Nội dung

Tập các thanh ghi (8086)

4 thanh ghi đoạn

1 chương trình Assembly chia ra các đoạn (Segment) chứa dữ liệu, code và stack:

- Code segment: chứa các mã lệnh thực thi. Thanh ghi CS chứa địa chỉ bắt đầu của Code segment.
- Data segment: chứa biến, hằng số, dữ liệu chương trình. Thanh ghi DS chứa địa chỉ bắt đầu của Data segment.
- Stack segment: chứa dữ liệu và địa chỉ trả về của các chương trình con, các dữ liệu lưu trữ theo cấu trúc Stack. Thanh ghi SS chứa địa chỉ bắt đầu của Stack segment.
- ES (Extra Segment): thanh ghi đoạn dữ liệu phụ

2 thanh ghi chỉ số

thường dược dùng để đánh chỉ số cho các địa chỉ mảng, xâu; đôi khi sử dụng trong phép tính số học.

- SI (Source Index): được sử dụng làm địa chỉ nguồn cho các phép toán với xâu.

- DI (Destination Index): được sử dụng làm địa chỉ đích cho các phép toán với xâu.

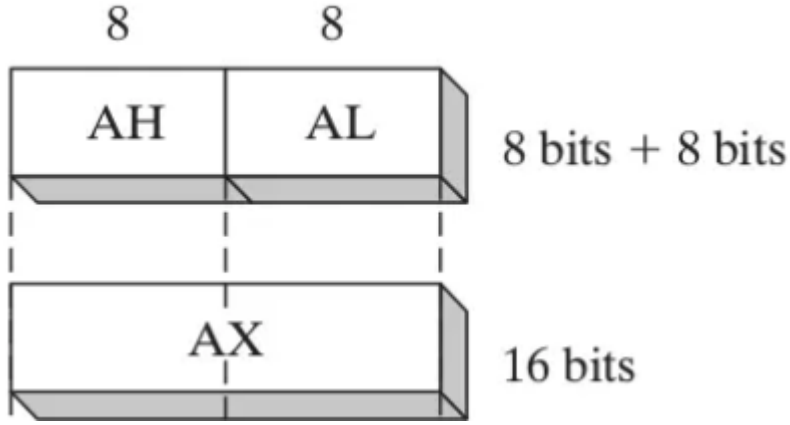
3 thanh ghi con trỏ

- IP (Instruction Pointer): thanh ghi con trỏ lệnh
- SP (Stack Pointer): con trỏ ngăn xếp
- BP (Base Pointer): thanh ghi con trỏ cơ sở.

4 thanh ghi dữ liệu đa năng

- AX (Accumulator Register): thanh chứa - thanh ghi tích lũy (thường dùng để nhập xuất và các lệnh tính toán số học.
- BX (Base Register): thanh ghi cơ sở (thường dùng để đánh dấu địa chỉ, lưu địa chỉ bắt đầu của 1 mảng)
- CX (Count Register): thanh ghi đếm (thường dùng trong vòng lặp, đếm lần lặp)
- DX (Data Register): thanh ghi dữ liệu (thường dùng trong nhập xuất như AX)

Mỗi thanh ghi đa năng này lưu được 16 bit và có thể chia thành 2 đoạn nhỏ hơn:



Nhỏ nhất là 2 thanh ghi 8 bit AH và AL. AH chứa phần bit cao và AL chứa phần bit thấp của thanh ghi AX.

