLUDE chứa khối lệnh mà ta muốn các module khác thêm vào khi dịch.

Cú pháp:

INCLUDE tên file

(file chứa khối lệnh cần được đưa vào vị trí mà lệnh INCLUDE đang đứng)

Cơ chế: Khi chương trình hợp ngữ gặp lệnh INCLUDE thì sẽ tìm đến đường dẫn chứa file đã được xác định sau INCLUDE và đưa toàn bộ khối lệnh mà tệp này chứa xen vào vị trí mà lệnh INCLUDE đang đứng của module chương trình. Hay nói cách khác nội dung của tệp INCLUDE được đặt vào đúng vùng nhớ của chương trình mà lệnh INCLUDE đang được xác định.

Ví dụ:

Giả sử có một chương trình trong file A.ASM có nội dung như sau:

```
…
.CODE
Mov BX,10
Add AX,BX
INCLUDE  B.ASM
Sub AX,CX
```

File B.ASM có nội dung như sau:

```
Mov CX,3
Mov DX,4
```

Kết quả dịch chương trình A.ASM sẽ như sau:

```
.CODE
Mov BX,10
Add AX,BX
Mov CX,3
Mov DX,4
Sub AX,CX
```

Qua ví dụ trên ta thấy trong quá trình dịch chương trình A.ASM, khi đến dòng lệnh INCLUDE B.ASM

Thì chương trình dịch lấy tất cả các lệnh của B.ASM đặt vào vị trí mà lệnh INCLUDE đang đứng. ngoài ra các lệnh INCLUDE còn có tính chất lồng nhau với những mức khác nhau, có nghĩa là trong file INCLUDE có thể gọi 1 file INCLUDE khác.

Dưới đây là cơ chế mà chương trình dịch tìm các file INCLUDE:

- Nếu trong lệnh INCLUDE chỉ rõ tên ổ đĩa, đường dẫn, tên file thì chương trình dịch sẽ tìm theo sự xác định ở trên
- Nếu trong lệnh INCLUDE chỉ xác định tên file thì chương trình dịch tiến hành tìm file này ở thư mục hiện thời. Trường hợp không tìm thấy thì sẽ tìm file đó ở thư mục được chỉ ra trong câu lệnh:

  TASM –i <đường dẫn đến file >
- Nếu không tìm thấy file INCLUDE trong tất cả các trường hợp trên thì chương trình dịch sẽ báo không tìm thấy file INCLUDE.

2.3.5 Macro

Trước khi đi vào tìm hiểu có chế hoạt động, cách viết các macro ta hãy tìm hiểu một số lệnh thường được sử dụng để viết các macro.

a. Các lệnh lặp và điều khiển điều kiện khi dịch

Các lệnh lặp có chức năng là thực hiện một khối lệnh nhiều lần theo số lần lặp (do người lập trình đặt sẵn). Ta hãy xem xét một số lệnh lặp thường hay được sử dụng.

1. Lệnh: REPT

Chức năng: lặp một khối lệnh

Cú pháp:

```
REPT <số lần lặp>
    Khối lệnh
ENDM   ; kết thúc lệnh lặp
```

Ví dụ:

```
REPT 100
    DB ?
ENDM
```

Khi tiến hành dịch nó sẽ thực hiện khai báo như sau:

```
DB ?
DB ?
 …
DB ?
```

; 100 lần khai báo

Trong trường hợp này giống như khai báo

```
DB 100 Dup(?)
```

Tuy nhiên, hai cách này cũng không hoàn toàn giống nhau.

Ví dụ 2:

```
REPT 100
    DB Count
```

```
    Count= Count+1
ENDM
```

Khi dịch ra sẽ được triển khai như sau:

```
DB 0
DB 1
…
DB 99
```

2.Lệnh: IRP

Chức năng: Lệnh này cho phép lặp một khối lệnh theo số lượng danh sách các tham số với sự thay đổi các giá trị của tham số trong khối lệnh.

Cú pháp:

```
IRP tên tham số <danh sách tham số>
    Khối lệnh
ENDM
```

Ví dụ:

```
IRP varX <0,2,4,6,8,10>
     DB varX
ENDM
```

Khi dịch khối lệnh trên sẽ được dịch như sau:

```
     DB 0
     DB 2
  …
     DB 10
```

3. Lệnh: IF và IFE

Chức năng: Lệnh điều khiển IF báo cho chương trình dịch hợp ngữ biết phải thực hiện việc dịch khối lệnh khi giá trị của biểu thức khác 0.

Cú pháp:

```
IF <biểu thức>
    Khối lệnh
ENDIF
```

Hoặc:

```
IF <biểu thức>
    Khối lệnh 1
ELSE
    Khối lệnh 2
ENDIF
```

Ví dụ:

```
IF is8086 ; nếu CPU là 8086 thì không cho phép lưu vào
    Mov AX,EFh ; một hằng số trực tiếp vào ngăn xếp
   Push AX
ELSE
    Push EFh ; nếu không phải thì có thể
```

```
ENDIF
```

Lệnh điều khiển IFE giống IF song việc dịch được tiến hành khi giá trị của biểu thức bằng 0

```
IFE 0
    Khối lệnh
ENDIF
```

Luôn dịch khối lệnh bên trong. IFE và ENDIF

4. Lệnh: IFDEF và IFNDEF

Chức năng: Lệnh điều khiển IFDEF báo cho chương trình dịch hợp ngữ biết phải thực hiện khối lệnh khi nhãn đã được khai báo.

Cú pháp:

```
IFDEF Nhãn
    Khối lệnh
ENDIF
```

Hoặc:

```
IF Nhãn
    Khối lệnh 1
ELSE
    Khối lệnh 2
ENDIF
```

Lệnh điều khiển IFNDEF giống lệnh trên nhưng với điều kiện ngược lại là chương trình dịch sẽ thực hiện khối lệnh nếu nhãn chưa được khai báo.

Ví dụ:

Tránh việc khai báo hai lần giá trị khởi đầu của một biến

```
varX DB 1
….
IF varX
        Display "khai báo trùng hợp biến varX "
 Err
ELSE
 varX DB 10
ENDIF
```

5. Lệnh: IFB và IFNB

Chức năng: Các lệnh này được dùng để kiểm tra việc truyền tham số cho macro. Chúng báo cho chương trình dịch hợp ngữ biết phải thực hiện khối lệnh nếu tham số là rỗng.

Cú pháp:

```
IFB Tham số
    Khối lệnh
ENDIF
```

Hoặc:

```
IFB Tham số
```

```
    Khối lệnh 1
ELSE
    Khối lệnh 2
ENDIF
```

Lệnh điều khiển IFNB giống lệnh trên nhưng với điều kiện ngược lại là chương trình dịch sẽ thực hiện khối lệnh nếu thông số không phải là dấu trống.

6. Lệnh: IFDIF và IFIDN

Chức năng: Lệnh IFDIF báo cho chương trình dịch hợp ngữ biết phải thực hiện khối lệnh nếu hai tham số bằng nhau.

Cú pháp:

```
IFDIF Tham số1, Tham số 2
    Khối lệnh
ENDIF
```

Hoặc:

```
IFDIF Tham số1, Tham số 2
    Khối lệnh 1
ELSE
    Khối lệnh 2
ENDIF
```

Lệnh điều khiển IFIDN giống lệnh IFDIF nhưng với điều kiện ngược lại là chương trình dịch sẽ thực hiện khối lệnh nếu hai tham số phải là khác nhau.

Chú ý: Tên tham số của 2 lệnh trên có phân biệt chữ hoa, chữ thường (Case sensitive)

b. Cơ bản về Macro

Macro bao gồm một tên đại diện và một khối lệnh. Khi chương trình dịch hợp ngữ gặp tên này ở đâu trong chương trình thì khối lệnh sẽ được dịch và đặt khối mã lệnh vào vị trí mà chương trình gọi tên macro.

Về cơ chế thực hiện thì macro giống với file INCLUDE. Nghĩa là, mỗi lần chương trình dịch gặp tên của macro hay tên file INCLUDE thì toàn bộ khối lệnh được định nghĩa trong macro hay file INCLUDE sẽ được dịch và đặt vào vị trí lời gọi.

Tuy nhiên, giữa file INCLUDE và MACRO có một số điểm khác nhau sau đây. Nhìn chung, macro mạnh hơn file INCLUDE vì:

- Macro có thể truyền tham số.
- Macro có thể chứa các nhãn cục bộ.
- Việc dịch Macro diễn ra nhanh hơn vì macro là 1 phần của chương trình (không phải là file) nên không phải đọc từ đĩa ngoài.
- Macro còn sử dụng một số lệnh điều khiển điều kiện hoặc lặp khi thực hiện.

c. Khai báo và sử dụng Macro

Trước khi sử dụng , ta phải khai báo macro theo cú pháp sau:

Tên macro MACRO các tham số

Thân macro

ENDM

Ví dụ:

```
Cong MACRO
Xor AX,AX
Add AX,BX
Add AX,CX
Add AX,DX
ENDM
```

Để sử dụng Macro này ta chỉ cần đưa tên của macro vào vị trí gọi.

```
…
Mov BX,10
Mov CX,100
Mov DX,1000
Cong
…
```

Chú ý:

- Chương trình con thường được sử dụng khi cần tiết kiệm vùng nhớ vì với chương trình con thì các mã lệnh được dịch ra cho một nhiệm vụ nào đó chỉ có một lần và nó sẽ được gọi ở bất kỳ nơi nào trong chương trình khi cần đến. Tuy nhiên Macro được đánh giá là thực hiện nhanh hơn vì nó thực hiện một nhóm lệnh như 1 phần của chương trình gọi nó và không phải sử dụng lệnh CALL và RET.
- Macro linh hoạt hơn so với chương trình con. Chúng sử dụng các lệnh điều khiển để trao đổi giữa tham trị và tham số hình thức. Nếu trong 1 chương trình có nhiều đoạn mã lệnh thực hiện các nhiệm vụ gần giống nhau thì macro rất hiệu quả.

d. Trao đổi tham số

Một macro có thể có 0,1 hoặc nhiều tham số. Macro ở ví dụ trên là macro không có tham số. Ta xem xét một số macro có tham số thông qua các ví dụ.

Ví dụ:

Dùng 1 macro để khai báo một mảng kiểu byte, có độ dài Length và tất cả các phần tử được gán giá trị ban đầu là Value.

```
Mang MACRO Value, Length
     REPT Length
 DB Value
ENDM
```

Để sử dụng Macro ở trên, ta chỉ việc truyền các giá trị cho các tham số. Chẳng hạn, cần khởi tạo 10 phần tử và giá trị khởi tạo là 0 thì ta làm như sau:

```
MangByte LABEL BYTE
Mang 0,10
```

Các giá trị 0 và 10 được truyền vào tham số Value và Length khi Macro được sử dụng. 0 và 10 là tham số thực trong khi Value và Length là tham số hình thức. Khi macro được gọi thì các tham số hình thức được thay bởi tham số thực. Nghĩa là:

```
MangByte LABEL BYTE
```

```
        REPT 10
    DB 0
ENDM
```

2.4. KẾT NỐI HỢP NGỮ VỚI CÁC NGÔN NGỮ BẬC CAO

Phần này giới thiệu cách thức kết nối một chương trình hợp ngữ với các ngôn ngữ bậc cao như C và Pascal. Việc chuyển đổi một đoạn chương trình từ ngôn ngữ bậc cao sang dạng hợp ngữ sẽ làm cho tốc độ thực hiện của chương trình sẽ được cải thiện đáng kể. Trong nhiều trường hợp, nó còn làm đơn giản cho người lập trình khi viết các đoạn chương trình liên quan đến thao tác phần cứng và các thiết bị ngoại vị thông qua các dịch vụ ở mức BIOS.

2.4.1 Ngôn ngữ C và Hợp ngữ

Để liên kết các đoạn chương trình hợp ngữ vào ngôn ngữ C hoặc Pascal thì người ta thường sử dụng một trong hai cách: sử dụng inline assembly hoặc viết tách biệt các module.

a . Sử dụng inline assembly

Chèn các khối lệnh hợp ngữ vào chương trình được viết bằng ngôn ngữ C. Đây là phương pháp nhanh và đơn giản. Người lập trình chỉ phải thêm từ khóa **asm** đứng trước mỗi lệnh. Với phương pháp này, ta có thể dễ dàng đưa các lệnh của hợp ngữ vào giữa các dòng lệnh của C.

Cú pháp đầy đủ của một dòng lệnh inline-assembly

```
asm  [<Nhãn>:] <lệnh> <các toán hạng>
```

hoặc cũng có thể dùng cả một khối lệnh hợp ngữ được gói bên trong cặp dấu {}. Trong nhiều trường hợp dạng sau được sử dụng thuận tiện hơn. Đặc biệt khi có nhiều hơn 1 lệnh hợp ngữ.

```
asm  {
[<Nhãn 1>:] <lệnh 1> <các toán hạng 1>
[<Nhãn 2>:] <lệnh 2> <các toán hạng 2>
….
[<Nhãn n>:] <lệnh n> <các toán hạng n>
}
```

Mỗi khi chương trình dịch của C gặp từ khóa asm trong dòng lệnh inline assembly thì chương trình dịch sẽ chuyển dòng lệnh hợp ngữ này vào và dịch với việc qui chiếu biến C ra dạng tương ứng của hợp ngữ để thực hiện.

Dưới đây là một ví dụ minh họa cả hai dạng cú pháp trên. Trong ví dụ này in hai xâu kí tự đã được định nghĩa sẵn lên màn hình.

Chương trình được viết theo dạng cú pháp thứ nhất

```
#include <stdio.h>
#include <conio.h>
void main()
```

```
{
 char xau1 []=”Hello World $”;
 char xau2 []=”Hello Vietnam $”;
 asm mov dx,offset xau1;
 asm mov ah,09;
 asm int 21h;
/*xuống dòng */
 asm mov ah,02;
 asm mov dl,13;
 asm int 21h;
/*về đầu dòng */
 asm mov dl,10;
 asm int 21h;
 printf (“%s”, xau2);
getch();/*chờ người dùng gõ vào 1 phím*/
}
```

Chương trình được viết theo dạng cú pháp thứ hai

```
#include <stdio.h>
#include <conio.h>
void main()
{
 char xau1 []=”Hello World $”;
 char xau2 []=”Hello Vietnam $”;
 asm  {
        mov dx,offset xau
        mov ah,09
        int 21h
        /*xuống dòng */
        mov ah,02
        mov dl,13
        int 21h
        /*về đầu dòng */
        mov dl,10
        int 21h
    }
  printf (“%s”, xau2);  /* in xâu 2*/
  getch(); /*chờ người dùng gõ vào 1 phím*/

}
```

Chú ý rằng: mọi lời giải thích sẽ phải tuân thủ theo cách của chương trình C.

Chương trình dịch C khi gặp từ khóa asm thì các biến xau1, xau2 của C sẽ được ánh xạ sang các biến tương ứng của hợp ngữ. Nghĩa là, với từ khóa asm ta có thể đặt câu lệnh hợp ngữ ở bất kỳ đâu trong đoạn mã chương trình chương t rình C.

Qua trình dịch của chương trình C có chứa các dòng lệnh hợp ngữ như sau:

- Chương trình dịch C (turbo C) sẽ dịch file chương trình nguồn (phần mở rộng .C) từ dạng .C sang dạng hợp ngữ (đuôi .asm).
- Chương trình TASM sẽ dịch tiếp file .asm sang file .obj
- Trình liên kết TLINK sẽ thực hiện việc liên kết để tạo file .exe.

Trong trường hợp chương trình chỉ chứa các lện C mà không có inline-assembly thì chương trình dịch sẽ dịch trực tiếp file nguồn C sang file .OBJ.

Các cách truy xuất biến của ngôn ngữ C;

- *Truy xuất trực tiếp:*

Các biến được khai báo trong C được coi như các biến "toàn cục" sử dung chung cho cả C và các inline- assembly. Ví dụ chương trình dưới đây tính tổng 2 số nguyên x và y rồi lưu kết quả vào biến sum.

```
#include <stdio.h>
#include <conio.h>
void main()
{
 int x,y, Sum;
 /*Nhập x và y từ bàn phím*/
 printf ("x =  "); scanf("%d",&x);
 printf ("y =  "); scanf("%d",&y);
 asm  {
       mov ax,x
       add ax,y
       mov Sum,ax
 }
 printf ("Tong la: %d", Sum);  /* in tong*/
 getch(); /*chờ người dùng gõ vào 1 phím*/

}
```

- *Truy xuất gián tiếp qua thanh ghi chỉ số*:

Sử dụng một thanh ghi làm chỉ số của mảng. Ví dụ dưới đây ta tính tổng các phần tử của một mảng gồm 6 số nguyên đã được khai báo trước.

```
#include <stdio.h>
#include <conio.h>
void main()
{
 int Sum;
```

```
    int A[]=(3,2,1,5,6,7};
 asm  {
       mov bx,offset A /*bx chỉ số của phần tử đầu tiên */
       xor ax,ax /* ax chứa tổng */
       mov cx,6 /* số phần tử */
    Cong:
       add al,[bx]
       inc bx
       loop Cong
       mov Sum,ax
 }
 printf ("Tong la: %d", Sum);  /* in tong*/
 getch(); /*chờ người dùng gõ vào 1 phím*/

}
```

- *Truy xuất đến tham số truyền cho hàm:*

Trong cách truy xuất này, ta có thể dùng biến kiểu con trỏ (pointer) làm tham số truyền của hàm.

Ví dụ 1:

Chương trình ví dụ sau in ra 1 xâu kí tự được nhập từ bàn phím và xâu kí tự này được truyền vào một tham số của hàm InXau.

```
#include <stdio.h>
#include <conio.h>
void InXau(char *xau);
/*Chương trình con in ra một xâu kí tự*/
void InXau(char *xau)
{
 asm {
       mov ah,9
       mov dx, offset xau
       int 21h
 }
}
/*chương trình chính*/
void main()
{
 char *s1;
 /*Nhập vào 1 xâu từ bàn phím*/
 printf ("Nhap vao xau:  "); scanf("%s",&s1);
 /*In xau vừa nhập*/
 InXau(s1);
 getch(); /*chờ người dùng gõ vào 1 phím*/

}
```

Ví dụ 2: Viết hàm di chuyển con trỏ màn hình đến vị trí (x,y) trên màn hình (giống lệnh gotoxy(x,y) trong Pascal.

```
#include <stdio.h>
#include <conio.h>
void gotoxy(int x,int y);
/* Hàm di chuyển con trỏ màn hình đến vị trí x,y trên màn hình
*/
void gotoxy(int x,int y)
{
 asm{
        mov ax,x
        /*hoành độ lưu trong dl */
        mov dl,al
        mov ax,y
        /*tung độ lưu trong dl */
        mov dh,al
        /*đặt vị trí con trỏ*/
        mov ah,02
        mov bh,00
        int 10h  /*ngăt phục vụ màn hình*/
 }
}
/*chương trình chính*/
void main()
{
 int x=50, y=10;
 gotoxy(x,y);
 printf ("(%d,%d)",x,y);
 getch(); /*chờ người dùng gõ vào 1 phím*/

}
```

Ví dụ 3: Các lệnh nhảy có thể được thực hiện bên trong các hàm trong C. Dưới đây là một hàm nhận đầu vào là 1 kí tự ch, hàm sẽ kiểm tra kí tự ch có nằm trong khoảng từ [‘a’…’z’] hay không. Nếu ch thuộc khoảng (đóng) đó thì sẽ đổi kí tự ch từ thường sang hoa.

```
char upcase(char ch)
{
       asm mov al,ch; /*lưu kí tự trong al*/
       asm cmp al,'a';  /*là kí tự đứng trước 'a'*/
       asm jb khongxet;
       asm cmp al,'z'; /*là kí tự đứng sau 'z'*/
       asm ja khongxet;
       asm and al,5fh;
```

```
      khongxet:
}
```

- *Các kết quả trả về từ hàm*

Các kết quả trả về từ hàm được liệt kê trong bảng dưới đây:

Kiểu	Thanh ghi	Dữ liệu (byte)
char	AL	1
short int	AL	1
int	AX	2
unsigned int	AX	2
dword	DX:AX	4
pointer	DX:AX	4

- Lệnh điều khiển #pragma inline

Cú pháp: #pragma inline

Ví dụ: viết chương trình tìm giá trị nhỏ nhất trong 2 số bằng ngôn ngữ C có xen inline assembly

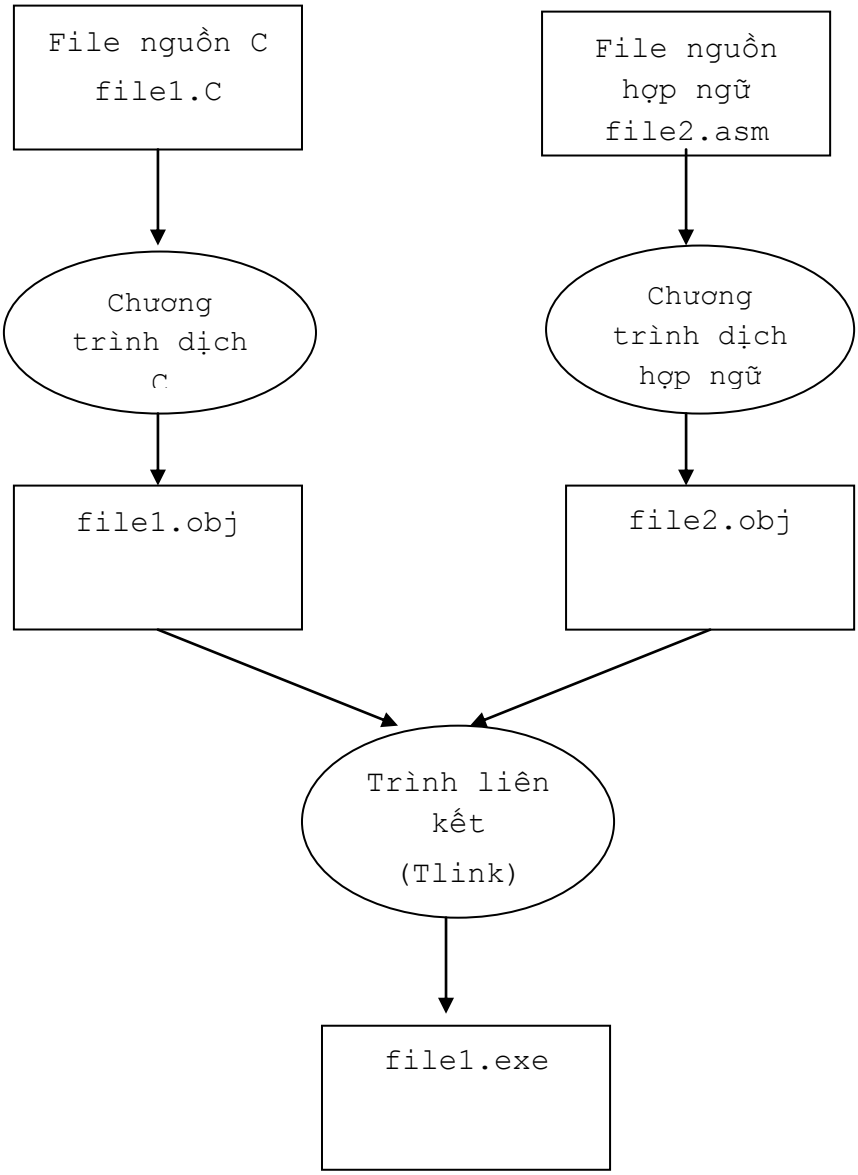
```
#pragma inline
#include <stdio.h>
#include <conio.h>
int min(int x, int y);
/*chương trình chính*/
void main()
{
 int m,n;

 /*Nhập vào 2 số từ bàn phím*/
 printf ("m= "); scanf("%d",&m);
 printf ("n= "); scanf("%d",&n);
 /*In min*/
 printf ("So be la:  %d", min(m,n));
 getch(); /*chờ người dùng gõ vào 1 phím*/

}
int min(int x, int y);
/*Chương trình con tìm min*/
int min(int x, int y)
{
```

```
  asm {
        mov ax,m
        cmp ax,n
        jb thoat
        mov ax,n
  thoat:
        return(_ax);
  }
}
```

b . Viết tách biệt các module hợp ngữ và C

Trong phương pháp trên thì cả lệnh C và hợp ngữ cùng được chứa trong 1 file. Phương pháp trên khá nhanh và hiệu quả đối với các chương trình nhỏ (đoạn mã chương trình bé hơn 64KB). Đối với các chương trình lớn thì các module được tổ chức trong các file khác nhau. Ta có thể viết các module C và hợp ngữ hoàn toàn tách biệt, sau đó tiến hành dịch riêng rẽ từng module sau đó liên kết chúng với nhau trước khi cho chạy. Cuối cùng ta thu được một file thực hiện được (exe) bằng cách trộn các file được viết bằng C và hợp ngữ.

Dưới đây là mô tả cho phương pháp thực hiện này:



Khi ta đã soạn xong chương trình nguồn file1.C và file2.asm thì ta có thể dịch và liên kết bằng lệnh:

tcc file1 file2.asm

Lệnh dịch trên sẽ được thực hiện như sau:

- trình biên dịch turbo C dịch file1.C thành file1.asm
- trình biên dịch tcc sẽ gọi trình biên dịch tasm để dịch file2.asm thành file2.obj
- trình biên dịch tcc sẽ gọi trình liên kết Tlink để liên kết hai file file1.obj và file2.obj thành file1.exe.

Việc viết tách biệt module ra với nhau rất có lợi cho các chương trình có nhiều lệnh hợp ngữ. Không những thuận lợi cho việc bảo trì mà phương pháp này còn tận dụng tối đa khả năng của trình biên dịch hợp ngữ và tránh được các nhược điểm của inline-assembly. Tuy nhiên, để thực hiện được sự liên kết theo cách này thì khi viết các module hợp ngữ người lập trình phải bắt

buộc tuân thủ tất cả các qui định của việc liên kết với module C. Đó là các vấn đề liên quan đến segment, chuyển đổi tham số, cách qui chiếu đến các biến của C, và bảo tồn các biến thanh ghi.

- *Các vấn đề cần phải giải quyết khi viết tách các module C và module hợp ngữ:*

1. Module hợp ngữ phải sử dụng sự sắp xếp các đoạn bộ nhớ (segment) tương thích với ngôn ngữ C.

Đây là vấn đề liên quan đến việc khai báo và sử dụng mô hình bộ nhớ và các đoạn bộ nhớ (segment). Dưới đây là một số lệnh điều khiển đơn giản có liên quan đến các qui định của C.

+ Lệnh điều khiển DOSSEG báo cho trình biên dịch TASM sắp xếp các đoạn bộ nhớ (segment) theo thứ tự như qui định của Intel. Ngôn ngữ C và hầu hết các ngôn ngữ bậc cao khác cũng phải sắp xếp các đoạn bộ nhớ theo qui cách này. Như vậy thứ tự sắp xếp các đoạn bộ nhớ trong module hợp ngữ cũng phải tuân thủ theo qui cách này.

+ Lệnh điều khiển MODEL báo cho trình biên dịch TASM biết kích thước mô hình bộ nhớ. Theo sau lệnh . MODEL là các kiểu mô hình bộ nhớ (*Tiny, Small, Compact, Medium, Large và Huge*) giống như việc chọn các tùy chọn trong môi trường C khi dịch. Lệnh điều khiển .MODEL còn được mặc định về dạng (NEAR hoặc FAR) của chương trình con được xây dựng bởi lệnh điều khiển PROC.

+ Các lệnh điều khiển .CODE, .DATA, .FARDATA, và .CONST của nhóm lệnh điều khiển segment đơn giản cũng tạo được những segment tương thích với C.

