CHƯƠNG 3 LẬP TRÌNH ASSEMBLY CHO HỆ VI XỬ LÝ

Trong chương trước chúng ta đã tìm hiểu về cấu trúc và tập lệnh của bộ vi xử lý 8088. Trong chương này chúng ta sẽ tìm hiểu cách lập trình Assembly cho các hệ vi xử lý được xây dựng trên bộ vi xử lý 8088 (và họ vi xử lý Intel 80x86 nói chung). Sở dĩ ta dùng ngôn ngữ lập trình Assembly để viết phần mềm cho hệ vi xử lý là vì nó có các ưu điểm sau:

- Sử dụng trực tiếp tập lệnh của bộ vi xử lý nên quá trình điều hành chức năng rất sát với cấu trúc phần cứng của hệ thống, tận dụng triệt để khả năng của phần cứng mà không một ngôn ngữ lập trình bậc cao nào làm được.
- Có tốc độ thực hiện nhanh hơn nhiều so với các ngôn ngữ bậc cao. Do vậy nó rất thích hợp với các chức năng yêu cầu thời gian thực chẳng hạn như thao tác với các tín hiệu biến đổi nhanh.

Các chương trình viết bằng ngôn ngữ Assembly phải được dịch ra ngôn ngữ máy (dạng nhị phân) vì đây là dạng duy nhất mà hệ vi xử lý có thể hiểu được. Có nhiều chương trình biên dịch nhưng thông dụng nhất hiện nay Macro Assembler của hãng Microsoft và Turbo Assembler của hãng Borland. Chúng ta sẽ sử dụng Macro Assembler 6.0 để biên dịch các chương trình Assembly. Chương trình biên dịch MASM 6.0 có rất nhiều file nhưng tối thiểu cần những file sau:

- MASM.EXE để biên dịch chương trình sang ngôn ngữ máy
- LINK.EXE để liên kết các chương trình và tạo ra một chương trình chạy được có đuôi exe.
- EXE2BIN để chuyển chương trình đuôi exe sang đuôi com.

I - KHUNG CHƯƠNG TRÌNH ASSEMBLY

1. Bộ ký tự của Assembly

Một ngôn ngữ bất kỳ từ ngôn ngữ giao tiếp của con người tới ngôn ngữ máy tính đều xây dựng trên một bộ ký tự. Các ký tự ghép lại thành các từ có nghĩa gọi là từ vựng. Các từ lại được viết thành các câu tuân theo cú pháp và ngữ pháp của ngôn ngữ để diễn tả hành động sự việc cần thực hiện. Bộ ký tự của Assembly gồm có:

- Các chữ cái latin: 26 chữ hoa A-Z, 26 chữ thường a-z.
- Các chữ số thập phân: ‘0’ - ‘9’
- Các ký hiệu phép toán, các dấu chấm câu và các ký hiệu đặc biệt: + - * / @ ? $, . : [ ] ( ) < > { } & % ! \ # v.v...
- Các ký tự ngăn cách: space và tab

2. Từ khóa

Từ khóa là các từ của riêng Assembly như tên các thanh ghi, tên lệnh dạng gợi nhớ của bộ vi xử lý, tên toán tử... Các từ khóa này đòi hỏi người lập trình phải dùng đúng như Assembly quy định. Các từ khóa có thể viết bằng chữ hoa hoặc chữ thường đều được.

3. Tên tự đặt

Tên là một dãy ký tự dùng để biểu thị tên hằng, tên biến, tên nhãn, tên chương trình con, tên đoạn nhớ... Tên do người lập trình tự đặt nhưng phải tuân theo quy tắc sau:

Quy tắc đặt tên: Tên chỉ gồm chữ cái, chữ số và một số ký tự đặc biệt như ? @ _ $ Chữ đầu của tên bắt buộc phải là chữ cái. Trong tên không có dấu cách. Tên có thể dài từ 1 đến 35 ký tự.

4. Cấu trúc một lệnh Assembly

Một chương trình Assembly bao gồm các dòng lệnh, một dòng lệnh có thể là một lệnh thật dưới dạng gợi nhớ của bộ vi xử lý hoặc một hướng dẫn cho chương trình dịch (assembler directive, đôi khi gọi là lệnh giả). Lệnh thật sẽ được dịch ra mã máy còn lệnh giả thì không

được dịch, vì nó chỉ có tác dụng chỉ dẫn cho chương trình dịch thực hiện công việc. Ta có thể viết các dòng lệnh bằng chữ hoa hoặc chữ thường đều được vì chúng được coi là tương đương nhau.