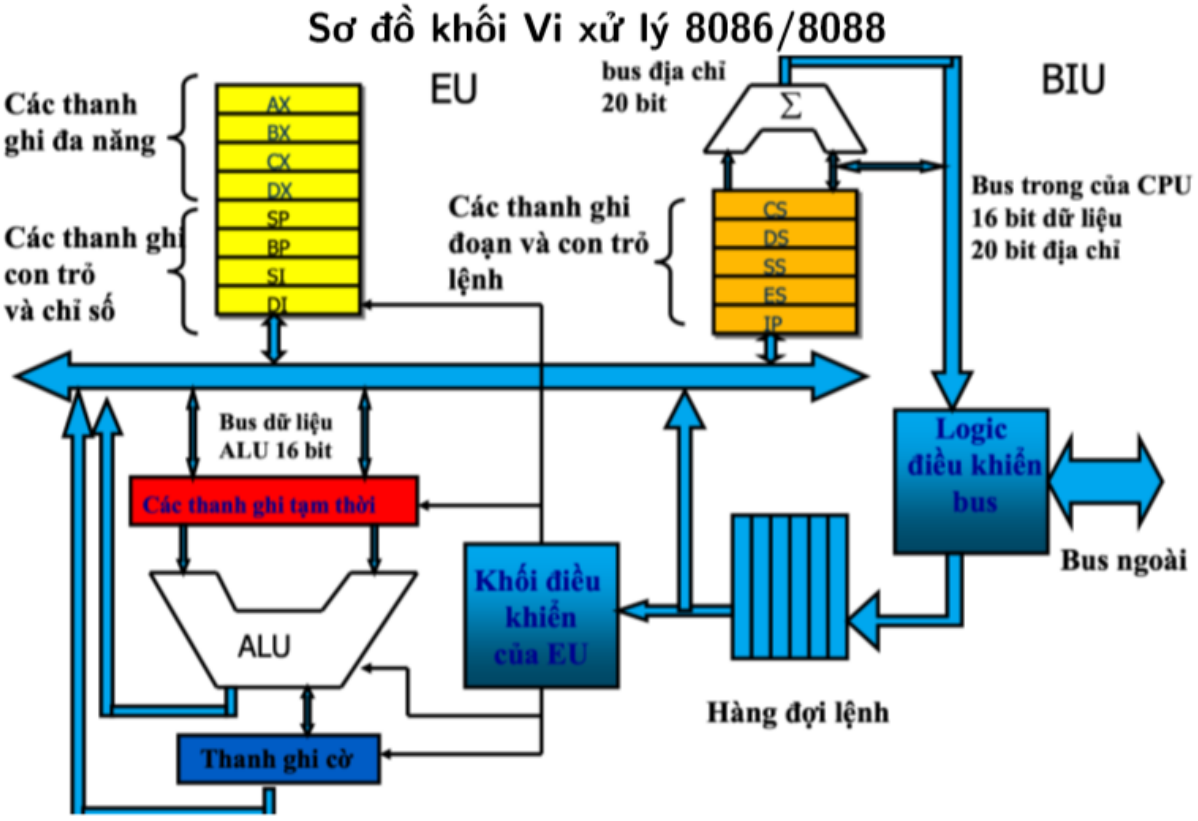
Áp dụng tương tự cho 3 thanh ghi còn lại.

Thanh ghi trạng thái FR (8086)

- ZF: Zero Flag, ZF = 1 nếu kết quả là 0 và ZF = 0 nếu kết quả khác 0.
- SF: Sign Flag, SF = 1 nếu kết quả âm và SF = 0 nếu kết quả dương
- CF: Carry Flag, CF = 1 nếu có nhớ/ mượn ở bit trái nhất.
  VD: 10 + 11 = **1**01 => CF = 1
- AF: Auxiliary Carry Flag, AF = 1 nếu có nhớ ở bit trái nhất của nibble (4 bit).
  VD: 0000 1111 + 0001 1001. Khi cộng phần 1111 với 1001 thì có 1 bit nhớ sang nibble tiếp theo (1111 + 1001 = **1**1110) => AF = 1.