Ví dụ: Tính tổng của dãy số nguyên sodau + (sodau+1) +(sodau+2) + …+socuoi, với socuoi>sodau. Chương trình được tổ chức làm hai file. File hợp ngữ Tong.asm chứa đoạn chương trình tính tổng còn file ngôn ngữ C InTong.C sẽ chứa đoạn chương trình in kết quả của tổng này.

Module hợp ngữ được viết như sau:

```
.MODEL Small
.DATA
 EXTRN  _sodau: WORD
  XTRN  _socuoi: WORD
 PUBLIC Tong dw ?
.CODE
      PUBLIC _Sum
      _Sum PROC
        Mov CX, _socuoi
        Sub CX, _sodau
        Inc CX ; CX chứa số lượng các số
        Mov BX,_sodau ;  BX chứa số đầu tiên
        Xor AX,AX ; AX chứa tổng
 TinhTong:
        Add AX,BX
        Inc BX   ; BX= sốkế tiếp
        Loop TinhTong ; tiep tuc tinh tong neu CX<>0
        Mov Tong,AX
        Ret
 _Sum ENDP
```

```
  END
```

Hàm _Sum sẽ được chương trình của C gọi từ mô hình dịch Small của turbo C với câu lệnh: Sum().

Dưới đây là module C của file InTong.C

```
extrn int sodau;
extrn int socuoi;
extrn int Sum();
void main()
{
 printf ("So dau:  "); scanf("%d",&sodau);
 printf ("So cuoi "); scanf("%d",&socuoi);
       printf ("Tong la: %d ", Sum());
}
```

Để tạo được file chạy được (đuôi exe) ta thực hiện lệnh sau (giả sử tất cả các file liên quan đều cùng nằm trong 1 thư mục với tcc và turbo C được cài đặt trên ổ C trong thư mục tc).

```
tcc -ms -Ic:\tc\include -Lc:\tc\lib InTong Tong.asm
```

Chú ý:

Nếu muốn liên kết –Sum với mô hình bộ nhớ dạng khác chẳng hạn compact thì ta phải chọn tùy chọn compact trong khi dịch bằng tcc và trong chương trình Tong.asm ta phải sửa lệnh .Model Small thành .Model Compact

Khi muốn sử dụng đoạn bộ nhớ kiểu FAR trong Tong.asm thì ta phải sử dụng lệnh điều khiển .FARDATA.

1. Các khai báo PUBLIC, EXTERNAL và sự tương thích kiểu dữ liệu

Ta đã tìm hiểu về chương trình được tổ chức thành nhiều module và được lưu trữ trên nhiều file khác nhau.. Chương trình được liên kết bằng các module của C và hợp ngữ cũng là chương trình nhiều file, do đó phải thỏa mãn các yêu cầu về khai báo nhãn (tên biến, tên hàm…) giữa các module với nhau, cụ thể là:

Trong các module viết bằng hợp ngữ phải:

Khai báo PUBLIC trước những nhãn (tên biến, tên hàm…) mà các file khác sẽ sử dụng đến bằng cú pháp:

```
PUBLIC _tên nhãn 1, _ tên nhãn 2,…
khai báo nhãn (xác định kích cỡ)
```

Ví dụ:

Với tên nhãn là biến nhớ:

```
PUBLIC _giatri1, _giatri2
giatri1 DB 10
giatri1 DW 1000
```

Với tên nhãn là tên hàm:

```
PUBLIC _Sum
_Sum PROC
   < Các lệnh trong thân hàm>
_Sum ENDP
```

Khai báo EXTRN trước những biến ngoài được file này sẽ sử dụng đến. Cú pháp như sau:

```
EXTRN _tên nhãn 1: kiểu nhãn 1,
      _tên nhãn 2: kiểu nhãn 2,…
```

Ví dụ:

Với tên nhãn là biến nhớ:

```
EXTRN _x1: BYTE, _x2: WORD
```

Với tên nhãn là hàm:

```
EXTRN _Ham: PROC
```

+ Sự tương thích giữa các kiểu về khai báo dữ liệu được cho ở trong bảng sau:

Kiểu khai báo dữ liệu trong C	Kiểu khai báo dữ liệu trong hợp ngữ
Unsigned char	Byte
Char	Byte
Enum	Word
Unsigned short	Word
Short	Word
Unsigned int	Word
Int	Word
Unsigned long	Dword
Long	Dword
Float	Dword
Double	Qword
Long double	Tword
Near	Word
Far	Dword

Ví dụ:chương trình tính giai thừa

Chương trình được tổ chức thành hai module: module C và module hợp ngữ. Mỗi module có một nhiệm vụ như sau:

Module C: đọc số cần tính giai thừa, gọi chương trình con thực hiện việc tính giai thừa và in kết quả ra màn hình. Module này được lưu trong file file1.C

```
#include <stdio.h>
#include <conio.h>
extern GiaiThua();
```

```
int number, ketqua;
/*chương trình chính*/
void main()
{
 int m,n;
 /*Nhập vào 1 số từ bàn phím*/
 printf ("Nhap vao 1 so: "); scanf("%d",&number);
 GiaiThua();
 /*In min*/
 printf ("Ket qua la:  %d", ketqua);
 getch(); /*chờ người dùng gõ vào 1 phím*/

}
```

Module hợp ngữ: tính giai thừa. Module này được lưu trong file file2.asm

```
.MODEL Small
.DATA
 EXTRN  _number: WORD,_ketqua: WORD
 temp dw ?
.CODE
      PUBLIC _GiaiThua
      _GiaiThua PROC
       Mov _ketqua, 1 ; ket qua tinh giai thua
       Mov temp, 2 ;bat dau nhan tu 1*2
       Mov CX,_number ; so cac thua so
       Dec CX
 Tinh:
       Mov AX,_ketqua ; AX chua ket qua
       Mul temp  ; nhan ket qua voi số kế tiếp
       Mov _ketqua, AX
       Inc temp
       Loop Tinh ; tiep tuc tinh giai thua neu CX<>0
       Ret
 _GiaiThua ENDP
  END
```

Sau khi soạn xong, có thể tiến hành dịch và liên kết chương trình bằng lệnh:

```
tcc –ms –Ic:\tc\include –Lc:\tc\lib file1 file2.asm
```

Sau khi sửa lỗi, chương trình sẽ được dịc thành file file1.exe.

c . Một số điểm cần lưu ý

Khi viết chương trình C và hợp ngữ liên kết với nhau ta cần chú ý hai điểm:

- Bảo vệ các thanh ghi:

Một chương trình con (hàm hoặc thủ tục) viết bằng hợp ngữ được liên kết với chương trình C phải bảo tồn các thanh ghi đoạn, đó là các thanh ghi: BP,SP,CS,DS và SS. Giá trị của các thanh ghi này phải được lưu vào ngăn xếp bằng các lệnh PUSH trước các lệnh khác trong các

chương trình con hoặc macro. Ở phần cuối các chương trình con (trước lệnh ret) thì các lệnh này phải được khôi phục lại bằng các lệnh POP.

- Giá trị trả lại của các hàm:

Giống ngôn ngữ C và các ngôn ngữ khác, các hàm được xây dựng bằng bằng hợp ngữ khi liên kết với C cũng có thể trả về một giá trị (tên hàm mang một giá trị). Xong các giá trị trả về của các hàm được viết bằng hợp ngữ tuân thủ các qui định sau:

Kiểu giá trị trả về	Nơi chứa giá trị trả về
Unsigned char	AX
Char	AX
Enum	AX
Unsigned short	AX
Short	AX
Unsigned int	AX
Int	AX
Unsigned long	DX:AX
Long	DX:AX
Float	Đỉnh ngăn xếp 8087, thanh ghi ST(0)
Double	Đỉnh ngăn xếp 8087, thanh ghi ST(0)
Long double	Đỉnh ngăn xếp 8087, thanh ghi ST(0)
Near	AX
Far	DX:AX

d. Một số ví dụ về truyền tham số giữa các hàm của C và hợp ngữ.

Ví dụ 1: Viết chương trình tính giai thừa với yêu cầu: kết quả của hàm tính giai thừa là một đối số ra của hàm (chứ không phải là giá trị trả lại của hàm [giống ví dụ trong phần b, 2 của mục 3.3.1]).

Module C: đọc số cần tính giai thừa, gọi chương trình con thực hiện việc tính giai thừa. Lấy và in kết quả (từ đối số của hàm) ra màn hình. Module này được lưu trong file file1.C

```
#include <stdio.h>
#include <conio.h>
extern GiaiThua(int number, int near *ketqua);
/*chương trình chính*/
void main()
{
 int n, kq;
 /*Nhập vào n từ bàn phím*/
```

```
 printf ("Nhap vao 1 so: "); scanf("%d",&n);
 GiaiThua(n,&kq);
 /*In min*/
 printf ("Ket qua la:  %d", ketqua);
 getch(); /*chờ người dùng gõ vào 1 phím*/
}
```

Module hợp ngữ: tính giai thừa và lưu kết quả vào đối của hàm. Module này được lưu trong file file2.asm

```
.MODEL Small
.DATA
 Gtri DW ?
 temp dw ?
.CODE
      PUBLIC _GiaiThua
      _GiaiThua PROC
       ARG  _number: WORD,_ketqua: WORD
       Push BP    /* bao ve gia tri thanh ghi BP */
       Mov BP,SP  /*BP,SP tro den dau stack*/
       Mov Gtri, 1 ; ket qua tinh giai thua
       Mov temp, 2 ;bat dau nhan tu 1*2
       Mov CX,_number ; so cac thua so
       Dec CX
 Tinh:
       Mov AX, Gtri ; AX chua ket qua
       Mul temp  ; nhan ket qua voi số kế tiếp
       Mov Gtri, AX
       Inc temp
       Loop Tinh ; tiep tuc tinh giai thua neu CX<>0
       Mov BX, _ketqua
       Mov [BX],AX
       Pop Bp
       Ret
 _GiaiThua ENDP
       END
```

Sau khi soạn xong, có thể tiến hành dịch và liên kết chương trình bằng lệnh:

```
tcc –ms –Ic:\tc\include –Lc:\tc\lib file1 file2.asm
```

Sau khi sửa lỗi, chương trình sẽ được dịc thành file file1.exe.

2.4.2 Ngôn ngữ Pascal và Hợp ngữ

Về nguyên lý, thì liên kết giữa hợp ngữ với Pascal giông như việc liên kết giữa hợp ngữ với C. tuy nhiên cũng có một số qui tắc riêng khi thực hiên liên kết giữa hợp ngữ và Pascal.

Cũng như hợp ngữ và C, có hai cách để liên kết giữa hợp ngữ và Pascal là dùng inline assembly và viết tách biệt giữa các module hợp ngữ và module Pascal.

a . Sử dụng inline assembly trong Pascal

Phương pháp này thích hợp cho người lập trình phát triển các chương trình nhỏ. Trong phương pháp này người lập trình sẽ chèn một khối lệnh hợp ngữ vào một chương trình được viết bằng ngôn ngữ Pascal. Đây là phương pháp khá đơn giản và nhanh.

```
 asm
[<Nhãn 1>:] <lệnh 1> <các toán hạng 1>
[<Nhãn 2>:] <lệnh 2> <các toán hạng 2>
….
[<Nhãn n>:] <lệnh n> <các toán hạng n>
end;
```

Khi các chương trình dịch của Pascal gặp từ khóa asm trong dòng lệnh inline-assembly thì chương trình dịch sẽ chuyển khối lệnh hợp ngữ này vào và dịch với việc qui chiếu biến Pascal ra dạng tương ứng của hợp ngữ để thực hiện.

Dưới đây là một ví dụ đơn giản để minh họa:

Ví dụ 1: viết chương trình tìm giá trị nhỏ nhất (min) cho hai số bằng ngôn ngữ Pascal có chèn các dòng lệnh dạng inline-assembly. Giả sử file chương trình là Min.pas

```
Program Min;
Uses crt;
Var
 m,n: integer;
 function min(int x, int y): integer;
  /*Chương trình con tìm min*/
 begin
  asm
              mov ax,x;
              cmp ax,y;
              jb thoat;
              mov ax,y;
             mov x,ax;
       thoat:
        min=x;
  end;
 end;
Begin
  Clrscr;
 /*Nhập vào 2 số từ bàn phím*/
 write (“m= ”); readln(m);
 write (“n= ”); readln(n);
 /*In min*/
 write (“So be la:  ”, min(m,n));
 readln(); /*chờ người dùng gõ vào 1 phím*/
End.
```

Để thực hiện chương trình ta gõ lệnh:

```
tpc  Min.pas <enter>
```

Chương trình sẽ được dịch ra file exe.

b .Viết tách biệt nhiều module hợp ngữ và Pascal riêng rẽ

Giống như viết tách biệt module và hợp ngữ, khi viết tách biệt giữa Pascal và hợp ngữ cũng phải xử lý một số vấn đề tương tự như: các lệnh điều khiển dịch, các vấn đề liên kết thông tin qua các biến, bảo vệ và khôi phục giá trị các thanh ghi đoạn và sự tương thích về kiểu dữ liệu.

Trong các vấn đề trên, hầu hết các vấn đề đều được giải quyết tương tự như việc đối với C và hợp ngữ. Sự tương thích về kiểu dữ liệu có đôi chút khác biệt, dưới đây là bảng tương thích kiểu dữ liệu giữa hợp ngữ và Pascal.

Kiểu khai báo dữ liệu trong Pascal	Kiểu khai báo dữ liệu trong hợp ngữ
Byte	Byte
Word	Word
Shortint	Byte
Integer	Word
Real	Fword
Single	DWord
Double	QWord
Extended	TByte
Comp	Qword
Pointer	Dword

Ví dụ: Tính tổng các phần tử trong một dãy số nguyên dương khi biết số đầu tiên và số các phần tử cần tính.

Chương trình được thành hai module. Module hợp ngữ có nhiệm vụ tính tổng của các phần tử. Module này được lưu vào file sum.asm

```
.MODEL Small
.DATA
 EXTRN  _sodau: WORD,_sophantu: WORD
.DATA ?
 Tong dw ?
.CODE
      PUBLIC Sum
      Sum PROC
        Mov CX, [sophantu]
        Mov AX, [sodau]
        Mov [tong],AX
```

```
 TinhTong:
       Inc AX
       Add [tong],AX
       Loop TinhTong ; tiep tuc tinh tong neu CX<>0
       Mov AX, Tong
       Ret
  Sum ENDP
        END
```

Module Pascal được lưu trong file InTong.pas

```
Program Tong;
Uses crt;
{F+}
Var
 Sodau: integer;
 Sophantu: integer;
 function Sum: integer; external;
 {$I sum.obj}
{F-}
Begin
 Clrscr;
 write ("So dau:  "); readln(sodau);
 write ("So phan tu:  "); readln(sophantu);
 write ("Tong la:  ", Sum);
End.
```

Để tạo ra file chạy (.exe) ta tiến hành qua các bước sau:

- Dịch module hợp ngữ (để tạo file sum.obj)

```
Tasm sum <Enter>
```

- Dịch module Pascal có lên kết với file sum.obj

```
Tpc –ml Intong <enter>
```

2.5 CÁC CHƯƠNG TRÌNH NGẮT

Ngắt là một cơ chế yêu cầu CPU tạm dừng công việc (task) đang thực hiện để thực hiện 1 công việc khác. Nói cụ thế hơn, ngắt yêu cầu CPU tạm dừng chương trình đang thực hiện để thực hiện một chương trình con phục vụ ngắt.

Người ta tạm chia ngắt ra làm hai loại: ngắt cứng và ngắt mềm. Các ngắt mềm được kích hoạt bằng lệnh INT n trong đó n là số hiệu ngắt dưới dạng một số hexa. Ngắt cứng khác vơi ngắt mềm ở chỗ không được kích hoạt bằng một lệnh INT n trong chương trình mà được kích hoạt bằng các tác động của các tín hiệu linh kiện điện tử như bàn phím, ổ đĩa,..

Phần này giới thiệu về các ngắt và các dịch vụ ở mức BIOS và mức hệ điều hành DOS và cách viết chương trình thường trú và chương trình con ngắt.

2.5.1 Ứng dụng các ngắt của BIOS & DOS

Máy tính có 256 ngắt được đánh số hiệu từ 00h đến FFh. Trong đó các ngắt có số hiệu từ 00h đến 1Fh là các ngắt của BIOS, còn các ngắt còn lại từ 20h đến FFh là các ngắt của DOS.

Dưới đây ta sẽ tìm hiểu các ngắt theo từng nhóm ngắt.

a. Các ngắt của BIOS & DOS

Địa chỉ	Số hiệu ngắt	Chức năng
Các ngắt phục vụ hệ thống		
0-3	0	CPU: chia cho 0
4-7	1	CPU: chạy từng bước của DEBUG
8-B	2	CPU: ngắt NMI (hiện thông báo halt)
C-F	3	CPU: thực hiện đến điểm dừng (break point)
10-13	4	CPU: tràn số (overflow)
14-17	5	In nội dung ra mà hình
18-1B	6	Phục vụ liên lạc
1C-1F	7	Dự trữ
Các ngắt cứng		
20-23	8	IRQ0: CLK (18.2 lần/s) nối từ chip 8253
24-27	9	IRQ1: bàn phím
28-2B	A	IRQ2: đầu vào của 8259 thứ 2
2C-2F	B	IRQ3: giao diện nối tiếp
30-33	C	IRQ4:giao diện nối tiếp
34-37	D	IRQ5: thường nối với máy in nối tiếp
38-3B	E	IRQ6: phục vụ đĩa mềm
3C-3F	F	IRQ5: thường nối với máy in song song
Các ngắt thực sự đặc trưng cho BIOS		
40-43	10	Màn hình (I/O video)
44-47	11	Xác định cấu hình
48-4B	12	Cho biết kich cỡ của RAM
4C-4F	13	Thâm nhập đĩa mềm, đĩa cứng.
50-53	14	Giao diện nối tiếp

54-57	15	Giao diện với cassete
58-5B	16	Kiểm tra bàn phím
5C-5F	17	Truy nhập máy in song song
60-63	18	Gọi BASIC trong ROM
64-67	19	Khởi động nóng hệ thống (Ctrl+Alt+Del)
68-6B	1A	Thông báo thời gian
6C-6F	1B	Quản lý phím Ctrl+Break
70-73	1C	Dành cho đồng hồ
74-77	1D	Địa chỉ bảng tham số cho màn hình
78-7B	1E	Cho biết các tham số của đĩa mềm
7C-7F	1F	Địa chỉ các bảng font các kí tự mở rộng
Các ngắt của DOS		
80-83	20	Kết thúc chương trình dạng COM
84-87	21	Các hàm của DOS
88-8B	22	Địa chỉ kết thúc chương trình
8C-8F	23	Địa chỉ thủ tục Ctrl+Break
90-93	24	Báo lỗi đĩa
94-97	25	Đọc đĩa mềm, đĩa cứng
98-9B	26	Ghi đĩa
9C-9F	27	Kết thúc chương trình và thường trú
A0-A3	28	Dành cho các hàm không được DOS cung cấp dữ liệu
	29-3F	Dự trữ
	40	BIOS phục vụ đĩa mềm
	41	Địa chỉ của bảng đĩa cứng 1
	42-49	Dự trữ
	4A	Hẹn giờ
	4B-6F	Dự trữ
	70-77	Ngắt cứng của 8259 thứ 2
	78-7F	Dự trữ
	80-F0	Dùng cho bộ thông dịch BASIC

	F1-FF	Dự trữ

b. Cơ chế hoạt động khi một ngắt được kích hoạt

Khi có một yêu cầu ngắt số hiệu N đến chân CPU và nếu yêu cầu ngắt này được CPU đáp ứng Khi đó CPU sẽ thực hiện các công việc sau:

1. Cất nội dung của thanh ghi cờ (FR) vào đỉnh của ngăn xếp. (Bằng việc tự động thực hiện câu lệnh PUSHF).

2.Cấm các ngắt khác tác động vào CPU để CPU chạy ở chế độ bình thường. Đặt các cờ IF=0 và TF =0 bằng cách thực hiện các lệnh: CLI và CLT.

3. Cất địa chỉ đoạn (segment) của chương trình gọi chương trình ngắt vào ngăn xếp bằng lệnh PUSH CS.

4. Cất địa chỉ lệch (offset) của lệnh kế tiếp của chương trình gọi chương trình ngắt vào ngăn xếp PUSH IP.

5. Lấy địa chỉ mới của chương trình con phục vụ ngắt số hiệu N trong bảng vector ngắt bằng cách lấy địa chỉ offset và segment của ngắt N từ bảng vector ngắt.

IP=[N*4]

CS=[N*4+2]

6. Khi gặt lệnh cuối cùng của chương trình con phục ngắt (lênh IRET). Bộ vi xử lý sẽ quay lại chương trình gọi ngắt tại địa chỉ trả về và khôi phục các giá trị của các thanh ghi từ ngăn xếp bằng các lệnh sau:

POP IP

POP CS

POPF

Giải thích cho mục 5. Ta biết rằng các địa chỉ của chương trình con phục vụ ngắt được lưu vào trong một bảng có kích thước 1K từ địa chỉ 0000h đến 03FFh của bộ nhớ RAM. Bảng vector ngắt lưu địa chỉ của 256 chương trình con phục vụ ngắt và mối địa chỉ chiếm 4 byte trong đó 2 byte dành cho địa chỉ đoạn (segment) và 2 byte dành cho địa chỉ lệch (offset). Như bảng ở trên, địa chỉ của chương trình con phục vụ ngắt 0 chiếm byte 0-3, ngắt 1 chiếm byte 4-7 … và chương trình con phục vụ ngắt thứ N sẽ có địa chỉ 4*N. Trong đó 2 byte [4*N] và [4*N+1] là địa chỉ lệch (offset) và 2 byte [4*N+2] và [4*N+3] là địa chỉ đoạn (segment).

2.5.2 Chương trình thường trú và chương trình ngắt

a. Chương trình thường trú

- *Khái niệm về chương trình thường trú*

Chương trình thường trú (Terminate and Stay Resident- TSR) là chương trình có thể chạy "sau" chương trình khác, hỗ trợ khả năng kích hoạt, khả năng nằm lại bộ nhớ sau khi chạy xong. Sau đó khi ta chạy một chương trình khác với một điều kiện nào đó nó sẽ được kích hoạt để hoạt động trở lại.

Với chương trình bình thường khi chạy sẽ được một chương trình tải (Program Loader trong command.com) nạp vào vùng nhớ do DOS cấp phát. Khi chương trình thực hiện xong thì

vùng nhớ đã cấp phát cho nó được giải phóng và DOS sẽ đánh dấu lại vùng nhớ này để cấp phát cho chương trình khác. Với chương trình thường trú thì bước cuối cùng không xảy ra, chương trình thường trú làm cho DOS đánh dấu lại miền dành cho DOS và vùng bị nó chiếm, do vậy sau này DOS sẽ không cấp phát vùng nhớ này cho chương trình khác, và như vậy nó được bảo vệ chống bị viết đè bởi chương trình khác, bằng cách này thì chương trình thường trú trở thành một "bộ phận" của DOS.

Chỉ có file dạng COM với cấu trúc nằm gọn trong một đoạn mới dễ dàng trở thành chương trình thường trú. Ngoài hợp ngữ, người ta có thể viết chương trình thường trú trên các ngôn ngữ lập trình bậc cao khác như ngôn ngữ C, Pascal …

- *Viết chương trình thường trú*

Chương trình thường trú được viết giống như chương trình thông thường và viết thêm một đoạn mã của chương trình thường trú vào vùng nhớ dành cho DOS, đoạn mã đó sẽ không được kích hoạt nếu không được trao điều khiển. Việc thêm một đoạn mã tiếp dau vùng dành cho DOS được thực hiện bằng các chương tình con phục vụ ngắt. Đó là ngắt số 27H hoặc hàm 31H của ngắt 21H.

Có hai cách để làm cho chương trình thường trú được kích hoạt là: dùng ngắt và ấn một tổ hợp phím (hot-key).

Các chương trình thường trú thường sửa nội dung của vector ngắt trong bẳng vector ngắt để làm cho nó trỏ đến địa chỉ của mình trong bộ nhớ, nhờ thế mỗi khi ngắt tương ứng được gọi thì chương trình thường trú lại được trao điều khiển. việc làm này được gọi là chặn vector ngắt. Chẳng hạn, nhiều chương trình POP-UP thường sửa ngắt bàn phím (vector ngắt bàn phím (số 9) nằm tại địa chỉ: 0:0024h) làm cho nó trỏ đến địa chỉ của mình và cất địa chỉ của INT 9h để cho nó làm nhiệm vụ khi cần thiết.

- *Các bước viết chương trình thường trú*

Bước 1: Lấy và đặt lại vector ngắt bằng các dịch vụ của DOS

Ta luôn cần đến các chương trình con phục vụ ngắt đã có của hệ thống để không phải viết lại trong chương trình của mình đoạn chương trình đã có, vì vậy cần lấy nội dung của vector ngắt cũ cất vào miền dữ liệu của chương trình thường trú, khi nào cần sẽ trả lại giá trị này cho vector ngắt mà chương trình thường trú đã thay đổi. Giá trị của một vector ngắt là 2 từ tương ứng với địa chỉ đoạn (CS) và địa chỉ lệch (IP) của chương trình con phục vụ ngắt tương ứng.

+ Lấy vector ngắt (Get Interrupt Vector)

Hàm 35h:

Ý nghĩa: Lấy địa chỉ của một ngắt từ bảng vector ngắt.

Đầu vào: AH=35h

AL=số hiệu vector ngắt

Int 21h

Đầu ra: ES:BX = giá trị của vector ngắt.

Ví dụ: Đoạn chương trình sau lấy vector ngắt của bàn phím (INT 9h)

```
MODEL Tiny
.CODE
            Org 100h
```

```
            Jmp Load_Prog
             ; vùng dữ liệu
             SohieuNgat EQU 9h
             NgatCu DW 2 DUP(0); luu dia chi ngat cu
      Load_Prog PROC
        Mov AH,35
        Mov AL, SohieuNgat
        Int 21h
        Mov NgatCu,BX ; lay dia chi  lech
        Mov NgatCu[2],ES ; lay dia chi doan
        ….
 Load_Prog ENDP
```

+ Đặt giá trị cho một vector ngắt (Set Interrupt Vector)

Ý nghĩa: Đặt lại địa chỉ của vector ngắt .

Đầu vào: AH=25h

AL=địa chỉ vector ngắt

Int 21h

DS:DX = Địa chỉ của chương trình ngắt.

Ví dụ: Đoạn chương trình sau đặt lại địa chỉ của vector ngắt.

```
MODEL Tiny
.CODE
             Org 100h
            Jmp Load_Prog
             ; vùng dữ liệu
             SohieuNgat EQU 9h
             NgatCu DW 2 DUP(0); luu dia chi ngat cu
      Load_Prog PROC
        Mov AH,35h
        Mov AL, SohieuNgat
        Int 21h
        Mov NgatCu,BX ; lay dia chi  lech
        Mov NgatCu[2],ES ; lay dia chi doan
        Mov AH,25
        Mov DX,offset Prog ; Dat vector ngat moi, tro vao PROG
        Int 21h
        ….
 Load_Prog ENDP
```

Sau đó, mỗi lần ngắt bàn phím được kích hoạt thì chương trình PROG của ta sẽ được thực hiện. Tất nhiên sau khi thực hiện chương trình PROG này thì ta phải đặt đặt lại điạc chỉ của vector ngắt bàn phím.