Một dòng lệnh của Assembly có thể có những trường sau (không nhất thiết phải có đủ các trường):

Tên Mã lệnh Các toán hạng Chú giải

Ví dụ:

LAP: MOV AH,[BX] ; Copy nội dung của ô nhớ có địa chỉ DS:BX vào AH

Dòng lệnh trên có đủ 4 trường. Trường tên là nhãn LAP, trường mã lệnh là lệnh MOV, trường toán hạng là các thanh ghi AH và BX, trường chú giải đặt sau dấu chấm phẩy

MAIN PROC

và

MAIN ENDP

Hai dòng lệnh này là hai lệnh giả, ở trường tên có tên thủ tục là MAIN, ở trường mã lệnh có lệnh giả PROC và ENDP. Đây là hai lệnh giả để bắt đầu và kết thúc một thủ tục có tên là MAIN.

- Trường tên
  Trường tên có thể là tên nhãn, tên biến hoặc tên thủ tục (chương trình con). Các tên và nhãn này sẽ được trình biên dịch gán bằng các địa chỉ cụ thể của ô nhớ. Một nhãn kết thúc bằng dấu hai chấm (:).
- Trường mã lệnh
  Chứa các lệnh thật hoặc lệnh giả
- Trường toán hạng
  Đối với các lệnh thật thì trường này chứa các toán hạng của lệnh. Tùy từng loại lệnh mà có thể không có, có 1 hoặc 2 toán hạng trong một lệnh.
  Đối với các lệnh giả thì trường này chứa các thông tin khác liên quan đến lệnh giả.
- Trường chú giải
  Lời giải thích phải được bắt đầu bằng dấu chấm phẩy. Trường chú giải dành cho người lập trình để ghi các lời giải thích cho các lệnh của chương trình, giúp cho người đọc chương trình dễ hiểu các thao tác của chương trình lớn. Khi đọc thấy dấu chấm phẩy, chương trình dịch bỏ qua không dịch từ sau dấu chấm phẩy đến hết dòng. Người lập trình có thể lợi dụng đặc điểm này để loại bỏ một dòng lệnh nào đó trong chương trình.

5. Các dạng hằng dùng trong Assembly

- Hằng số nhị phân: gồm một dãy các chữ số 0 và 1, kết thúc bằng chữ B. Ví dụ: 10011101B

- Hằng số hex: gồm một dãy các số từ 0 đến 9 và các chữ cái từ A đến F (a đến f), kết thúc bằng chữ H. Đối với các số bắt đầu bằng chữ thì phải thêm 0 đằng trước để báo cho chương trình dịch biết đó là số không phải là tên. Ví dụ: 7AC5H, 0ABH

- Hằng số thập phân: gồm một dãy các số từ 0 đến 9, có hoặc không có chữ D theo sau. Ví dụ: 34 hoặc 34D.

- Hằng ký tự: là một ký tự bất kỳ đặt giữa hai dấu phẩy trên. Ví dụ: ‘A’

- Hằng xâu ký tự: là một dãy ký tự bất kỳ đặt giữa hai dấu phẩy trên. Ví dụ: ‘Nhap’

6. Khai báo biến và hằng

a) Khai báo biến

Biến là tên ô nhớ dùng để cất giữ dữ liệu. Khai báo biến là đặt tên cho ô nhớ và xác định ô nhớ có kích thước 1 byte, 1 từ hay 1 từ kép. Các tên biến sẽ được trình biên dịch gán cho một địa chỉ nhất định trong bộ nhớ khi dịch chương trình.

- Khai báo biến kiểu byte

Tên biến DB Giá trị khởi đầu

Ví dụ:
B1 DB 4

Ví dụ trên định nghĩa biến kiểu byte có tên là B1 và dành 1 byte bộ nhớ cho nó, trong byte đó có chứa giá trị 4.

Nếu không muốn biến chứa giá trị khởi đầu ta có thể dùng toán tử ? vào vị trí giá trị khởi đầu.

Ví dụ:
B2 DB ?

Ví dụ trên chỉ định nghĩa biến byte có tên là B2 và dành 1 byte bộ nhớ cho nó.

- Khai báo biến kiểu từ

Tên biến DW Giá trị khởi đầu

Ví dụ:
W1 DW 42H

Ví dụ này định nghĩa biến từ có tên là W1 và dành 2 byte bộ nhớ cho nó, trong đó chứa giá trị khởi đầu là 42H.

Muốn biến không chứa giá trị khởi đầu ta dùng toán tử ? và vị trí giá trị khởi đầu.

Ví dụ:
W2 DW ?

- Khai báo biến kiểu từ kép

Tên biến DD Giá trị khởi đầu

Ví dụ:
DW1 DD 1000

- Khai báo biến mảng

Biến mảng là biến hình thành từ một dãy liên tiếp các phần tử (ô nhớ) có cùng kiểu byte từ hoặc từ kép. Khai báo biến mảng là đặt tên cho một dãy liên tiếp các byte từ hoặc từ kép trong bộ nhớ đồng thời cung cấp các giá trị ban đầu tương ứng. Số phần tử của mảng được xác định qua số giá trị khởi đầu.

