CƠ BẢN VỀ LẬP TRÌNH HỢP NGỮ

1. CÚ PHÁP LỆNH HỢP NGỮ

Một chương trình hợp ngữ bao gồm một loạt các mệnh đề (statement) được viết liên tiếp nhau, mỗi mệnh đề được viết trên 1 dòng

Một mệnh đề có thể là:

- Một lệnh (Instruction): Được trình biên dịch chuyển thành mã máy
- Một chỉ dẫn của Assembler: Không được chuyển thành mã máy.

Một mệnh đề của Assembler (ASM) bao gồm 4 trường:

Name	Operation	Operand(s)	Comment

Ví dụ 1 mệnh đề:

START: MOV CX,5 ;khởi tạo thanh ghi CX

Ví dụ chỉ dẫn của ASM

Main Proc ;tạo một thủ tục có tên là MAIN

CƠ BẢN VỀ LẬP TRÌNH HỢP NGỮ

1.1. Tên (Name Field)

Dùng cho nhãn lệnh, tên thủ tục và tên biến

Tên này sẽ được ASM chuyển thành địa chỉ bộ nhớ

Cách đặt tên:

- Có thể dài từ 1 - 31 ký tự
- Trong tên chứa các ký tự từ a-z, các số
- Có thể chứa các ký tự đặc biệt như hỏi chấm (?), @, gạch dưới (_), dấu chấm (.)
- Dấu chấm không đặt bên trong của tên
- Trong tên không có dấu cách, không bắt đầu bằng số
- Tên phân biệt chữ HOA và chữ thường

CƠ BẢN VỀ LẬP TRÌNH HỢP NGỮ

1.1. Tên (Name Field)

Tên hợp lệ	Tên không hợp lệ
COUNTER1	TOW WORDS
@CHARACTER	2ABC
SUM_OF_DIGITS	A45.28
DONE?	YOU&ME
.TEST	ADD-REPEAT

1.2. Toán tử (Operation)

Đối với 1 lệnh trường toán tử chứa ký hiệu của mã phép toán (operation code = OPCODE) ASM sẽ chuyển ký hiệu mã phép toán thành mã máy.

Thông thường ký hiệu mã phép toán mô tả chức năng của phép toán, ví dụ ADD, SUB, INC, DEC, INT,...

Đối với chỉ dẫn của ASM, trường toán tử chứa một opcode giả (pseudo operation code = pseudo-op) ASM không chuyển thành mã máy mà hướng dẫn ASM thực hiện một việc gì đó như tạo ra một thủ tục, định nghĩa các biến,...

1.3. Toán hạng (Operand(s))

Trong một lệnh trường toán hạng chỉ ra các số liệu tham gian trong lệnh đó.

Một lệnh có thể không có toán hạng, 1 toán hạng hoặc 2 toán hạng.

Ví dụ:

```
NOP             ;không có toán hạng
INC AX          ;1 toán hạng
ADD WORD1,2     ;2 toán hạng cộng 2 với nội dung của từ nhớ WORD1
```

Trong các lệnh 2 toán hạng, toán hạng đầu là toán hạng đích, đó thường là thanh ghi hoặc vị trí nhớ dùng để lưu trữ kết quả. Toán hạng thứ 2 là toán hạng nguồn, toán hạng nguồn thường không bị thay đổi sau khi thực hiện lệnh

1.4. Chú thích (Comment)

Trường chú thích là 1 tùy chọn của mênh đề trong ngôn ngữ ASM được lập trình viên dùng để thuyết minh về câu lệnh

Một chú thích được bắt đầu bằng dấu chấm phẩy (;)

Dòng chú thích cũng có thể được dùng để ngăn cách các phần khác nhau của chương trình.

Ví dụ:

```
;
;Khởi tạo các thanh ghi
;
MOV AX,0
MOV BX,0
```

2. Một số quy ước khi lập trình hợp ngữ

Khi giới thiệu các câu lệnh viết bằng hợp ngữ, các câu lệnh cần được bao quát tất cả các trường hợp do đó có một số qui ước khi thiết lập cú pháp các lệnh như sau:

Tên qui ước	Tên qui ước đại diện cho	Ví dụ Lệnh sử dụng tên qui ước	Ví dụ khi sử dụng
Rn	Các thanh ghi ở các Bank thanh ghi Khi sử dụng thay n bằng các số từ 0 đến 7: R0, R1, R2, R3, R4, R5, R6, R7	Mov A,Rn	Mov A,R2
direct	Ô nhớ có địa chỉ là direct, direct được thay bằng địa chỉ từ 00H đến FFH khi viết chương trình.	Mov A,direct	Mov A,30H

CƠ BẢN VỀ LẬP TRÌNH HỢP NGỮ

2. Một số quy ước khi lập trình hợp ngữ

Tên qui ước	Tên qui ước đại diện cho	Ví dụ Lệnh sử dụng tên qui ước	Ví dụ khi sử dụng
#data	Dữ liệu 8 bit, khi sử dụng data có thể viết dưới dạng : • số nhị phân (Vd: #00110011b) • số thập lục phân (Vd: #0A6H) • số thập phân (Vd: #21)	Mov A,#data	Mov A,#20H
@Ri	Ô nhớ có địa chỉ gián tiếp, đây là địa chỉ của một ô nhớ, địa chỉ này được xác định gián tiếp bằng giá trị của thanh ghi R0 hoặc R1 (chỉ được sử dụng hai thanh ghi R0 hoặc R1 để lưu giá trị này)	Mov A,@Ri	Mov A,@R1

CƠ BẢN VỀ LẬP TRÌNH HỢP NGỮ

2. Một số quy ước khi lập trình hợp ngữ

- #data: là giá trị cần thiết lập trong một ô nhớ, data được ghi trong chương trình assembly với qui định về cách viết số như ở bên dưới, các số này sau đó được trình biên dịch chuyển thành các số nhị phân tương ứng.

Ví dụ: khi ghi #95H đây là giá trị được thiết lập trong từng bit của ô nhớ.(các bit của ô nhớ có giá trị là 10010101).

- Còn khi ghi 95H thì hiểu đây là ô nhớ có địa chỉ là 95H.
- Đối với các ô nhớ được định tên bằng kí hiệu chẳng hạn P0,P1,A,B,TH0... thì được sử dụng tên đó thay cho địa chỉ cần sử dụng.

Ví dụ: hai lệnh sau đây là như nhau Mov TH0,#43H và Mov 8CH,#43H vì thanh ghi TH0 có địa chỉ là 8CH.

2. Một số quy ước khi lập trình hợp ngữ

Trình biên dịch Assembler cho phép sử dụng các loại số sau trong chương trình:

- Số Binary (số nhị phân): Số nhị phân khi viết cần thêm phía sau giá trị bằng kí tự "B". Các số này phải là số nhị phân 8 bit. Khi giá trị cần thiết lập là các giá trị cần cho từng bit trong byte thì dùng cách biểu diễn bằng số nhị phân

Ví dụ: Khi cần thiết lập giá trị cho một byte mà các bit 0,1 xen kẽ nhau thì nên biểu diễn bằng số 01010101B cho dễ kiểm tra.

2. Một số quy ước khi lập trình hợp ngữ

- **Hexadecimal** (số thập lục phân-ghi tắt là hex): Số hex khi viết cần thêm phía sau giá trị bằng kí tự "H".