Bước 2: Làm cho chương trình ở lại thường trú

Có thể làm cho chương trình ở lại thường trú bằng cách sử dụng INT 27H hoặc dịch vụ 31H của INT 21H. Trong phần này ta chỉ xem xét việc sử dụng INT 27H.

Nói chung tất cả các chương trình thường trú đều tự nạp ns vào bộ nhớ sau đó tự loại bỏ phần "đuôi" của mình- đó là đoạn mã thực hiện nạp chương trình vào bộ nhớ hay còn được gọi là phần tạm trú. Dưới đây là khung của chương trình thường trú bằng cách chặn ngắt.

```
MODEL Tiny
.CODE
        Org 100h
        Jmp Load_Prog
              ; vùng dữ liệu
        PROG PROC
               ; các lệnh của chương trình được viết ở đây
        PROG ENDP

        Load_PROG PROC
               ; các lệnh của phần thường trú của chương trình được viết ở
        đây
              Mov, DX, offset Load_Prog
              Int 21h
        Load_PROG ENDP
```

b. Chương trình con phục vụ ngắt

Về cơ bản chương trình con phục vụ ngắt (để cho ngắn gọn ta gọi là *chương trình ngắt*) giống như một chương trình dạng COM mà ta đã tìm hiểu từ các phần trước. Tuy nhiên, có một số điểm lưu ý khi ta muốn viết một chương trình con phục vụ ngắt đó là:

- Bảo vệ các thông tin trạng thái và khôi phục lại khi kết thúc chương trình ngắt
- Lệnh cuối cùng của chương trình ngắt là lệnh IRET

Do vậy, dưới đây là "khung" của một chương trình ngắt, nó được sử dụng khi viết một chương trình ngắt.

```
.MODEL Tiny
.CODE
      Org 100h
      ; vùng dữ liệu
 SohieuNgat EQU 9h
 NgatCu DW 2 DUP(0); luu dia chi ngat cu
      Jmp Load_Prog
Start:
TenCTN PROC
 Push AX
 Push BX
      Push CX
      Push DX
      Push DI
      Push SI
```

```
        Push DS
        Push ES
        ; Thân chương trình thường trú
        Pop ES
        Pop DS
        Pop SI
        Pop DI
        Pop DX
        Pop CX
        Pop BX
        Pop AX
  IRET
TenCTN ENDP
;------------------------------
Load_Prog PROC
      Mov AH,35
   Mov AL, SohieuNgat
   Int 21h
   Mov NgatCu,BX ; lay dia chi  lech
   Mov NgatCu[2],ES ; lay dia chi doan
   Mov AH,25
       Mov DX,offset TenCTN ; vao PROG
       Int 21h
 Exit:
       Mov DX,offset Load_Prog ; giữ lại cho thường trú
   Int 27h
Load_Prog ENDP
End Start
```

2.6 MỘT SỐ VÍ DỤ MINH HỌA

Ví dụ 0: Viết chương trình in ra nhập vào một kí tự nhưng in ra màn hình kí tự kế tiếp. Chẳng hạn, khi nhập vào kí tự ‘a’ thì mà hình lại hiện ra kí tự ‘b’.

Bài giải

Ta sử dụng hàm 08 của ngắt 21h để nhập 1 kí tự không hiện lên màn hình rồi sau đó dung hàm 02 để in kí tự kế tiếp (tăng mã ASCII lên 1) ra màn hình.

```
.MODEL Tiny
.CODE
      Org 100h
      Jmp Start
Start:
      Mov AH,08h ; nhập 1 kí tự không hiện lên màn hình
      Int 21h
```

```
        Mov DL,AL ; chuyển mã ASCII của kí tự vào DL
        Inc DL  ; DL chứa kí tự kế tiếp
        Mov AH,02h ; In ra màn hình
        Int 21h ;
        Int 20h ; trở về DOS
End Start
```

Ví dụ 1: Viết chương trình in ra 256 kí tự của bảng mã ASCII

Bài giải

Ta sử dụng một vòng lặp FOR-DO và dùng DL đê chứa mã ASCII của các kí tự trong bảng mã ASCII. CX chứa số kí tự cần in (256). Mỗi kí tự cách nhau bởi 1 dấu cách. Chương trình được viết theo khung của chương trình COM

```
.MODEL Tiny
.CODE
        Org 100h
        Jmp Start
Start:
        Mov CX,256 ; số kí tự cần in
        Mov DL,0 ; kí tự đầu tiên
        Mov AH,2 ; hàm 2 ngắt 21h in ra 1 kí tự lên màn hình
Tiep:
        Int 21h
        Mov BL,DL ; dùng BL để chứa tạm mã ASCII của kí tự
        Mov DL,32
        Int 21 ; In dấu cách
        Mov DL,BL ; lấy lại kí tự in cuối cùng
        Inc DL ; sang kí tự tiếp theo
        Loop Tiep ; In kí tự kế tiếp
        Int 20h ; trở về DOS
End Start
```

Ví dụ 2: Viết chương trình nhập vào một dãy các kí tự rồi hiển thị nó theo thứ tự ngược lại.

Bài giải

Ta có thể sử dụng ngăn xếp để giải quyết bài toán này. Mỗi khi có ký tự được nhập vào sẽ được PUSH vào ngăn xếp, sau khi nhập xong (bằng cách gõ Enter) thì các kí tự trong ngăn xếp sẽ được POP ra và hiển thị theo thứ tự ngược lại so với ban đầu.

```
.MODEL small
.STACK 100h
.DATA
        NhapXau db 'Nhap vao day ki tu:  ','$'
        InXau db 'Day ki tu in ra theo thu tu nguoc lai la: ','$'
        xuongdong db 13,10,'$'
.CODE
Start:
```

```
        Mov AX,@Data
        Mov DS,AX
        Mov  AH,9
        Mov DX, offset NhapXau
   Int 21h ; in lời mời nhập xâu
   Xor CX,CX ; CX=0
   Mov AH,1  ; Nhap ki tu
DocVao:
      Int 21h
        Cmp AL,13 ; co phai Enter khong?
        JE ThoiDoc ; Neu là Enter, dung lai
        Push AX ; Cho vao ngan xep
        Inc CX ; Tang de dem so ki tu da nhap
        Jmp DocVao
ThoiDoc:
        Mov AH,9
        Mov DX, offset xuongdong
        Int 21h ;xuong dong va ve dau dong
        Mov DX, offset InXau
        Int 21h ; in lời mời in xâu InXau
        Mov AH,2
HienThi:
        Pop DX   ; In tung ki tu trong ngan xep
        Int 21h
        Loop HienThi ; In ra cho den khi CX=0
        Mov AH,4Ch ; Tro ve DOS
        Int 21h
End Start
```

Ví dụ 3: Nhập vào hai số nguyên x và y (0<=x,y<=9), tính hiệu x-y và in kết quả ra màn hình.

Bài giải:

Bài toán được chia thành 3 phần:

- Nhập x và y
- Tính hiệu x-y
- In kết quả

Một số lưu ý: khi nhập vào bằng hàm 01 của ngắt 21h thì AL sẽ chứa mã ASCII của kí tự vừa nhập. Chẳng hạn, khi ta nhập vào số 3 thì AL=33h (mã ASCII của 3), do vậy để nhận được số thực sự ta phải đem trừ đi 30h. Ngược lại, khi in ra thì đang ở dạng số phải đổi sang mã ASCII bằng cách cộng thêm 30h.

Để thực hiện được phép trừ hai số. Ta tiến hành phép so sánh x và y, nếu x>y ta lấy x trừ đi y, ngược lại ta lấy y trừ đi x và in dấu trừ trước kết quả.

```
.MODEL small
```

```
.STACK 100h
.DATA
      stringX db 'x= ','$'
      stringY db 'y= ','$'
      xuongdong db 13,10,'$'
      Hieu db 'x-y = ','$'
      X  db ?
      Y db ?
.CODE
Start:
      Mov AX,@Data
      Mov DS,AX
      Mov  AH,9
      Mov DX, offset stringX
      Int 21h   ; in xâu 'x = '
      Call Nhap  ; gọi chương trình con Nhập
      Mov x,AL
      Mov  AH,9
      Mov DX, offset xuongdong
      Int 21h   ; in xâu xuống dòng và về đầu dòng
      Mov DX, offset xuongdong
      Int 21h   ; in xâu 'y = '
      Call Nhap  ; gọi chương trình con Nhập
      Mov y, AL
      Mov  AH,9
      Mov DX, offset xuongdong
      Int 21h   ; in xâu xuống dòng và về đầu dòng
      Mov DX, offset xuongdong
      Int 21h   ; in xâu 'x-y = '
      Mov DL,x  ; DL=x
      Cmp DL,y ; so sánh x với y
      Sub DL,y
      Call Inra ; gọi chương trình con Inra
      Jmp Ketthuc ; nhảy
      Jb Behon ; nhảy nếu x<y
      Mov AL,y
      Sub AL,x
      Mov AH,2
      Mov DL,'-' ; In dấu trừ trước kết quả
      Int 21h
      Mov DL,AL
      Call Inra ; gọi chương trình con Inra
Ketthuc:
      Mov AH,4Ch
```

```
        Int 21h
End Start
;----------------------------------------
; chương trình con nhập, trả lại số nhập được trong AL
;----------------------------------------
Nhap Proc
        Mov AH,1 ; hàm 01 nhập vào 1 kí tự
Nhaplai:
        Int 21h
        Cmp AL,30h ; nhỏ hơn kí tự '0'
        Jb NhapLai ; nhập lại
        Cmp AL,39h ; lơn hơn kí tự '9'
        Ja NhapLai ; nhập lại
        Sub AL,30h ; đổi mã ASCII sang số
Nhap Endp
;----------------------------------------
;chương trình con In ra 1 số trong khoảng 0..9 trong DL
;---------------------------------------

Inra Proc
        Mov AH,2 ; in ki tự
        Add DL,30h ; đổi sang mã ASCII
        Int 21h
Inra Endp
;---------------------------------------
```

Ví dụ 4: Nhập vào một xâu kí tự rồi in xâu đó ra màn hình

Bài giải:

Đây là một bài tập không khó, tuy nhiên có một số vấn đề mà người học lập trình cần biết trước khi viết chương trình giải bài toán này. Đó là cấu trúc của vùng đệm (buffer) khi lưu trữ xâu kí tự. Chẳng hạn, xâu 'Hello' được lưu trữ trong vùng đệm như sau:

Chứa độ dài lớn nhất của xâu	Chứa độ dài thực của Xâu	Kí tự đầu tiên					Kí tự kết thúc xâu
255	5	H	E	l	l	o	$

Byte đầu tiên của vùng đệm chứa độ dài lớn nhất của xâu, byte thứ 2 chứa độ dài thực. Xâu thực sự được chứa từ byte thứ 3 trở đi. Tuy nhiên, thông thường thì người dùng kết thúc việc nhập

bằng phìm Enter có mã ASCII là 13 nhưng kí tự kết thúc xâu lại là $ nên chương trình phải xử lý việc này bằng cách sau:

Lấy địa chỉ offset của xâu đem cộng với nội dung byte thứ hai rồi cộng với 2 thì sẽ trỏ đến byte cuối cùng của xâu đang chứa mã ASCII của Enter (13) rồi thay thế mã này bởi kí tự kết thúc xâu là '$'.

Dưới đây là chương trình hợp ngữ:

```
.MODEL Tiny
.CODE
      Org 100h
      Jmp Start
      XauIn db 'Nhap xau: ','$'
      XauOut db 'Xau vua nhap: ','$'
      Xuongdong db 13,10,24h
      Buffer db 100 dup(?)  ; Khai bao buffer
Start:
      Mov AH,9
      Mov DX, offset XauIn
      Int 21h
      Mov AH,0Ah
      Mov DX, offset Buffer
      Mov BX,DX ; BX va DX cung tro den Buffer
      Mov BYTE PTR[BX],100 ; Do dai lon nhat cua
      Int 21h
      Mov AH,9
      Mov DX, offset Xuongdong ; xuong dong va ve dau dong
      Int 21h
      Mov DX, offset XauOut  ; Xau kq
      Int 21h
      Mov DX,BX ; BX,DX cung tro den Buffer
      Add BL,[BX+1] ; Cong vao do dai thuc cua xau vao BX
      Add BX,2    ; Tro den byte cuoi cung
      Mov BYTE PTR[BX],'$' ; Thay the byte cuoi cung boi $
      Add DX,2 ; bo qua hai byte dau
      Int 21h  ; in xau ra
      Int 20h ; trở về DOS
End Start
```

Ví dụ 5: Viết chương trình tạo một thư mục với tên thư mục được nhập từ bàn phím.

Bài làm

Trước hết, ta phải nhập vào 1 xâu ký tự tên thư mục. Sau đó sử dụng hàm 39h để tạo thư mục. Kết thúc việc tạo thư mục ta kiểm tra cờ Carry (CF), nếu cờ Carry bằng 1 thì việc tạo thư mục đã bị lỗi, ngược lại là tạo thành công. Lệnh JC (Jump if Carry equals to 1) thực hiện việc đó.

```
.MODEL Tiny
.CODE
```

```
        Org 100h
        Jmp Start
        TenThuMuc db 'Nhap ten thu muc: ','$'
        OK db 'Tao thu muc thanh cong','$'
        NotOK db 'Tao thu muc khong thanh cong','$'
        Xuongdong db 13,10,24h
Start:
        Mov AH,9
        Mov DX, offset TenThuMuc
        Int 21h
        Mov AH,0Ah
        Mov DX, offset Buffer
        Mov BX,DX    ; BX va DX cung tro den Buffer
        Mov BYTE PTR[BX],100 ; Do dai lon nhat cua
        Int 21h
        Mov AH,9
        Mov DX, offset Xuongdong ; xuong dong va ve dau dong
        Int 21h
        Mov DX,BX     ; BX,DX cung tro den Buffer
        Add BL,[BX+1] ; Cong vao do dai thuc cua xau vao BX
        Add BX,2      ; Tro den byte cuoi cung
        Mov BYTE PTR[BX],'$' ; Thay the byte cuoi cung boi $
        Add DX,2      ;bo qua hai byte dau, DX=ten thu muc
        Mov AH,39h    ; ham tao thu muc
        Int 21h
        JC Error      ; CF=1, bi loi
        Mov AH,9
        Mov DX, offset OK ; thanh cong
        Int 21h
        Jmp Ketthuc
Error:
        Mov AH,9
        Mov DX, offset NotOK ; Khong thanh cong
        Int 21h
Ketthuc:
        Int 20h ; trở về DOS
End Start
```

Ví dụ 6: Hãy định nghĩa trước một mảng các số nguyên trỏ bởi biến Mang. Hãy sắp xếp theo chiều tăng dần mảng số nguyên này rồi in kết quả lên màn hình.

Bài làm

Đây là một bài toán hay gặp khi học các ngôn ngữ lập trình. Ta sử dụng thuật giải sắp xếp chèn (INSERTION SORT) để giải bài toán này. Ý tưởng như sau: tìm phần tử lớn nhất của mảng rồi đặt phần tử đó vào cuối dãy, sau đó lại tiếp tục quá trình này với các phần tử còn lại. Tại mỗi

lần tìm thì một phần tử được đặt đúng chỗ. Ở bước lặp thứ i có i phần tử được đặt đúng chỗ và ta chỉ cần tìm phần tử lớn nhất trong n-i phần tử còn lại.

Ta tổ chức chương trình này thành một chương trình chính và một chương trình con. Chương trình con Exchange sẽ làm nhiệm vụ hoán đổi vị trí của hai phần tử của dãy.

```
.MODEL small
.STACK 100h
.DATA
      Thongbao db 'Day da sap xep: ','$'
      xuongdong db 13,10,'$'
      Mang  db  8,4,3,1,2,5,1
        Db '$'
.CODE
Start:
      Mov AX,@Data
      Mov DS,AX
      Mov  AH,9
      Mov DX, offset Thongbao
      Int 21h   ; in thongbao
      Mov DX, offset xuongdong
      Int 21h   ; nhay xuong dong
      Mov BX,7  ; BX= so phan tu cua mang
      Mov DX, offset Mang ; DX tro vao mang
      Dec BX ; so vòng lặp bên ngoài
Lap:
      Mov SI,DX ; SI trỏ vào đầu mảng
      Mov CX,BX ;Số lần lặp ở vòng lặp bên trong (tìm max)
      Mov DI, SI ;giả sử phần tử thứ 1 là max
      Mov AL, [DI] ; AL= max
TimMax:
      Inc SI   ; phần tử kế tiếp
      Cmp [SI],AL ; phần tử mới > max?
      JB Tiep ; không lớn hơn max
      Mov DI,SI ; lớn hơn max, DI trỏ vào max
      Mov AL,[DI]; Đưa max vào AL
Tiep:
      Loop TimMax
      Call DoiCho
      Dec BX
      JNZ Lap
; In Mang
 Mov BX,DX ; BX trỏ đến phần tử đầu tiên
 Mov CX,7 ; in cả 7 phần tử
 Mov AH,2
```

```
InMang:
 Mov DL,[BX]
 Add DL,30h
 Int 21h  ; in ra
 Mov DL,32 ; in dấu cách giữa các phần tử cho dễ xem
 Int 21h
 Inc BX  ; sang phần tử kế tiếp
 Loop InMang

Ketthuc:
      Mov AH,4Ch
      Int 21h
End Start
;---------------------------------------
; chương trình con DoiCho
;---------------------------------------
DoiCho Proc
      Push AX
      Mov AL,[SI]
      XCHG AL,[DI]
      Pop AX
      Ret
Nhap Endp
;---------------------------------------
```

2.7 TÓM TẮT

Chương này trình bày các vấn đề cơ bản của lập trình hợp ngữ: tạo và thực hiện một chương trình hợp ngữ, các cấu trúc lập trình cơ bản , chương trình con và Macro.

Trong phần viết và thực hiện một chương trình chúng ta đã tìm hiểu cấu trúc đầy đủ một lệnh hợp ngữ, cách thức khai báo hằng và biến. Phần này cũng trình bày khung của chương trình hợp ngữ, cách tạo, dịch và chạy một chương trình hợp ngữ.

Công cụ quan trọng của các ngôn ngữ lập trình là các cấu trúc lập trình. Hợp ngữ chỉ hỗ trợ các lệnh nhảy (jump) và so sánh (compare) cho phép người lập trình cài đặt các cấu trúc này. Phần các cấu trúc lập trình cơ bản trình bày cú pháp, cách cài đặt các cấu trúc: tuần tự, điều kiện (IF-THEN), điều kiện phân nhánh (IF-THEN-ELSE), lựa chọn (CASE), các cấu trúc lặp xác định trước (FOR-DO) và lặp không xác định trước (WHILE-DO, REPEAT-UNTIL).

Phần tiếp theo là các vấn đề liên quan đến chương trình con. Ngoài các vấn đề cơ bản liên quan đến chương trình con như: cơ chế, cấu trúc của chương trình con, cách thức truyền tham số, phần này cũng đề cập đến việc chia nhỏ một chương trình lớn thành các chương trình con và gói chúng vào các module hay thư viện nhờ các lệnh điều khiển PUBLIC, EXTRN, GLOBAL. Ngoài ra, cùng với chương trình con, Macro là một lựa chọn khi viết chương trình hợp ngữ. Macro

không những thực hiện nhanh, mềm dẻo và khá hiệu quả. Chúng ta cũng đã khảo sát về các lệnh điều khiển hay được sử dụng trong các Macro, cách khai báo và sử dụng Macro.

Phần trình bày tiếp theo là kết nối hợp ngữ với các ngôn ngữ bậc cao. Điển hình là ngôn ngữ C va Pascal. Đây là phần quan trọng và có ý nghĩa thực tế. Trong một hệ thống hay ứng dụng, các thành phần có thể được phát triển trên các ngôn ngữ khác nhau và chúng phải được kết hợp với nhau trong khi thực hiện. Chẳng hạn, nhân (kernel version 1.0) của hệ điều hành Linux co khoảng 10 ngàn dòng lệnh hợp ngữ (chiếm khoảng 6% tổng số dòng lệnh) mà rất khó để thay thế bởi bất kỳ ngôn ngữ khác, phần còn lại được viết bằng ngôn C dưới dạng các modules khác nhau.

Phần cuối cùng là giới thiệu về ngắt, cách xây dựng chương trình con ngắt và chương trình thường trú.

2.8 BÀI TẬP

2.8.1 Câu hỏi trắc nghiệm

1. Phát biểu nào sau đây là đúng nhất cho một chương trình .COM:

A. Chương trình gồm 2 đoạn: một đoạn chứa mã lệnh đoạn còn lại chứa dữ liệu khai báo.

B. Chương trình gồm 1 đoạn chứa đồng thời mã lệnh và dữ liệu khai báo.

C. Chương trình gồm 1 đoạn chỉ chứa mã lệnh.

D. Chương trình gồm 1 đoạn chỉ chứa dữ liệu khai báo.

2. Phát biểu nào sau đây là đúng nhất cho một chương trình .EXE:

A. Chương trình gồm ít nhất 3 đoạn: một đoạn chứa mã lệnh, một đoạn chứa ngăn xếp và đoạn còn lại chứa dữ liệu khai báo.

B. Chương trình gồm nhiều nhất 3 đoạn: Stack, Data và Code.

C. Chương trình gồm đúng 3 đoạn: Stack, Data và Code.

D. Có thể gộp mã lệnh và dữ liệu khai báo vào chung một đoạn.

3. Phát biểu nào sau đây là đúng nhất cho một chương trình con hợp ngữ.

A. Chương trình con luôn luôn được gọi bởi chương trình chính.

B. Chương trình con gồm 3 đoạn: Stack, Data và Code.

C. Lệnh cuối cùng của chương trình con là lệnh End Start.

D. Lệnh cuối cùng của chương trình con là lệnh RET.

4. Phát biểu nào sau đây là đúng nhất cho việc kết hợp giữa hợp ngữ và C bằng cách sử dụng inline assembly:

A. inline assembly chỉ chứa các lệnh MOV, INC, DEC, ADD,SUB

B. Các lệnh inline assembly được viết cùng với các lệnh C trong file chương trình C

C. Các lệnh inline assembly được viết cùng với các lệnh C trong file chương trình hợp ngữ.

D. Trong cùng một chương trình C, các inline assembly phải được viết tách biệt với các lệnh của C.

5. Phát biểu nào sau đây là đúng nhất cho việc kết hợp giữa hợp ngữ và C bằng cách sử dụng viết tách biệt module hợp ngữ và C:

A. Trong các module C chỉ chứa các lệnh C.

B. Trong module hợp ngữ không thể gọi được các hàm viết từ C

C. Trong module C có thể gọi các hàm từ module hợp ngữ và trong module hợp ngữ cũng có thể gọi các hàm từ module C.

D. Các module C và hợp ngữ được dịch và thực hiện độc lập với nhau.

6. Phát biểu nào sau đây là sai đối với việc tương thích kiểu giữa module hợp ngữ và module C:

A. Kiểu unsigned char của C không tương thích với kiểu byte của hợp ngữ.

B. Kiểu char của C không tương thích với kiểu word của hợp ngữ

C. Kiểu short của C tương thích với kiểu word của hợp ngữ

D. Kiểu far * của C tương thích với kiểu dword của hợp ngữ.

7. Phát biểu nào sau đây là sai đối với kiểu giá trị trả lại của hàm và nơi đặt giá trị trả lại trong module hợp ngữ:

A. Kiểu giá trị unsigned long được đặt vào DX:AX.

B. Kiểu giá trị enum được đặt vào AX

C. Kiểu giá trị float được đặt vào đỉnh ngăn xếp 8087 thanh ghi ST(0)

D. Kiểu giá trị unsigned long được đặt vào AX

8. Khi một ngắt được đáp ứng yêu cầu, các lệnh sẽ được thực hiện theo thức tự sau:

A. PUSHF, CLI, PUSH CS, PUSH IP.

B. PUSH IP, PUSH CS, PUSH IP, CLI,.

C. PUSH CS, PUSH IP, CLI, PUSHF .

D. CLI, PUSH CS, PUSH IP, PUSHF

9. Phát biểu nào sau đây là đúng nhất đối chương trình thường trú:

A. Có thể viết chương trình thường trú sử dụng khung của chương trình .EXE.

B. Hỗ trợ khả năng kích hoạt và nằm lại bộ nhớ sau khi chạy xong.

C. Được kích hoạt bởi một tổ hợp phím nóng (hot-key).

D. Hoạt động giống như các chương trình bình thường khác.

10. Phát biểu nào sau đây là đúng nhất đối chương trình thường trú:

A. Vùng nhớ cấp phát cho chương trình thường trú không được giải phóng để cấp phát cho chương trình khác khi nó thực hiện xong .

B. Vùng nhớ cấp phát cho chương trình thường trú được giải phóng để cấp phát cho chương trình khác khi nó thực hiện xong

C. Khi thực hiện lần đầu tiên nó không cần chương trình tải (program loader) tải vào vùng nhớ cấp phát cho nó.

D. Mỗi lần thực hiện chương trình thường trú sẽ được tải vào vùng nhớ được cấp phát cho nó.

2.8.2 Bài tập lập trình

1. Viết chương trình in ra màn hình 26 chữ cái 'A' …'Z'.

2. Viết chương trình nhập vào 1 số nguyên n (0≤ n≤ 9), tính tổng S=1+2+3+…+n rồi in kết quả ra màn hình.

3. Viết chương trình nhập vào hai số nguyên x và y (0≤ x,y ≤ 9), tính tổng x+y rồi in kết quả ra màn hình. Yêu cầu: Chương trình được tổ chức như sau: gồm 1 chương trình chính, 2 chương trình con hoặc 2 macro, 1 dùng để nhập vào 1 số và 1 dùng để in kết quả .

4. Viết chương trình nhập vào từ bàn phím hai số nguyên a và b với 0 ≤ a,b ≤256. Tính tích của chúng và in kết quả ra màn hình. Yêu cầu chương trình phải được tổ chức thành các chương trình con hoặc macro.

5. Viết chương trình nhập vào một tên file rồi xóa file đó. In ra màn hình thông báo xóa thành công hay không.

6. Viết chương trình tạo một file backup từ một file văn bản nguồn. Tên của file văn bản được nhập vào từ bàn phím.

7. Dùng trình tiện ích Debug để chạy và gỡ lỗi chương trình ví dụ 1 phần 2.4

8. Viết chương trình nhập vào một số nguyên N (0<N<256), hãy in ra N dưới các dạng: thập phân, nhị phân và hexa. Yêu cầu chương trình được tổ chức như sau:

   a. Dùng inline-assembly và C

   b. Viết tách biệt module C và hợp ngữ

   c. Viết tách biệt module Pascal và hợp ngữ

9. Viết chương trình nhập vào từ phàn phím hai số nguyên dương (0<x,y<256), hãy tính tổng hiệu, tích thương của chúng rồi in ra màn hình. Yêu cầu chương trình được tổ chức như sau:

   a. Dùng inline-assembly và C

   b. Viết tách biệt module C và hợp ngữ

10. Lập chương trình thực hiện nhiệm vụ copy một file có kích thước tùy ý. Tên file nguồn và đích nhận vào từ bàn phím. Thông báo lỗi ra màn hình. Yêu cầu chương trình được tổ chức như sau:

    a. Dùng inline-assembly và C

    b. Viết tách biệt module C và hợp ngữ

2.9 TÀI LIỆU THAM KHẢO

2. Văn Thế Minh. Kỹ thuật Vi xử lý. Nhà XB Giáo dục 1997.
3. Đặng Thành Phu. Turbo Assembler và Ứng dụng. NXB Khoa học và Kỹ thuật 1998.
4. Nguyễn Minh San. Cẩm nang Lập trình hệ thống (bản dịch). NXB Tổng cục Thống kê.2001.