Tên biến mảng DB/DW/DD Các giá trị khởi đầu

Ví dụ:
M1 DB 4,5,6,7,8,9

Ví dụ trên định nghĩa biến mảng có tên là M1 và dành 6 byte liên tiếp cho nó để chứa các giá trị khởi đầu tương ứng là 4, 5 ,6 ,7 ,8 , 9. Phần tử đầu của mảng là 4 và có địa chỉ trùng với địa chỉ của tên biến (M1), phần tử thứ hai là 5 và có địa chỉ là M1+1...

Khi chúng ta muốn khởi đầu các phần tử của mảng với cùng một giá trị chúng ta có thể dùng thêm toán tử DUP. Toán tử DUP dùng để lặp lại các dữ liệu với số lần quy định. Cú pháp: Count DUP(Các dữ liệu) -> lặp lại các dữ liệu với số lần Count.

Ví dụ:
M2 DB 20 DUP(0)
M3 DB 20 DUP(?)

Ví dụ trên định nghĩa một biến mảng có tên là M2 gồm 20 byte để chứa 20 giá trị khởi đầu bằng 0 và một biến mảng khác có tên là M3 gồm 20 byte nhưng không chứa giá trị khởi đầu.

Chú ý:
+ Toán tử DUP có thể dùng lồng nhau để định nghĩa 1 mảng
Ví dụ:
M4 DB 4,3,2,2 DUP(1,2 DUP(5),6)

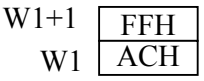
Khai báo này tương đương với khai báo sau:

M4 DB 4,3,2,1,5,5,6,1,5,5,6

+ Đối với các bộ vi xử lý của Intel, khi ta lưu trữ một từ trong bộ nhớ thì *byte thấp của nó sẽ được để ở ô nhớ có địa chỉ thấp, byte cao để ở ô nhớ có địa chỉ cao*.

Ví dụ:

W1 DW 0FFACH



khi đó byte thấp ACH sẽ được để tại địa chỉ W1, còn byte cao FFH sẽ được để tại địa chỉ tiếp theo W1+1.

- Khai báo biến kiểu xâu ký tự

Biến kiểu xâu ký tự là trường hợp đặc biệt của biến mảng kiểu byte, trong đó các phần tử của mảng là các ký tự. Một xâu ký tự có thể định nghĩa bằng các ký tự hoặc bằng mã ASCII của các ký tự đó.

Ví dụ:

```
Xaukt   DB   'ABCDE'
hoặc
Xaukt   DB   41h,42h,43h,44h,45h
hoặc
Xaukt   DB   41h,42h,'C','D',45h
```

b) Khai báo hằng

Các hằng trong chương trình Assembly được gán tên để làm cho chương trình dễ đọc hơn. Hằng có thể là kiểu số hoặc kiểu ký tự. Việc gán tên cho hằng được thực hiện bằng lệnh giả EQU như sau:

Tên hằng EQU Giá trị của hằng

Ví dụ:

```
CR  EQU  0Dh
LF  EQU  0Ah
CHAO     EQU  'Hello'
```

Vì lệnh giả EQU không dành chỗ của bộ nhớ cho tên hằng nên ta có thể khai báo hằng ở bất kỳ đâu trong chương trình. Tuy nhiên người ta thường đặt các khai báo hằng trong đoạn dữ liệu.

7. Khung của một chương trình Assembly

Một chương trình mã máy trong bộ nhớ thường bao gồm các vùng nhớ khác nhau để chứa mã lệnh, chứa dữ liệu của chương trình và một vùng nhớ được dùng làm ngăn xếp phục vụ hoạt động của chương trình. Chương trình viết bằng ngôn ngữ Assembly cũng phải có cấu trúc tương tự để khi dịch nó sẽ tạo ra mã máy có cấu trúc như trên, tức là đoạn mã lệnh sẽ được dịch và để trong vùng nhớ mã lệnh, đoạn dữ liệu sẽ được dịch và để trong vùng nhớ dữ liệu và đoạn ngăn xếp sẽ được dịch và tạo ra vùng nhớ ngăn xếp cho chương trình.