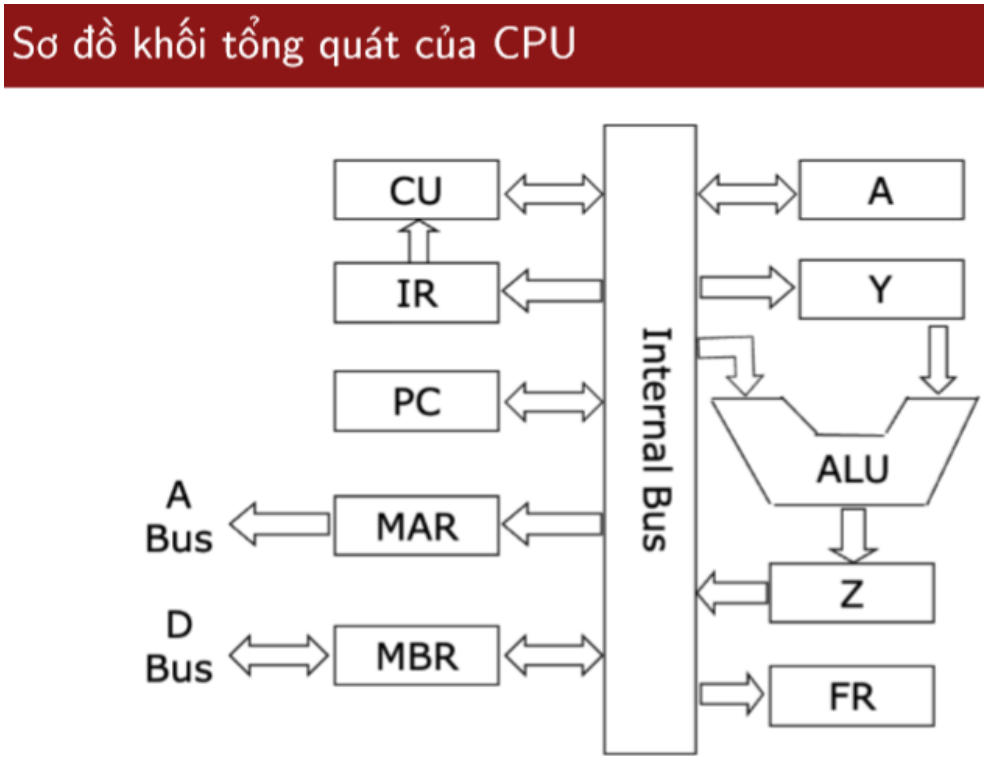
- OF: Overflow Flag, OF = 1 nếu có tràn bit và OF = 0 nếu không có
- PF: Parity Flag, PF = 1 nếu tổng số bit 1 trong kết quả là số lẻ, PF = 0 ngược lại
- IF: Interrupt Flag, IF = 1: ngắt được phép, IF = 0: cấm ngắt.
- TF: Trap flag, hỗ trợ thực thi chương trình theo Single-step mode để Debug.

Trong các cờ trên có DF, IF và TF là 3 cờ điều khiển; 6 cờ còn lại là cờ trạng thái.



Thanh ghi lệnh IR: lưu trữ lệnh đang được xử lý, lấy lệnh từ MBR và chuyển nó đến CU để giải mã lệnh.

Thanh ghi địa chỉ bộ nhớ MAR (Memory Address Register) **và thanh ghi nhớ đệm MBR** (Memory Buffer Register).



Chương trình hợp ngữ của 8086

I. Khai báo biến, hằng, mảng và xâu kí tự

1. Biến

Cú pháp: <tên biến> <kiểu dữ liệu> <giá trị khởi đầu>

Với 3 kiểu dữ liệu thường gặp:

- DB (Define byte): Biến byte (8bit)
- DW (Define word): Biến word (16bit)
- DD (Define double word): Biến double word (32bit)

Giá trị khởi đầu là giá trị được gán khi biến được khởi tạo. Nếu không muốn có giá trị đầu khi khởi tạo thì sử dụng ký tự '?'.

VD: X DB 10 ;Khai báo biến X và giá trị khởi tạo 10

Y DW ? ;Khai báo biến Y và không khởi tạo giá trị

2. (Biến) Mảng

Cú pháp: <tên mảng> <kiểu dữ liệu> <giá trị 1>, <giá trị 2>,…

Các giá trị được gán vào khi biến được khởi tạo. Nếu muốn khởi tạo không giá trị đầu thì sử dụng '?'. Nếu muốn khởi tạo nhiều biến cùng 1 giá trị dùng cú pháp (lệnh DUP có thể lồng nhau):

<tên mảng> <kiểu dữ liệu> <số phần tử> DUP(giá trị)

VD: mang DB 1,2,6,4,5 ;Khởi tạo mảng có 5 phần tử giá trị 1, 2, 6, 4, 5

Y DB 5 DUP(?) ;Khởi tạo mảng 5 phần tử không giá trị.