CHƯƠNG 3. CÁC CÔNG CỤ HỖ TRỢ

Phần này trình bày về các công cụ hỗ trợ cho việc lập trình hợp ngữ bao gồm: trình tiện ích Debug, chương trình mô phỏng Emu8086, và công cụ phát triển các ứng dụng viết bằng hợp ngữ trên môi trường Windows 32 bít - RadASM .

3.1 BỘ GỠ RỐI DEBUG

3.1.1 Tổng quan về Debug

DEBUG là một trình tiện ích trợ giúp người lập trình tác động đến quá trình thực hiện một công việc (task) nào đó. Đồng thời Debug là một chương trình dùng để gỡ lỗi chương trình.

Debug hỗ trợ cho người dùng các nhóm lệnh sau:

Thao tác với bộ nhớ:

- lệnh hiển thị nội dung ô nhớ (lệnh D)
- lệnh sửa nội dung ô nhớ (lệnh E)
- điền thông tin vào một vùng nhớ (lệnh F)
- chuyển nội dung từ vùng nhớ này sang vùng nhớ khác (lệnh M).

Thao tác với các files:

- đặt tên file (lệnh N)
- nạp nội dung một file vào bộ nhớ (lệnh L)
- chạy file dang .COM hoặc .EXE (lệnh G)

Truy cập đến các sector trên đĩa (lệnh L,W)

Soạn thảo và thực hiện một chương trình hợp ngữ (lệnh A,G)

Theo dõi quá trình thực hiện 1 chương trình

- xem, sửa chữa trạng thái thanh ghi (lệnh R)
- chạy từng bước (lệnh T, lệnh P)

Thực hiện một số thao tác vào/ra đối với các thiết bị ngoại vi (lệnh I và O)

Dịch ngược từ mã máy sang hợp ngữ (lệnh U)

Tìm kiếm (lệnh S)

3.1.2 Sử dụng Debug

a. Khởi động Debug

Cách 1: Tại thư mục DOS:

C:\DOS>

gõ debug <Enter>

C:\DOS> debug <Enter>

Dấu nhắc của Debug là dấu –

Cách 2:

Gõ C:\DOS> debug tenfile.exe <Enter>

Khi đó cả trình debug và chương trình người dùng (tenfile.exe) đều được đưa vào bộ nhớ RAM.

b. Một số lưu ý

- Địa chỉ của vùng nhớ (address): được thể hiện dưới dạng SEGMENT:OFFSET, chẳng hạn như:

  DS:0300 hoặc

  9D0:01FF

  Nếu ở đoạn hiện tại, ta có thể chỉ cần dùng địa chỉ offset, ví dụ:

  02FFh
- Khoảng (range): thể hiện địa chỉ một vùng nhớ:

  address L value

  ví dụ: DS:1FF L 10
- Mỗi lệnh của Debug gồm 1 kí tự duy nhất
- Giữa tên lệnh và tham số có ít nhất 1 dấu cách
- Dùng dấu cách giống như dùng dấu phẩy (,)
- Kết thúc 1 lệnh đang được thực hiện bằng Ctrl+C hoặc Ctrl+Break
- Lệnh được thực hiện nếu gõ tên lệnh và gõ enter

3.1.3 Các lệnh của Debug

1. Lệnh: A

Chức năng: Soạn thảo và dịch trực tiếp các lệnh hợp ngữ.

Cú pháp: **A [địa chỉ] <Enter>**

Trong đó **[địa chỉ]** là địa chỉ offset của ô nhớ (dạng hexa) mà ta cần đặt mã lệnh vào.

Ví dụ:

```
-A 1FF0
xxxx:1FF0  mov AH,9
xxxx:1FF2  mov AH,9
```

- Nếu chưa xác định được địa chỉ ban đầu thì lệnh sẽ tự đông đưa vào địa chỉ offset 0100h
- Nếu phát hiện lỗi trong một lệnh, Debug sẽ đưa ra thông báo ERROR và hiện lại địa chỉ ô nhớ có lỗi đó để người sử dụng sửa lại lệnh đó cho đúng
- Muốn trở lại dấu nhắc Debug thì nhấn hai lần Enter
- Lệnh A luôn sử dụng địa chỉ tuyệt đối, nghĩa là luôn đi kèm với 1 địa chỉ chính xác.

2. Lệnh: C

Chức năng: So sánh nội dung của hai vùng nhớ.

Cú pháp: **C khoảng,địa chỉ <Enter>**

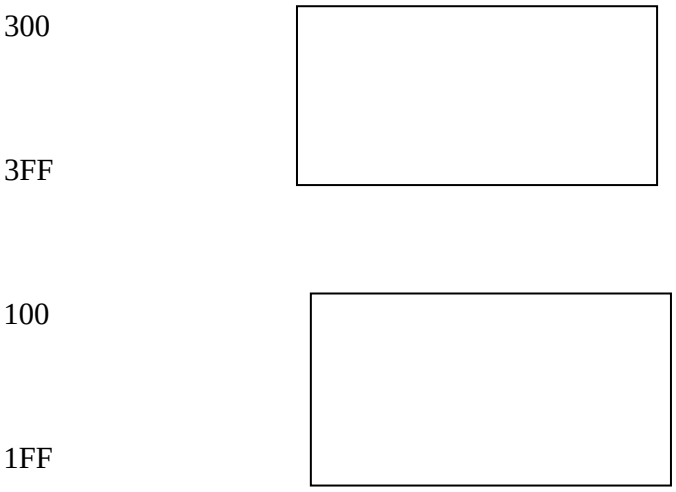
Trong đó khoảng bao gồm địa chỉ đầu tiên và địa chỉ cuối cùng của một vùng nhớ, địa chỉ là 1 địa chỉ offset bắt đầu của 1 vùng nhớ khác.

Ví dụ:

-C 100, 1FF, 500 <Enter>

Hoặc:

-C 100 L 100, 500 <Enter>



Hình 3.1 Hai vùng nhớ khác nhau

3. Lệnh: D

Chức năng: Hiển thị nội dung của một vùng nhớ.

Cú pháp: **D [khoảng |địa chỉ] <Enter>**

Trong đó khoảng bao gồm địa chỉ đầu tiên và địa chỉ cuối cùng của một vùng nhớ, hoặc địa chỉ là 1 địa chỉ offset của 1 ô nhớ khác.

Ví dụ:

-D 100, 1FF <Enter>

Hoặc:

-D 100 L 11 <Enter>

Nếu sử dụng lệnh:

-D <address> <Enter>

Thì nội dung các ô nhớ từ địa chỉ đã cho đến 128 byte kế tiếp sẽ được hiện lên. Nếu tiếp tục

-D <Enter>

Thì nội dung 128 byte kế tiếp theo đó được hiện lên.

4. Lệnh: E

Chức năng: Hiện nội dung ô nhớ và cho phép sửa nội dung ô nhớ.

Cú pháp:

Cách 1*:* **E <địa chỉ> <danh sách các giá trị> <Enter>**

Ví dụ:

-E 01FF:0100 ‘ABC’9A <Enter>

Thì các ô nhớ từ 01FF:0100 đến 01FF:0103 sẽ lần lượt được điền các các giá trị là mã ASCII của A, B, C và giá trị 9A.

Cách 2*:* **E <địa chỉ> <Enter>**

Thì nội dung của ô nhớ có địa chỉ trên sẽ hiện lên, ta có thể thay đổi giá trị mới và sau đó nếu muốn thay đổi nội dung của ô nhớ kế tiếp thì nhấn dấu cách (SPACE), còn nếu muốn dừng lại thì nhấn Enter để trở lại màn hình debug.

Ví dụ:

-E 01FF:0100 9A

Nếu muốn sửa giá trị của ô nhớ này thì đưa vào giá trị mới vào rồi nhấn Enter đển hiện nội dung của ô nhơ kế tiếp 01FF:0101 và cứ tiếp tục như vậy cho đến khi nhấn Enter nếu muốn dừng.

5. Lệnh: F

Chức năng: Nạp các giá trị trong danh sách vào một vùng nhớ.

Cú pháp: **F <khoảng> <danh sách các giá trị> <Enter>**

Ví dụ:

-F 01FF:0100 L 4,‘ABC’9A <Enter>

Thì các ô nhớ từ 01FF:0100 đến 01FF:0103 sẽ lần lượt được điền các các giá trị là mã ASCII của A, B, C và giá trị 9A.

6. Lệnh: G

Chức năng: Cho thực hiện một chương trình đang hiệu chỉnh. Việc thực hiện chương trình sẽ dừng lại khi đạt đến địa chỉ dừng. Sau đó’, hiển thị các thanh ghi và dòng lệnh thực hiện tiếp theo của chương trình

Cú pháp: **G [<địa chỉ đầu>] [<địa chỉ dừng>] <Enter>**

Nếu gõ lệnh - **G [<địa chỉ đầu>] <Enter>**

Thì chương trình sẽ được thực hiện từ địa chỉ đầu cho đên lệnh cuối cùng của chương trình. Nếu là chương trình con thì dừng lại khi gặp lệnh RET, nếu là macro thì dừng lại khi gặp lệnh ENDM.

Nếu gõ lệnh - **G <Enter>**

Thì chương trình sẽ được thực hiện từ địa chỉ đã được nạp vào cặp thanh ghi CS:IP cho đến lệnh cuối cùng của chương trình

7. Lệnh: H

Chức năng: Cộng và trừ hai giá trị hexa và hiển thị kết quả của tổng và hiệu lên màn hình.

Cú pháp: **H <giá trị 1> <giá trị 2> <Enter>**

Ví dụ: - H 10, 0f <Enter>

1F, 01

Kết quả của tổng là 1F và của hiệu là 01

8. Lệnh: I

Chức năng: Đọc và hiển thị giá trị của một cổng lên màn hình.

Cú pháp: **I <địa chỉ của cổng vào> <Enter>**

Ví dụ: - I 1f <Enter>

26

26 là giá trị đọc được từ cổng 1F.

9. Lệnh: L

Chức năng: Chuyển nội dung 1 file hoặc nội dung sector của đĩa vào vùng nhớ.

Cú pháp:

Dạng 1: **L [<địa chỉ>,[, ổ đĩa, sector, số sector]] <Enter>**

Đọc số liệu từ sector đầu của ổ đĩa, với số lượng sector và bắt đầu từ địa chỉ được chỉ ra ở tham số thứ nhất.

Ví dụ 1: - L 01FA:0100 1 0A 30 <Enter>

Đọc số liệu của 48 sector (30h) bắt đầu từ sector 0A của ổ đĩa B và vùng nhớ có địa chỉ 01FA:0100.

(Các ổ đĩa được kí hiệu như sau: 0 là ổ đĩa A, 1 là ổ đĩa B, 2 là ổ đĩa C, 3 là ổ đĩa D).

Nếu tên file được đặt tên bởi lệnh –N <tên file> thì:

- lệnh –L sẽ nạp nội dung của file vào vùng nhớ mặc định CS:0100
- lệnh –L <địa chỉ> sẽ nạp nội dung của file vào vùng nhớ có địa chỉ <địa chỉ>.

Ví dụ 2:

- N cong2so <Enter>
- L <Enter>

Nội dung của file cong2so sẽ được nạp vào vùng nhớ có địa chỉ đầu là CS:0100

Ví dụ 3:

- N cong2so <Enter>
- L 03FF <Enter>

Nội dung của file cong2so sẽ được nạp vào vùng nhớ có địa chỉ đầu là CS:03FF

10. Lệnh: M

Chức năng: Chuyển nội dung từ một vùng nhớ sang một vùng nhớ khác.

Cú pháp: **-M <khoảng>,<địa chỉ> <Enter>**

Nếu trong khoảng và địa chỉ không xác định đoạn thì sẽ lấy DS là địa chỉ đoạn..

Ví dụ nếu sử dung 1 trong 3 lệnh sau:

- M 01FF:0100, 010E, 01FF:0200 <Enter>
- M 01FF:0100 L F, 01FF:0200 <Enter>
- M 01FF:0100 010E 01FF:0200 <Enter>

Thì 15 byte từ vùng nhớ có địa chỉ 01FF:0100 đến 01FF:010E sẽ được chuyển đến vùng nhớ có địa chỉ bắt đầu là: 01FF:0200

11. Lệnh: N

Chức năng: Đặt tên cho file.

Lệnh này thường được đi kèm với các lệnh –L và –W dùng tên file đó .

Cú pháp: **-N <tên file>.EXE (hoặc .COM) <Enter>**

Ví dụ:

- N Tenfilemoi.exe <Enter>
- L <Enter>

12. Lệnh: O

Chức năng: Đưa một byte dữ liệu ra cổng.

Cú pháp: **-O <địa chỉ cổng ra>, <giá trị> <Enter>**

Ví dụ:

- O 02F, 20 <Enter>

13. Lệnh: Q

Chức năng: Thoát khỏi Debug và trở về DOS.

Cú pháp: **-Q <Enter>**

14. Lệnh: R

Chức năng: Hiển thi và sửa đổi nội dung các thanh ghi.

Cú pháp: **-R [thanh ghi | F] <Enter>**

Có các trường hợp sau:

- R <enter> hiển thị và sửa đổi nội dung các thanh ghi.
- RAX <enter> hiển thị nội dung của thanh ghi AX và cho phép sửa nội dung đó, ví dụ: AX 101A. Nếu không muốn thay đổi giá trị thì nhấn Enter, còn muốn sửa giá trị mới

thì nhập giá trị mới vào rồi nhấn Enter. Muốn hiển thị nội dung của các thanh ghi kế tiếp (BX, CX, DX) thì nhấn dấu cách (SPACE).

- R F <Enter> hiện và sửa nội dung của thanh ghi cờ.

15. Lệnh: S

Chức năng: Tìm trong vùng nhớ xác định bởi khoảng các kí tự trong danh sách.

Cú pháp: **-S <khoảng>,<danh sách> <Enter>**

Nếu khoảng không xác định thì đoạn ngầm định là thanh ghi DS.

Ví dụ 1:

- S 01FF:0100 0110 20 <Enter>

Hoặc:

- S 01FF:0100 L 10 20 <Enter>

Thì sẽ tìm các ô nhớ có nội dung bằng 20h trong vùng nhớ từ 01FF:0100 đến 01FF:0110. Kết quả là tất cả địa chỉ của các ô nhớ có nội dung bằng 20 thì sẽ được hiển thị, chẳng hạn có 3 ô nhớ có nội dung bằng 20 thì màn hình sẽ liệt kê như sau:

01FF:0100

01FF:0104

01FF:0105

Ví dụ 2:

- S 01FF:0100 0110 ‘ABC’2E <Enter>

Sẽ tìm 4 ô liên tiếp chứa giá trị là mã ASCII của A, B, C và 2E.

16. Lệnh: T

Chức năng: Thực hiện một hay nhiều lệnh bắt đầu từ địa chỉ CS:IP hoặc từ địa chỉ được chỉ ra ở dấu = , và hiển thị trạng thái toàn bộ các thanh ghi sau mỗi lệnh.

Cú pháp: **-T [= địa chỉ][, số lệnh] <Enter>**

- địa chỉ là địa chỉ của lệnh đầu tiên sẽ thực hiện
- số lệnh được thực hiện trong chế độ này.

Có các trường hợp sau:

- T <enter> lệnh tại địa chỉ CS:IP sẽ được thực hiện.
- T 10 <enter> 10 lệnh bắt đầu từ địa chỉ CS:IP sẽ được thực hiện. Trạng thái của tất cả các thanh ghi sau mỗi lệnh sẽ được hiện ra 1 cách liên tục.
- T=2FF,10 <enter> sẽ thực hiện 16 lệnh bắt đầu từ lệnh tại địa chỉ CS:02FF.

17. Lệnh: P

Chức năng: Giống lệnh T nhưng thực hiện cả 1 chương trình con.

Cú pháp: **-P [= địa chỉ][, số lệnh] <Enter>**

18. Lệnh: U

Chức năng: Dịch ngược các lệnh dưới dạng mã máy nằm trong vùng nhớ sang dạng hợp ngữ và hiển thị địa chỉ, mã máy và lệnh dạng gợi nhớ lên màn hình.

Cú pháp: **-U [khoảng][địa chỉ]<Enter>**

Có các trường hợp sau:

- U <khoảng> <enter> các lệnh nằm trong vùng nhớ sẽ được dịch ngược.
- U <địa chỉ> <enter> dịch ngược bắt đầu từ địa chỉ cho đến 128 byte kế tiếp.
- U <enter> dịch ngược bắt đầu từ địa chỉ CS:IP đến 128 byte kế tiếp.

19. Lệnh: W

Chức năng: Ghi dữ liệu lên đĩa.

Cú pháp: **-W [địa chỉ [,ổ đĩa, sector đầu, số sector] <Enter>**

Dữ liệu trong vùng nhớ bắt đầu từ địa chỉ ghi lên ổ đĩa vào sector đầu tiên cho đến sector cuối do số sector xác định.

Ví dụ:

- N cong2so.exe <enter>
- W 01FF:0200, 1, 2A,5 <Enter>

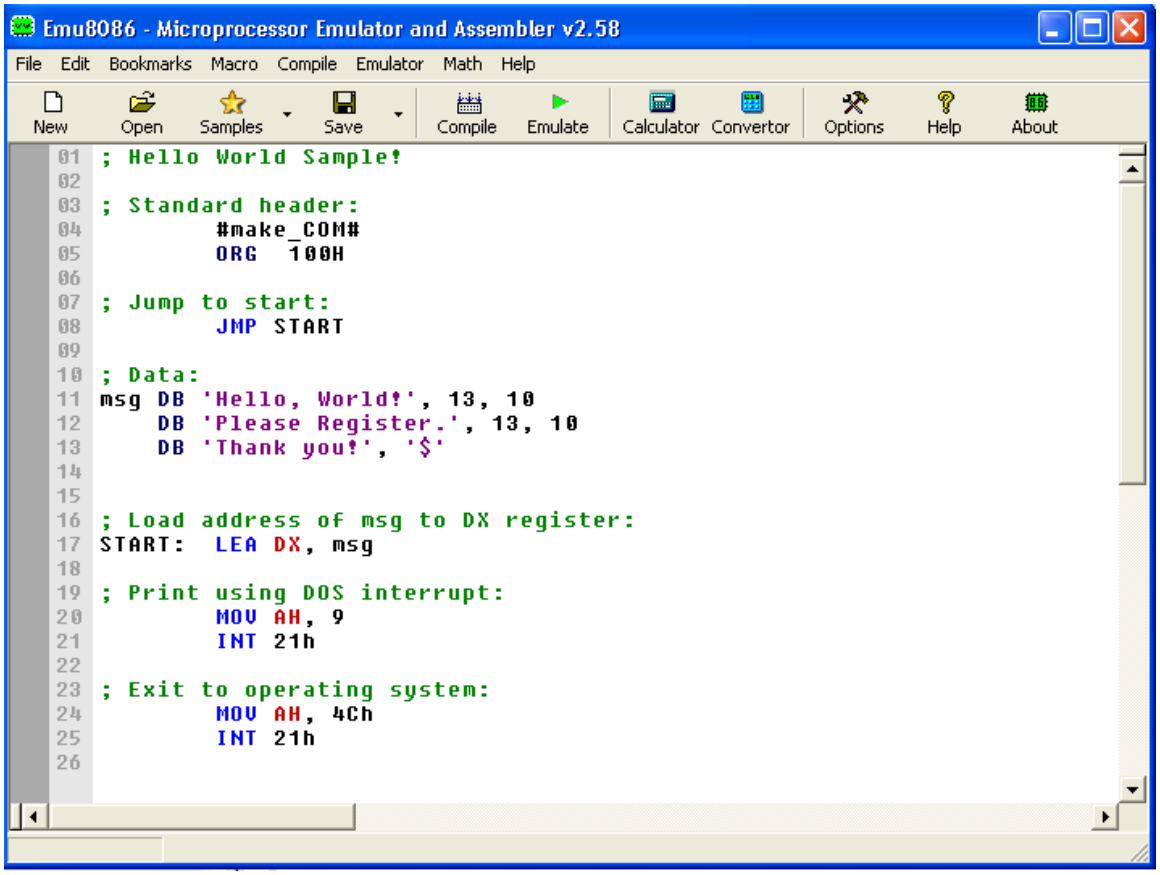
Dữ liệu từ vùng nhớ xác định bởi 01FF:0200 được ghi vào ổ đĩa B từ sector 2A và ghi vào 5 sector với tên file là cong2so.exe.

3.2 CHƯƠNG TRÌNH MÔ PHỎNG EMU8086

Hiện nay, hầu hết các desktop tại các phòng thực hành tại Việt nam chạy hệ điều hành 32-bit như windows XP, NT, 2000… thì người lập trình không thể gọi ngắt bằng chương trình người dùng được. Thay vì gọi ngắt, các hệ điều hành 32-bit cung cấp một tập các hàm giao diện lập trình ứng dụng gọi là API (Application Programming Interface) cho phép người lập trình gọi hàm. Để người mới học lập trình hệ thống có thể lập trình với các ngắt mà không bị giới hạn bởi phiên bản khác nhau của các hệ điều hành của Microsoft thì cách tốt nhất là học lập trình trên môi trường mô phỏng Emulator 8086. Phần này chúng tôi giới thiệu về phần mềm mô phỏng CPU 8086 của công ty phần mềm Emu8086 Inc., phiên bản 2.58. Các phiên bản mới hơn có thể được download tại địa chỉ của trang web: www.emu8086.com.

3.2.1 Các chức năng soạn thảo, dịch và thực hiện chương trình.

Dưới đây là màn hình cho phép người sử dụng viết một chương trình hợp ngữ hoặc chạy thử một ví dụ có sẵn:



Hình 3.2 Màn hình soạn thảo chương trình Emu8086

New: tạo một chương trình mới, khi đó người dùng sẽ được hỏi xem sẽ tạo file chương trình dạng nào: COM, EXE, BIN hay BOOT .

Open: mở một file chương trình nguồn hợp ngữ.

Samples: Liệt kê các file chương trình mẫu có sẵn do chương trình mô phỏng cung cấp.

Save: Lưu file chương trình nguồn

Compile: dịch file chương trình nguồn

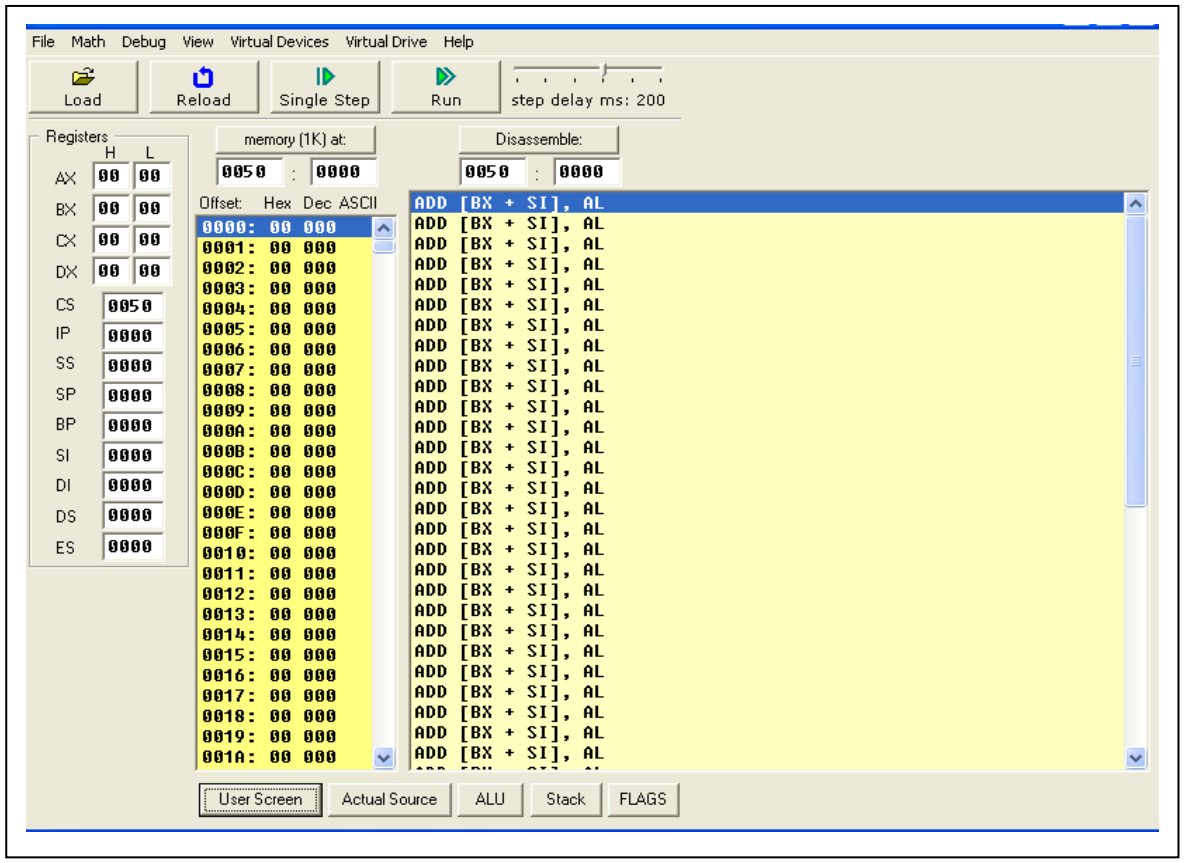
Emulate: cho phép thực hiện chương trình nguồn. Các trạng thái của quá trình thực hiện chương trình được hiển thị trên màn hình mô phỏng dưới đây.

Calculator: người dùng có thể nhập 1 vào một biểu thức với các số là: có dấu, số dạng word hoặc số dạng byte để tính toán. Kết quả tính toán được hiển thị một trong các dạng số thập phân, nhị phân, hexa hoặc số bát phân (cơ số 8).

Convertor: Bộ chuyển đổi gữa các cơ số. Emu8086 hỗ trợ chuyển đổi giữa các cơ số 16 thành cơ số 10 có dấu hoặc không dấu. Chuyển đổi từ cơ số 8 thành thành cơ số 2 (nhị phân). Một mã ASCII gồm 2 số hexa cũng có thể được chuyển đổi thành thập phân hoặc nhị phân.

3.2.2 Chức năng mô phỏng quá trình thực hiện chương trình.

Dưới đây là màn hình mô phỏng trạng thái thực hiện một chương trình.



Hình 3.2 Màn hình mô phỏng trạng thái thực hiện chương trình

Các chức năng chính:

Load: tải chương trình. Trước khi thực hiện thì chương trình sẽ được tải vào trong bộ nhớ. Chương trình có thể ở dạng các file thực hiện được như EXE, COM, BIN, BOOT hoặc dưới dạng file nguồn ASM.

Reload: người dùng có thể tải lại 1 chương trình.

Single Step: chạy chương trình theo chế độ từng lệnh. Với chế độ này, người dùng có thể quan sát trạng thái các thanh ghi, bộ nhớ trong…

Run: chế độ chạy tất cả các lệnh trong chương trình.

Trên màn hình, người dùng có thể quan sát trạng thái các thanh ghi và đoạn bộ nhớ sử dụng cho đoạn mã lệnh của chương trình.

Phần registers mang nội dung của các thanh ghi trong đó các thanh ghi AX,BX,CX và DX được chia làm 2 nửa. phân cao (H) và phần thấp (L). Ngoài ra, ta có thể xem nội dung các thanh ghi đoạn, con trỏ lệnh, ngăn xếp…

Phần bộ nhớ lưu trữ đoạn mã chương trình. Địa chỉ đoạn (dạng hexa) được lưu trong thanh ghi CS. Danh sách địa chỉ offset được hiển thị dưới các dạng hexa và thập phân.