Trước khi tìm hiểu khung của một chương trình Assembly ta xem xét các khai báo có trong chương trình:

a) Khai báo quy mô sử dụng bộ nhớ

Kích thước bộ nhớ dành cho đoạn mã và đoạn dữ liệu trong một chương trình được xác định bằng lệnh giả .MODEL. Lệnh này phải được đặt trước các lệnh khác trong chương trình nhưng đặt sau lệnh giả khai báo loại CPU. Cú pháp:

.MODEL Kiểu_kích_thước_bộ_nhớ

Kiểu kích thước bộ nhớ	Mô tả
TINY	Mã lệnh và dữ liệu gói gọn trong một đoạn 64 KB
SMALL	Mã lệnh gói gọn trong một đoạn 64 KB. Dữ liệu gói gọn trong

	một đoạn 64 KB
MEDIUM	Mã lệnh không gói gọn trong một đoạn 64 KB Dữ liệu gói gọn trong một đoạn 64 KB
COMPACT	Mã lệnh gói gọn trong một đoạn 64 KB Dữ liệu không gói gọn trong một đoạn 64 KB
LARGE	Mã lệnh không gói gọn trong một đoạn 64 KB Dữ liệu không gói gọn trong một đoạn 64 KB Không có mảng nào lớn hơn 64 KB
HUGE	Mã lệnh không gói gọn trong một đoạn 64 KB Dữ liệu không gói gọn trong một đoạn 64 KB Các mảng có thể lớn hơn 64 KB

b) Khai báo đoạn ngăn xếp

Việc khai báo đoạn ngăn xếp là để dành ra một vùng nhớ đủ lớn dùng làm ngăn xếp phục vụ cho hoạt động của chương trình. Cú pháp:

```
.STACK  Kích_thước
```

Kích_thước quyết định số byte dành cho ngăn xếp. Thông thường với 100 - 256 byte là đủ để làm ngăn xếp và ta có thể khai báo kích thước cho ngăn xếp như sau:

```
.STACK 100
```

hoặc

```
.STACK 100H
```

c) Khai báo đoạn dữ liệu

Đoạn dữ liệu chứa toàn bộ các khai báo biến và hằng của chương trình. Các khai báo trong đoạn dữ liệu đặt sau lệnh giả .DATA

Ví dụ:

```
.DATA
MSG     DB    'Hello!$'
B1      DB    100
CR      EQU   0DH
LF      EQU   0AH
```

d) Khai báo đoạn mã

Đoạn mã chứa mã lệnh của chương trình, tức là các lệnh của chương trình sẽ viết ở đây. Để bắt đầu đoạn mã ta dùng lệnh giả .CODE

Bên trong đoạn mã, các lệnh của chương trình có thể tổ chức thành chương trình chính và chương trình con như sau:

```
.CODE
Tên_CTChính     PROC

    ;Các lệnh của chương trình chính
            .
            .
    CALL Tên_CTCon  ;Gọi chương trình con
            .
            .
Tên_CTChính     ENDP
;Khai báo các chương trình con ở đây
Tên_CTCon       PROC

    ;Các lệnh của chương trình con
```

```
    RET   ;Trở về
Tên_CTCon       ENDP
```

e) Khung chương trình Assembly để dịch ra chương trình .EXE

```
.MODEL   SMALL
.STACK   100H
.DATA
    ;Các khai báo biến và hằng để tại đây
.CODE
MAIN     PROC
    ;Khởi đầu cho đoạn DS
    MOV AX,@DATA
    MOV DS,AX
    ;Các lệnh của chương trình để tại đây
        ......
    ;Trở về DOS dùng hàm 4CH của INT 21H
    MOV AH,4CH
    INT 21H
MAIN     ENDP
    ;Các chương trình con (nếu có) khai báo tại đây

END MAIN        ;Kết thúc toàn bộ chương trình
```

Dòng cuối cùng của chương trình ta dùng lệnh giả END và tiếp theo là MAIN để kết thúc toàn bộ chương trình. Ta có nhận xét rằng MAIN là tên của chương trình chính nhưng về thực chất nó là nơi bắt đầu các lệnh của chương trình trong đoạn mã lệnh.

Khi một chương trình EXE được nạp vào bộ nhớ, DOS sẽ tạo ra một mảng gồm 256 byte làm *đoạn mào đầu chương trình* (Program Segment Prefix, PSP) dùng để chứa các thông tin liên quan đến chương trình và đặt nó vào ngay phía trước phần bộ nhớ chứa mã lệnh của chương trình. Trong khi đưa các thông số liên quan đến chương trình vào PSP, DOS đã sử dụng đến các thanh ghi DS và ES. Do vậy DS và ES không chứa giá trị địa chỉ của đoạn dữ liệu của chương trình. Để chương trình có thể chạy đúng ta phải có các lệnh khởi tạo cho thanh ghi đoạn DS (hoặc cả ES nếu cần) để chứa địa chỉ đoạn dữ liệu của chương trình.

```
MOV AX,@DATA
MOV DS,AX
;MOV ES,AX      ;Nếu cần
```

trong đó @DATA là địa chỉ của đoạn dữ liệu.