X DB 1,2, 5 DUP(4) ;Khởi tạo mảng gồm 2 phần tử 1, 2 và 5 phần tử giá trị 4.

3. Hằng

Cú pháp: <tên hằng> EQU <giá trị>

Hằng có thể là số hoặc chuỗi kí tự, cũng có thể sử dụng hằng này để định nghĩa 1 biến mảng khác.

VD: MAX EQU 100 ;Khai báo hằng MAX giá trị 100

A1 EQU 'Hello' ;Khai báo hằng A1 = 'Hello'

A2 EQU A1, '$' ;Khai báo biến mảng A2 = 'Hello$'

4. (Biến) Xâu kí tự

Cú pháp: <tên xâu> <kiểu dữ liệu> <xâu>

Có thể được định nghĩa như 1 chuỗi ký tự hoặc 1 mảng các ký tự (giá trị mỗi phần tử là mã ASCII của kí tự đó)

Ký tự '$' báo hiệu đã hết xâu.

VD: xau1 DB 'can cu$' ;Khai báo xau1 = 'can cu$'

number DB 30h, 31h, 32h, '$' ;Khai báo number = '012$'

Chú ý: CRLF DB 13, 10, '$' ;Đây là xâu dùng để xuống dòng và về đầu dòng (13 – CR (Carriage Return) và 10 – LF (Line feed)), hiểu đơn giản CRLF giống "\n" trong C/C++.

II. Khung chương trình hợp ngữ

Khung cơ bản của 1 chương trình gồm 4 phần:
.Model
.Stack
.Data
.Code

1. Khai báo quy mô sử dụng bộ nhớ (.Model)

Cú pháp: .model <kiểu kích thước bộ nhớ>

Các kiểu kích thước bộ nhớ:

- Tiny (hẹp): mã lệnh và dữ liệu gói gọn trong 1 đoạn.
- **Small** (nhỏ): mã lệnh gói gọn trong 1 đoạn, dữ liệu gói gọn trong 1 đoạn.
- Medium (vừa): mã lệnh không gói gọn trong 1 đoạn, dữ liệu gói gọn trong 1 đoạn.
- Compact (gọn): mã lệnh gói gọn trong 1 đoạn, dữ liệu không gói gọn trong 1 đoạn.
- Large (lớn): mã lệnh không gói gọn trong 1 đoạn, dữ liệu không gói gọn trong 1 đoạn, không có mảng lớn hơn 64K.
- Huge (rất lớn): mã lệnh không gói gọn trong 1 đoạn, dữ liệu không gói gọn trong 1 đoạn, có mảng lớn hơn 64K.

2. Khai báo đoạn ngăn xếp (.Stack)

Cú pháp: .stack <kích thước ngăn xếp>

Nếu không khai báo chương trình dịch tự gán kích thước 1KB, kích thước khá lớn, thường chỉ dùng 100-256 byte. (.stack 100/256)

3. Khai báo đoạn dữ liệu (.Data)

Khai báo các biến, mảng, hằng, xâu ở đây.

4. Khai báo đoạn mã (.Code)

- Nơi chứa mã lệnh của chương trình, gồm chương trình chính và các chương trình con (nếu có).
- Trong đoạn mã sẽ bao gồm các thủ tục, được khai báo bởi 2 lệnh giả PROC và ENDP. Lệnh PROC để bắt đầu và ENDP để kết thúc. Một chương trình chính được khai báo theo mẫu như sau:

```
<tên chương trình chính> PROC
;code nằm ở đây
CALL <tên chương trình con>; gọi chương trình con
<tên chương trình chính> ENDP
```

