BÀI GIẢNG MÔN

HỢP NGỮ

Quảng Ngãi - 2015

Mục lục

Chương 1 - CƠ BẢN VỀ HỢP NGỮ

Trong chương này sẽ giới thiệu những nguyên tắc chung để tạo ra, dịch và chạy một chương trình hợp ngữ trên máy tính. Cấu trúc ngữ pháp của lệnh hợp ngữ trong giáo trình này được trình bày theo Macro Assembler (MASM) dựa trên CPU 8086 .

1.1 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 (Assembler =ASM) chuyển thành mã máy.
- một chỉ dẫn của Assembler (Assembler directive) : ASM không chuyển thành mã máy

Các mệnh đề của ASM gồm 4 trường :

Name Operation Operand(s) Comment

các trường cách nhau ít nhất là một ký tự trống hoặc một ký tự TAB
ví dụ lệnh đề sau :

```
START : MOV CX,5 ; khoi tạo thanh ghi CX
```

Sau đây là một chỉ dẫn của ASM :

```
MAIN PROC ; tạo một thủ tục có tên là MAIN
```

1.1.1 Trường Tên (Name Field)

Trường tên được dùng cho nhãn lệnh, tên thủ tục và tên biến. ASM sẽ chuyển tên thành địa chỉ bộ nhớ. Tên có thể dài từ 1 đến 31 ký tự. Trong tên chứa các ký tự từ a-z, các số và các ký tự đặc biệt sau : ? ,@, _, $ và dấu. Không được phép có ký tự trống trong phần tên. Nếu trong tên có ký tự. thì nó phải là ký tự đầu tiên. Tên không được bắt đầu bằng một số. ASM không phân biệt giữa ký tự viết thường và viết hoa .
Sau đây là các ví dụ về tên hợp lệ và không hợp lệ trong ASM .

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

1.1.2 Trường toán tử (operation field)

Đối với 1 lệnh trường toán tử chứa ký hiệu (symbol) 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 pseudo-op thành mã máy mà hướng dẫn ASM thực hiện một việc gì đó ví dụ tạo ra một thủ tục, định nghĩa các biến ...

1.1.3 Trường các toán hạng (operand(s) field)

Trong một lệnh trường toán hạng chỉ ra các số liệu tham gia trong lệnh đó. Một lệnh có thể không có toán hạng, có 1 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 (destination operand). Toán hạng đích thường làthanh ghi hoặc vị trí nhớ dùng để lưu trữ kết quả. Toán hạng thứ hai là toán hạng nguồn. Toán hạng nguồn thường không bị thay đổi sau khi thực hiện lệnh . Đối với một chỉ dẫn của ASM, trường toán hạng chứa một hoặc nhiều thông tin mà ASM dùng để thực thi chỉ dẫn .

1.1.4 Trường chú thích (comment field)

Trường chú thích là một tuỳ chọn của mệnh đề trong ngôn ngữ ASM. Lập trình viên dùng trường chú thích để thuyết minh về câu lệnh. Điều này là cần thiết vì ngôn ngữ ASM là ngôn ngữ cấp thấp (low level) vì vậy sẽ rất khó hiểu chương trình nếu nó không được chú thích một cách đầy đủ và rỏ ràng. Tuy nhiên không nên có chú thích đối với mọi dòng của chương trình, kể cả nnhững lệnh mà ý nghĩa của nó đã rất rỏ ràng như :

```
NOP ; không làm chi cả
```

Người ta dùng dấu chấm phẩy (;) để bắt đầu trường chú thích . ASM cũng cho phép dùng toàn bộ một dòng cho chú thích để tạo một khoảng trống ngăn cách các phần khác nhau cuả chương trình ,ví dụ :

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

1.2 Các kiểu số liệu trong chương trình hợp ngữ

CPU chỉ làm việc với các số nhị phân. Vì vậy ASM phải chuyển tất cả các loại số liệu thành số nhị phân. Trong một chương trình hợp ngữ cho phép biểu diễn số liệu dưới dạng nhị phân, thập phân hoặc thập lục phân và thậm chí là cả ký tự nửa .