Ví dụ: Chương trình hiện lên màn hình dòng chữ CHAO CAC BAN

```
.MODEL SMALL
.STACK   100H
.DATA
    CRLF        DB    13,10,'$'
    CHAO        DB    'CHAO CAC BAN!$'
.CODE
MAIN     PROC
;Khoi tao thanh ghi DS
    MOV AX,@DATA
    MOV DS,AX
    ;Xuong dong moi
```

```
        MOV AH,9
        LEA DX,CRLF
        INT 21H
        ;Hien thi loi chao
        MOV AH,9
        LEA DX,CHAO
        INT 21H
        ;Xuong dong moi
        MOV AH,9
        LEA DX,CRLF
        INT 21H
        ;Tro ve DOS
        MOV AH,4CH
        INT 21H
MAIN    ENDP
END MAIN
```

f) Khung chương trình Assembly để dịch ra chương trình .COM

Chương trình đuôi .COM ngắn gọn và đơn giản hơn nhiều so với chương trình đuôi .EXE. Trong chương trình đuôi .COM, đoạn mã, đoạn dữ liệu và đoạn ngăn xếp được gộp lại trong một đoạn duy nhất là đoạn mã. Việc tạo ra tệp này không những tiết kiệm được thời gian và bộ nhớ khi chạy chương trình mà còn tiết kiệm được cả không gian nhớ khi lưu trữ chương trình trên ổ đĩa.

Khung của chương trình Assembly để dịch ra đuôi .COM như sau:

```
.MODEL TINY
.CODE
    ORG   100H
START: JMP CONTINUE
    ;Các khai báo biến và hằng để tại đây

CONTINUE:
MAIN    PROC
    ;Các lệnh của chương trình chính để tại đây

    ;Trở về DOS
    INT 20H
MAIN    ENDP
    ;Các chương trình con (nếu có) khai báo ở đây

END START
```

Ta nhận thấy ở ngay đầu đoạn mã là lệnh giả ORG 100H dùng để gán địa chỉ bắt đầu của chương trình tại 100h trong đoạn mã, chừa lại vùng nhớ 256 byte (từ địa chỉ 0 đến 255) cho đoạn mào đầu chương trình (PSP).

Lệnh JMP sau nhãn START dùng để nhảy qua toàn bộ phần bộ nhớ dành cho việc khai báo dữ liệu. Đích của lệnh nhảy này là đầu chương trình chính.

Khi kết thúc chương trình COM, để trở về DOS ta dùng ngắt INT 20H của DOS để làm cho chương trình gọn hơn.

Để kết thúc toàn bộ chương trình ta dùng lệnh giả END theo sau là nhãn START, vì START tương ứng với địa chỉ lệnh đầu tiên của chương trình trong đoạn mã.

Ví dụ: Chương trình hiện lên màn hình dòng chữ CHAO CAC BAN

```
.MODEL TINY
.CODE
    ORG 100H
START: JMP CONTINUE
    CRLF        DB    13,10,'$'
    CHAO        DB    'CHAO CAC BAN!$'
CONTINUE:
MAIN    PROC
    ;Xuong dong moi
    MOV AH,9
    LEA DX,CRLF
    INT 21H
    ;Hien thi loi chao
    MOV AH,9
    LEA DX,CHAO
    INT 21H
    ;Xuong dong moi
    MOV AH,9
    LEA DX,CRLF
    INT 21H
    ;Tro ve DOS
    INT 20H
MAIN    ENDP
END START
```

II – BIÊN SOẠN VÀ DỊCH CHƯƠNG TRÌNH ASSEMBLY

Để viết và dịch các chương trình Assembly ta theo các bước sau:

Bước 1: Soạn thảo chương trình

Dùng các phần mềm soạn thảo văn bản dạng TEXT (như NC, PASCAL, C) để tạo ra tệp văn bản chương trình Assembly. Sau đó ghi tệp chương trình Assembly ra đĩa với đuôi .ASM

Bước 2: Dịch chương trình sang ngôn ngữ máy

Dùng chương trình dịch MASM để dịch tệp chương trình đuôi .ASM sang mã máy dưới dạng tệp đuôi .OBJ. Nếu trong bước này chương trình có lỗi về cú pháp thì chương trình dịch sẽ báo lỗi và ta phải quay lại Bước 1 để sửa.

Cách làm như sau: giả sử chương trình MASM nằm trên thư mục gốc ổ C, dấu nhắc của DOS là C:\>, khi đó từ dấu nhắc của DOS gõ

MASM Tên tệp chương trình; ↵

Dấu chấm phẩy sau tên tệp chương trình để báo cho MASM chỉ tạo tệp .OBJ, không tạo ra các tệp khác. Tên tệp chương trình có thể gõ đủ cả đuôi .ASM hoặc không gõ cũng được.