Ngoài ra, người dùng có thể có thể xem:

- kết quả hiển thị lên mà hình (nhắp chuột vào nút User Screen).

- mã nguồn của chương trình (nhắp chuột vào nút Actual Source).
- trạng thái ALU (nhắp chuột vào nút ALU).
- nội dung của ngăn xếp (nhắp chuột vào nút Stack).
- nội dung của thanh ghi cờ (nhắp chuột vào nút FLAG)

3.2.3 Các chương trình mẫu.

Emu8086 cung cấp cho người dùng 54 chương trình mẫu. Chúng rất có ích cho người học lập trình hợp ngữ. Từ các chương trình đơn giản như Hello world cho đến một số chương trình thao tác với một số thiết bị ngoại vi điển hình như màn hình, máy in…Để chạy thử các chương trình mẫu này, người dùng nhắp chuột vào nút Samples/ More Samples để chọn ra một file chương trình để chạy thử. Dưới đây là các giải thích cho 1 một số chương trình mẫu.

Chương trình Calculate SUM. Chương trình tính tổng các phần tử trong một mảng V1 đã được định nghĩa trước và lưu kết quả vào biến V2.

Dưới đây là nội dung chương trình (lời giải thích được dịch ra tiếng Việt):

```
#make_BIN#
; Tính tổng các phần tử trong mảng V1
; Lưu kết quả vào biến V2.
; Số phần tử của mảng:
MOV CX, 5
; AL chứa tổng các phần tử:
MOV AL, 0
; BX là chỉ số của mảng:
MOV BX, 0
; Tính tổng:
Tong:
ADD AL, V1[BX]
; có thể thay đổi giá trị của mảng
; đặt giá trị phần tử bằng chỉ số
MOV V1[BX], BL
; phần tử kế tiếp:
INC BX
; lặp cho đến khi CX=0:
; tính tổng của tất cả các phần tử
LOOP Tong
; lưu kết quả vào biến V2:
MOV V2, AL
HLT
; Khai báo biến:
V1 DB 4, 3, 2, 1, 0
V2 DB 0
```

Ở chương trình trên có một số điểm khác so với các chương trình ta thường thấy. Lệnh đầu tiên của chương trình là #make_BIN#. Chương trình sẽ được viết dưới dạng file binary.

Các biến của chương trình được khai báo ở phần cuối.

Chương trình tính tính tổng hai số nguyên được nhập từ bàn phím nằm trong khoảng [-32768..32767] rồi in kết quả ra mà hình.

Các file chương trình liên quan: calc.asm và emu8086.inc

Trong file emu8086.inc chứa một số chương trình con và macro được gọi từ calc.asm.

Dưới đây là chương trình calc với các lời giải thích đã được viết lại bằng tiếng Việt.

```
; Đây là chương trình  nhập vào 2 số nguyên
; trong khoảng [-32535, 32536] từ người dùng
; Tính tổng của chúng
; rồi in kết quả ra màn hình
; chương trình dạng COM
#make_COM#
include 'emu8086.inc'
ORG     100h
; Nhảy qua đoạn khai  báo biến, hằng
JMP     START
; khai báo biến:
num  DW ?
START:
; Nhập vào số thứ nhất:
CALL    PTHIS
DB 13, 10, 'Calculation Range: [-32768..32767]', 13, 10
DB 13, 10, 'Enter first number: ', 0
; Gọi chương trình con scan_num để nhập 1 số, kết quả trả lại
; là một số được lưu trong thanh ghi CX
CALL    scan_num
; Lưu số thứ nhất vào biến num:
MOV     num, CX
; nhập vào số thứ 2:
CALL    PTHIS
msg2 DB 13, 10, 'Enter second number: ', 0
CALL    scan_num
; cộng các số:
ADD     num, CX
JO      overflow
; In kết quả bằng chương trình con PTHIS
CALL    PTHIS
DB 13, 10, 'The sum is: ', 0
MOV     AX, num
CALL    print_num
JMP     exit
```

```
; xử lý lỗi tràn:
overflow:
   PRINTN 'We have overflow!'
exit:
RET
;================================
; Khai báo việc sử dụng các chương trình con
; hoặc macro trong  emu8086.inc
; Chương trình con SCAN_NUM đọc vào 1 số
; từ người dùng và lưu vào thanh ghi CX
DEFINE_SCAN_NUM
; Chương trình con PRINT_NUM in ra
; một số có dấu nằm trong AX
; Chương trình con  PRINT_NUM_UNS  in ra
; một số không dấu nằm trong AX
; do PRINT_NUM gọi đến
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS
; Chương trình con  PTHIS in ra giá trị rỗng (NULL)
; xâu được định nghĩa sau lệnh
; CALL PTHIS:
DEFINE_PTHIS
;================================
END
```

Dưới đây là các macro và chương trình con trong file INCLUDE emu8086.inc được gọi đến bởi chương trình trên.

Macro scan_num (trong đó các lời giải thích đã được viết lại bằng tiếng Việt)

```
;*************************************************************
; Đây là macro
; nhận vào một số nguyên có dấu
; và lưu trong thanh ghi CX:
DEFINE_SCAN_NUM         MACRO
; Khai báo các biến cục bộ
LOCAL make_minus, ten, next_digit, set_minus
LOCAL too_big, backspace_checked, too_big2
LOCAL stop_input, not_minus, skip_proc_scan_num
LOCAL remove_not_digit, ok_AE_0, ok_digit, not_cr
JMP     skip_proc_scan_num
SCAN_NUM        PROC    NEAR
        PUSH    DX
        PUSH    AX
```

```
        PUSH    SI
        MOV     CX, 0
        ; reset flag:
        MOV     CS:make_minus, 0
next_digit:
        ; nhập vào 1 kí tự từ bàn phìm
        ; đặt vào trong AL, dùng dịch vụ BIOS phục vụ bàn phím
        MOV     AH, 00h
        INT     16h
        ; và in ra:
        MOV     AH, 0Eh
        INT     10h
        ; Kiểm tra xem có phải là dấu âm:
        CMP     AL, '-'
        JE      set_minus
        ; phím Enter – hoàn thành việc nhập số
        CMP     AL, 13  ; 13 là mã ASCII của phím Enter?
        JNE     not_cr
        JMP     stop_input
not_cr:
        CMP     AL, 8 ; Có nhấn phím 'BACKSPACE'?
        JNE     backspace_checked
        MOV     DX, 0             ; có, thì bỏ đi số cuối cùng
        MOV     AX, CX            ;chia
        DIV     CS:ten          ; chia cho 10.
        MOV     CX, AX
        PUTC    ' '                 ; xóa dấu cách
        PUTC    8                   ; backspace again.
        JMP     next_digit
backspace_checked:
        ; chỉ cho phép nhập vào số
        CMP     AL, '0'
        JAE     ok_AE_0
        JMP     remove_not_digit
ok_AE_0:
        CMP     AL, '9'
        JBE     ok_digit
remove_not_digit:
        PUTC    8       ; phím backspace.
```

```
        PUTC    ' '     ; xóa nếu kí tự nhập được không phải là số
        PUTC    8       ; phím backspace.
        JMP     next_digit ; đợi nhập vào chữ số kế tiếp.
ok_digit:
        ; nhân CX với 10
        PUSH    AX
        MOV     AX, CX
        MUL     CS:ten                  ; DX:AX = AX*10
        MOV     CX, AX
        POP     AX
        ; kiểm tra lại nếu số quá lớn
        ;
        CMP     DX, 0
        JNE     too_big
        ; Đổi từ mã ASCII ra số thực sự
        SUB     AL, 30h
        ; add AL to CX:
        MOV     AH, 0
        MOV     DX, CX      ; lưu lại
        ADD     CX, AX
        JC      too_big2    ; nhảy nếu số quá lớn
        JMP     next_digit
set_minus:
        MOV     CS:make_minus, 1
        JMP     next_digit
too_big2:
        MOV     CX, DX   ; khôi phục lại giá trị đã được sao chép
        MOV     DX, 0    ; trước khi sao lưu DX=0
too_big:
        MOV     AX, CX
        DIV     CS:ten  ; Đảo lại chữ số cuối
        MOV     CX, AX
        PUTC    8       ; backspace.
        PUTC    ' '     ; xóa đi số nhập vào cuối cùng.
        PUTC    8       ; backspace again.
        JMP     next_digit ; chờ nhấn Enter hoặc phím xóa lùi.
stop_input:
        ; kiểm tra cờ:
        CMP     CS:make_minus, 0
```

```
        JE      not_minus
        NEG     CX
not_minus:
        POP     SI
        POP     AX
        POP     DX
        RET
make_minus      DB      ?       ; sử dụng biến này như 1 cờ.
ten             DW      10      ; dùng để nhân.
SCAN_NUM        ENDP
skip_proc_scan_num:
```

Dưới đây là Macro DEFINE_PRINT_NUM in ra một số nguyên nằm trong AX (các lời giải thích được viết lại bằng tiếng Việt)

```
;*************************************************************
; Trong macro này định nghĩa chương trình con DEFINE_PRINT_NUM
; để in ra một số nguyên trong AX
; gọi đến chương trình con PRINT_NUM_UNS để in ra một số có dấu
; chương trình con liên quan:
; DEFINE_PRINT_NUM  và DEFINE_PRINT_NUM_UNS !!!
DEFINE_PRINT_NUM        MACRO
; khai báo các nhãn cục bộ
LOCAL not_zero, positive, printed, skip_proc_print_num
JMP     skip_proc_print_num
PRINT_NUM       PROC    NEAR
        PUSH    DX
        PUSH    AX

        CMP     AX, 0
        JNZ     not_zero 1

        PUTC    '0'
        JMP     printed
not_zero:
        ; Kiểm tra dấu của AX,
        CMP     AX, 0
        JNS     positive
        NEG     AX
        PUTC    '-'
positive:
        CALL    PRINT_NUM_UNS
printed:
        POP     AX
```

```
        POP     DX
        RET
PRINT_NUM       ENDP
skip_proc_print_num:
DEFINE_PRINT_NUM        ENDM

;*************************************************************
```

Dưới đây là đoạn chương trình của macro DEFINE_PRINT_NUM_UNS, macro chứa một chương trình con PRINT_NUM_UNS, in ra một số nguyên không dấu.

```
; Macro này định nghĩa một thủ tục in ra màn hình một số nguyên
; không dấu trong AX
; với giá trị  từ 0 đến 65535
DEFINE_PRINT_NUM_UNS    MACRO
;khai báo các nhãn cục bộ
LOCAL begin_print, calc, skip, print_zero, end_print, ten
LOCAL skip_proc_print_num_uns
JMP     skip_proc_print_num_uns
PRINT_NUM_UNS   PROC    NEAR
; cất các giá trị thanh ghi vào ngăn xếp
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        ; Cờ cấm in số 0 trước 1 số
        MOV     CX, 1
        ; két quả của AX/ 10000 luôn nhỏ hơn 9).
        MOV     BX, 10000       ; số chia.
        ; AX =0?
        CMP     AX, 0
        JZ      print_zero
begin_print:
        ; kiểm tra số chia (nếu là 0 thì nhảy đến nhãn end_print:
        CMP     BX,0
        JZ      end_print
      ; tránh in số 0 trước số cần in
        CMP     CX, 0
```

```
        JE      calc
        ; nếu AX<BX thì kết quả của phép chia là 0:
        CMP     AX, BX
        JB      skip
calc:
        MOV     CX, 0   ; thiết lập cờ.
        MOV     DX, 0
        DIV     BX      ; AX = DX:AX / BX   (DX=số dư).
        ; in số cuối cùng
        ; AH =0, bị bỏ qua
        ADD     AL, 30h    ; chuyển sang mã ASCII
        PUTC    AL
        MOV     AX, DX  ; lấy số dư từ phép chia cuối cùng.
skip:
        ; tính BX=BX/10
        PUSH    AX
        MOV     DX, 0
        MOV     AX, BX
        DIV     CS:ten  ; AX = DX:AX / 10   (DX=số dư).
        MOV     BX, AX
        POP     AX
        JMP     begin_print
print_zero:
        PUTC    '0'
end_print:
      ; khôi phục lại giá trị thanh ghi ban đầu
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        RET
ten             DW      10      ; định nghĩa số chia.
PRINT_NUM_UNS   ENDP
skip_proc_print_num_uns:
DEFINE_PRINT_NUM_UNS    ENDM
```

```
;*************************************************************
```

3.3 CÔNG CỤ LẬP TRÌNH HỢP NGỮ TRÊN WINDOWS

Phần này giới thiệu về hợp ngữ ở mức cao (High-Level Assembly) hay HLA và các công cụ sử dụng để lập trình trên windows.

3.3.1 Công cụ hỗ trợ lập trình hợp ngữ trên windows

Dưới đây là công cụ để hỗ trợ người lập trình viết chương trình bằng hợp ngữ bậc cao (HLA) trên windows.

- Ngôn ngữ HLA
- Thư viện chuẩn HLA
- RadASM: môi trường phát triển ứng dụng cho HLA.
- Bộ công cụ gỡ rối OllyDbg (Olly Debugger)
- Chương trình xử lý đầu ra để output ra các file PE/COFF (như MASM, FASM)
- Một bộ liên kết (linker) có thể kết nối được các file output từ bộ dịch.(MS link)
- Một trình biên dịch tài nguyên (resource compiler) như rc.exe của Microsoft.
- Một số tiện ích cho phép xây dựng ứng dụng (như nmake của Microsoft hoặc make của Borland)
- Các module thư viện Win32.

3.3.2 Sử dụng công cụ phát triển RadASM

RadASM viết tắt của cụm từ Rapid Application Development Assemblers là một môi trường phát triển các ứng dụng viết bằng Hợp ngữ trên Windows 32 bít. RadASM được phát triển bởi Ketil Olsen, bao gồm các công cụ hỗ trợ các trình hợp ngữ sau:

MASM - **Hutch's** MASM32 Package

FASM - **Thomasz Grysztar's** Flat Assembler
NASM - The Netwide Assembler
TASM - **Borland's** Turbo Assembler

HLA - **Randall Hyde's** High Level Assembler

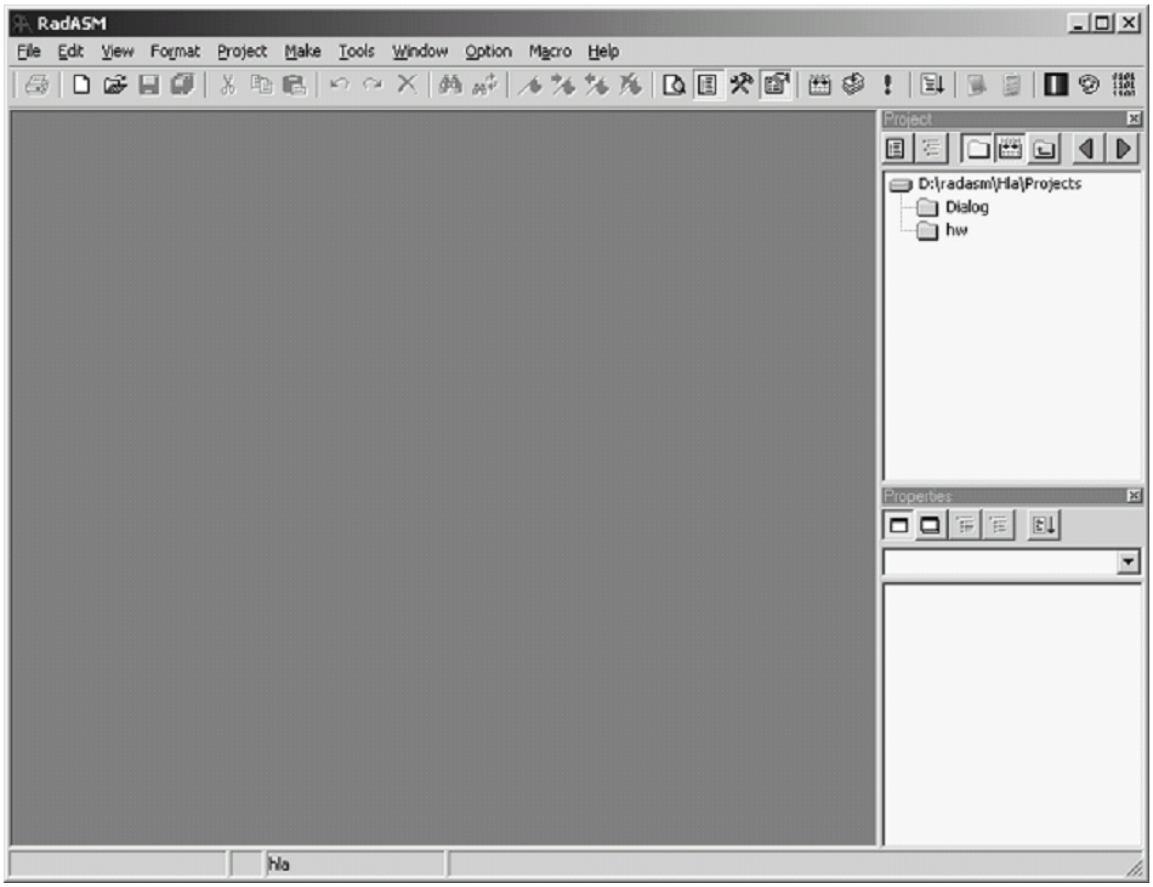
GoAsm - Jeremy Gordon's GoTools Assembler

Các ứng dụng được phát triển bằng RadASM dưới dạng các project. Một project là các files chứa các mô đun hợp ngữ để tạo ra một ứng dụng hoặc thành phần (component) có thể sử dụng được. Cụ thể, project bao gồm các file mã nguồn, các file tài nguyên và các đối tượng dialog. Mỗi project có một RAP (RadASM Project) file, chứa các thông tin liên quan đến project như các lệnh biên dịch, các danh sách file và các tùy chọn khác.

Dưới đây là hướng dẫn sử dụng dung công cụ RadASM

a. Khởi động RadASM

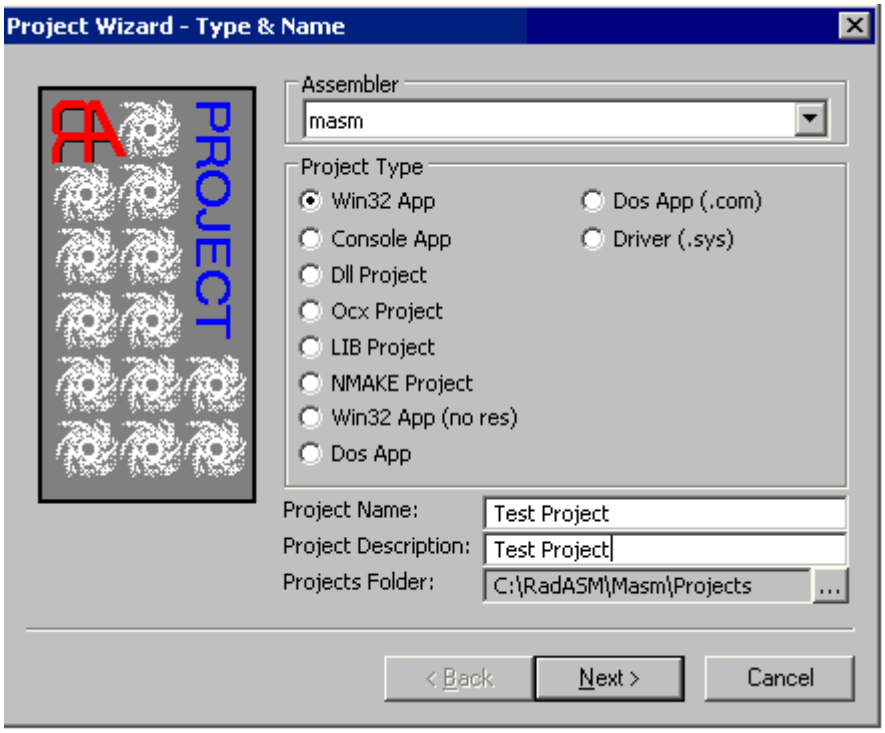
Khi khởi động, cửa sổ chính màn hình RadASM như sau:



Hình 3.3 Cửa sổ khởi động RadASM

Nó bao gồm menu chính, một cửa số dành cho các file thuộc về một dự án va cửa sổ thuộc tính của các file/ các đối tượng của dự án.

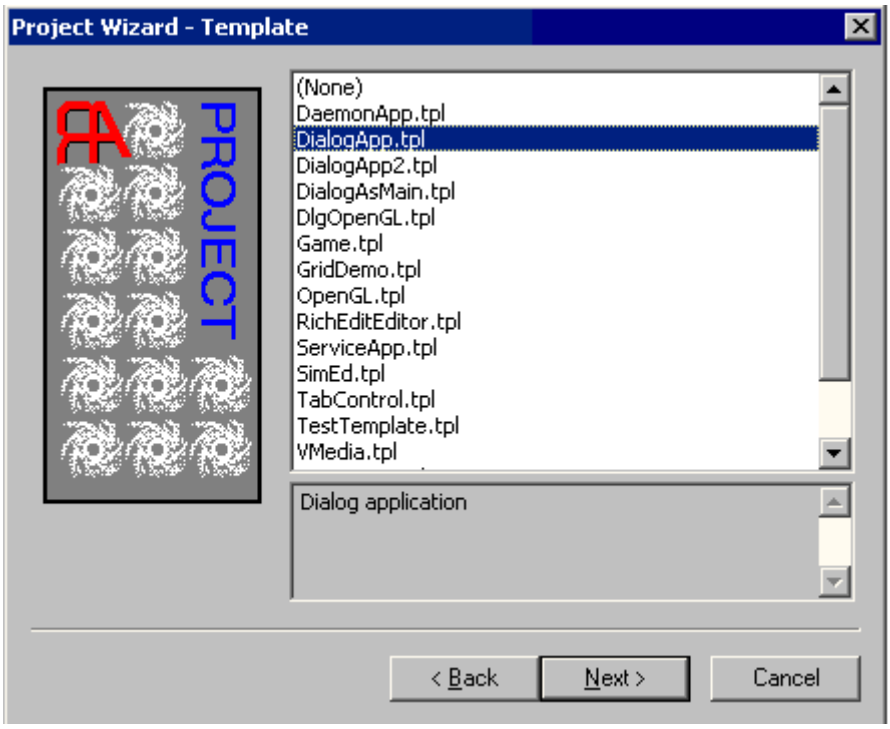
Để thực hiện tạo một Project mới, người dùng vào menu File, chọn new project. Khi đó màn hình thông tin của project mới sẽ được hiện ra như hình sau:



Hình 3.3 Cửa sổ chọn kiểu project

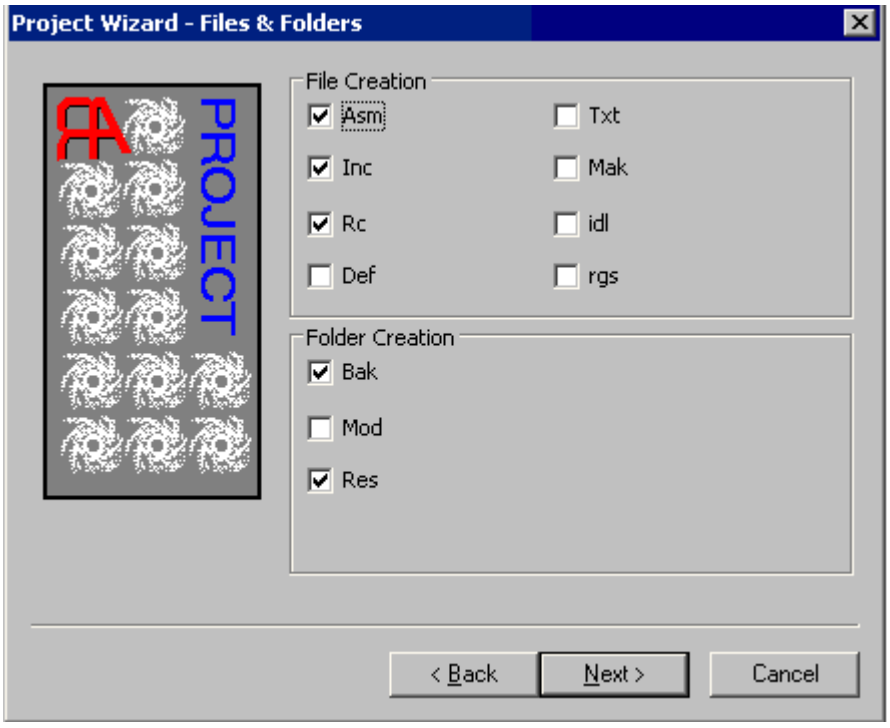
Người lập trình có thể chọn một trong số các mẫu ứng dụng của project đã có sẵn là một trong bốn loại: ứng dụng Console (input và output của chương trình đều hiển thị trên cửa số console), ứng dụng kiểu hộp thoại (chương trình [input và output] và người dùng tương tác với nhau qua các hộp thoại), ứng dụng windows thông thường, tạo các hàm để dịch ra thư viện liên kết động (DLL), tạo thư viện OCX, tạo thư viện LIB, NMAKE, ứng dụng trên Windows hay DOS.

Tiếp đến, người lập trình đưa vào tên project, các mô tả của project, thư mục chứa project và template sử dụng cho project.



Hình 3.4 Cửa sổ chọn kiểu template

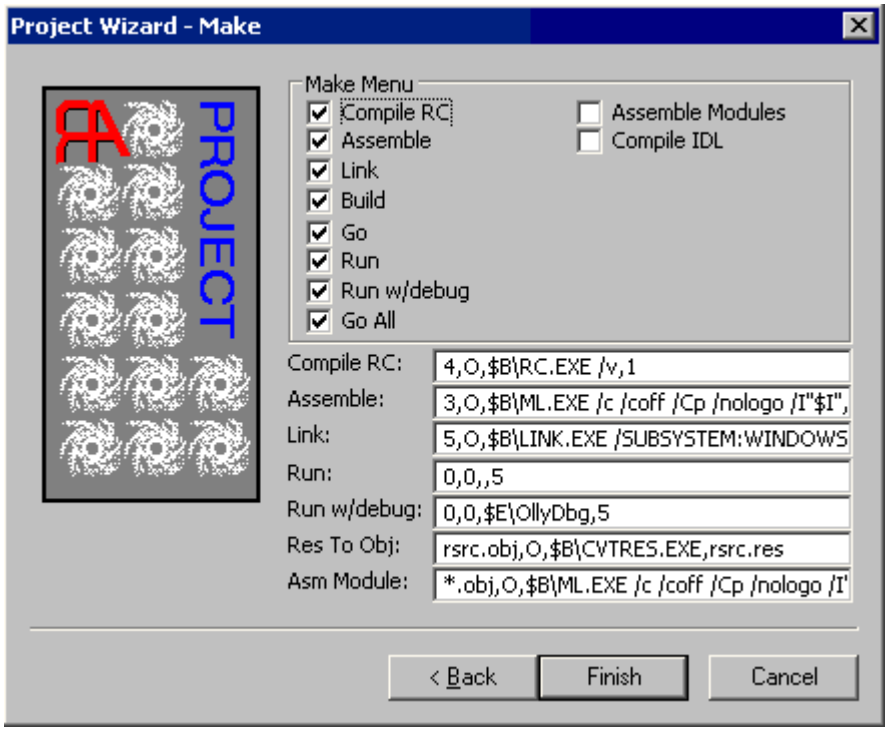
Sau khi chọn được template phù hợp, người lập trình sẽ chọn kiểu file chương trình và các thư mục cần lưu các file vào trong đó.