- Một chương trình con được khai báo như sau:

```
<tên chương trình con> PROC
;code nằm ở đây
RET             ;kết thúc chương trình con
<tên chương trình con> ENDP
```

Sau đây là 1 khung cho chương trình hợp ngữ để rồi sau khi được dịch, nối trên máy IPM PC sẽ tạo ra 1 tệp chương trình chạy được ngay đuôi .EXE:

```
.model small

.stack 100

.data

    ;khai báo các biến và hằng

.code

    MAIN PROC

        ;khởi tạo DS

        MOV AX, @data    ;Ta phải dùng thanh ghi AX làm trung gian cho khởi đầu DS do những lí do kỹ thuật trên vi xử lý 8086, không cho phép chuyển giá trị số (chế độ địa chỉ tức thì) vào các thanh ghi đoạn. (AX có thể thay bằng thanh ghi khác)

        MOV DS, AX

        ;code nằm ở đây

        ;kết thúc chương trình

        MOV AH, 4CH

        INT 21H

    MAIN ENDP

        ;các chương trình con (nếu có) nằm ở đây
```

END MAIN

Danh sách các lệnh thường gặp của 8086

I. Các lệnh trao đổi dữ liệu

(<gốc> = <toán hạng gốc>; <đích> = <toán hạng đích>)

Lệnh	Cú pháp	Chức năng	VD

MOV (Move)	**MOV <đích>, <gốc>**
	Với <đích> có thể là thanh ghi, địa chỉ 1 ô nhớ hay 1 biến.
	<gốc> có thể là hằng, biến, thanh ghi, địa chỉ ô nhớ.
		Gán giá trị của <gốc> vào <đích>	MOV AX, BX ;Đặt giá trị thanh ghi BX vào thanh ghi AX
			MOV CX, 'A' ;CX = 41h
			MOV BL, DX ;Không hợp lệ vì DX là 16bit còn BL là 8 bit
			MOV DL, [DI] ;DL = [DS:DI]

LEA (Load Effective Address)	**LEA <đích>, <gốc>** Với <đích> là các thanh ghi 16 bit (BX, CX, DX, BP, SI, DI) <gốc> là địa chỉ 1 vùng nhớ hay tên của 1 biến.	Chuyển địa chỉ hiệu dụng hoặc địa chỉ lệch (offset) của <gốc> vào <đích>.	LEA SI, a ;nạp địa chỉ biến a vào thanh ghi SI LEA CX, [BX] ;nạp địa chỉ ô nhớ có địa chỉ [DS:BX] vào CX LEA DI, xauKT ;chuyển địa chỉ offset của biến xauKT vào DI, thao tác này thường được gọi là trỏ DI vào đầu biến xauKT.
PUSH (16bit)	**PUSH <gốc>**	Đẩy giá trị <gốc> vào stack.	PUSH AX ;đẩy giá trị của AX vào stack PUSH 0Ah ;đẩy giá trị 0Ah vào stack
POP (16bit)	**POP <gốc>**	Lấy giá trị trên cùng stack rồi gán vào <gốc>.	POP AX ;lấy giá trị trên cùng stack rồi gán vào AX.

XCHG	**XCHG <Operand1>, <Operand2>**	Tráo đổi giá trị 2 toán hạng.	MOV BX, 100
			MOV AX, 200
			XCHG AX, BX ;AX <- 100, BX <- 200

II. Các lệnh tính toán số học

Lệnh	Cú pháp	Chức năng	VD
ADD	**ADD <đích>, <gốc>** Với <đích> và <gốc> giống MOV.	Lấy giá trị của <gốc> cộng vào giá trị của <đích>, kết quả đặt vào <đích>	ADD AX, BX ADD AX, 120
SUB	**SUB <đích>, <gốc>** Với <đích> và <gốc> giống MOV.	Lấy giá trị của <đích> trừ giá trị của <gốc>, kết quả đặt vào <đích>.	SUB BX, AX SUB DL, [SI] ;DL = DL – [DS:SI]
INC	**INC <đích>**	Tăng giá trị của <đích> lên 1.	INC AX
DEC	**DEC <đích>**	Giảm giá trị của <đích> đi 1.	DEC BX