Bước 3: Liên kết các tệp đuôi .OBJ để tạo thành một tệp chương trình chạy được đuôi .EXE

Cách làm như sau: giả sử chương trình liên kết LINK nằm trên thư mục gốc ổ C, dấu nhắc của DOS là C:\>, khi đó từ dấu nhắc của DOS ta gõ lệnh

LINK Têntệp1 + Têntệp2 + …;↵

Chương trình liên kết sẽ lấy tên tệp đầu tiên (Têntệp1) để đặt tên cho tệp đuôi .EXE cuối cùng. Dấu chấm phẩy sau cùng để báo cho chương trình LINK không hỏi tên các tệp.

Bước 4: Nếu chương trình viết để dịch ra đuôi .COM thì ta phải dùng chương trình EXE2BIN của DOS để dịch tiếp tệp .EXE ra tệp chương trình chạy được đuôi .COM

Cách làm như sau: giả sử chương trình EXE2BIN nằm trên thư mục gốc ổ C, dấu nhắc của DOS là C:\>, khi đó từ dấu nhắc của DOS ta gõ lệnh

EXE2BIN Têntệp.EXE Têntệp.COM ↵

Chú ý: Với trình biên dịch MASM phiên bản 6.x trở lên, nếu chỉ biên dịch và liên kết một tệp chương trình thì ta có thể gộp Bước 2, 3, 4 thành một bước bằng cách dùng chương trình kết hợp cả biên dịch và liên kết ML. Tại dấu nhắc của DOS gõ vào như sau:

ML Têntệp1.ASM ↵

III - THỰC HIỆN CÁC CẤU TRÚC ĐIỀU KHIỂN CHƯƠNG TRÌNH BẰNG ASSEMBLY

Thông thường khi lập trình chúng ta cần đến các cấu trúc điều khiển chương trình sau:

+ Cấu trúc rẽ nhánh IF-THEN
+ Cấu trúc lựa chọn CASE
+ Cấu trúc lặp với số lần xác định FOR-DO
+ Cấu trúc lặp với số lần không xác định REPEAT-UNTIL và WHILE-DO

Để thực hiện các cấu trúc điều khiển chương trình này chúng ta sử dụng các lệnh nhẩy và lệnh lặp của bộ vi xử lý. Cụ thể như sau:

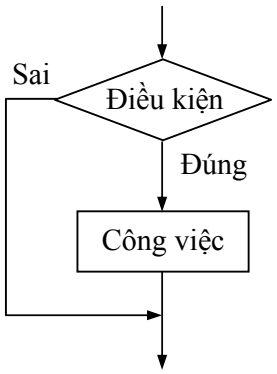
1. Cấu trúc rẽ nhánh IF-THEN

Có hai dạng

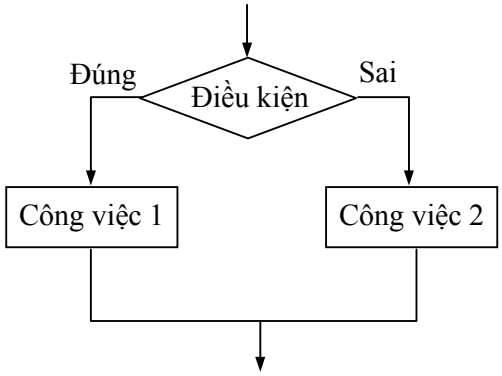
(1) IF <Điều kiện> THEN <Công việc>

(2) IF <Điều kiện> THEN <Công việc 1> ELSE <Công việc 2>

Hoạt động của cấu trúc này có thể diễn tả bằng lưu đồ sau



Cấu trúc IF-THEN



Cấu trúc IF-THEN-ELSE

Ví dụ: 1. Viết đoạn chương trình gán BX ← |AX|

Giải

Dùng cấu trúc IF để kiểm tra nếu AX < 0 thì đổi dấu. Sau cấu trúc IF dùng lệnh MOV để đưa AX vào BX.

```
IF_:
   CMP AX,0
   JAE ENDIF_
   NEG AX
ENDIF_:
MOV BX,AX
```

2. Giả sử AL và BL chứa mã ASCII của ký tự. Kiểm tra nếu AL<=BL thì hiện ra màn hình ký tự trong AL, còn không thì hiện ký tự trong BL.