Nếu số hex bắt đầu là A,B,C,D,E,F thì cần thêm số "0" phía trước để trình biên dịch nhận biết được đó là số Hex, không lầm giá trị số với các kí tự chữ khác.

Khi sử dụng các giá trị dành riêng cho một công việc nào đó, việc ghi nhớ bằng số nhị phân rất rắc rối và khó nhớ, khi đó số hex được sử dụng, vì số hex là cách viết ngắn gọn của số nhị phân.

Ví dụ: 69H, 0A3H

2. Một số quy ước khi lập trình hợp ngữ

- Số Decimal (số thập phân): Số thập phân khi viết không cần cần thêm kí tự hoặc thêm sau giá trị bằng kí tự "D".

Khi tính toán: cộng trừ nhân chia, nếu sử dụng số nhị phân hoặc số hex sẽ gây khó khăn cho người viết chương trình, trong trường hợp này số thập phân được sử dụng

Ví dụ: 45, 27, 68D

Chú ý: Địa chỉ của các ô nhớ, của các bit nhớ, địa chỉ của ROM luôn được viết bằng số thập lục phân và cũng tuân theo qui tắc viết số như phía trên.

CƠ BẢN VỀ LẬP TRÌNH HỢP NGỮ

2. Một số quy ước khi lập trình hợp ngữ

- Ký tự và một chuỗi các ký tự phải được đặt giữa hai dấu ngoặc đơn hoặc 2 dấu ngoặc kép

Ví dụ: 'A' và "HELLO"

Các ký tự đều được chuyển thành ASCII bởi ASM, do đó khai báo 'A' và 41h là giống nhau

Sau khi chương trình hoàn tất phải kết thúc bằng câu lệnh **END**

Các câu lệnh này báo cho trình biên dịch biết phần kết thúc của chương trình, trình biên dịch bỏ qua tất cả các câu lệnh sau lệnh **END**

CƠ BẢN VỀ LẬP TRÌNH HỢP NGỮ

2. Một số quy ước khi lập trình hợp ngữ

Số	Loại
10111	Thập phân
10111b	Nhị phân
64223	Thập phân
-2183	Thập phân
1B4DH	Hex
1B4D	Số không hợp lệ
FFFFH	Số không hợp lệ
0FFFFH	Hex

3. Các biến trong chương trình hợp ngữ

Vai trò của biến trong ASM cũng giống như trong các ngôn ngữ cấp cao. Mỗi biến có 1 loại dữ liệu và được gán một địa chỉ bộ nhớ sau khi dịch chương trình.

Bảng sau liệt kê các toán tử giả dùng để định nghĩa các loại số liệu:

Toán tử giả	Định nghĩa loại số liệu
DB	Byte
DW	Word (DoubleByte)
DD	DoubleWord
DQ	QuadWord (4 Word liên tiếp)
DT	TenBytes (10 Byte liên tiếp)

3.1. Biến Byte

Cú pháp định nghĩa biến Byte có dạng:

Name DB <Giá trị ban đầu>

Ví dụ:

Alpha DB 4 ;Chỉ dẫn này sẽ gán tên Alpha cho 1 byte nhớ mà giá trị ban đầu của nó là 4

Nếu giá trị của byte là không xác định thì đặt dấu hỏi chấm (?) vào vị trí giá trị ban đầu. Ví dụ: **byt DB ?**

Vùng giá trị mà biến Byte lưu trữ được là -128 đến 127 đối với số có dấu và từ 0 đến 255 đối với số không dấu

3.2. Biến Word

Cú pháp định nghĩa biến Word có dạng:

Name DW <Giá trị ban đầu>

Ví dụ:

wrd DW -2 ;Chỉ dẫn này sẽ gán tên wrd cho 1 word nhớ mà giá trị ban đầu của nó là -2

Nếu giá trị của word là không xác định thì đặt dấu hỏi chấm (?) vào vị trí giá trị ban đầu. Ví dụ: **wrd DW ?**

Vùng giá trị mà biến Word lưu trữ được là -32768 đến 32767 đối với số có dấu và từ 0 đến 56535 đối với số không dấu

3.3. Biến mảng (Arrays)

Trong ASM, một mảng là một loạt các byte hoặc word nhớ liên tiếp nhau.

Ví dụ để định nghĩa 1 mảng 3 byte có tên Byt_Array mà giá trị ban đầu của nó là 10h, 20h, 30h thì ta có thể viết:

Byt_Array DB 10h,20h,30h

- Byt_Array là tên được gán cho byte đầu tiên
- Byt_Array + 1 là tên của byte thứ 2
- Byt_Array + 2 là tên của byte thứ 3

3.3. Biến mảng (Arrays)

Giả sử ASM gán địa chỉ OFFSET là 0200h cho mảng Byt_Array thì nội dung nhớ sẽ như sau:

Symbol	Address	Contents
Byt_Array	200h	10h
Byt_Array + 1	201h	20h
Byt_Array + 2	202h	30h

3.3. Biến mảng (Arrays)

Ví dụ có 1 chỉ dẫn định nghĩa 1 mảng 4 phần tử có tên W_Array:

W_Array DW 1000,40,29887,329

Giả sử mảng bắt đầu tại địa chỉ 0300h thì bộ nhớ sẽ như sau:

Symbol	Address	Contents
W_Array	300h	1000d
W_Array + 2	302h	40d
W_Array + 4	304h	29887d
W_Array + 6	306h	329d

3.3. Biến mảng (Arrays)

**** Byte thấp và byte cao của Word***

Đôi khi chúng ta cần truy xuất tới byte thấp và byte cao của 1 biến Word, giả sử chúng ta định nghĩa:

WORD1 DW 1234h

Byte thấp của WORD1 chứa 34h, còn byte của WORD1 chứa 12h

Ký hiệu địa chỉ của byte thấp là WORD1 còn địa chỉ của byte cao là WORD1 +1

3.3. Biến mảng (Arrays)

**** Chuỗi các ký tự (character strings)***

Một mảng các mã ASCII có thể được định nghĩa bằng 1 chuỗi các ký tự.

Ví dụ: LETTERS DW 41h,42h,43h

tương đương LETTERS DW 'ABC'

Bên trong 1 chuỗi, ASM sẽ phân biệt chữ hoa và chữ thường. Vì vậy chuỗi 'abc' sẽ được chuyển thành 3byte: 61h,62h,63h

Trong ASM cũng có thể tổ hợp các ký tự và các số trong một định nghĩa, ví dụ: MSG DB 'HELLO',0AH,0DH,'$'

tương đương MSG DB 48H,45H,4CH,4CH,4FH,0AH,0DH,24H

3.4. Các hằng (constants)

Trong một chương trình các hằng có thể đặt tên nhờ chỉ dẫn EQU (equates). Cú pháp của EQU là:

NAME EQU constants

Ví dụ: LF EQU 0AH

Sau khi có khai báo trên thì LF được dùng thay cho 0AH trong chương trình, vì vậy ASM sẽ chuyển các lệnh:

MOV DL,0AH

và MOV DL,LF thành cùng 1 mã máy

3.4. Các hằng (constants)

Cũng có thể dùng EQU để định nghĩa một chuỗi, ví dụ:

```
PROMPT EQU 'TYPE YOUR NAME'
```

Sau khi có khai báo này, thay cho

```
MSG DB 'TYPE YOUR NAME'
```

chúng ta có thể viết là:

```
MSG DB PROMPT
```

4. Các lệnh cơ bản

4.1. Lệnh MOV

Lệnh MOV dùng để chuyển số liệu giữa các thanh ghi, giữa 1 thanh ghi và một vị trí nhớ hoặc để di chuyển trực tiếp một số đến một thanh ghi hoặc 1 vị trí nhớ.