1.2.1 Các số

Một số nhị phân là một dãy các bit 0 và 1 va 2phải kết thúc bằng h hoặc H
Một số thập phân là một dãy các chữ só thập phân và kết thúc bởi d hoặc D (có thể không cần)
Một số hex phải bắt đầu bởi 1 chữ số thập phân và phải kết thúc bởi h hoặc H .
Sau đây là các biểu diễn số hợp lệ và không hợp lệ trong ASM :

Số	**Loại**
10111	thập phân
10111b	nhị phân
64223	thập phân
-2183D	thập phân
1B4DH	hex
1B4D	số hex không hợp lệ
FFFFH	số hex không hợp lệ
0FFFFH	số hex

1.2.2 Các ký tự

Ký tự và một chuỗi các ký tự phải được đóng giữa hai dấu ngoặc đơn hoặc hai dấu ngoặc kép. Ví dụ ‘A’ và “HELLO”. Các ký tự đều được chuyển thành mã ASCII bởi ASM. Do đó trong một chương trình ASM sẽ xem khai báo ‘A’ và 41h (mã ASCII của A) là giống nhau .

1.3 Các biến (variables)

Trong ASM biến đóng vai trò như trong ngôn ngữ cấp cao. Mỗi biến có một loại dữ liệu và nó được gán một địa chỉ bộ nhớ sau khi dịch chương trình. Bảng sau đây liệt kê các toán tử giả dùng để định nghĩa các loại số liệu .

PSEUDO-OP	STANDS FOR
DB	define byte
DW	define word (doublebyte)
DD	define doubeword (2 từ liên tiếp)
DQ	define quadword (4 từ liên tiếp)
DT	define tenbytes (10 bytes liên tiếp)

1.3.1. Biến byte

Chỉ dẫn của ASM để định nghĩa biến byte có dạng như sau :

NAME DB initial_value

Ví dụ :

```
ALPHA DB 4
```

Chỉ dẫn này sẽ gán tên ALPHA cho một byte nhớ trong bộ nhớ mà giá trị ban đầu của nó là 4. Nếu giá trị của byte là không xác định thì đặt dấu chấm hỏi (?) vào giá trị ban đầu. Ví dụ :

```
BYT DB ?
```

Đối với biến byte vùng giá trị khả dĩ mà nó lưu trữ được là -128 đến 127 đối với số có dấu và 0 đến 255 đối với số không dấu .

1.3.2 Biến từ

Chỉ dẫn của ASM để định nghĩa một biến từ như sau :

NAME DW initial_value

Ví dụ :

```
WRD DW -2
```

Cũng có thể dùng dấu ? để thay thế cho biến từ có giá trị không xác định. Vùng giá trị của biến từ là -32768 đến 32767 đối với số có dấu và 0 đến 56535 đối với số không dấu .

1.3.3 Mảng (arrays)

Trong ASM một mảng là một loạt các byte nhớ hoặc từ nhớ liên tiếp nhau. Ví dụ để định nghĩa một mảng 3 byte gọi là B_ARRAY mà giá trị ban đầu của nó là 10h,20h và 30h chúng ta có thể viết :

```
B_ARRAY DB 10h,20h,30h
```

B_ARRAY là tên được gán cho byte đầu tiên
B_ARRAY+1 là tên của byte thứ hai
B_ARRAY+2 là tên của byte thứ ba

Nếu ASM gán địa chỉ offset là 0200h cho mảng B_ARRAY thì nội dung bộ nhớ sẽ như sau :

SYMBOL	ADDRESS	CONTENTS
B_ARRAY	200h	10h
B_ARRAY+1	201h	20h
B_ARRAY+2	202h	30h

Chỉ dẫn sau đây sẽ định nghĩa một mảng 4 phần tử có tên là W_ARRAY:

```
W_ARRAY DW 1000,40,29887,329
```

Giả sử mảng bắt đầu tại 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

Byte thấp và byte cao của một từ

Đôi khi chúng ta cần truy xuất tới byte thấp và byte cao của một biến từ. Giả sử chúng ta định nghĩa :

```
WORD1 DW 1234h
```

Byte thấp của WORD1 chứa 34h, còn byte cao của WORD1 chứa 12h
Ký hiệu địa chỉ của byte thấp là WORD1 còn ký hiệu địa chỉ của byte cao là WORD1+1 .

Chuỗi các ký tự (character strings)

Một mảng các mã ASCII có thể được định nghĩa bằng một chuỗi các ký tự
Ví dụ :

```
LETTERS DW 41h,42h,43h
```

tương đương với

```
LETTERS DW 'ABC '
```

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

```
MSG DB 48H,45H,4CH,4Ch,4FH,0AH,0DH,24H
```

1.4 Các hằng (constants)

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

NAME EQU constant

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 một mã máy .
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

```
MSG DB PROMPT
```

1.5 Các lệnh cơ bản

CPU 8086 có hàng trăm lệnh, trong chương này ,chúng ta sẽ xem xét 7 lệnh đơn giản của 8086 mà chúng thường được dùng với các thao tác di chuyển số liệu và thực hiện các phép toán số học.