Giải

Thuật giải
IF AL<=BL THEN
    Hiện ký tự trong AL
ELSE
    Hiện ký tự trong BL
ENDIF

Đoạn chương trình như sau:

```
IF_:
   CMP AL,BL
   JA ELSE_
   MOV AH,2
   MOV DL, AL
   INT 21H
   JMP ENDIF_
ELSE_:
   MOV AH,2
   MOV DL, BL
   INT 21H
ENDIF_:
```

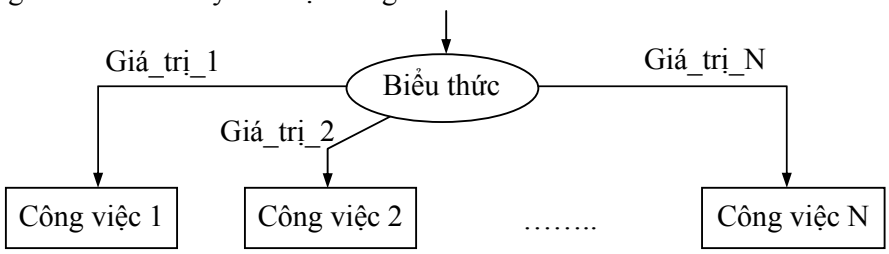
2. Cấu trúc lựa chọn CASE

Khác với cấu trúc rẽ nhánh IF-THEN chỉ rẽ được hai nhánh, cấu trúc lựa chọn có thể rẽ nhiều nhánh. Dạng tổng quát như sau:

```
CASE <Biểu thức> OF
   Giá_trị_1: Công_việc_1
   Giá_trị_2: Công_việc_2


   Giá_trị_N: Công_việc_N
END_CASE
```

Hoạt động của cấu trúc này thể hiện bằng lưu đồ sau:



Ví dụ: Nếu AX chứa số âm thì đưa -1 vào BX, nếu AX chứa 0 thì đưa 0 vào BX, nếu AX chứa số dương thì đưa 1 vào BX.

Giải

Đoạn chương trình như sau:

```
CASE_:
      CMP AX,0
      JL AM
      JE KHONG
      JG DUONG
   AM:
```

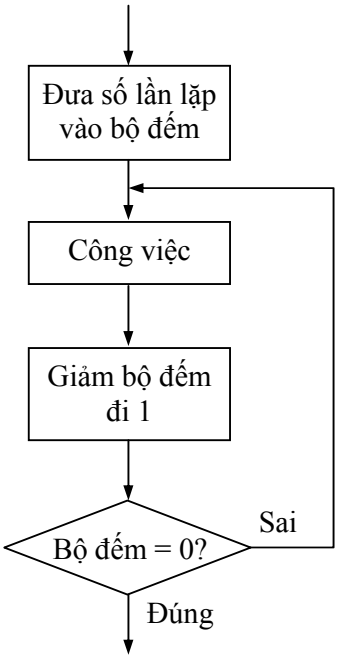
```
    MOV BX,-1
    JMP ENDCASE_
KHONG:
    MOV BX,0
    JMP ENDCASE_
DUONG:
    MOV BX,1
ENDCASE_:
```

3. Cấu trúc lặp với số lần xác định FOR-DO

Dạng tổng quát

```
FOR <Số lần lặp> DO
    <Công việc>
ENDFOR
```

Hoạt động của cấu trúc này thể hiện bằng lưu đồ sau:



Ví dụ: Hiện thị một dòng 80 ký tự '$'
Đoạn chương trình như sau:

```
FOR_:
    MOV CX,80
    MOV AH,2
    MOV DL,'$'
    INT 21H
    LOOP FOR_
ENDFOR_:
```

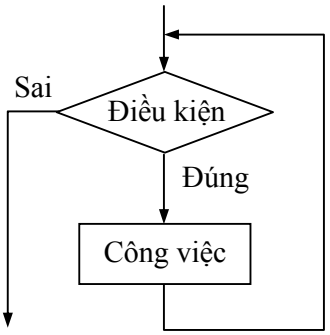
4. Cấu trúc lặp với số lần không xác định

a) Kiểm tra điều kiện trước

Dạng tổng quát

```
WHILE <Điều kiện> DO <Công việc>
```

Hoạt động của cấu trúc này được thể hiện qua lưu đồ sau:



Cấu trúc WHILE-DO

Ví dụ: Đếm số ký tự đọc được từ bàn phím,khi gặp ký tự CR thì thôi.
Đoạn chương trình như sau:

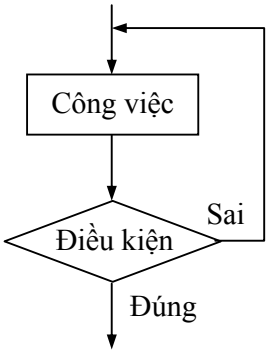
```
MOV CX,-1       ;Chứa số ký tự đếm được
MOV AL,'$'      ;Để AL khác ký tự CR
WHILE_:
   CMP AL,0DH       ;Có khác ký tự CR không?
   JE ENDWHILE_     ;Không khác, tức bằng CR thì thoát
   INC CX
   MOV AH,1
   INT 21H
   JMP WHILE_
ENDWHILE_:
```

b) Kiểm tra điều kiện sau

Dạng tổng quát

```
REPEAT <Công việc> UNTIL <Điều kiện>
```

Hoạt động của cấu trúc này được thể hiện qua lưu đồ ở trên:



Cấu trúc REPEAT-UNTIL

Ví dụ: Đếm số ký tự đọc được từ bàn phím, khi gặp ký tự CR thì thôi.