Hình 3.5 Cửa sổ chọn kiểu file chương trình

Trong hình 3.5 ta muốn tạo ra các file mã nguồn hợp ngữ (asm) và file tài nguyên. Ta muốn lưu chúng trong thư mục RES và lưu bản sao của chúng trong thư muc BAK.

Dịch và tạo file thực hiện chương trình như sau:



Hình 3.5 Cửa sổ dịch và tạo file thực hiện chương trình

3.4 TÓM TẮT

Chương này đã trình bày về các công cụ lập trình hỗ trợ lập trình bằng hợp ngữ. Ta đã tìm hiểu về bộ gỡ rối Debug-một công cụ quan trọng để giúp người lập trình hệ thống gỡ lỗi cho chương trình. Trong phần Debug ta đã đề cập hai vấn đề chủ yếu: cách sử dụng trình tiện ích Debug và tìm hiểu các lệnh do Debug cung cấp. Tiếp đến là tìm hiểu một chương trình mô phỏng cho bộ xử lý 8086 là Emu8086. Chương trình mô phỏng này không những cung cấp cho người học các cơ chế xử lý lệnh, các trạng thái trong quá trình thực hiện một chương trình…mà còn cung cấp cho người học lập trình hệ thống một môi trường phát triển các chương trình hợp ngữ đơn giản và chương trình mẫu theo nhiều dạng khác nhau.

Phần cuối cùng giới thiệu về một công cụ được sử dụng phổ biến trong lập trình các ứng dụng bằng hợp ngữ trên Windows, đó là RadASM. Đây là một IDE (Intergrated Development Environment) cho phép người lập trình phát triển các môdun riêng rẽ, được biên dịch và kết hợp lại với nhau thành một ứng dụng hoàn chỉnh.

3.6 BÀI TẬP

3.6.1 Câu hỏi trắc nghiệm

1. Để so sánh nội dung của hai vùng nhớ, ta dùng lệnh nào dưới đây của Debug

A. Lệnh A

B. Lệnh C

C. Lệnh D

D. Lệnh E

2. Để cho thực hiện một chương trình , ta dùng lệnh nào dưới đây của Debug

A. Lệnh G

B.Lệnh L

C. Lệnh T

D. Lệnh P

3. Phát biểu nào dưới đây đúng và đầy đủ nhất về chương trình mô phỏng Emu8086.

A. Là một hệ soạn thảo chương trình

B. Hỗ trợ cho người lập trình viết các chương trình con và Macro

C. Mô phỏng quá trình thực hiện chương trình

D. Cung cấp môi trường cho người lập trình viết các chương trình hợp ngữ và Mô phỏng quá trình thực hiện chương trình

4. Phát biểu nào dưới đây đúng và đầy đủ nhất về công cụ RadASM.

A. Là một chương trình dịch hợp ngữ

B. Là một công cụ cho phép chạy các chương trình hợp ngữ trên windows

C. Là một công cụ cho phép chạy các chương trình hợp ngữ trên hệ điều hành DOS.

D. Cung cấp môi trường cho phép người lập trình viết các ứng dụng hợp ngữ dưới dạng các project trong Windows.

5. Phát biểu nào dưới đây đúng và đầy đủ nhất về một project được phát triển trên công cụ RadASM.

A. Có thể bao gồm nhiều file như các file nguồn, file tài nguyên, dialogs…

B. Chỉ gồm 1 file có phần mở rộng là ASM

C. Sau khi dịch thì file project phải là dạng .EXE

D. Sau khi dịch thì file project phải là dạng .COM

3.7 TÀI LIỆU THAM KHẢO

1. Đặng Thành Phu. Turbo Assembler và Ứng dụng. Nhà XB Khoa học và Kỹ thuật. 1998.
2. Nguyễn Minh San. Cẩm nang Lập trình Hệ thống (tập 2). Bản dịch. Nhà XB Tài chính Thống Kê. 1996.
3. Nguyễn Đình Việt. Giáo Trình nhập môn Hợp ngữ và Lập trình Hệ thống. Hà nội. 1998.
4. Website: www.emu8086.com

5. Website: http://members.a1.net/ranmasaotome/main.html

6. Website: http://www.radasm.com/

7. Randall Hyde. Entire Windows Assembly Programming. University California Riverside, USA. 2003.

CHƯƠNG 4: LẬP TRÌNH PHỐI GHÉP VỚI THIẾT BỊ NGOẠI VI

Trong chương này, chúng ta sẽ tìm hiểu về lập trình phối ghép và điều khiển một số thiết bị ngoại vi điển hình dựa trên các dịch vụ cung cấp bởi BIOS (Base Input Output System) và ngắt của DOS phục vụ cho các thiết bị đó. Chương này trình bày các vấn đề về lập trình phối ghép với Modem, bàn phím, màn hình, và ổ đĩa.

4.1 LẬP TRÌNH PHỐI GHÉP VỚI MODEM

Việc truyền thông tin giữa các thành phần nằm gần nhau như các bộ phận của một hệ vi xử lý đĩa cứng, màn hình, CPU…có thể thực hiện thông qua bus song song mở rộng hoặc qua các mạch phối ghép song song. Trong đó một nhóm bít (8, 16 , 32 hoặc 64) được truyền từ bộ phận này sang bộ phận khác trên 1 tập các đường truyền tín hiệu cáp. Ngược lại, trong trường hợp phải truyền tín hiệu giữa các đối tượng ở xa nhau thì không thể dùng nhiều dây tín hiệu đồng thời vì lý do kinh tế. Phương thức truyền tin nối tiếp phù hợp với yêu cầu thực tế này. Trong phương thức truyền tin nối tiếp thì ở đầu phát, dữ liệu dưới dạng song song được chuyển thành dữ liệu dạng nối tiếp, tín hiệu nối tiếp sau đó được truyền đi liên tiếp theo từng bit trên một đường dây đến đầu thu. Tại đầu thu, tín hiệu nối tiếp sẽ được biến đổi ngược lại thành dạng song song để truyền cho các thành phần đứng gần nhau.

Thành phần chính của Modem (Bộ điều tần và giải điều tần) là một mạch thu phát dị bộ vạn năng (UART) 8250A. Chính vì vậy, các phần sau ta sẽ tìm hiểu về các vấn đề liên quan đến lập trình bằng Hợp ngữ cho vi mạch này.

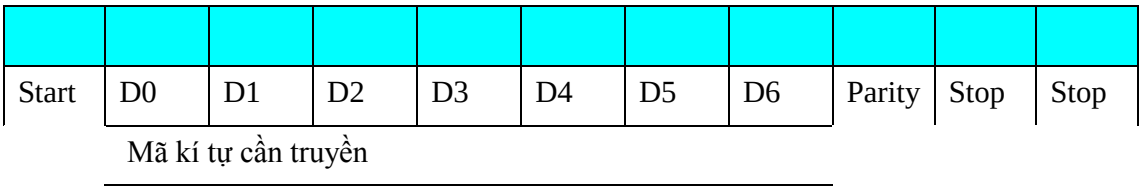
4.1.1 Cơ bản về truyền tin nối tiếp

Có hai kiểu truyền thông nối tiếp: đồng bộ và dị bộ. Trong phương thức đồng bộ thì dữ liệu được truyền đi theo từng bản tin (một đoạn văn bản) với một tốc độ xác định. Túy theo giao thức truyền thông mà mỗi bản tin truyền đi có các cấu trúc khác nhau. Chẳng hạn, nếu dùng giao thức truyền thông tin hệ 2 đồng bộ (BISYNC: Binary Synchronous Communication Protocol) thì cấu trúc một bản tin như sau:

SYN	SYN	SOH	HEADER	STX	TEXT	ETX	BCC
Kí tự đồng bộ	Kí tự đồng bộ	Bắt đầu phần header	phần header	Bắt đầu phần nội dung	Phần nội dung bản tin	Kết thúc phần nội dung	Kí tự kiểm tra khối

Ngược lại, trong phương thức truyền không đồng bộ, dữ liệu được truyền đi theo từng kí tự. Kí tự cần truyền đi được gắn thêm bit đánh dấu ở đầu để đánh dấu bắt đầu kí tự. 1 hoặc 2 bít

cuối cùng để đánh dấu kết thúc kí tự. Vì mỗi kí tự được nhận dạng riêng biệt nên nó có thể truyền đi bất cứ khi nào. Dưới đây là cấu trúc của một khung truyền (frame) theo phương pháp không đồng bộ.



Tùy theo loại mã ta sử dụng để mã hóa kí tự (baudot, ASCII, EBCDIC) mà độ dài cho mã kí tự có thể là 5,6,7,8 bit. Tùy theo hệ thông struyền tin, bên cạch các bít mã dữ liệu còn có thể có bit parity dùng để kiểm tra lỗi khi truyền.

Việc truyền tin dị bộ được thực hiện nhờ một UART (Universal Asynchronous Receiver Transmitter) ở đầu phát và một UART ở đầu thu. Khi có kí tự phát, mạch 8251A sẽ tạo ra khung cho kí tự bằng cách thêm vào các bit start, parity và stop, rồi gửi liên tiếp từng bít ra đường truyền. Bên phía thu, một mạch 8251A khác sẽ nhận kí tự, tháo bỏ khung truyền, kiểm tra parity, rồi chuyển sang dạng song song để CPU đọc.

4.1.2 Các thanh ghi của UART 8250A/16450

Mạch 8250A là một mạch thu phát dị bộ vạn năng (UART) được sử dụng rất phổ biến để phối ghép với cổng thông tin nối tiếp như cổng COM theo chuẩn RS 232C. Dưới đây ta tìm hiểu về các thanh ghi bên trong của UART 8250A.

Dưới đây là bảng liệt kê tổ hợp chân tín hiệu có thể chọn ra các thanh ghi bên trong của UART 8250A.

DLA	A2	A1	A0	Chọn ra thanh ghi
0	0	0	0	Thanh ghi đệm thu (RBR), thanh ghi giữ phát (THR)
0	0	0	1	Thanh ghi cho phép tạo yêu cầu ngắt (IER)
1	0	0	0	Thanh ghi cho số chia phần thấp (LSB)
1	0	0	1	Thanh ghi cho số chia phần cao (MSB)
X	0	1	0	Thanh ghi nhận dạng nguồn gốc yêu cầu ngắt (HR)
X	0	1	1	Thanh ghi điều khiển đường truyền (LCR)
X	1	0	0	Thanh ghi điều khiển modem (MCR)
X	1	0	1	Thanh ghi trạng thái đường dây (LSR)
X	1	1	0	Thanh ghi trạng thái modem (MSR)
X	1	1	1	Thanh ghi nháp

Thanh ghi điều khiển đường truyền (Line Control Register)

Thanh ghi này còn có tên khác là thanh ghi định khuôn dạng dữ liệu vì nó quyết định khuôn dạng dữ liệu trên đường truyền. Cấu trúc của thanh ghi LCR được biểu diên như sau:

D7	D6	D5	D4	D3	D2	D1	D0
DLAB	SBCB	SP	EPS	PEN	STB	WLS1	WLS1

- ***Bít D7 (DLAB)***: Bit truy nhập số chia
  1: truy nhập số chia
  0: truy nhập IER,THR và RBR

- ***Bít D6 (SBCB)***: Bit điều khiển gián đoạn
  1: buộc S_{out} =0
  0: không hoạt động
- ***Bít D5 (SP)***: Đảo parity
  1: parity chẵn
  0: không hoạt động
- ***Bít D4 (ESP)***: chọn tạo/kiểm tra parity chẵn
  1: parity chẵn
  0: parity lẻ
- ***Bít D3 (PEN)***: cho phép tạo/kiểm tra parity
  1: cho phép
  0: cấm
- ***Bít D2 (STB)***: số bit stop
  1: 1,5 hoặc 2 bít
  0: 1bít
- ***Bít D1, D0 (WLS1, WLS0)***: Chọn độ dài từ
  00: 5 bít
  01: 6 bít
  10: 7 bít
  11: 8 bít

Thanh ghi đệm giữ phát (Transmitter Holding Register- THR)

Ký tự cần phát đi phải được ghi từ CPU vào thanh ghi này trong khi bit DLAB=0. Sau đó khi truyền 8250A lấy kí tự từ đây, đóng khung cho nó như đã định và đưa từng bít ra chân S_{out}

Thanh ghi đệm thu (Receiver Buffer Register- RBR)

Khi 8250A nhận được một ký tự qua chân S_{in} , nó tháo bỏ khung truyền cho kí tự và giữa kí tự tại thanh ghi đệm thu để chờ CPU đọc vào. CPU chỉ đọc được kí tự trong thanh ghi này khi bít DLAB=0.

Thanh ghi cho phép tạo yêu cầu ngắt (Interrupt Enable Register- IER)

Thanh ghi này dùng để cho phép/ cấm các nguyên nhân gây ra ngắt rkhác nhau. Trong khi mạch 8250A hoạt động có thể tác động đượctới CPU thông qua chân INTRPT của UART. Mỗi bít trong các bít D3,D2,D1,D0 ở mức cao cho phép các hiện tượng tương ứng với bít đó được đưa ra yêu cầu ngắt đối với CPU.

Dưới đây là cấu trúc của thanh ghi cho phép tạo yêu cầu ngắt.

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	MODEM	RLINE	TxEMPTY	RxRDY

- ***Bít D3 (MODEM)*** = 1: cho phép các thay đổi trạng thái của modem gây ra ngắt.
- ***Bít D2 (RLINE)*** = 1: cho phép các tín hiệu trạng thái đường truyền thu gây ra ngắt.
- ***Bít D1 (TxEMPTY)*** = 1: cho phép gây ra ngắt khi đệm giữ phát rỗng.
- ***Bít D0 (RxRDY)*** = 1: ho phép gây ra ngắt khi đệm thu đầy..

Thanh ghi nhận dạng nguồn yêu cầu ngắt (Interrupt Identification Register- IIR)

Thanh ghi nhận dạng ngắt (chỉ để đọc ra) chứa mã mức ưu tiên cao nhất của yêu cầu ngắt tại chân INTRPT của 8250A đang chờ phục vụ. Do vậy khi cần xử lý các yêu cầu ngắt theo kiểu thăm dò, CPU cần đọc bít ID_0 của thanh ghi này để biết là có yêu cầu ngắt và kiểm tra các bít ID_2- ID_1 để xác định được nguồn gốc của yêu cầu ngắt.

Mỗi lần UART được reset thì chỉ có yêu cầu ngắt ở mức ưu tiên số 1 sẽ được phục vụ. Điều này có thể thay đổi bằng cách dùng mặt nạ che đi các yêu cầu ngắt nào đó bằng cách ghi vào thanh ghi IER các bít thích hợp.

Dưới đây là cấu trúc của thanh ghi nhận dạng nguồn yêu cầu ngắt.

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	ID_2	ID_1	ID_0

- ***Bít D2, D1 (ID_2, ID_1)***: mã hóa các yêu cầu ngắt có mức ưu tiên cao nhất đang chờ được phục vụ.
- ***Bít D0 (ID_0)*** = 0: có yêu cầu ngắt
  = 1: không có.

Thanh ghi điều khiển modem (Modem Control Register- MCR)

Đây là thanh ghi điều khiển các tín hiệu ra của modem thông qua việc điều khiển các tín hiệu tại chân DTR và RTS của mạnh UART.

Cấu trúc của thanh ghi điều khiển modem như sau:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	LOOP	OUT2	OUT1	RTS	DTR

- ***Bít D4 (LOOP)*** = 1: Nối vòng cục bộ
  = 0: bình thường.
- ***Bít D3, D2 (OUT2, OUT1)*** = 1: đưa OUTi=1
  = 0: đưa OUTi=0.
- ***Bít D1 (RTS)*** = 1: đưa RTS=1
  = 0: đưa RTS=0.
- ***Bít D0 (DTR)*** = 1: đưa RTS=1
  = 0: đưa RTS=0.

Khi thiết lập D0=DTR=1 ta có thể điều khiển tín hiệu tại chân DTR của mạch 8250A đạt mức tích cực thấp để báo UART sẵn sàng làm việc.

Tương tự, D1=RTS=1 ta có thể điều khiển tín hiệu tại chân RTS của mạch 8250A đạt mức tích cực thấp để báo UART sẵn sàng truyền phát ký tự.

Ngoài ra, ta có thể điều khiển được các đầu ra phụ OUT1 và OUT2. Bằng các bít D2=OUT1=1 và D3=OUT2=1 để điều khiển tín hiệu tại các chân này sao cho OUT1=1 và OUT2=1.

Bít D4=1 cho phép điều khiển mạch 8250A làm việc ở chế độ nối vòng cục bộ để kiểm tra chức năng của UART.

Khi D4=1 thì cấu hình sau được thiết lập:

S_{out} =1

S_{in} =1: không nối với bên ngoài

Các thanh ghi dịch của phần phát với phần thu được nối vòng với nhau

Các chân điều khiển vào của modem không được nối ra ngoài mà được nối ở bên trong mạch với các chân điều khiển ra của modem..

Thanh ghi trạng thái modem (Modem Status Register- MSR)

Thanh ghi này còn được gọi là thanh ghi trạng thái vào từ cổng nối tiếp RS232C vì nó cho biết trạng thái hiện thời của các tín hiệu điều khiển modem từ đường truyền.

Dưới đây là cấu trúc của thanh ghi trạng thái modem:

D7	D6	D5	D4	D3	D2	D1	D0
RLSD	R1	DSR	CTS	RLSD*	RI*	DSR*	CTS*

Bít D7,D6,D5, D4 (RLSD,RI,DSR,CTS): Có giá trị các bít OUT2, OUT1, DTR, RTS trong MCR khi bít LOOP=1.

- ***Bít D3 (RLSD)*** = 1: nếu có sự thay đổi của các tín hiệu tương ứng so với lần đọc trước.
- ***Bít D2 (RI)*** = 1: Nếu RI có biến đổi từ mức thấp lên mức cao

Dấu * đứng trước các chân tín hiệu để chỉ ra rằng trong khi 8250A hoạt động, nếu có sự thay đổi của các tín hiệu đó thì các bít tương ứng sẽ được lập. Với tín hiệu RI thì đó là sự biến đổi từ mức thấp lên mức cao.

Thanh ghi trạng thái đường truyền (Line Status Register- LSR)

Thanh ghi trạng thái đường truyền cho ta biết được trạng thái của việc truyền tín hiệu trên đường truyền.

Dưới đây là cấu trúc của thanh ghi trạng thái đường truyền:

D7	D6	D5	D4	D3	D2	D1	D0
0	TSRE	THRE	BI	FE	PE	OE	RxDR

- ***Bít D6 (TSRE)***: thanh ghi dịch phát rỗng.

=1: khi 1 ký tự được phát đi, bít này bị xóa khi có một kí tự chuyển từ THR sang TSR

- ***Bít D5 (THRE)***: thanh ghi giữ phát rỗng.

=1: khi ký tự đã được chuyển từ THR sang TSR, bít này bị xóa khi CPU đưa kí tự tới thanh ghi THR.

- ***Bít D4 (BI)***: ngắt gây ra sự gián đoạn khi truyền.

=1: khi tín hiệu đầu vào phần thu ở mức thấp lâu hơn thời gian dành cho một ký tự, bit này bị xóa khi CPU đọc thanh ghi LSR.

- ***Bít D3 (FE)***: lỗi khung truyền.

=1: báo có lỗi về khung truyền, bit này bị xóa khi CPU đọc thanh ghi LSR.

- ***Bít D2 (PE)***: lỗi parity.

=1: báo có lỗi parity, bit này bị xóa khi CPU đọc thanh ghi LSR.

- ***Bít D1 (OR)***: lỗi do nhận tín dữ liệu bị ghi đè.

=1: có hiện tượng ghi đè do CPU chưa kịp đọc thanh ghi đệm thu, bit này bị xóa khi CPU đọc thanh ghi LSR.

- ***Bít D0 (RxDR)***: sẵn sàng nhận dữ liệu.

=1: đã nhận được một kí tự và đăth nó trong thanh ghi đệm thu (RBR), bit này bị xóa khi CPU đọc thanh ghi RBR.

4.1.3 Dịch vụ của BIOS phục vụ Modem

Như ta đã biết truyền thông nối tiếp giữa hai máy tính được tực hiện qua cổng COM và theo chuẩn RS 232C. Phần này sẽ giới thiệu về lập trình truyền thông qua các cổng COM và ngắt BIOS phục vụ vào/ra của các cổng COM.

a. *Địa chỉ các cổng COM và ngắt 14h-dịch vụ BIOS cho cổng COM*

Bảng địa chỉ cổng COM:

Cổng	Địa chỉ I/O	Yêu cầu ngắt
COM1	3F8-3FF	IRQ 4
COM2	2F8-2FF	IRQ 3
COM3	338-33F	IRQ 5
COM4	238-23F	IRQ 5

Các cổng COM3 và COM4 có thể không cần sử dụng mức ngắt như trên. Chúng có thể sử dụng các mức ngắt IRQ2, IRQ5, IRQ7 hoặc IRQ9.

Dưới đây ta sẽ tìm hiểu về các chức năng của ngắt 14h –dịch vụ BIOS dành cho các cổng COM.

Hàm 0h:

Ý nghĩa: Khởi tạo cổng COM.

Đầu vào: AH=0

DX=số hiệu cổng COM (0-3)

AL= tham số khởi tạo trong đó định dạng của AL như sau:

D7	D6	D5	D4	D3	D2	D1	D0
Tôc độ truyền (bits/giây) 000: 110 bits/ giây 001: 150 bits/ giây 010: 300 bits/ giây 011: 600 bits/ giây 100: 1200 bits/ giây 101: 2400 bits/ giây			*Bit chẵn /lẻ* x0: không chẵn lẻ 01: lẻ 11: chẵn		*Độ dài bit stop* 0 : 1 bít 1 : 2 bít	*Độ dài mã kí tự:* 00: 5 kí tự 01: 6 kí tự 10: 7 kí tự 11: 8 kí tự	

110: 4800 bits/ giây 111: 9600 bits/ giây			

Int 14h

Đầu ra: AH = trạng thái của cổng vừa khởi tạo.

Hàm 1h:

Ý nghĩa: Ghi một kí tự ra cổng COM

Đầu vào: AH=1

DX=số hiệu cổng COM (0-3)

AL= Mã ASCII của kí tự

Int 14h

Đầu ra: AH = trạng thái lỗi của cổng. Bít 7=1: có lỗi, bít 7=0 không có lỗi.

Hàm 2h:

Ý nghĩa: Nhận một kí tự vào từ cổng COM

Đầu vào: AH=2

DX=số hiệu cổng COM (0-3)

Đầu ra: AL= Mã ASCII của kí tự nhận được. AH = trạng thái lỗi của cổng. Bít 7=1: có lỗi, bít 7=0 không có lỗi.

Hàm 3h:

Ý nghĩa: Đọc trạng thái cổng COM

Đầu vào: AH=3

DX=số hiệu cổng COM (0-3)

Đầu ra: AH = trạng thái cổng COM và AL-= trạng thái modem. Chi tiết như sau.

Dạng thức của AH:

Bit 7	Lỗi qua thời gian (timeout)
Bit 6	Thanh ghi phát rỗng
Bit 5	Thanh ghi nhận rỗng
Bit 4	Cho phép ngắt
Bit 3	Lỗi khung truyền
Bit 2	Lỗi parity
Bit 1	Lỗi đường truyền

Bit 0	Dữ liệu đã có trong bộ đệm

Dạng thức của AL:

Bit 7	Phát hiện vật mang dữ liệu (Data Carrier Detect)
Bit 6	Chỉ thị chuông (Ringing Indicator)
Bit 5	Sẵn sàng thiết lập dữ liệu (Data Set Ready)
Bit 4	Xóa gửi (Clear To Send)
Bit 3	Phát hiện sóng mang dữ liệu Delta(Delta Data Carrier Detect)
Bit 2	Chỉ thị chuông ở đuôi (Tailing Edge Ring Indicator)
Bit 1	Sẵn sàng thiết lập dữ liệu Delta (Delta Data Set Ready)
Bit 0	Xóa gửi Delta (Delta Data Set Ready)

4.1.4 Ví dụ về lập trình cho Modem

Ví dụ 1: Khởi tạo chế độ làm việc cho cổng COM3 với các thông số: 6 bít mã kí tự truyền , tốc độ truyền 2400 bits/ giây, parity chẵn, một bít stop.

Dạng thức của AL

D7	D6	D5	D4	D3	D2	D1	D0
101: 2400 bits/ giây			*Bit chẵn /lẻ* 11: chẵn		*Độ dài bit stop* 0: 1 bít	*Độ dài mã kí tự:* 01: 6 kí tự	

AL=1011 1001=B9h

Đoạn mã chương trình sẽ được viết như sau:

```
Mov AH,0 ; khởi tạo
Mov DX,2 ; cổng COM3 có số hiệu 2
Mov AL,B9h
Int 14h
```

Ví dụ 2: Khởi tạo chế độ làm việc cho cổng COM1 với các thông số: 7 bít mã kí tự truyền , tốc độ truyền 4800 bits/ giây, parity chẵn, một bít stop không điều khiển gán đoạn ở S_{out}.

Địa chỉ cổng của thanh ghi điều khiển đường truyền là 3FB (tính từ địa chỉ cơ sở 3F8)

Giá trị của của thanh ghi điều khiển đường truyền là:

D7	D6	D5	D4	D3	D2	D1	D0

0	0	0	1	1	0	1	0

Giá trị này bằng 1AH

Địa chỉ cổng của thanh ghi số chia LSB là 3F8 với DLAB=1

Địa chỉ cổng của thanh ghi số chia MSB là 3F9 với DLAB=1

Giả thiết dùng xung đồng hồ tần số 1,8432 Mhz ở đầu vào của UART và ta muốn có tốc độ truyền 4.800 bits/giây, ta cần tính số chia để ghi giá trị số chia vào thanh ghi số chia

Số chia = 1.843200/(4.800x16)=24

Đoạn mã chương trình sẽ được viết như sau:

```
Mov AL,80 ; đưa vào LCR để tạo ra DLAB=1
Mov DX,3FBH ; địa chỉ LCR
Out DX,AL
Mov AL,24 ; đưa vào LSB của số chia
Out DX,AL
Mov AL,0 ; đưa vào MSB của số chia
Mov DX,3F9H ; địa chỉ MSB của số chia
Out DX,AL
Mov AL,1AH ; qui định khuôn dạng dữ liệu
Mov DX,3FBH ; địa chỉ LCR
Out DX,AL
…
```

4.2 LẬP TRÌNH PHỐI GHÉP VỚI BÀN PHÍM

Phần này giới thiệu về một số lập trình phối ghép cơ bản với bàn phím, màn hình. Chúng ta sẽ sử dụng những chức năng và dịch vụ do BIOS và DOS cung cấp để lập trình với các thiết bị này. Chi tiết về các chức năng đã được trình bày ở phần 3.4.1.

4.2.1 Cơ bản về bàn phím

a. Cơ bản về bàn phím

Trong số các thiết bị thu nhận tín hiệu: bàn phím, chuột, cần điều khiển, bút quang, màn hình cảm biến, bảng vẽ vector, thiết bị quét, máy ảnh số, thiết bị nhận dạng ảnh, tiếng nói… thì bàn phím là một trong những thiết bị vào thông dụng nhất. Bàn phím bao gồm một tập hợp các phím mà mỗi phím hoạt động như một công tắc cảm biến lực chuyển lực nhấn thành một đại lượng điện. Bộ vi điều khiển 8042 của máy tính và bộ vi điều khiển của bàn phím liên lạc với nhau tuần tự và đồng bộ qua hai đầu dây dẫn: dây dữ liệu và dây đồng hồ. hai vi điều khiển này làm việc việc với nhau theo nguyên tắc chủ/tớ (master/slaver) trong đó 8042 là chu còn 8048 là tớ.