Cú pháp của lệnh MOV là:

MOV Destination,Source

Ví dụ::

```
MOV  AX,WORD1   ;lấy nội dung từ WORD1 đưa vào thanh ghi AX
MOV  AX,BX      ;AX lấy nội dung của BX , BX không thay đổi
MOV  AH,'A'     ;AX  lấy giá trị 41h
```

4. Các lệnh cơ bản

4.2. Lệnh XCHG

Lệnh XCHG (Exchange) dùng để trao đổi nội dung giữa 2 thanh ghi, giữa 1 thanh ghi và một vị trí nhớ.

Cú pháp của lệnh MOV là:

XCHG Destination,Source

Ví dụ::

```
XCHG  AH,BL          ;trao đổi nội dung của 2 thanh ghi AH và BL
XCHG  AX,WORD1;trao đổi nội dung của AX với WORD1
```

4. Các lệnh cơ bản

4.3. Lệnh ADD, SUB

Lệnh ADD, SUB dùng để cộng, trừ nội dung giữa 2 thanh ghi, giữa 1 thanh ghi và một vị trí nhớ hoặc cộng (trừ) một số với (khỏi) một thanh ghi hoặc một vị trí nhớ

Cú pháp của lệnh là:

ADD Destination,Source

SUB Destination,Source

Ví dụ::

```
ADD  WORD1,AX
SUB  AX,DX                ;AX = AX - DX
```

4. Các lệnh cơ bản

4.3. Lệnh ADD, SUB

Ghi chú: Việc cộng hoặc trừ trực tiếp giữa 2 vị trí nhớ là không đươck phép. Để giải quyết vấn đề này người ta phải di chuyển byte (word) nhớ đến một thanh ghi mới sau đó mới thực hiện cộng hoặc trừ

Ví dụ:

```
MOV AL,BYTE2
ADD BYTE1,AL
```

4. Các lệnh cơ bản

4.4. Lệnh INC, DEC

Lệnh INC, DEC dùng để cộng thêm, trừ bớt nội dung của thanh ghi hoặc một vị trí nhớ

Cú pháp của lệnh là:

INC Destination

DEC Destination

Ví dụ::

```
INC  WORD1
INC AX
DEC BL
```

4. Các lệnh cơ bản

4.5. Lệnh NEG

Lệnh NEG để đổi dấu (lấy bù 2) của 1 thanh ghi hoặc một vị trí nhớ.

Cú pháp của lệnh là:

NEG Destination

Ví dụ: Giả sử AX=0002h

NEG AX

Sau khi thực hiện lệnh ta có AX=FFFEH

Lưu ý: 2 toán hạng trong các lệnh trên phải cùng loại (cùng là byte hoặc word)

5. Ví dụ sử dụng các lệnh trong ASM

Giả sử A và B là 2 biến WORD, chúng ta sẽ chuyển các mệnh đề sau ra ngôn ngữ của ASM:

- Mệnh đề B=A:

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

- Mệnh đề A=5-A

```
        mov AX,5        ;đưa 5 vào AX
        SUB AX,A        ;AX=5-A
        mov A,AX        ;A=5-A
hoặc    NEG A           ;A=-A
        ADD A,5         ;A=5-A
```

CƠ BẢN VỀ LẬP TRÌNH HỢP NGỮ

5. Ví dụ sử dụng các lệnh trong ASM

- Mệnh đề A=B-2*A:

```
mov AX,B        ;AX=B
sub AX,A        ;AX=B-A
sub AX,A        ;AX=B-2*A
mov A,AX        ;A=B-2*A
```

CƠ BẢN VỀ LẬP TRÌNH HỢP NGỮ

6. Cấu trúc của một chương trình hợp ngữ

Một chương trình ngôn ngữ máy bao gồm mã (code), số liệu (data) và ngăn xếp (stack).

Mỗi 1 phần chiếm một đoạn bộ nhớ, mỗi 1 đoạn chương trình được chuyển thành 1 đoạn bộ nhớ bởi ASM

6. Cấu trúc của một chương trình hợp ngữ

6.1. Các kiểu bộ nhớ (memory models)

Độ lớn của mã và số liệu trong một chương trình được quy định bởi chỉ dẫn của MODEL nhằm xác định kiểu bộ nhớ dùng trong chương trình.

Cú pháp của chỉ dẫn MODEL là:

.MODEL memory_model

Kiểu MODEL	Mô tả
SMALL	code và data nằm trong một đoạn
MEDIUM	code nhiều hơn 1 đoạn, data trong 1 đoạn
COMPACT	data nhiều hơn 1 đoạn, code trong 1 đoạn
LARGE	code và data nhiều hơn 1 đoạn, array không quá 64KB
HUGE	code và data nhiều hơn 1 đoạn, array lớn hơn 64KB

6. Cấu trúc của một chương trình hợp ngữ

6.2. Đoạn số liệu

Đoạn số liệu của chương trình chứa các khai báo biến, khai báo hằng,... Để bắt đầu đoạn số liệu chúng ta dùng chỉ dẫn DATA với cú pháp khai báo như sau:

```
.DATA          ;bắt đầu bằng .DATA
        ;sau đó là phần khai báo biến, hằng, mảng
Ví dụ: .DATA
            WORD1 DW 2
            WORD2 DW 5
            msg DB 'This is a message'
            mask EQU 10010010B
```

6. Cấu trúc của một chương trình hợp ngữ

6.3. Đoạn ngăn xếp

Mục đích của việc khai báo đoạn ngăn xếp là dành 1 vùng bộ nhớ (vùng stack) để lưu trữ cho stack.

Cú pháp của lệnh như sau:

.STACK size

Nếu không khai báo size thì 1KB sẽ được dành cho vùng stack

Ví dụ:

```
.stack 100h    ;dành 256 bytes cho vùng stack
```

6. Cấu trúc của một chương trình hợp ngữ

6.4. Đoạn mã

Đoạn mã chứa các lệnh của chương trình, bắt đầu bằng chỉ dẫn .CODE

Bên trong đoạn mã các lệnh thường được tổ chức thành thủ tục (procedure) với cấu trúc như sau:

```
Ten_thu_tuc PROC
        ;thân của thủ tục
Ten_thu_tuc ENDP
```

6. Cấu trúc của một chương trình hợp ngữ

6.4. Đoạn mã

Một chương trình hợp ngữ thường có cấu trúc:

```
.model SMALL
.stack 100h
.data
        ;định nghĩa số liệu tại đây
.code
main PROC
        ;thân của main
main ENDP
;các thủ tục khác nếu có
end
```

6.5. *Chương trình đầu tiên*

- Chương trình đọc 1 ký tự từ bàn phím và in ký tự đó ra ở dòng dưới

```
.model small
.stack 100h
.code
main proc
    ;Hien thi dau nhac
    mov ah,2
    mov dl,'?'
    int 21h

    ;Nhap 1 ky tu
    mov ah,1     ;ham doc ky tu
    int 21h      ;ky tu duoc dua vao al
    mov bl,al

    ;Nhay den dong moi
    mov ah,2     ;ham xuat 1 ky tu
    mov al,0dh   ;thuc hien CR
    mov dl,0ah   ;thuc hien LF
    int 21h

    ;Xuat ky tu
    mov DL,BL    ;Dua ky tu vao DL
    int 21h

    ;Tro ve DOS
    mov ah,4ch   ;ham thoat ve DOS
    int 21h
main endp
end mainp
```

7. Các lệnh vào ra với INT (INTerrupt)

CPU thông tin với các thiết bị ngoại vi thông qua các cổng IO. Lệnh IN và OUT của CPU cho phép truy xuất đến các cổng này.