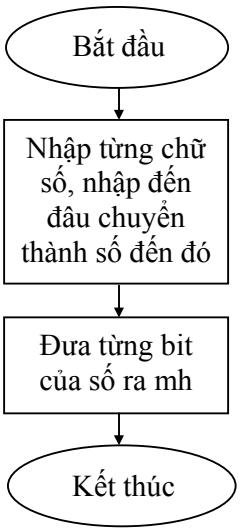
Đoạn chương trình như sau:

```
MOV CX,0          ;Chứa số ký tự đếm được
REPEAT_:
   MOV AH,1
   INT 21H
   CMP AL,0DH     ;Có bằng ký tự CR không?
   JE UNTIL_      ;Bằng CR
   INC CX
   JMP REPEAT_
UNTIL_:
```

IV - MỘT SỐ CHƯƠNG TRÌNH CỤ THỂ

Viết chương trình assembly cho máy tính cá nhân nhập vào một số hệ 10 trong khoảng -32768 đến 32767, sau đó hiện số nhập vào ra màn hình ở dạng nhị phân.

Lưu đồ thuật giải:



Chương trình hoàn chỉnh:

```
;Nhap tung so, nhap den dau chuyen thanh so den do
.MODEL  SMALL
HienXau MACRO xau
        PUSH AX
        PUSH DX
        LEA DX,xau
        MOV AH,09H
        INT 21H
        POP DX
        POP AX
        ENDM
.STACK 100H
.DATA
   TB1 DB 'Nhap vao mot so thap phan: $'
   TB2 DB 13,10,'Dang nhi phan la: $'
.CODE
  MAIN PROC
```

```
  ;Khoi tao thanh ghi DS
  MOV AX,@DATA
  MOV DS,AX
  ;Xoa man hinh
  MOV AH,0FH
  INT 10H
  MOV AH,0
  INT 10H
  ;Hien thong bao nhap so thap phan
  HIENXAU TB1
  ;Nhap tung so, nhan so tinh duoc truoc do voi 10 roi cong voi so vua nhap
  XOR BX,BX       ;Xoa BX = 0, BX dung de chua so he 10 nhap vao
  MOV CX,10       ;So nhan
  MOV DI,0        ;Ghi nho ve dau: DI=0 la duong, DI=1 la am
  NHAP:
  MOV AH,1
  INT 21H
  CMP AL,13
  JE THOI         ;Neu an Enter thi thoat
  CMP AL,'-'
  JNE TIEP
  MOV DI,1
  JMP NHAP
  TIEP:SUB AL,30H       ;Chuyen chu so thanh so
  XOR AH,AH
  PUSH AX               ;Dua so vua nhap vao ngan xep
  MOV AX,BX             ;Dua so tinh duoc truoc do vao AX de nhan voi 10
  MUL CX                ;Nhan so tinh duoc truoc do voi 10
  MOV BX,AX             ;Dua tich sang BX
  POP AX                ;Lay lai so de trong ngan xep
  ADD BX,AX             ;Cong so do voi BX roi de ket qua trong BX
  JMP NHAP              ;Quay lai nhap so tiep theo
  THOI:
  ;Kiem tra xem co nhap vao dau - hay khong
  CMP DI,0
  JE DUARA        ;Khong nhap vao dau tru
  NEG BX          ;Co nhap dau -

  ;Dua so he 10 ra man hinh duoi dang he 2
  DUARA:
  HIENXAU TB2
  CALL HIENBIN    ;Goi chuong trinh con
  ;Tro ve DOS
  MOV AH,4CH
  INT 21H
MAIN ENDP
;Khai bao chuong trinh con
HIENBIN PROC
  ;Cat cac thanh ghi de khong bi anh huong
  PUSH AX
  PUSH BX
  PUSH CX
```

```
   PUSH DX

   MOV CX,16
   MOV AH,2
   MOV DH,0
   HIEN:
   XOR DL,DL
   ROL BX,1
   ADC DL,30H
   INT 21H
   INC DH
   CMP DH,4
   JNE HIENTIEP
   MOV AH,2
   MOV DL,' '
   INT 21H
   MOV DH,0
   HIENTIEP:
   LOOP HIEN
   ;Lay lai noi dung cua cac thanh ghi da cat
   POP DX
   POP CX
   POP BX
   POP AX
   RET          ;Tro ve chuong trinh chinh
 HIENBIN ENDP
END MAIN
```