ADC	**ADC <đích>, <gốc>**	Tương tự như ADD nhưng ADD là cộng không nhớ, ADC là cộng có nhớ (ADD with Carry)	
NEG	**NEG <đích>**	Đảo dấu giá trị <đích>	NEG AL

- **Lệnh** MUL: thực hiện phép nhân trên số không dấu.
  Cú pháp: MUL <gốc>
  Chức năng:
  + Với <gốc> là số 8 bit: AX = AL × <gốc> (VD: MUL BL;AX = AL × BL)
  + Với <gốc> là số 16 bit: DXAX = AX × <gốc> (phần bit thấp lưu ở AX, bit cao lưu ở DX) (VD: MUL BX ;DXAX = AX × BX)
  Lấy số 8 bit nhân với số 16 bit: Giả sử muốn lấy ô nhớ có địa chỉ DS:SI 8bit nhân với thanh ghi BX 16 bit, ta có thể làm bằng cách để số 16 bit tại <gốc>, số 8bit vào AL, sau đó mở rộng sang AH để thành 16 bit

```
MOV [SI], 74      ;[DS:SI] = 74
MOV BX, 123FH     ;BX = 123FH
MOV AL, [SI]      ;AL = [DS:SI]
MOV AH, 00H       ;AH = 00H -> AX = AL
MUL BX            ;DXAX = AX x BX
```

  Ta có thể dùng IMUL để nhân trên số có dấu.
- **Lệnh** DIV: thực hiện phép chia trên số không dấu.
  Cú pháp: DIV <gốc>
  Chức năng: + Với <gốc> là số 8 bit: AL = AX / <gốc> (chia nguyên)
  AH = AX % <gốc> (chia dư)
  + Với <gốc> là số 16 bit: AX = DXAX / <gốc> (chia nguyên)
  DX = DXAX % <gốc> (chia dư)
  Nếu <gốc> = 0 hoặc thương lớn hơn FFH (với phép chia 8bit) hoặc FFFFH (với phép chia 16bit) thì CPU thực hiện lệnh ngắt INT 0
  Ta có thể dùng IDIV để chia trên số có dấu.

III. Các lệnh so sánh logic

Lệnh	Cú pháp	Chức năng	VD
AND	**AND <đích>, <gốc>** Với <đích> và <gốc> giống MOV.	<đích> = <đích> && <gốc> Thường dùng để lấy đi 1 số bit nhất định của 1 toán hạng.	AND BL, AL AND BL, 0Fh ;Che 4 bit đầu của BL (VD: BL: 1011 0111 && 0Fh (15): 0000 1111 => BL = 0000 0111)
OR	**OR <đích>, <gốc>** Với <đích> và <gốc> giống MOV.	<đích> = <đích> || <gốc>	OR AL, BL OR AL, F0h (240) (F0h: 1111 0000)
XOR	**XOR <đích>, <gốc>** Với <đích> và <gốc> giống MOV.	<đích> = <đích> ^ <gốc>. Thường dùng để xoá 1 toán hạng về 0 bằng cách XOR với chính nó hoặc dùng để đảo bit.	XOR BL, BL XOR AL, BL ;AL = AL ^ BL (VD: AL: 1011 0110 và BL: 1111 1111 => AL: 0100 1001)

NOT	**NOT <đích>**	Đảo (bù 1) <đích>	NOT AL
CMP	**CMP <đích>, <gốc>**	So sánh <đích> và <gốc>, lệnh này chỉ tác động đến các cờ của thanh ghi FR. Thường dùng để tạo điều kiện trong các lệnh nhảy.	