Tuy nhiên hầu hết các ứng dụng không dùng lệnh IN và OUT vì:

- Các địa chỉ cổng thay đổi theo từng loại máy tính
- Có thể lập trình cho các IO dễ dàng hơn nhờ các chương trình con (routine) được cung cấp bởi các hãng chế tạo máy tính.

Có 2 loại chương trình phục vụ IO là các routine của BIOS và rountine của DOS

7. Các lệnh vào ra với INT (INTerrupt)

INT (interrupt) là lệnh dùng để gọi các chương trình con của BIOS và DOS.

Cú pháp của lệnh:

INT interrupt_number

Ở đây, interrupt_number là số chỉ định 1 routine

Lệnh INT phổ biến nhất được dùng là 21h, đây là lệnh thường dùng để gọi một số các hàm của DOS để nhập vào hoặc in ra ký tự.

7. Các lệnh vào ra với INT (INTerrupt)

*** Hàm 01h:** Nhập vào 1 ký tự từ bàn phím và hiện ký tự vừa nhập ra màn hình. Nếu không có ký tự nhập thì hàm sẽ đợi cho đến khi nhập.

Khi sử dụng:

- Gọi AH=01h
- Sau khi gọi, AL sẽ chứa mã ASCII của ký tự nhập

Mã ASM:

```
MOV AH,01H
INT 21h
```

7. Các lệnh vào ra với INT (INTerrupt)

* **Hàm 02h:** Xuất ra 1 ký tự trong thanh ghi DL ra màn hình tại vị trí hiện hành của con trỏ.

Khi sử dụng:

- Gọi AH=02h
- Sau khi gọi, DL sẽ chứa mã ASCII của ký tự sẽ hiển thị

Mã ASM:

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

Đoạn mã trên sẽ hiển thị ký tự A trong thanh ghi DL ra màn hình

7. Các lệnh vào ra với INT (INTerrupt)

*** Hàm 09h:**

- INT 21H, hàm **09h** dùng để hiển thị chuỗi ký tự
- Nhập vào thanh ghi DX địa chỉ của chuỗi để hiển thị
- Phải có ký tự **$** ở cuối chuỗi, tuy nhiên ký tự này không được in ra màn hình.
- Nếu chuỗi chứa ký tự điều khiển thì chức năng tương ứng sẽ được thực hiện.

7. Các lệnh vào ra với INT (INTerrupt)

*** Hàm 09h:**

Ví dụ muốn in lên màn hình chuỗi "HELLO!" thì chuỗi này cần định nghĩa như sau:

```
.DATA
        msg DB 'HELLO!$'
        ...
        MOV AH,09H
        LEA DX,msg
        INT 21h
```

Trong đoạn mã trên, LEA (Load Effective Address) thực hiện đưa địa chỉ offset của biến msg vào DX

7. Các lệnh vào ra với INT (INTerrupt)

*** Hàm 0Ah:**

Nhập và chuỗi ký tự từ bàn phím (tối đa 255 ký tự) và dùng phím ENTER để kết thúc chuỗi.

Khi sử dụng, gọi AH=0Ah thì DX sẽ chứa địa chỉ lưu chuỗi

Chuỗi cần có dạng như sau:

- Byte 0: Số byte tối đa cần đọc (Kể cả ký tự ENTER)
- Byte 1: Số byte đã đọc
- Byte 2: Lưu các ký tự đọc

7. Các lệnh vào ra với INT (INTerrupt)

*** Hàm 0Ah:**

Khi sử dụng hàm 0AH để nhập chuỗi ta khai báo biến:

```
max db 101                ;Byte 0: Số byte tối đa cần nhập là 101
len db 0                  ;Byte 1: Số byte đã nhập, ban đầu 0 có
chuoi db 101 dup ('$')    ;Byte 2: Lưu các ký tự đã nhập
```

Trong đó:

- **max** là ô nhớ chứa độ dài của mảng **chuoi**
- **len** chứa độ dài chuỗi nhập vào (ban đầu là 0)
- **chuoi** chứa chuỗi nhập và từ bàn phím (kết thúc nhập chuỗi bằng phím Enter)

Ghi chú: Mảng chuỗi cần khai báo toàn bộ các phần tử là '$' để đảm bảo tất cả các chuỗi khi nhập vào đều kết thúc là '$' để kết hợp hàm 09H khi in chuỗi

7. Các lệnh vào ra với INT (INTerrupt)

* Hàm 0Ah:

Ta gọi hàm 0AH để nhập chuỗi:

```
mov ah,0ah      ;gọi hàm 0ah
lea dx,max      ;giới hạn số ký tự sẽ nhập vào
int 21h         ;nhập chuỗi ký tự
```

In chuỗi đã nhập bằng hàm 09H:

```
mov ah,09h      ;gọi hàm 09h
lea dx,chuoi    ;đưa chuỗi đã nhập vào dx
int 21h         ;in chuỗi ký tự
```

7. Các lệnh vào ra với INT (INTerrupt)

*** Hàm 0Ah:**

Toán tử dup: Toán tử này thường được dùng để khai báo 1 mảng các phần tử có cùng kiểu dữ liệu.

Khai báo dữ liệu với toán tử dup:

<số lần> DUP (<giá trị>)

Trong khai báo trên, <giá trị> sẽ được lặp lại <số lần>

Ví dụ:

Mang1 db 10 dup(0)

Tương đương với

Mang1 db 0,0,0,0,0,0,0,0,0,0

7. Các lệnh vào ra với INT (INTerrupt)

*** Hàm 4Ch:** Kết thúc chương trình

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

*** Hàm ngắt 10:**

- Xóa màn hình:

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

- Chuyển tọa độ con trỏ:

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

7. Các lệnh vào ra với INT (INTerrupt)

Ví dụ:

```
.model small
.stack 100h
.data
    msg DB "Hello!$"
.code
    mov AX,@data
    mov DS,AX

    lea DX,msg      ;mov DX,OFFSET msg
    mov AH,9
    int 21h

    mov AH,4Ch
    int 21h
end
```

Bài tập áp dụng:

1. Viết chương trình hiện ra 2 dòng chữ:

    Chao mung ban den voi Assembly

    Assembly that de

2. Viết chương trình nhập vào 1 ký tự, xuất ra ký tự đó và ký tự đứng trước và ký tự đứng sau
3. Viết chương trình nhập vào ký tự thường, in ra ký tự HOA tương ứng.
4. Viết chương trình nhập vào ký tự HOA, in ra ký tự thường tương ứng