Khi có một thao tác phím, bàn phím phát ra một tín hiệu điện (IRQ1) gửi đến CPU. CPU sẽ tạm dừng công việc và gọi ngắt Int 9 đọc cổng bàn phím (địa chỉ 60H) để nhận mã Scan của tác động phím vừa xảy ra. Ngoài ra, nó sẽ đọc cờ bàn phím đó là một word ở địa chỉ 40:0017 để kết

hợp với mã scan sinh ra mã ASCII tương ứng với tác động phím ở trên. Cặp hai byte gồm mã Scan và mã ASCII sẽ được Int 9h đưa vào vùng đệm bàn phím theo thứ tự mã ASCII đứng trước mã Scan. Chúng nằm ở đó cho tới khi được ngắt 16h phục vụ.

Trong trường hợp ngắt 9h phát hiện thấy từ cổng 60H các mã Scan ứng với tổ hợp phím đặc biệt, ví dụ Ctrl_Alt_Del, PrintScreen, Ctrl_Numlock, Ctrl_Break, SysReq thì nó sẽ hiểu đây là các lệnh khiến ROM-BIOS phải thực hiện ngay chứ không lưu lại các tổ hợp phím này trong bộ đệm bàn phím .

- Khi gặp Cltr_Alt_Del, ROM BIOS gọi ngắt 19h (Boot strap loader)
- Khi gặp PrtScr, ROM BIOS gọi ngắt 5h (hard copy)
- Khi gặp Cltr_Numlock, ROM BIOS thi hành một vòng lặp vô hạn cho đến khi một phím khác được nhấn.
- Khi gặp Cltr_Break, ROM BIOS gọi ngắt 1Bh – kết thúc chương trình đang thực hiện, trả điều khiển lại cho DOS.
- Khi gặp SysReq, ROM BIOS gọi ngắt 15h với giá trị 8500H trong AX, khi nhả phím này int 15h cũng được gọi nhưng với AX=8501. Dịch vụ 85H của ngắt 15H chỉ chứa một lệnh IRET, không gây tác động gì.

b. Bộ đệm bàn phím và các thao tác

Mỗi phím trên bàn phím được bộ xử lý của bàn phím gán cho một mã Scan đặc trưng cho vị trí của phím trên bàn phím và cho trạng thái của phím (nhấn, không nhấn, nhấn chưa nhả..). Bộ đệm bàn phím (Keyboard buffer) là một miền nhớ 16 word trong RAM. Thuộc vùng dữ liệu của BIOS. Dưới đây là một phần vùng dữ liệu của BIOS (DTA) liên quan đến bộ đệm và trạng thái bàn phím:

Địa chỉ	Nội dung
40:0000-40:0006	Địa chỉ các bộ phối ghép RS232 (1-4)
40:0008-40:000F	Địa chỉ các bộ phối ghép máy in (1-4)
40:0010	Cờ thiết bị (do ngắt 11h trả lại)
40:0012	Dấu hiệu kiểm tra của nhà sản xuất
40:0013	Dung lượng nhớ tính theo đơn vị KB
40:0015	Bộ nhớ của kệnh vào/ra
40:0017	Cờ bàn phím
40:0019	Các số vào bằng phím alt
40:001A	Vị trí Head của vùng đệm
40:001C	Vị trí Tail của vùng đệm
40:001E-40:003D	Bộ đệm bàn phím

Vùng đệm bàn phím được tổ chức theo một hàng đợi quay vòng (circular queue) trong đó thao tác đọc và ghi vào bộ đệm là độc lập với nhau, điều này làm cho việc ghi tác động phím vào

và đọc ra theo kiểu mã phím (ASCII+Scan) nào được đưa vào bộ đệm trước thì sẽ được lấy ra trước.

Con trỏ Head lưu trữ địa chỉ dành cho thao tác đọc, đó là vị trí sẽ đọc ký tự tiếp theo ra khỏi bộ đệm bàn phím. Sau mỗi thao tác đọc Head được tăng lên 1 word. Word tại địa chỉ 40:001A trong vùng DTA chứa địa chỉ của Head.

Con trỏ Tail lưu trữ địa chỉ sẽ ghi tác động phím tiếp theo vào bộ đệm bàn phím. Sau mỗi thao tác ghi Tail được tăng lên 1. Word tại địa chỉ 40:001C chứa địa chỉ của Tail.

Khi cả Head và Tail cùng trỏ tới word cuối cùng của bộ đệm và nếu có một tác động phím nữa xảy ra thì cả Head va Tail cùng trỏ tới đầu vùng đệm (40:001E)

Cứ mỗi phím được đọc ra thì Head lại tiến gần đến Tail, khi tất cả vùng đệm đã được đọc hết thì Head sẽ đuổi kip Tail và cả hai cùng trỏ tới cùng một địa chỉ, lúc đó bộ đệm là rỗng.

Khi có lời gọi ngắt int 9 thì chương trình của ta chiếm quyền điều khiển, nó sẽ gọi đến chương trình xử lý ngắt bàn phím cũ để nhận tác động phím và đặt cặp byte mã ASCII và Scan vào bộ đệm bàn phím. Đồng thời gán cho DS địa chỉ đoạn của vùng dữ liệu BIOS và kiểm tra xem Tail có trong bộ đệm bàn phím không, word nằm trước Tail sẽ tương ứng với phím vừa mới nhận vào. Đọc byte mã Scan vào thanh ghi DH và byte mã ASCII vào DL. Sau đó, kiểm tra word trong DX có phải là hot-key hay không, nếu không phải sẽ nhảy đến kết thúc.

4.2.2 Dịch vụ của BIOS và DOS phục vụ bàn phím

- Ngắt 16h của BIOS

Hàm 0h:

Ý nghĩa: Chờ đọc một kí tự từ bàn phím (nếu có kí tự trong vùng đệm bàn phím thì sẽ nhận được ký tự đó, còn không thì chờ đến khi bàn phím được nhấn.

Đầu vào: AH=0

Int 16h

Đầu ra: Nếu AL<>0 thì

AL chứa mã ASCII của ký tự

AH chứa mã SCAN của ký tự

Nếu AL= 0thì

AL chứa mã bàn phím mở rộng

Hàm 1h:

Ý nghĩa: Kiểm tra xem trong vùng đệm của bàn phím có ký tự hay không (không đợi đến khi ký tự có trong vùng đệm mà trả ngay điều khiển lại cho chương trình)?.

Đầu vào: AH=01

Int 16h

Đầu ra: Nếu ZF=1 không có ký tự trong vùng đệm bàn phím

Nếu ZF=0 thì:

Nếu AL<>0 thì:

AL chứa mã ASCII của ký tự

AH chứa mã SCAN của ký tự

Nếu AL= 0 thì:

AL chứa mã bàn phím mở rộng

Hàm 02h:

Ý nghĩa: Kiểm tra trạng thái một số phím đặc biệt của bàn phím (Insert, Caplock, NumLock, Scroll Lock).

Đầu vào: AH=02

Int 16h

Đầu ra:

AL chứa kết quả các trạng thái hay cờ bàn phím , có ý nghĩa như sau:

7	6	5	4	3	2	1	0
1: chế độ Insert	1: chế độ Cap Lock	1: Num Lock bị ấn	1: Scroll Lock bị ấn	1: Alt bị ấn	1: Ctrl bị ấn	1: Shift trái bị ấn	1: Shift phải bị ấn

- Một số hàm phục vụ bàn phím của ngắt 21h của DOS

Hàm 06h:

Ý nghĩa: Đọc một kí tự từ bàn phím hoặc đưa kí tự ra màn hình. Nếu đọc vào một kí tự thì

Đầu vào: AH=6

Int 21h

DL=0FFh (nếu DL<>0FFh sẽ đưa ra màn hình)

Đầu ra: Nếu ZF=0 thì có kí tự trong vùng đệm bàn phím và:

AL chứa mã ASCII của ký tự

AH chứa mã SCAN của ký tự

Nếu ZF= 1 thì

Vùng đệm bàn phím rỗng

Hàm 07h:

Ý nghĩa: Chờ đọc một kí tự từ bàn phím

Đầu vào: AH=07

Int 21h

Đầu ra: AL chứa mã ASCII của ký tự (AL=0 sẽ không có ký tự nào)

AH chứa mã SCAN của ký tự

Hàm 0Bh:

Ý nghĩa: Đọc trạng thái bộ đệm bàn phím

Đầu vào: AH=0B

Int 21h

Đầu ra: AL =0FFh có kí tự trong bộ đệm

AL =00h không có kí tự trong bộ đệm

Hàm 0Ch:

Ý nghĩa: xóa bộ đệm bàn phím, sau đó gọi hàm vào kí tự có số chức năng đặt trong AL

Đầu vào: AL =số hàm của kí tự.

4.2.3 Phương pháp lập trình bàn phím và một số chương trình ví dụ

Có hai cách lập trình bàn phím:

- Lập trình hệ thống, truy nhập lập trình qua các bộ vi điều khiển 8042, 8044 qua các cổng 60H, 61H và 64H.
- Lập trình ứng dụng, dùng các hàm BIOS (int 9h, Int 16H) hoặc các hàm 0Ah, 0Bh, 0Ch của int 21h.

Cổng 60H là cổng A của 8255A và là nơi để CPU và bàn phím trao đổi thông tin bao gồm các lệnh (thường là của chương trình con ngắt) điều khiển bàn phím hoặc ký tự từ bàn phím vào. Đối với người lập trình thì cổng 60H được coi như thanh ghi mã lệnh của bàn phím.

Mã lệnh EDH là mã lệnh tắt, bật đèn

Ví dụ 0: Viết chương trình thiết lập mật khẩu là kí tự A thi khởi động máy

```
.MODEL small
.STACK 100h
.DATA
      matkhau db 'P','$'
      Saimatkhau db 'Sai mat khau ','$'
      Nhapmatkhau db 'Nhap mat khau: ','$'
      xuongdong db 13,10,'$'
.CODE
Start:
      Mov AX,@Data
      Mov DS,AX
Lap:
      Mov  AH,9
      Mov DX, offset Nhapmatkhau
 Int 21h ; in lời mời nhập xâu
 Mov AH,0  ; Nhap ki tu
     Int 16h
      Cmp AL,matkhau ; co phai Enter khong?
      JZ Done ; Neu là Enter, dung lai
      Mov  AH,9
      Mov DX, offset Saimatkhau
      Int 21h ;xuong dong va ve dau dong
      Jmp Lap
```

```
Done:
      Mov AH,4Ch ; Tro ve DOS
      Int 21h
End Start
```

Sau khi dịch và hợp dịch chương trình trên ta đặt tên chương trình vào cuoi file autoexec.bat. Khi máy khởi động thì chương trình trên sẽ được tự dộng thực hiện.

Ví dụ 1: Viết chương trình tự động bật phím CapLock (sau khi chạy chương trình này phím Caplock sẽ được bật lên).

Ta hãy xem cấu trúc của cờ bàn phím (chi tiết xem phần 3.4.1):

7	6	5	4	3	2	1	0
1: chế độ Insert	1: chế độ Cap Lock	1: Num Lock bị ấn	1: Scroll Lock bị ấn	1: Alt bị ấn	1: Ctrl bị ấn	1: Shift trái bị ấn	1: Shift phải bị ấn

Cờ bàn phím là byte ở tại địa chỉ 40:0017 trong vùng đệm bàn phím. Để đèn của phím CapLock là ON thì giá trị của cờ bàn phím được đặt bằng: 01000000 =40h. ta chỉ cần gán giá trị 40h vào byte có địa chỉ 40:0017 là xong.

```
.MODEL Tiny
.CODE
      Org 100h
      Jmp Start
Start:
      Mov AX,40h ; AX chứa địa chỉ đoạn dữ liệu DTA
      Mov DS, AX ; cho DS trỏ tới vùng DTA
      Mov BX,0017h ; DS:BX chứa địa chỉ của byte chứa cờ
      ; trạng thái bàn phím
      Mov byte PTR[BX] ,40h ; đặt cờ bàn phím bằng 40h
      Int 20h ; trở về DOS
End Start
```

Ví dụ 2: Viết chương trình tự động bật phím CapLock , NumLock, ScollLock (sau khi chạy chương trình này phím Caplock, NumLock, ScollLock sẽ được bật lên). Yêu cầu chương trình thực hiện qua các cổng 60H, 61H và 64H.

Chương trình sẽ kiểm tra trạng thái nhấn phím hay chưa.

Cấu trúc của byte điều khiển các đèn LED trên bàn phím như sau:

7	6	5	4	3	2	1	0

					Cap Lock	Num Lock	Scroll Lock

Để các đèn Caplock, NumLock, ScollLock sẽ được bật lên thì ta phải gửi giá trị 00000111 =07h ra cổng 60H.

```
.MODEL Tiny
.CODE
      Org 100h
      Jmp Start
Start:
      Mov AL,EDHh ; lệnh tắt/bật đèn
      Out 60H, AL ; đưa ra cổng bàn phím
Kiemtra:
      In AL,64h  ;Kiểm tra trạng thái bàn phím
      Test AL,02h ; Nếu bộ đệm bàn phím bằng đầy
      Jnz Kiemtra ; kiểm tra lại
      Mov Al,07 ; Nếu bộ đệm trống sẽ bật đèn
      Out 60H, AL ; đưa ra mã bật đèn ra cổng bàn phím
      Int 20h ; trở về DOS
End Start
```

Ví dụ 3: Viết chương trình cấm bàn phím hoạt động. Biết rằng lệnh khóa ADH là lệnh cấm bàn phím.

Kiểm tra trạng thái bàn phím trước khi cấm.

```
.MODEL Tiny
.CODE
      Org 100h
      Jmp Start
Start:
Kiemtra:
      In AL,64H ; đọc trạng thái
      Test AL,02h ; Nếu bộ đệm bàn phím bằng đầy
      Jnz Kiemtra ; kiểm tra lại
      Mov Al,ADh ; Nếu bộ đệm trống sẽ bật đèn
      Out 64H, AL ; đưa ra cấm bàn phím ra cổng 64h
      Int 20h ; trở về DOS
End Start
```

Ví dụ 4: Viết chương trình xử lý bàn phím đơn giản. Chương trình kiểm tra các phím chữ cái và phân biệt chữ hoa, chữ thường.

```
.MODEL small
.STACK 100h
.DATA
      Table db  16 dup(0)
```

```
            db  'qwertyuiop',0,0,0,0 ; hang tren
         db  'asdfghjkl',0,0,0,0 ; hang giua
         db  'zxcvbnm'; hang duoi
         db 16 dup(0) ; vung danh cho chu hoa
         db 'QERTYUIOP',0,0,0,0 ; chu hoa cho hang tren
         db 'ASDFGHJKL',0,0,0,0,0 ; chu hoa cho hang giua
         db 'ZXCVBNM'; chu hoa cho hang duoi
.CODE
Start:
      Mov AX,@Data
      Mov DS,AX
      Cli; xoa ngat
      Push DS ;
      Mov AX, seg TryKB ; DS:AX tro den Checkbanphim
      Mov DS,AX
      Mov DX, offset TryKB ; Checkbanphim
      Mov Al,9
      Mov Ah,25H  ; dat lai dia chi vector ngat
      Int 21h
      Pop DS
      Sti ; cho phep ngat
      ;---------------------------------
      ;---------  Chuong trinh xu ly ngat ban phim--------
      TryKB proc far
      Push AX
      Push BX
      Push CX
      Push DI
      Push ES
      ; nhan ma Scan va tra loi ban phim
      In Al,60h ; nhan ma Scan
      Mov AH,AL ; chuyen vao AH
      Push AX ; dua vao stack
      In AL,61h ; doc cong PB cua 8255A
      Or AL,10000000b ; dua bit 7 len bit 1
      Out 61h,AL ; dua ra cong PB
      ; Cho ES tro den doan du lieu
      Mov AX,40H ; dua AX den cuoi bo nho
      Mov ES,AX
      Pop AX  ; AL= ma scan
; kiem tra phim shift
 Cmp Al,42 ; co nhan phim shift ko?
     Jnz checkkey ; neu khong thi kiem tra tiep
      Mov BL,1
```

```
      Or ES:[17h],BL đat bit 1 của byte trang thai =1
      Jmp Thoat ; thoat khoi
PhimKetiep:
 Test AL,10000000b; ma nha phim
 Jnz Thoat
 Mov BL,ES:[17h]  ; neu ko doc dc trang thai phim shift
 Test BL,00000011b; nhan phim shift?
 Jz DoiMa ; neu ko doi lai ma
    Add AL,100 ;doi ra ki tu hoa(dia chi 100 byte ke tiep)
DoiMa:
 Mov BX, offset table ;
 Xlat table ; doi ma scan sang ma ASCII
 Cmp Al,0 ; tra ve 0?
 Jz Thoat
; Kiem tra do dem ban phim da day cua?
 Mov BX, 1AH  ; nap con tro bo dem ban phim
 Mov CX,ES:[BX]
 Mov DI,ES:[BX]+2
 Cmp CX,60
 Jz Tieptuc
 Add, CX,2
      Cmp CX,DI
      Jz Thoat
; Bo dem chua day, nap the ki tu vao
Tieptuc:
      Mov ES:[DI],AL
 Cmp DI,60H
      Jnz Naptiep
      Mov DI,28 ; dua dia chi hien tai ve 28+2=30
Naptiep:
      Add DI,2
      Mov ES:[BX],DI
      ; ket thuc
Thoat:
 Pop ES
 Pop DI
 Pop CX
 Pop BX
 Pop AX
    Mov AL,20h ; tra lai ngat
 Out 20h,AL
 IRET
      TryKB endp
      ;-----------------------------------------------
```

```
End Start
```

4.3 LẬP TRÌNH PHỐI GHÉP VỚI MÀN HÌNH

4.3.1. Cơ bản về màn hình

Bộ nhớ Video

Màn hình của máy tính hiện đại ngày nay đều sử dụng bộ nhớ video (bộ nhớ màn hình) để chứa nội dung hình ảnh được hiển thị và các thông tin liên quan. Bộ nhớ video còn được gọi là bộ đệm khung (frame buffer). Từ thế hệ Pentium, bộ vi xử lý còn có cổng gia tốc đồ họa AGP (Accelerated Graphics Port) Cổng này cho phép bộ vi xử lý dồ họa truy cập trực tiếp vào bộ nhớ hệ thống cho các phép tính đồ họa nhưng vẫn có bộ nhớ video riêng để lưu trữ nội dung các điểm ảnh màn hình. Phương pháp này cho phép sử dụng bộ nhớ hệ thống mềm dẻo hơn mà không làm ảnh hưởng tới tốc độ máy tính. Cổng AGP ngày nay trở thành chuẩn trong các máy tính hiện đại.

Dưới đây là bảng dung lượng bộ nhớ video và khả năng hiển thị màn hình.

Dung lượng bộ nhớ	Kích thước màn hình	Chiều sâu mầu	Số màu
1MB	1024x768 800x600	8-bit 16-bit	256 65,536
2MB	1024x768 800x600 1280x1024	8-bit 16-bit 24-bit	256 65,536 16.7 million
4MB	1024x768	24-bit	16.7 million
6MB	1280x1024	24-bit	16.7 million
8MB	1600x1200	32-bit	16.7 million

Bộ chuyển đổi số/tương tự

Tín hiệu ra màn hình không phải dạng số mà là dạng tín hiệu tương tự. Tín hiệu ra phải đi qua một bộ biến đổi bộ nhớ tương tự/ số (RAMDAC- RAM Digital Analog Converter) đọc nội dung bộ nhơ video rồi chuyển sang tín hiệu tương tự. Qua trình này lặp lại theo tần số làm tươi màn hình.

Bộ xử lý Video

Bộ xử lý video biên dịch và thực hiện lệnh đồ họa thành từng điểm ảnh cụ thể để đưa ra RAMDAC. Bộ vi xử lý video có cấu trúc hoàn chỉnh và phức tạp như một bộ vi xử lý thực sự. Nhờ có bộ xử lý video nên đã giảm tính toán cho từng điểm ảnh để hiển thị đối với bộ xử lý. Thay vào đó CPU chỉ cần truyền tham số căn bản của đối tượng cần hiển thị sang bộ xử lý video. Bộ xử lý video sẽ tính từng điểm ảnh cần hiển thị từ các tham số nhận được, giải phòng CPU khỏi nhiệm vụ này.

Đồ họa ba chiều

Để phần mềm bắt kịp được với phần cứng trong quá trình phát triển của hiển thị đồ họa ba chiều, các hệ điều hành cung cấp một thư viện giao diện lập trình ứng dụng (Application Programming Interface). Giống như BIOS, API cung cấp các lệnh đồ họa chuẩn ra các chương trình xử lý video thay vì lập trình trực tiếp nó. Phần mềm điều khiển sẽ phải biên dịch tiếp nhưng lệnh này sang lệnh máy.

4.3.2 Dịch vụ của BIOS phục vụ màn hình – ngắt 10h

Màn hình làm việc ở một trong hai chế độ: văn bản (text) và đồ họa (graphics).

Ở chế độ văn bản, các kí tự được trình bày trong các ma trận điểm 5x7 với 25 dòng và 80 cột. Màn hình là hình ảnh của video RAM. Do vậy ở chế độ text một trang màn hình cần tối thiểu là 25 dòng x 80 cột x 2 (1 byte mã ASCII và 1 byte thuộc tính kí tự) =4000 bytes. Byte thuộc tính có dạng như sau:

C	Red	Green	Blue	I	Red	Green	Blue
Nhấp nháy	Màu nền			Đậm nhạt	Màu chữ		

Dưới đây là một số giá trị thường dùng của thuộc tính:

Giá trị	Vỉ màu
00	Không hiển thị
01	Kí tự bình thường
07	Kí tự bình thường
09	In đậm
70	Nghịch ảnh
81	Nhấp nháy
87	Kí tự bình thường và Nhấp nháy
F0	Nghịch ảnh và Nhấp nháy

Dưới đây liệt kê một số chức năng của BIOS về chế độ văn bản của màn hình.

Hàm 00h:

Ý nghĩa: Đặt chế độ cho màn hình

Đầu vào: AH=00

AL = chế độ màn hình

Int 10h

Trong đó chế độ màn hình = 0: 40 x 25 trắng đen.

= 1: 40 x 25 16 màu..

= 2: 80 x 25 trắng đen (card màu).

= 3: 80 x 25 16 màu.

= 7: 80 x 25 trắng đen (card mono).

Hàm 01h:

Ý nghĩa: Đặt kích thước con trỏ

Đầu vào: AH=01

CH = tọa độ hàng

CL = tọa độ cột

Int 10h

Hàm 02h:

Ý nghĩa: Đặt vị trí con trỏ

Đầu vào: AH=02

BH = số trang màn hình

DH=số dòng

DL = số cột

Int 10h

Hàm 03h:

Ý nghĩa: Đọc vị trí con trỏ

Đầu vào: AH=03

BH = số trang màn hình

Int 10h

Đầu ra: DH=số dòng

DL = số cột

CH= tọa độ hàng của con trỏ

CL = tọa độ cột của con trỏ

Hàm 05h:

Ý nghĩa: Đặt trang màn hình hoạt động

Đầu vào: AH=05

BL = số trang màn hình

Int 10h

Hàm 06h:

Ý nghĩa: Cuộn màn hình lên (dùng để xác lập vùng cửa sổ văn bản hình chữ nhật)

Đầu vào: AH=06

AL=số trang để trắng hoặ dòng cuộn (AL=0 để trắng toàn màn hình)

(CH,CL) = tọa độ trên bên trái màn hình

(DH,DL) = tọa độ dưới bên phải màn hình

BH= thuộc tính của vùng để trống của màn hình.

Int 10h

Hàm 07h:

Ý nghĩa: Cuộn màn hình xuống

Đầu vào: AH=07

AL=số trang để trắng hoặ dòng cuộn (AL=0 để trắng toàn màn hình)

(CH,CL) = tọa độ trên bên trái màn hình

(DH,DL) = tọa độ dưới bên phải màn hình

BH= thuộc tính của vùng để trống của màn hình.

Int 10h

Hàm 08h:

Ý nghĩa: Đọc kí tự và thuộc tính của nó tại vị trí con trỏ

Đầu vào: AH=08

BH=số trang

Int 10h

Đầu ra: AL =mã ASCII của kí tự

BL= thuộc tính của kí tự.

Hàm 09h:

Ý nghĩa: Viết các kí tự và thuộc tính vào vị trí con trỏ đang đứng (vị trí con trỏ không đổi). Đưa kí tự ra, đặt màu cho kí tự.

Đầu vào: AH=09

BH= số trang màn hình

CX = số lần kí tự được đưa ra màn hình

AL = mã ASCII của kí tự

BL= thuộc tính của kí tự.

Int 10h

Hàm 0Ah:

Ý nghĩa: Viết các kí tự không có thuộc tính vào vị trí con trỏ đang đứng (vị trí con trỏ chuyển sang phải). Không đặt màu cho kí tự.

Đầu vào: AH=0Ah

BH=số trang màn hình

CX = số lần kí tự được đưa ra màn hình

AL =mã ASCII của kí tự

Int 10h

Hàm 0Eh:

Ý nghĩa: Viết các kí tự theo kiểu teletype ra màn hình (vị trí con trỏ chuyển sang phải).

Đầu vào: AH=0Eh

BH=số trang màn hình

BL = màu của kí tự

AL =mã ASCII của kí tự

Int 10h

Hàm 0Fh:

Ý nghĩa: Lấy kiểu màn hình hiện hành.

Đầu vào: AH=0Fh

Int 10h

Đầu ra:

AH=số cột của màn hình

BH = số trang

AL =chế độ hiện thời của màn hình

Hàm 13h:

Ý nghĩa: Hiển thị một dãy kí tự.

Đầu vào: AH=13h

BH=số trang màn hình

DL= số cột bắt đầu hiển thị

DH= số dòng bắt đầu hiển thị

ES:BP =địa chỉ đầu của vùng nhớ chứa dãy kí tự cần hiển thị

CX=độ dài của dãy kí tự.

Int 10h

Ở chế độ đồ họa có thêm một số hàm sau:

Hàm 0h:

Ý nghĩa: Chọn kiểu màn hình.

Đầu vào: AH=00h

AL = 0Dh: 320 x 200, 16 màu

= 0Eh: 640 x 200 16 màu..

= 0Fh: 640 x 350, trắng đen.

= 10h: 640 x 350 16 màu.

= 11h: 640 x 480 2 màu.

= 12h: 640 x 480 16 màu.

= 13h: 320 x 200 256 màu.

(chỉ với card VGA)

Int 10h

Hàm 0Bh:

Ý nghĩa: Chọn bộ màu.

Đầu vào: AH=0Bh

BH=0: chọn màu cho nền BL=0-15

=1: chọn bộ màu cho điểm.

Int 10h

Hàm 0Ch:

Ý nghĩa: Hiển thị một điểm.

Đầu vào: AH=0Ch

DX=số hàng

CX=số cột

AL = số màu của điểm.

BH=số trang màn hình

Int 10h

Hàm 0Dh:

Ý nghĩa: Đọc thông tin của một điểm.

Đầu vào: AH=0Ch

DX=số hàng

CX=số cột

BH=số trang màn hình

Int 10h

Đầu ra: AL = số màu của điểm.

4.3.3. Ví dụ về Lập trình phối ghép với màn hình

Dưới đây là một số ví dụ về lập trình màn hình có sử dụng các chức năng và dịch vụ do BIOS cung cấp đã trình bày ở phần 3.4.1.