IV. Các lệnh dịch và quay

Lệnh	Cú pháp	Chức năng	VD
SHL	**SHL <đích>, CL**	Dịch trái logic toán hạng <đích> CL bit.	SHL AL, CL ;nếu CL = 5 thì dịch trái AL 5 bit Nếu CL = 1 có thể viết: SHL AL, 1
SHR	**SHR <đích>, CL**	Dịch phải logic toán hạng <đích> CL bit.	SHR BL, CL ;nếu CL = 4 thì dịch phải BL 4 bit Nếu CL = 1 có thể viết: SHR AL, 1
SAL	**SAL <đích>, CL**	Dịch trái toán học toán hạng <đích> CL bit.	
SAR	**SAR <đích>, CL**	Dịch phải toán học toán hạng <đích> CL bit.	
ROL	**ROL <đích>, CL**	Quay trái toán hạng <đích> CL bit.	
ROR	**ROR <đích>, CL**	Quay phải toán hạng <đích> CL bit.	

V. Các lệnh nhảy

Cú pháp chung: <tên lệnh> <nhãn đích> (nhảy đến nhãn nếu phù hợp điều kiện lệnh, thường được dùng với lệnh CMP)

Chú ý các lệnh màu đỏ là dành cho so sánh các số có dấu, nếu chỉ so sánh các số dương thì dùng các lệnh màu xanh để tránh sai sót.

Instruction	Description
JMP (Jump)	Nhảy không điều kiện
JA/ JNBE	Nhảy nếu lớn hơn
JB/ JNAE	Nhảy nếu nhỏ hơn
JE/ JZ (Jump if Equal (Zero))	Nhảy nếu bằng (0)
JNE/ JNZ (Jump if not Equal (Zero))	Nhảy nếu không bằng (0)
JG/ JNLE (Jump if Greater)	Nhảy nếu lớn hơn
JL/ JNGE (Jump if Lower)	Nhảy nếu nhỏ hơn
JNG/ JLE (Jump if not Greater/ if Lower or Equal)	Nhảy nếu không lớn hơn
JNL/ JGE (Jump if not Lower/ if Greater or Equal)	Nhảy nếu không nhỏ hơn
JS (Jump if Signed)	Nhảy nếu có dấu (SF = 1)
JNS (Jump if Not Signed)	Nhảy nếu không có dấu (nếu kết quả dương) (SF = 0)
JC (Jump if Carry)	Nhảy nếu có nhớ (CF = 1)
JNC (Jump if Not Carry)	Nhảy nếu không có nhớ (CF = 0)
JO (Jump if Overflow)	Nhảy nếu tràn (OF = 1)

JNO (Jump if Not Overflow)	Nhảy nếu không tràn (OF = 0)

VI. Lệnh LOOP

Cú pháp: LOOP <Nhãn đích>

Trong đó: <Nhãn đích> là một nhãn lệnh và nó phải đứng trước lệnh lặp LOOP không quá 126 byte.

Tác dụng: Khi gặp lệnh này chương trình sẽ lặp lại việc thực hiện các lệnh sau <Nhãn đích> đủ n lần, với n được đặt trước trong thanh ghi CX. Sau mỗi lần lặp CX tự động giảm 1 đơn vị (CX = CX - 1) và lệnh lặp sẽ dừng khi CX = 0.

Lệnh Loop thường được sử dụng để cài đặt các đoạn chương trình lặp với số lần lặp xác định, được cho trước trong thanh ghi CX (tương tự For trong các nnlt bậc cao).

Ngoài ra còn có:

LOOPE (LOOPZ): Lặp nếu bằng (cờ không) hoặc số lần lặp do CX xác định.

LOOPNE (LOOPNZ): Lặp nếu không bằng (cờ không xoá) hoặc số lần do CX xác định.

VII. Các lệnh điều khiển vi xử lý

Các lệnh này tác động lên thanh ghi cờ FR làm thay đổi trạng thái hoạt động vi xử lý.