Đáp án bài 02

```
.model small
.stack
.data
    msg1 db "Hay nhap vao 1 ky tu: $"
    msg2 db 0dh,0ah,"Ky tu da nhap: $"
    msg3 db 0dh,0ah,"Ky tu dung truoc la: $"
    msg4 db 0dh,0ah,"Ky tu dung sau la: $"
    tg db ?
.code
    mov ax,@data
    mov ds,ax

    lea dx,msg1
    mov ah,09h
    int 21h

    mov ah,01h
    int 21h

    mov tg,al

    lea dx,msg2
    mov ah,09h
    int 21h

    mov ah,02h
    mov dl,tg
    int 21h

    lea dx,msg3
    mov ah,09h
    int 21h

    mov ah,02h
    mov dl,tg
    dec dl
    int 21h

    lea dx,msg4
    mov ah,09h
    int 21h

    mov ah,02h
    mov dl,tg
    inc dl
    int 21h

    mov ah,4ch
    int 21h
end
```

```
.model small
.stack 100h
.data
    msg1 db 'Nhap vao ky tu chu thuong: $'
    msg2 db 0dh,0ah,'Chuyen sang ky tu chu HOA la: '
    char db ?,'$'
.code
    main proc
        mov ax,@data
        mov ds,ax

        lea dx,msg1
        mov ah,9
        int 21h

        mov ah,1
        int 21h
        sub al,20h
        mov char,al

        lea dx,msg2
        mov ah,9
        int 21h

        mov ah,4ch
        int 21h
    main endp
end main
```

Chuyển từ ký tự thường sang ký tự HOA tương ứng

8. Cấu trúc các thanh ghi

- Thanh ghi là một vùng nhớ đặc biệt dạng RAM nằm trong CPU, việc thâm nhập các thanh ghi được thực hiện bằng tên của thanh ghi
- Người lập trình thường dùng thanh ghi làm toán hạng thay thế cho biến nhớ, vì vậy làm cho chương trình chạy nhanh hơn vì:
    + Các thanh ghi nằm trên CPU nên dữ liệu lấy nhanh hơn.
    + Vùng nhớ cache là vùng nhớ nằm trên CPU

8.1. Nhóm 1: Thanh ghi cờ

- Người lập trình ASM hay dùng trạng thái các bit cờ làm điều kiện cho các lệnh nhảy có điều kiện.

x	x	x	x	O	D	I	T	S	Z	x	A	x	P	x	C	Cờ
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Bit

+ x: không được định nghĩa.

6 bit cờ trạng thái thể hiện các trạng thái khác nhau của kết quả sau một thao tác nào đó, trong đó 5 bit cờ đầu thuộc byte thấp của thanh cờ là các cờ giống như của bộ vi xử lý 8 bit 8085 của Intel.

+ C hoặc CT (Carry flag): cờ nhớ. CF = 1 khi có nhớ.

+ P hoặc PF (Parity flag): cờ parity. PF phản ánh tính chẵn lẻ(parity) của tổng số bit có trong kết quả. PF = 1 khi tổng số bit 1 trong kết quả là chẵn.

8.1. Nhóm 1: Thanh ghi cờ

+ A hoặc AF (Auxiliary carry flag): cờ nhớ phụ, rất có ý nghĩa khi ta làm việc với các số BCD. AF = 1 khi có nhớ hoặc mượn từ một số BCD thấp (4 bit thấp) sang một số BCD cao (4 bit cao).

+ Z hoặc ZF (Zero flag): cờ rỗng, ZF = 1 khi kết quả bằng 0.

+ S hoặc SF (Sign flag): cờ dấu, SF = 1 khi kết quả âm.

+ O hoặc 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.

Ngoài ra bộ vi xử lí 8088 còn có các cờ điều khiển sau đây:

+ T hoặc TF (Trap flag): cờ bẫy, TF = 1 thì CPU làm việc ở chế độ chạy từng lệnh (chế độ này cần dùng khi cần tìm lỗi trong một chương trình).

+ I hoặc IF (Interrupt enable flag): cờ cho phép ngắt, IF = 1 thì CPU cho phép các yêu cầu ngắt được tác động.

+ D hoặc DF (Direction flag): cờ hướng, DF = 1 khi CPU làm việc với chuỗi kí tự theo thứ tự từ trái sang phải (hay còn gọi D là cờ lùi).

8.2. Nhóm 2: Thanh ghi đa năng (gồm 8 thanh ghi 16 bits)

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL
SI		
DI		
BP		
SP		

+ Trong đó H(high) thể hiện các bit cao, L(low) thể hiện các bit thấp.
+ Trong 4 thanh ghi AX, BX, CX và DX có 3 cách truy cập: truy cập theo 8 bit cao hoặc theo 8 bit thấp hoặc theo cả 16 bit. Các thanh ghi còn lại chỉ có một cách truy cập.

8.2. Nhóm 2: Thanh ghi đa năng (gồm 8 thanh ghi 16 bits)

+ AX (Accumulator, Acc): thanh chứa. Các kết quả của các thao tác thường được chứa ở đây (kết quả của phép nhân, chia). Nếu kết quả là 8 bit thì thanh ghi AL được coi là Acc.
+ BX (Base): thanh ghi cơ sở, thường chứa địa chỉ cở sở của một bảng dùng trong lệnh XLAT.
+ CX (Count): bộ đếm, CX thường được dùng để chứa số lần lặp trong trường hợp các lệnh LOOP, còn CL thường chứa số lần dịch hoặc quay trong các lệnh dịch hay quay thanh ghi.
+ DX (Data): thanh ghi dữ liệu, DX cùng AX tham gia vào các thao tác của phép nhân hoặc chia các số 16 bit. DX còn dùng để chứa địa chỉ của các cổng trong các lệnh vào ra trực tiếp (IN/OUT).
+ SI (Source index): chỉ số gốc hay nguồn, SI chỉ vào dữ liệu trong đoạn dữ liệu DS mà địa chỉ cụ thể đầy đủ tương ứng với DS : SI.

8.2. Nhóm 2: Thanh ghi đa năng (gồm 8 thanh ghi 16 bits)

+ DI (Destination index): chỉ số đích, DI chỉ vào dữ liệu trong đoạn dữ liệu DS mà địa chỉ cụ thể đầy đủ tương ứng với DS : DI.
+ BP (Base pointer): con trỏ cơ sở, BP luôn trỏ vào một dữ liệu nằm trong đoạn ngăn xếp SS. Địa chỉ đầy đủ của một phần tử trong đoạn ngăn xếp ứng với SS : BP.
+ SP (Stack pointer): con trỏ ngăn xếp, SP luôn trỏ vào đỉnh hiện thời của ngăn xếp SS. Địa chỉ đầy đủ của đỉnh ngăn xếp ứng với SS:SP.

Người lập trình chỉ dùng 7 thanh ghi sau: AX, BX, CX, DX, SI, DI, BP.

9. Các lệnh điều khiển

9.1. Lệnh nhảy

Một chương trình thông thường sẽ thực hiện lần lượt các lệnh theo thứ tự mà chúng được viết ra.

Tuy nhiên nhiều trường hợp cần phải chuyển điều khiển đến 1 phần khác của chương trình.

Trong trường hợp đó chúng ta sẽ sử dụng các lệnh nhảy hoặc các lệnh lặp.