Ví dụ 1: In ra màn hình 256 kí tự của bảng mã ASCII, mỗi kí tự có một thuộc tính khác nhau trong chế độ text.

```
.MODEL Tiny
.CODE
      Org 100h
      Jmp Start
Start:
      Mov AL,0
      Mov BL,0 ; thuộc tính 0 vơi kí tự co mã ASCII =0
      Mov DL,0 ; DL= so cot, bat dau tu cot 0
      Mov DH,4 ; Bat dau tu hang thu 4
      Mov CX, 255 ; in 256 kí tự
InTiep:
   Push AX
   Mov AH,2
       Int 10h  ; dat vi tri con tro
       Pop AX
       Mov AH,9 ; in ki tu trong AL voi thuoc
               ; thuoc tinh trong BL
     Push AX
   Push CX
   Mov CX,1 ; in 1 ki tu
   Int 10h
  Pop CX
  Pop AX
  Inc AL ; ki tu ke tiep
  Inc BL ; thuoc tinh ke tiep
  Inc DL ; sang cot ben canh
  Cmp DL,79 ; Cot cuoi cung?
  Jb TiepTuc
  Mov Dl,0 ; xuong dong ke tiep
  Inc DH
TiepTuc:
   Loop InTiep
  Int 20h
End Start
```

Ví dụ 2: Vẽ một hình chữ nhật trong chế độ đồ họa VGA

```
.MODEL Tiny
.CODE
      Org 100h
      Jmp Start
      TopRow dw 100
      TopCol dw 100
      BotRow dw 400
      BotCol dw 600
      Color db 4 ; mau do
Start:
      Mov AH,0h ; thiet lap che do do hoa
      Mov AL,12h ; VGA mode
      Int 10h
      Mov CX, BotCol
      Sub CX,TopCol
      Mov SI,TopRow
      Mov DI,TopCol
      ; Ve cach tren
TopSide:
     Call DrawPixel  ; goi chuong trinh con ve diem anh
      Inc DI  ;
      Loop TopSide
      Mov CX, BotRow
      Sub CX, TopRow
      Mov SI, TopRow
      ; Ve cach  2 canh ben
Side:
      Mov DI, TopCol
 Call DrawPixel  ; goi chuong trinh con ve diem anh
      Mov DI, BotCol
 Call DrawPixel  ; goi chuong trinh con ve diem anh
      Inc SI
      Loop Side
      Mov CX, BotCol
      Sub CX,TopCol
      Mov SI, BotRow
      Mov DI, TopCol
      ; Ve cach duoi
BotSide:
     Call DrawPixel  ; goi chuong trinh con ve diem anh
      Inc DI  ;
      Loop BotSide
```

```
        Mov AH, 1 ; Cho 1 phim go vao
        Int 21h
        Mov AH,0
        Mov AL,2
        Int 10h
        Int 20h
        ; chuong trinh con ve 1 diem anh
        DrawPixel Proc
        Push AX
        Push CX
        Push DX
        Mov DX, SI
        Mov CX,DI
        Mov AL, Color
        Mov AH,0CH
        Int 10h
        Pop DX
        Pop CX
        Pop AX
        Ret
        DrawPixel Endp
End Start
```

Ví dụ 3: Viết một điểm ảnh trực tiếp vào bộ nhớ hiển thị (không thông qua RAM).

Đây là cách nhanh nhất, tuy nhiên thủ tục viết vào bộ nhớ hiển thị phụ thuộc vào loai card điều khiển màn hình cụ thể.

Chương trình sau được thực hiện trong chế độ đồ họa (640x200x2).

```
VeDiemAnh Proc Near
; DX=hàng, CX= cột
; ES=B800H
; Giả sử chế độ đồ họa đã được khởi tạo như sau:
; AH=0, AL=6, Int 10h

        Xor BX,BX ; BX=0
        SHR DX,1 ; kiem ta xem diem anh o dong chan hay le
        Add BX, 8*1024 ; o dong chan
TinhTiep:
 Mov AX,DX ; DX =so hang
 Mov CL,2
        SHL DX,CX ; DX=DX*4
        Add DX,AX ; DX=5*DX
        Mov AX,CX
        Mov CL,4
        SHL DX,CL ; DX=DX*16
```

```
        Add BX,DX ; BX=offset cua dong chua diem anh
        Mov DX,AX
        AND DX,7 ; DX=DX Mod 8 = vi tri byte tren dong
        Mov CL,3 ; no chua diem anh
        SHR AX,CL; AX=AX div 8= vi tri byte tren dong
        Add, BX,AX ; [BX] la byte chua diem anh
        NEG DL
        Add DL,7
        Mov CL,DL
        Mov AL,1
        SHL AL,CL
        OR ES:[BX], AL
        Ret
VeDiemAnh Endp
```

4.4 LẬP TRÌNH ĐIỀU KHIỂN Ổ ĐĨA

4.4.1 Cơ bản về ổ đĩa

a. *Đĩa mềm*

Mỗi sector trên đĩa sẽ chứa các đặc trưng (directory) của các file. Mỗi đặc trưng của một file gồm 32 byte chứa các thông tin sau:

Byte	Nội dung
0h-7h	Tên file
8h-0Ah	Phần mở rộng
0Bh	Thuộc tính của file
0Ch-15h	Chưa dung đến
16h-17h	Giờ của lần thay đổi cuối cùng
18h-19h	Ngày của lần thay đổi cuối cùng
1Ah-1Bh	Chứa số ô của bảng FAT (File Allocation Table)
1Ch-20h	Chứa kích thước file

Byte thuộc tính có cấu trúc như sau:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		=1: lưu trữ	=1: thư mục con	=1: tên nhãn	=1: hệ thống	=1: thuộc tính ẩn	=1: chỉ đọc

b. *Đĩa cứng*

Đĩa cứng được chia thành các phần độc lập gọi là partition. Mỗi partition có thể dành cho các hệ xử lý vaf hệ điều hành sử dụng riêng rẽ. Trên đĩa dành ra một sector riêng để chứa bảng partition. Bảng partition chứa điểm vào của partition.

Dưới đây là nội dung điểm vào (partition entry) của partition :

Vị trí	Kích thước (byte)	Nội dung
00	1	Loại partition: 80H : khởi động được từ partition; 00H: không khởi động được từ partition; 0BH: partition với FAT 32; 0CH: partition với FAT 32 và ngắt 13 mở rộng;
01	3	Nới bắt đầu partition
04	1	Kiểu của hệ thống file: 0: không phải file của DOS; 1: file của DOS, FAT12; 4: file của DOS, FAT16; 5: file của DOS mở rộng; 6: đĩa lớn hơn 32MB;
05	3	Nơi bắt đầu partition
08	4	Cung bắt đầu partition
0C	4	Số cung trong partition

Dưới đây là các hệ thống file thông dụng :

- FAT16 của hệ điều hành DOS và Windows 3.x
- FAT32 trong VFAT (Virtual FAT) có trong windows 9x,
- NTFS có trong Windows NT, 2000, XP và Vista
- VFS có trong Linux

c. Cấu trúc bảng FAT

Bảng FAT ghi thông tin về vị trí của từng khối. Một đề mục FAT có thể dài 12, 16 hoặc 32 bít. Mỗi khối có độ rộng từ 1 cung (512 byte) đến 64 cung (32 KB). Chẳng hạn, đĩa mềm có 9 cung dành riêng cho FAT. Mỗi giá trị FAT tương ứng với một khối trên đĩa. Một file thường được lưu trữ trên nhiều khối khác nhau, những khối này không nhất thiết phải nằm liên tiếp nhau. Hệ điều hành luôn lưu trữ dữ liệu lên khối còn trống gần nhất.

Đại chỉ khỗi thứ nhất của file được lưu trữ trong điểm vào của thư mục (file entry directory). Hệ điều hành sẽ căn cứ vào địa chỉ này trên bảng FAT để tìm ra địa chỉ khối kế tiếp của file. Nếu xâu khối kết thúc, giá trị FAT của khối cuối cùng sẽ là FFFH (với FAT12).

Dưới đây là bảng chứa ý nghĩa của thông tin trong cung khởi động (boot sector).

Vị trí	Kích thước	Ý nghĩa
00	3	Lệnh JUMP nhảy về chương trình khởi động; byte thứ ba là một lệnh No-OP
03	8	Tên nhà sản xuất, phiên bản
0B	2	Số byte trên 1 cung
0D	1	Số cung trên 1 khối
0E	2	Số cung dành cho khởi động
10	1	Số bảng xếp đặt file FAT
11	2	Số điểm vào (entry) trên thư mục gốc
13	2	Số cung trên đĩa (trống nếu kích thước >32MB)
15	1	Loại đĩa (như byte đầu của FAT)
16	2	Số cung dành cho bảng xếp đặt file FAT
18	2	Số cung trên một track
1A	2	Số đầu từ
1C	2	Số cung ẩn
1E	4	Số cung nếu kích thước >32MB. Giá trị như tại địa chỉ 13H nếu kích thước nhỏ hơn 32MB.
22	1	Số ổ đĩa
23	1	Dữ trữ
24	1	Kí hiệu kết thuc cung khởi động
25	3	Số thứ tự
29	11	Tên đĩa (mã ASCII)
34	8	Nhận diện hệ FAT
3C-200	452	Chương trình khởi động

d. bảng FAT12

FAT12 dùng 1.5 byte để đánh địa chỉ của một khối (block). Để tiết kiệm dung tích của FAT, địa chỉ của hai khối gần nhau được lưu trữ chung trong 3 byte. Ba byte đầu tiên của FAT không được dung làm địa chỉ khối. Byte đầu tiên của FAT chứa mã nhận dạng đĩa (địa chỉ của boot sector). Hai byte kế tiếp luôn là FFH. Mã nhận dạng đĩa được liệt kê trong bảng sau :

Mã	Loại đĩa
FF	2 mặt, 8 cung
FE	1 mặt, 8 cung
FD	2 mặt, 9 cung
FC	1 mặt, 9 cung
F9	2 mặt, 15 cung
F8	Đĩa cứng
F0	Đĩa mềm 3.5 inch

Vì 3 byte đầu tiên của FAT đã bị chiếm, địa chỉ khối bắt đầu từ 2. Địa chỉ của khối 2 và 3 nằm trong 3 byte kế tiếp. Người lập trình cần tránh thay đối FAT vì hệ điều hành sẽ đảm nhận nhiệm vụ này. Khi cần phải đọc nội dung bảng FAT, người lập trình có thể làm như sau :

Tìm khối tiếp theo bằng cách :

- nhân số khối với 1.5
- đọc 2 byte của kết quả (làm tròn xuống)
- nếu số khối là chẵn, lấy 12 bít thấp. Ngược lại, lấy 12 bit cao.

Chuyển số khối thành số cung login tương ứng :

- số khối trừ đi 2
- nhân kết quả với số cung trong một khối

e. bảng FAT32

Với bảng FAT32, bảng xếp đặt file ảo (VFAT) được sử dụng. Hệ điều hành sử dụng các giá trị 4 byte trong FAT. Cấu trúc thư mục gốc cho phép tên file có thể dài đến 255 kí tự, thay vì 8 kí tự của phần tên và 3 kí tự của phần mở rộng như trong FAT16.

Ngoài ra, VFAT còn sử dụng một điểm vào (entry) trong thư mục với độ dài 32 byte cho 13 kí tự tiếp theo của tên file. Trên lý thuyết, tên file dài tối đa 255 kí tự có thể sẽ tiêu tốn hết 21 điểm vào của thư mục. Thư mục gốc có tất cả 512 điểm vào file (thư mục gốc chứa tối đa 512 file và thư mục con), trong đó chỉ chứa được 24 file có tên dài đến 255 kí tự. Tuy nhiên, các thư mục con bên trong lại không bị ảnh hưởng bởi giới hạn này.

Nhược điểm của FAT là chỉ quản lý được 1 ổ đĩa có kích thước giới hạn. Công nghệ sản xuất đĩa cứng sẽ cho ra đời các ổ đĩa lớn. Chẳng hạn, FAT16 chỉ quản lý được 2GB đĩa cứng.

Giả sử kích thước khối là B (thường là khoảng 4KB trên 1 block), dung lượng đĩa là V. Với FATn (n=12,16, hoặc 32) thì dung lượng có thể quản lý được theo công thức:

$V=2^n B$.

Công thức trên cho thấy, với n không đổi thì kích thước khối B phải tăng khi dung tích ổ đĩa tăng. Như vậy nếu 1 file có kích thước nhỏ hơn kích thước của một khối thì phần còn lại của khối sẽ bị lãng phí.

4.4.2 Dịch vụ của BIOS và DOS phục vụ ổ đĩa

a. Ngắt 13h- ngắt của BIOS phục vụ ổ đĩa

Hàm 0h:

Ý nghĩa: Reset lại ổ đĩa mềm, chỉ nên gọi hàm này khi gặp lỗi trong khi truy cập đĩa bằng 1 trong 6 chức năng của ngắt 13h.

Đầu vào: AH=0h

Int 13h

Đầu ra: AH= trạng thái lỗi

Hàm 01:

Ý nghĩa: Cho biết trạng thái đĩa

Đầu vào: AH=01h

DL=số ổ đĩa

Int 13h

Đầu ra: AH= trạng thái.

Hàm 02

Ý nghĩa: Đọc một hay nhiều sector

Đầu vào: AH=02h

DL=số ổ đĩa (0-3)

DH=mặt đĩa (0: mặt trên -1:mặt dưới)

CL=sector đầu cần đọc

CH=rãnh chứa sector đầu tiên cần đọc.

AL=số lượng sector cần đọc

ES:BX =địa chỉ vùng nhớ chứa thông tin đọc được.

Int 13h

Đầu ra: Nếu cờ CF=1 thì AH= mã lỗi.

Nếu cờ CF=0 thì AL= số sector đọc được.

Hàm 03:

Ý nghĩa: Ghi dữ liệu lên đĩa

Đầu vào: AH=03h

DL=số ổ đĩa (0-3)

DH=mặt đĩa (0: mặt trên -1:mặt dưới)

CL=sector đầu cần đọc

CH=rãnh chứa sector đầu tiên cần đọc.

AL=số lượng sector cần đọc

ES:BX =địa chỉ vùng nhớ cần ghi lên đĩa.

Int 13h

Đầu ra: Nếu cờ CF=1 thì AH= mã lỗi.

Nếu cờ CF=0 thì AL= số sector ghi thành công.

Hàm 04:

Ý nghĩa: Kiểm tra CRC (kiểm tra dư thừa vòng); không so sánh dữ liệu trên đĩa với dữ liệu trong vùng nhớ mà chỉ kiểm tra CRC.

Đầu vào: AH=04h

DL=số ổ đĩa (0-3)

DH=mặt đĩa (0: mặt trên -1:mặt dưới)

CL=sector đầu cần đọc

CH=rãnh chứa sector đầu tiên cần đọc.

AL=số lượng sector cần đọc

Int 13h

Đầu ra: Nếu cờ CF=1 thì AH= mã lỗi.

Nếu cờ CF=0 thì thành công.

Hàm 05:

Ý nghĩa: Tạo khuôn dạng (format) cho đĩa .

Đầu vào: AH=05h

AL=số lượng sector cần tạo trên 1 rãnh.

CH=số thứ tự của rãnh cần tạo (0-39 hoặc 0-79).

DH=số thứ tự của mặt đĩa (0,1).

ES:BX =trỏ đến một bảng chứa các tham số sau:

Byte 1: rãnh cần tạo khuôn.

Byte 2: mặt đĩa (0-trước, 1-sau).

Byte 3: số thứ tự của sector.

Byte 4: số byte của sector.

Ngoài ra, phải thêm thông tin nằm trên bảng tham số đĩa mềm gồm 11 byte.

Int 13h

Đầu ra: Nếu cờ CF=1 thì AH= mã lỗi.

Hàm 15h:

Ý nghĩa: Xác định loại ổ đĩa.

Đầu vào: AH=15h

DL=số ổ đĩa (0-3)

Int 13h

Đầu ra:

AH=kiểu ổ đĩa

= 0: không có ổ đĩa

= 1: ổ đĩa không phát hiện được sự thay đổi ổ đĩa

= 2: ổ đĩa phát hiện được sự thay đổi ổ đĩa

= 3: ổ đĩa cứng

Hàm 16h:

Ý nghĩa: Kiểm tra có sự thay đổi đĩa hay không.

Đầu vào: AH=16h

DL=số ổ đĩa (0-3)

Đầu ra:

AH= kết quả

= 0: đĩa chưa thay đổi

= 6: đĩa đã thay đổi sau lần truy cập cuối cùng.

b. Ngắt 21h của DOS phục vụ ổ đĩa

Hàm 0Eh

Ý nghĩa: Chọn ổ đĩa chủ (master disk)

Đầu vào: AH=0Eh

DL=số hiệu ổ đĩa cần chọn làm đĩa chủ

Đầu ra:

AL= số lượng ổ đĩa hiện có

Hàm 19h

Ý nghĩa: Cho biết ổ đĩa nào là chủ (master disk)

Đầu vào: AH=19h

Đầu ra:

AL= số hiệu ổ đĩa

Hàm 1Ah

Ý nghĩa: Khởi đầu bộ đệm truyền dữ liệu DTA (Data Transfer Area). Bộ đệm DTA có kích thước 128 byte và được DOS xếp ở vùng nhớ tiền tố đoạn chương trình (Program Segment Prefix). Nếu cần một vùng nhớ lớn hơn 128 byte thì sẽ được cấp ở vùng nhớ khác.

Đầu vào: AH=1Ah

DS:DX =địa chỉ mới của bộ đệm

Đầu ra:

Hàm 1Bh

Ý nghĩa: Đọc thông tin về ổ đĩa chủ (master disk)

Đầu vào: AH=1Bh

Đầu ra:

AL= số sector/cluster

CX = kích thước của 1 sector

DX = số lượng cluster

Hàm 1Ch

Ý nghĩa: Đọc thông tin về ổ đĩa bất kỳ

Đầu vào: AH=1Ch

DL=số hiệu ổ đĩa cần đọc

Đầu ra:

AL= số sector/cluster

CX = kích thước của 1 sector

DX = số lượng cluster

Hàm 2Fh

Ý nghĩa: Lấy địa chỉ DTA hiện hành

Đầu vào: AH=2Fh

Đầu ra:

ES:BX=địa chỉ vùng DTA hiện thời

Hàm 36h

Ý nghĩa: Không gian trống trên đĩa

Đầu vào: AH=36h

DL=số hiệu ổ đĩa

Đầu ra:

AX= số sector/cluster

BX = số cluster còn trống

DX = tổng số cluster trên đĩa

c. Ngắt 25h của DOS phục vụ ổ đĩa

Ý nghĩa: Đọc trực tiếp đĩa

Đầu vào: AL=số hiệu ổ đĩa

CX = số lượng sector cần đọc

DX = số sector logic

DS:BX =địa chỉ vùng nhớ chứa dữ liệu đọc ra

d. Ngắt 26h của DOS phục vụ ổ đĩa

Ý nghĩa: Ghi trực tiếp lên đĩa

Đầu vào : AL=số hiệu ổ đĩa

CX = số lượng sector cần đọc

DX = số sector logic

DS:BX =địa chỉ vùng nhớ chứa dữ liệu cần ghi lên đĩa.

4.4.3 Ví dụ về lập trình ổ đĩa

```
Ví dụ 1: Đọc nội dung bảng FAT.
.Model Tiny
.Code
Org 100h
Jmp Start
Buffer db 1024 dup(0) ; có thể lưu trữ được nội dung của 2 sector
Start:
     Mov bx, offset buffer ; trỏ ddeesns vùng đệm
     Mov dx,1 ; địa chỉ logic của cung
     Mov cx,2 ; đọc 2 cung
     Int 25h ; ngắt đọc ổ đĩa
     Pop cx
     Mov ax, 3; ax=địa chỉ khôi
     Mov cx,ax;
     Mov dx,ax; cả cx,dx cũng chứa địa chỉ khối
     Shr dx,1; chia dx cho 2
     Add cx,dx; nhân với 1.5 lần
     Add bx,cx ; cộng vào địa chỉ lệch của khối
     Mov dx,[bx]; đọc 2 byte ở địa chỉ bx và bx+1
     Test ax,1; nếu số hiệu khối là chẵn?
     Jz khoichan
     Mov cl,4
     Shr dx,cl  ; chia cho 16, địa chỉ khối nằm trong dx
     Jmp Continue
Khoichan:
     And dx,0000111111111111B; xóa 4 bít cao, đọc 12 bít thấp
Continue:
```

4.5 TÓM TẮT

Chương này đã trình bày về phương thức truyền thông tin nối tiếp với bộ điều hợp UART 8250A. Trong phần này, ta tìm hiểu về cơ chế truyền thông tin nối tiếp kiểu đồng bộ, dị bộ, các thanh ghi bên trong của UART 8250A và lập trình cho UART 8250A. Tiếp theo là phần lập trình phối ghép cho bàn phím: nguyên tắc hoạt động của bàn phím, bộ đệm bàn phím, vùng dữ liệu, sử dụng các dịch vụ của BIOS để lập trình điều khiển bàn phím thông qua một số ví dụ đơn giản mang tính minh họa. Tương tự, ta cũng tìm hiểu về cách lập trình điều khiển màn hình thông qua hai phương pháp: sử dụng dịch vụ của BIOS và truy nhập trực tiếp vào bộ nhớ màn hình. Tuy nhiên, cách thứ hai tương đối khó vi nó phụ thuộc vào loại card điều khiển màn hinh và người lập trình phải cài đặt một ánh xạ từ bộ nhớ RAM sang bộ nhớ màn hình, nên cách này được xem như một sự tham khảo cho người lập trình hệ thống.

Phần cuối cùng là phần giới thiệu trình hợp ngữ trong Windows thông qua công cụ RadASM. Ngoài một số đối tượng, chức năng được môi trường phát triển cho ứng dụng RadASM cung cấp dới dạng các thư viện, việc viết các ứng dụng bằng hợp ngữ trên windows cũng khá giống môi trường khác. Nên tác giả không đề cập kỹ ở cuốn sách này. Đây là phần chỉ mang tính chất giới thiệu cho độc giả.

4.6 BÀI TẬP

4.6.1 Câu hỏi trắc nghiệm

1. Lý do chính của việc truyền thông nối tiếp giữa hai đối tượng cần truyền tin cho nhau là:

A. Truyền thông nối tiếp là công nghệ mới

B. Truyền thông giữa hai đối tượng truyền/nhận thong tin ở khoảng cách xa nhau.

C. Truyền thông nối tiếp là nhanh hơn truyền thông song song

D. Truyền thông nối tiếp chính xác hơn song song.

2. Phát biểu nào sau đây là đúng nhất:

A. Truyền tin dị bộ truyền theo từng kí tự còn truyền đồng bộ truyền theo nhiều kí tự.

B. Truyền tin dị bộ là kiểu truyền tin song song còn truyền đồng bộ là truyền tin nối tiếp.

C. Truyền tin đồng bộ là truyền song song còn truyền tin dị bộ là kiểu truyền tin nối tiếp

D. Cách mã hóa kí tự trong hai kiểu truyền tin đồng bộ và dị bộ luôn luôn khác nhau..

3. Thanh ghi nào dưới đây quyết định khuôn dạng dữ liệu khi truyền:

A. Thanh ghi trạng thái modem.

B. Thanh ghi trạng thái đường truyền.

C. Thanh ghi nháp

D. Thanh ghi điều khiển đường truyền..

4. Phát biểu nào sau đây là đúng nhất cho một khung truyền kiểu dị bộ:

A. Dành ra ít nhất là 6 bít để mã hóa dữ liệu.

B. Dành ra ít nhất là 7 bít để mã hóa dữ liệu.

C. Có ít nhất 1 bít Start

D. Có ít nhất là 2 bít Stop.

5. Phát biểu nào sau đây là đúng nhất về bít parity trong khung truyền kiểu dị bộ:

A. Trong một khung truyền có ít nhất 1 bít parity.

B Trong một khung truyền có thể không có bít parity nào.

C. Trong một khung truyền có nhiều nhất 2 bít parity.

D. Bít parity lưu trạng.thái của khung truyền.

6. Khi người dùng nhấn phím Ctrl_NumLock, thì máy tính sẽ:

A. Lưu mã Scan của hai phím Ctrl và NumLock vào bộ đệm bàn phím.

B Thực hiện một vòng lặp mà không lưu lại gì vào trong vùng đệm bàn phím.

C. Lưu mã ASCII của hai phím Ctrl và NumLock vào bộ đệm bàn phím.

D. Lưu cả mã Scan và mã ASCII của hai phím Ctrl và NumLock vào bộ đệm bàn phím.

7. Tại một thời điểm, bộ đệm bàn phím có thể chứa được tối đa:

A. 16 mã phím.

B 8 mã phím

C. 32 mã phím

D. 24 mã phím

8. Nếu con trỏ Head bằng con trỏ Tail thì:

A. Bộ đệm bàn phím đầy.

B Có 8 mã phím

C. Có 16 mã phím

D. Bộ đệm bàn phím rỗng

4.6.2 Bài tập Lập trình

1. Hãy viết chương trình hợp ngữ tự động bật tổ hợp các phím CapLock, Insert, NumLock

2. hãy lập trình để khởi tạo chế độ làm việc một mạch UART 8250A cho cổng COM2 với các thông số:

a. 6 bít mã kí tự truyền , tốc độ truyền 1200 bits/ giây, parity lẻ, hai bít stop.

b. 7 bít mã kí tự truyền , tốc độ truyền 9600 bits/ giây, parity chẵn 1.5 bít stop.

3. Sử dụng các dịch vụ ngắt của BIOS phục vụ cho màn hình (ngắt 10h) trong chế độ dồ họa

để vẽ ra màn hình các hình sau:

a. Tam giác

b. Hình vuông

c. Hình tròn

d. Hình Parabol

4.7 TÀI LIỆU THAM KHẢO

1. Nguyễn Nam Trung. Cấu trúc máy Vi tính và Thiết bị ngoại vi. Nhà XB Khoa học Kỹ thuật.2000.

2. Hồ Khánh Lâm. Giáo trình Kỹ thuật Vi xử lý Tập 1&2. Nhà XB Bưu điện. 2006.

3. Đặng Thành Phu. Turbo Assembler và Ứng dụng. Nhà XB Khoa học và Kỹ thuật. 1998.

4.Nguyễn Minh San. Cẩm nang Lập trình Hệ thống (tập 2). Bản dịch. Nhà XB Tài chính Thống Kê. 1996.

5.Nguyễn Đình Việt. Giáo Trình nhập môn Hợp ngữ và Lập trình Hệ thống. Hà nội. 1998.

TÀI LIỆU THAM KHẢO

1. Đặng Thành Phu. Turbo Assembler và Ứng dụng. Nhà XB Khoa học và Kỹ thuật. 1998.

2. Nguyễn Nam Trung. Cấu trúc máy Vi tính và Thiết bị ngoại vi. Nhà XB Khoa học Kỹ thuật.2000.

3. Hồ Khánh Lâm. Giáo trình Kỹ thuật Vi xử lý Tập 1&2. Nhà XB Bưu điện. 2006.

4.Nguyễn Minh San. Cẩm nang Lập trình Hệ thống (tập 1&2)- bản dịch. Nhà XB Tài chính Thống Kê. 1996.

5.Nguyễn Đình Việt. Giáo Trình nhập môn Hợp ngữ và Lập trình Hệ thống. Hà nội. 1998.

6. webstie : www.emu8086.com

7. Văn Thế Minh. Kỹ thuật Vi xử lý. Nhà XB Giáo dục. 1997.

MỤC LỤC