Lệnh	Chức năng
CLD (Clear Direction Flag)	Xoá cờ hướng (DF = 0)
STD (Set Direction Flag)	Lập cờ hướng (DF = 1)
CLC (Clear Carry Flag)	Xoá cờ nhớ (CF = 0)
STC (Set Carry Flag)	Lập cờ nhớ (CF = 1)
CMC (Complement Carry Flag)	Đảo cờ nhớ (CF = !CF)

VIII. Các lệnh vận chuyển dữ liệu nâng cao

Lệnh	Chức năng	Ví dụ

LODSB	Nạp nội dung ô nhớ có địa chỉ chứa trong SI thuộc đoạn DS vào thanh ghi AL và tăng (nếu DF = 0) hoặc giảm (nếu DF = 1) nội dung của SI lên/ xuống 1.	MOV SI, 1000 ;SI <- 1000
		MOV [DS:SI], 200 ;[DS:SI] <- 200
		CLD ;DF <- 0
		LODSB ;AL <- 200
		;SI <- SI + 1

LODSW	Tương tự LODSB, khác là SI tăng/ giảm lên/ xuống 2.	
STOSB/ STOSW		
MOVSB/ MOVSW		

IX. Các lệnh ngắt 21H

Cú pháp: INT 21H. Chức năng của lệnh dựa vào giá trị AH.

Loại ngắt	Giá trị AH	Chức năng	VD

Loại 1	1	Đọc 1 kí tự từ bàn phím, AL sẽ lưu mã ASCII của phím vừa nhập, nếu phím vừa nhập là phím chức năng thì AL = 0	MOV AH, 1 ;AH = 1 INT 21H ;c/trình lúc này sẽ ngừng lại đến khi bạn nhập vào một phím

Loại 2	2	Hiện 1 kí tự có mã ASCII là giá trị của DL lên màn hình	MOV AH, 2
			MOV DL, 30h ;DL = '0'
			INT 21H ;in ra kí tự '0'

Loại 9	9	Hiện xâu kí tự có địa chỉ lệnh (offset) là giá trị của DX	test DB 'con meo$' MOV AH, 9 LEA DX, test ;gán địa chỉ lệch của test vào DX INT 21H ;in ra xâu test (không hiện '$')
Ngắt chương trình	AH = 4CH	Dừng chương trình	MOV AH, 4CH INT 21H ;dừng ctrình

Trường hợp với xử lí xâu (hàm 0Ah (AH = 10)): Nhập 1 xâu kí tự vào một biến đệm cho trước trong chương trình, biến này phải được chỉ bởi cặp thanh ghi [DS:DX] và biến đệm phải có dạng như sau:

- Byte 0: chứa số kí tự tối đa của xâu nhập vào

- Byte 1: chứa một giá trị không (= 0)

- Byte 2, byte 3, byte 4, ... chứa một trị rỗng (để trống), để chứa các kí tự sẽ được nhập vào sau này (khi chương trình thực hiện).

Để có được 1 biến như trên chương trình phải khai báo biến (tên biến là Xau_Nhap) như sau: Xau_Nhap DB 256, 0, 256 Dup (' ')

Như vậy Xau_Nhap là một biến kiểu byte, gồm 258 byte. Byte 0 (byte) chứa giá trị 256, byte 1 chứa giá trị 0, 256 byte tiếp theo chứa kí tự trống, được sử dụng để chứa các kí tự sẽ được nhập sau này. Xau_Nhap chứa tối đa 256 kí tự.

Cũng có thể sử dụng hàm 0Ah/21h để nhập một xâu kí tự vào vùng nhớ có địa chỉ xác định trong bô nhớ.

Sử dụng: Vào: Ah = 0Ah

DS:DX = <Địa chỉ Segment:Offset của xâu nhập>

Ra: DS:DX không thay đổi

Biến đệm có thay đổi như sau: Byte 0 không đổi, Byte 1 chứa tổng số kí tự đã được nhập vào và Byte 2, Byte 3,… chứa các kí tự đã được nhập.

Ví dụ: Với khai báo: Xau_Nhap DB 256, 0, 256 Dup (' ')

Nếu sau này chương trình nhập vào xâu: ‘Tin hoc’ thì: Byte 0: vẫn chứa số 256, Byte 1 chứa giá trị 7, đó chính là 7 kí tự trong xâu ‘Tin hoc’ và từ byte 2 đến byte 8 chứa lần lượt các kí tự trong xâu ‘Tin hoc’.