Để thực hiện được 1 lệnh nhảy có điều kiện thì CPU phải theo dõi thanh ghi cờ. Nếu điều kiện cho lệnh nhảy là đúng thì CPU sẽ điều chỉnh IP đến vị trí mới được chỉ định. Nếu điều kiện không đúng thì lệnh nhảy sẽ không thực hiện mà chương trình sẽ thực hiện lệnh tiếp theo sẽ thực hiện.

Lệnh nhảy gồm: Nhảy có dấu (dùng diễn dịch có dấu đối với kết quả)
Nhảy không dấu
Nhảy cờ (dùng cho các thao tác chỉ ảnh hưởng lên cờ)

9.1. Lệnh nhảy

* Nhảy có dấu:

Lệnh nhảy	Mô tả	Điều kiện cho lệnh nhảy
JG JNLE	nhảy nếu lớn hơn nhảy nếu không nhỏ hơn hoặc bằng	ZF = 0 và SF = OF
JGE JNL	nhảy nếu lớn hơn hoặc bằng nhảy nếu không nhỏ hơn hay bằng	SF = OF
JL JNGE	nhảy nếu nhỏ hơn nhảy nếu không lớn hơn hoặc bằng	SF <> OF
JLE JNG	nhảy nếu nhỏ hơn hoặc bằng nhảy nếu không lớn hơn	ZF = 1 hoặc SF <> OF

9.1. Lệnh nhảy

* Nhảy có điều kiện không dấu:

Lệnh nhảy	Mô tả	Điều kiện cho lệnh nhảy
JA JNBE	nhảy nếu lớn hơn nhảy nếu không nhỏ hơn hoặc bằng	CF = 0 và ZF = 0
JAE JNB	nhảy nếu lớn hơn hoặc bằng nhảy nếu không nhỏ hơn	CF = 0
JB JNA	nhảy nếu nhỏ hơn nhảy nếu không lớn hơn hoặc bằng	CF = 1
JBE JNA	nhảy nếu nhỏ hơn hoặc bằng nhảy nếu không phải lớn hơn	CF = 1 hoặc ZF = 1

9.1. Lệnh nhảy

* Nhảy 1 cờ:

Lệnh nhảy	Mô tả	Điều kiện cho lệnh nhảy
JE JZ	nhảy nếu bằng nhau nhảy nếu bằng không	ZF = 1
JNE JNZ	nhảy nếu không bằng nhau nhảy nếu không bằng không	ZF = 0
JC	Nhảy nếu có nhớ	CF = 1
JNC	Nhảy nếu không nhớ	CF = 0

9.2. Lệnh nhảy không điều kiện

* Lệnh JMP là một lệnh nhảy không điều kiện

Cú pháp:

JMP <Đích>

Trong đó <Đích> là một nhãn ở trong cùng 1 đoạn với lệnh JMP

9.3. Lệnh so sánh CMP

* Lệnh so sánh CMP (CoMPare) dùng để so sánh toán hạng nguồn với toán hạng đích. Kết quả sẽ không được cất giữ mà thường dùng trong các lệnh nhảy.

Cú pháp:

```
CMP <Đích,Nguồn>
```

Ví dụ:

```
CMP AX,BX
JG BELOW
```

Đoạn lệnh này có nghĩa nếu AX>BX thì nhảy đến nhãn BELOW

```
org 100h
;Thuc hien nhap vao tu ban phim 1 ky tu
;Neu khong phai ky tu so thi nhap lai
;Sau do hien thi ky tu vua nhap o dong tiep theo
.model small
.stack 100h
.data
    msg1 db 'Nhap vao ky tu so: $'
    msg2 db 13,10,'$'
    msg3 db 'Ban da nhap sai!$'
    tg db ?
.code
main proc
    mov ax,@data
    mov ds,ax

    ;Hien thi msg1
    LapMSG1:
    mov ah,09h
    lea dx,msg1
    int 21h

    ;Nhap vao ky tu
    mov ah,01h
    int 21h

    mov tg,al

    mov ah,09h
    lea dx,msg2
    int 21h

    ;Kiem tra ky tu nhap vao
    cmp tg,30h
    jl ThongBao
    cmp tg,39h
    jg ThongBao
    jmp HienThiNhap

    ;Hien thi msg3
    ThongBao:
    mov ah,09h
    lea dx,msg3
    int 21h

    mov ah,09h
    lea dx,msg2
    int 21h

    jmp LapMSG1

    HienThiNhap:
    mov ah,02h
    mov dl,tg
    int 21h

    mov ah,4ch
    int 21h

    main endp
end main
ret
```

```
org 100h

;Nhap 2 ky tu
;Phai co 1 ky tu so
;va 1 ky tu chu thuong

.model small
.stack 100h
.data
    msg1 db 'NHAP VAO 1 KY TU SO VA 1 KY TU CHU THUONG$'
    msg2 db 13,10,'Nhap vao 1 ky tu so: $'
    msg3 db 13,10,'Nhap vao 1 ky tu chu thuong: $'
    msg4 db 13,10,'$'
    msg5 db 13,10,'Ban phai nhap so!$'
    msg6 db 13,10,'Ban phai nhap ky tu chu thuong!$'
    so db ?
    chu db ?
.code
main proc
   mov ax,@data
   mov ds,ax

   mov ah,9
   lea dx,msg1
   int 21h


   Lap1: ;Lap khi nhap sai so
   mov ah,09h
   lea dx,msg2
   int 21h

   mov ah,01h
   int 21h

   mov so,al
```

```
cmp so,30h
jl tbNhapSai1
cmp so,39h
jg tbNhapSai1
jmp InSo

tbNhapSai1:
mov ah,09h
lea dx,msg5
int 21h


jmp Lap1

InSo:
mov ah,09h
lea dx,msg4
int 21h
mov ah,02h
mov dl,so
int 21h


Lap2: ;Lap khi nhap sai chu thuong
mov ah,09h
lea dx,msg3
int 21h

mov ah,01h
int 21h

mov chu,al
```

```
cmp chu,61h
jl tbNhapSai2
cmp chu,7ah
jg tbNhapSai2
jmp InChu

tbNhapSai2:
mov ah,09h
lea dx,msg6
int 21h


jmp Lap2

InChu:
mov ah,09h
lea dx,msg4
int 21h
mov ah,02h
mov dl,chu
int 21h

mov ah,4ch
int 21h
main endp
end main

ret
```

10. Lệnh LOOP

* Cú pháp:

LOOP <Nhãn đích>

Trong đó <Nhãn đích> là một nhãn lệnh đứng trước LOOP và không quá 126 byte

* Tác dụng của lệnh: Khi gặp lệnh này thì chương trình sẽ lặp lại việc thực hiện các lệnh sau <Nhãn đích> đủ n lần được đặt trước trong thanh ghi CX.

Lệnh LOOP thường sử dụng khi có sự lặp đi lặp lại của 1 yêu cầu nào đó.

Sau mỗi lần thực hiện thì CX tự động giảm 1 đơn vị

Lệnh sẽ dừng lại khi CX = 0

Lệnh thường được sử dụng để cài đặt trong các đoạn chương trình lặp với số lần lặp xác định được cho trước trong CX

10. Lệnh LOOP

* Ví dụ:

```
mov ax,1                ;đưa 1 vào AX
mov cx,4                ;nạp số lần lặp vào CX
Lap:
        add ax,1
        loop Lap
```

Sau 4 lần thực hiện lệnh ADD, giá trị trong AX là 1 + 1 + 1 + 1 + 1 = 5

```
;In ra cac so tu 1-9
.model small
.stack 100h
.data
    msg1 db 'IN RA TU 1-9',13,10,'$'
.code
    mov ax,@data
    mov ds,ax

    mov ah,09h
    lea dx,msg1
    int 21h

    mov dl,'0'
    mov cx,9

    InSo:
        add dl,1
        mov ah,02h
        int 21h
    loop InSo

    mov ah,4ch
    int 21h

    end
```

11. Cấu trúc rẽ nhánh IF

Cấu trúc rẽ nhánh IF...THEN: Sử dụng phép so sánh và các lệnh nhảy có điều kiện để thực hiện.

Có 2 dạng cấu trúc:

- IF <Điều kiện> THEN <Công việc>

Ví dụ:

```
ct_IF:
        cmp ax,0                ;so sánh AX với 0
        jae THEN_cong_viec      ;nhảy nếu AX >= 0
        neg ax                  ;thực hiện khi AX < 0
THEN_cong_viec:
        mov bx,ax               ;thực hiện khi AX >= 0
```

11. Cấu trúc rẽ nhánh IF

Cấu trúc rẽ nhánh IF...THEN: Sử dụng phép so sánh và các lệnh nhảy có điều kiện để thực hiện.

Có 2 dạng cấu trúc:

- IF <Điều kiện> THEN <Công việc>
- IF <Điều kiện> THEN <Công việc 1> ELSE <Công việc 2>

```
IF_:
    CMP     AL,BL   ;so sánh AL và BL
    JA      ELSE_   ;nhảy nếu AL > BL
    MOV     AH,2    ;lệnh được thực thi nếu AL <= BL
    MOV     DL,AL
    INT     21H
    JMP     ENDIF_
ELSE_:
    MOV     AH,2    ;lệnh được thực thi nếu AL > BL
    MOV     DL, BL
    INT     21H
ENDIF_:
```

```
.model small
.stack 100h
.data
    msg1 db 'TINH TONG 2 SO$'
    msgA db 13,10,'a = $'
    msgB db 13,10,'b = $'
    msgTong db 13,10,'Tong 2 so la: $'
.code
main proc
    mov ax,@data
    mov ds,ax

    mov ah,09h
    lea dx,msg1
    int 21h

    mov ah,09h
    lea dx,msgA
    int 21h

    mov ah,01h
    int 21h
    mov bl,al
    sub bl,30h ;Chuyển ký tự nhập vào thành số

    mov ah,09h
    lea dx,msgB
    int 21h

    mov ah,01h
    int 21h

    sub al,30h
    add bl,al

    mov ah,09h
    lea dx,msgTong
    int 21h

    mov ah,02h
    mov dl,bl
    add dl,30h
    int 21h

    mov ah,08h
    int 21h

    mov ah,4ch
    int 21h
    main endp
end
```

Từ bài toán tính tổng này, thực hiện bài toán:
- Nhập vào 2 số a và b (giả sử 9>a>5>b>0)
- Nếu nhập tiếp dấu + thì tính a+b
- Nếu nhập tiếp dấu - thì tính a-b
- Hiển thị kết quả

12. Cấu trúc rẽ nhánh CASE

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

CASE cũng sử dụng phép so sánh và các lệnh nhảy có điều kiện để thực hiện.

Cấu trúc lệnh:

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

12. Cấu trúc rẽ nhánh CASE

Ví dụ: Kiểm tra nếu
- AX âm thì đưa -1 vào BX
- AX bằng 0 thì đưa 0 vào BX
- AX dương thì đưa 1 vào BX

```
; case  AX
                CMP     AX,0            ;test AX
                JL      So_am           ;AX<0
                JE      Bang_Khong      ;AX=0
                JG      So_duong        ;AX>0
  So_am:
                MOV     BX,-1
                JMP     END_CASE
  Bang_Khong:
                MOV     BX,0
                JMP     END_CASE
  So_duong:
                MOV     BX,1
                JMP     END_CASE
  END_CASE :
```

13. Cấu trúc lặp FOR

Cấu trúc vòng lặp FOR..DO: Sử dụng phép so sánh và các lệnh nhảy có điều kiện để thực hiện một công việc lặp đi lặp lại với số lần lặp cho trước.

Cấu trúc:

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

13. Cấu trúc lặp FOR

Ví dụ: In ra một dòng 20 dấu '*'

```
        MOV   CX,20          ; CX chứa số lần lặp
        MOV   AH,2           ; hàm xuất ký tự
        MOV   DL,'*'         ;DL chứa ký tự '*'
Lap_For:
        INT   21h            ; in dấu '*'
        LOOP  Lap_For        ; lặp 20 lần
```

13. Cấu trúc lặp FOR

Ví dụ: In ra một dòng 26 ký tự hoc trong bảng mã ASCII

```
        MOV   CX,26          ; CX chứa số lần lặp
        MOV   BL,41h         ; Đưa chữ A vào BL
Lap_For:
        MOV   AH,02H
        MOV   DL,BL
        INT   21H            ; in chữ cái
        INC   BL             ;Tăng BL để đọc ký tự tiếp
LOOP    Lap_For              ; lặp 26 lần in được 26 ký tự
```

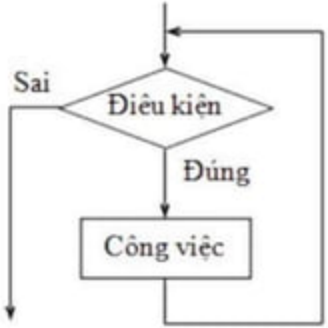
14. Cấu trúc lặp WHILE

Cấu trúc lặp WHILE phụ thuộc vào 1 điều kiện. Nếu điều kiện đúng thì thực hiện vòng WHILE còn nếu sai thì không thực hiện gì cả

Đây là cấu trúc lặp với số lần lặp không xác định, sử dụng phép so sánh và các lệnh nhảy (cả có điều kiện và không điều kiện) để thực hiện.

Có 2 dạng cấu trúc WHILE là:

- Kiểm tra điều kiện trước: WHILE <Điều kiện> DO <Công việc>



14. Cấu trúc lặp WHILE

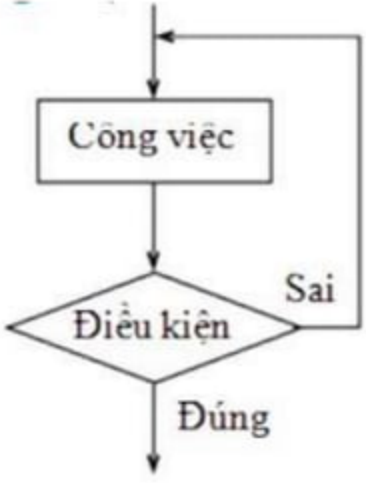
Ví dụ: Đếm số ký tự được nhập vào trên cùng 1 hàng:

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

14. Cấu trúc lặp WHILE

Có 2 dạng cấu trúc WHILE là:

- Kiểm tra điều kiện sau: REPEAT <Công việc> UNTIL <Điều kiện>



14. Cấu trúc lặp WHILE

Ví dụ: Đọc vào các ký tự cho đến khi gặp ký tự trống:

```
        MOV AH,1            ;hàm đọc 1 ký tự
REPEAT:
        INT 21h             ; đọc ký tự vào AL
;UNTIL
        CMP AL,' '          ;kiểm tra ký tự trên AL
        JNE   REPEAT
```

15. Lệnh LOGIC

Lệnh LOGIC có thể dùng riêng lẻ hoặc dùng để kết hợp các điều kiện trong các cấu trúc.

Cú pháp:

Not	[Toán hạng đích]
And	[Toán hạng đích], [Toán hạng nguồn]
Or	[Toán hạng đích], [Toán hạng nguồn]
Xor	[Toán hạng đích], [Toán hạng nguồn]
Test	[Toán hạng đích], [Toán hạng nguồn]

Trong đó:

- [Toán hạng đích], [Toán hạng nguồn] có thể là hằng số (trực hằng), biến, thanh ghi hay địa chỉ ô nhớ. [Toán hạng đích] không thể là hằng số.

Tác dụng: Mỗi lệnh logic thực hiện phép tính logic tương ứng trên các bít (tương ứng về vị trí) của [Toán hạng đích] và [Toán hạng nguồn], kết quả được ghi vào lại [Toán hạng đích]. Riêng lệnh Not, thực hiện phép đảo bít ngay trên các bít của [Toán hạng đích]. Hầu hết các lệnh logic đều ảnh hưởng đến các cờ CF, OF, ZF,...

- Lệnh Not (Logical Not): Thực hiện việc đảo ngược từng bít trong nội dung của [Toán hạng đích]. Lệnh này không làm ảnh hưởng đến các cờ.

Lệnh Not thường được sử dụng để tạo dạng bù 1 của [Toán hạng đích].

15. Lệnh LOGIC

- Lệnh And (Logical And): Thực hiện phép tính logic And trên từng cặp bít (tương ứng về vị trí) của [Toán hạng nguồn] với [Toán hạng đích], kết quả lưu vào [Toán hạng đích].

Lệnh And thường được sử dụng để xóa (= 0) một hoặc nhiều bít xác định nào đó trong một thanh ghi.

- Lệnh Or (Logical Inclusive Or): Thực hiện phép tính logic Or trên từng cặp bít (tương ứng về vị trí) của [Toán hạng nguồn] với [Toán hạng đích], kết quả lưu vào [Toán hạng đích].

Lệnh Or thường dùng để thiết lập (= 1) một hoặc nhiều bít xác định nào đó trong một thanh ghi.

- Lệnh Xor (eXclusive OR): Thực hiện phép tính logic Xor trên từng cặp bít (tương ứng về vị trí) của [Toán hạng nguồn] với [Toán hạng đích], kết quả lưu vào [Toán hạng đích].

Lệnh Xor thường dùng để so sánh (bằng nhau hay khác nhau) giá trị của hai toán hạng, nó cũng giúp phát hiện ra các bít khác nhau giữa hai toán hạng này.

- Lệnh Test: Tương tự như lệnh And nhưng không ghi kết quả vào lại [Toán hạng đích], nó chỉ ảnh hưởng đến các cờ CF, OF, ZF,...

A	B	A And B	A Or B	A Xor B	NotA
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

15. Lệnh LOGIC

Ví dụ: Chuyển AL bằng 0 có thể dùng:

- Mov AL,0 ; AL bằng 0
- Not AL ; AL = Not AL. Tức là AL = 0FFh

Ví dụ: Để xóa nội dung thanh ghi nào đó, trong hợp ngữ ta có thể sử dụng một trong các lệnh sau đây:

- Mov AX, 0
- Sub AX, AX
- Xor AX, AX ;các cặp bít giống nhau thì đều = 0

16. PUSH và POP

* PUSH dùng để cất nội dung một thanh ghi vào stack. Cú pháp:

PUSH <nguồn>

* POP dùng để lấy dữ liệu từ stack để đưa vào toán hạng đích. Cú pháp:

POP <đích>

Hai lệnh này thường được cùng sử dụng trong chương trình để cất và lấy dữ liệu từ stack. Có thể dùng nhiều lệnh PUSH và POP nhưng cần chú ý thứ tự ngược nhau giữa PUSH và POP:

```
PUSH AX
PUSH BX
PUSH CX
...
POP CX
POP BX
POP AX
```

17. Khung của 1 chương trình hợp ngữ

```
TITLE     Chương trình hợp ngữ
.MODEL    Kiểu kích thước bộ nhớ          ;Quy mô sử dụng bộ nhớ
.STACK    Kích thước                      ;Dung lượng stack
.DATA                                     ;Khai báo dữ liệu
          msg DB 'Hello$'
.CODE                                     ;Khai báo đoạn mã
main PROC
...
CALL      SubName                         ;Gọi chương trình con
...
main ENDP
SubName   PROC                            ;Định nghĩa chương trình con
...
RET
SubName ENDP
END main
```

Ví dụ: Đảo chuỗi 12345 thành 54321

```
.model small
.stack 100h

.code
    mov ax,12345
    mov bx,10
    mov cx,0

    Lap:
        xor dx,dx
        div bx
        push ax

        mov ah,02h
        or dl,30h
        int 21h

        pop ax

        cmp ax,0
    ja Lap

    mov ah,08h
    int 21h

    int 20h
end
```

Ví dụ: Nhập vào 1 chuỗi bất kỳ, in ra chuỗi thường, chuỗi HOA

```
.model small
.stack
.data
    tb1 DB 'Nhap vao 1 chuoi: $'
    tb2 DB 10,13,'Doi thanh chu thuong: $'
    tb3 DB 10,13,'Doi thanh chu HOA: $'
    s DB 100,?,101 dup('$')
.code
BEGIN:
    MOV AX, @data
    MOV DS,AX
    ;xuat chuoi tb1
    MOV AH,09h
    LEA DX,tb1
    INT 21h
    ;nhap chuoi s
    MOV AH,0Ah
    LEA DX,s
    INT 21h
    ;xuat chuoi tb2
    MOV AH,09h
    LEA DX,tb2
    INT 21h
    ;Goi chuong trinh con in chuoi thuong
    CALL InChuoiThuong
    ; xuat chuoi tb3
    MOV AH,09h
    LEA DX,tb3
    INT 21h
    ;Goi chuong trinh con in chuoi thuong
    CALL InChuoiHoa
    MOV AH,08h
    INT 21h

    MOV AH,4ch
    INT 21h
```

1

```
;Doi thanh chuoi ky tu thuong
InChuoiThuong PROC
    LEA SI,s+1
    XOR CX,CX
    MOV CL,[SI]
    INC SI
    LapThuong:
        MOV AH,02h
        MOV DL,[SI]
        CMP DL,'A'
        JB LT1
        CMP DL,'Z'
        JA LT1
        ADD DL,32
    LT1: INC SI
    INT 21h
    LOOP LapThuong
RET
InChuoiThuong ENDP
;Doi thanh chuoi ky tu hoa
InChuoiHoa PROC
    LEA SI,s+1
    XOR CX,CX
    MOV CL,[SI]
    INC SI
    LapHoa:
        MOV AH,02h
        MOV DL,[SI]
        CMP DL,'a'
        JB LH1
        CMP DL,'z'
        JA LH1
        SUB DL,32
    LH1: INC SI
INT 21h
LOOP LapHoa
RET
InChuoiHoa ENDP
END BEGIN
```

2