Trong phần sau đây, WORD1 và WORD2 là các biến từ, BYTE1 và BYTE2 là các biến byte .

1.5.1 Lệnh MOV và XCHG

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 một vị trí nhớ. Cú pháp của lệnh MOV là :

MOV Destination, Source

Sau đây là vài ví dụ :

```
MOV AX,WORD1 ; lấy nội dung của từ nhớ 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
```

Bảng sau cho thấy các trường hợp cho phép hoặc cấm của lệnh MOV

	Destination operand			
source operand	General Reg	Segment Reg	Memory Location	Constant
General Reg	Y	Y	Y	NO
Segment Reg	Y	NO	Y	NO
MemoryLocation	Y	Y	NO	NO
Constant	Y	NO	Y	NO

Lệnh XCHG (Exchange) dùng để trao đổi nội dung của 2 thanh ghi hoặc của một thanh ghi và một vị trí nhớ. Ví dụ :

```
XCHG AH,BL
XCHG AX,WORD1 ; trao đổi nội dung của thanh ghi AX và từ nhớ
WORD1.
```

Cũng như lệnh MOV có một số hạn chế đối với lệnh XCHG như bảng sau :

	Destination operand	
Source operand	General Register	Memory Locatin
General Memory	Y	Y
Memory Location	Y	No

1.5.2 Lệnh ADD, SUB, INC, DEC

Lệnh ADD và SUB được dùng để cộng và trừ nội dung của 2 thanh ghi, của một thanh ghi và một vị trí nhớ, hoặc cộng (trừ) một số với (khỏi) một thanh ghi hoặc một vị trí nhớ. Cú pháp là :

ADD Destination, Source

SUB Destination, Source

Ví dụ :

```
ADD WORD1, AX
ADD BL, 5
```

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

Vì lý do kỹ thuật, lệnh ADD và SUB cũng bị một số hạn chế như bảng sau:

	Destination operand	
Source operand	General Reg	Memory Loacation
Gen Memory	Y	Y
Memory Location	Y	NO
Constant	Y	Y

Việc cộng hoặc trừ trực tiếp giữa 2 vị trí nhớ là không được phép. Để giải quyết vấn đề này người ta phải di chuyển byte (từ) nhớ đến một thanh ghi sau đó mới cộng hoặc trừ thanh ghi này với một byte (từ) nhớ khác. Ví dụ:

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

Lệnh INC (incremrent) để cộng thêm 1 vào nội dung của một thanh ghi hoặc một vị trí nhớ. Lệnh DEC (decrement) để giảm bớt 1 khỏi một thanh ghi hoặc 1 vị trí nhớ. Cú pháp của chúng là:

INC Destination
DEC Destination

Ví dụ :

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

1.5.3 Lệnh NEG (negative)

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

NEG destination

Ví dụ : NEG AX ;
Giả sử AX=0002h sau khi thực hiện lệnh NEG AX thì AX=FFFEh
LƯU Ý : 2 toán hạng trong các lệnh trên đây phải cùng loại (cùng là byte hoặc từ)

1.6 Chuyển ngôn ngữ cấp cao thành ngôn ngữ ASM

Giả sử A và B là 2 biến từ . Chúng ta sẽ chuyển các mệnh đề sau trong ngôn ngữ cấp cao ra ngôn ngữ ASM .

1.6.1 Mệnh đề B=A

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

1.6.2 Mệnh đề A=5-A

```
MOV AX,5 ; đưa 5 vào AX
SUB AX,A ; AX=5-A
MOV A,AX ; A=5-A
```

cách khác :

```
NEG A ;A=-A
ADD A,5 ;A=5-A
```

*1.6.3 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
```

1.7 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 một phần chiếm một đoạn bộ nhớ. Mỗi một đoạn chương trình là được chuyển thành một đoạn bộ nhớ bởi ASM .

1.7.1 Các kiểu bộ nhớ (memory models)

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

.MODEL memory_model

Bảng sau cho thấy các kiểu bộ nhớ :

Model	**Description**
Small	code và data năm trong 1 đoạn
Medium	code nhiều hơn 1 đoạn , data trong 1 đoạn
Compact	data nhiều hơn 1 đoạn , code trong 1 đoạn
Large	code và dayta lớn hơn 1 đoạn , array không qúa 64KB
Huge	code ,data lớn hớn 1 đoạn , array lớn hơn 64KB

1.7.2 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 như sau :

.DATA

;khai báo tên các biến, hằng và mãng

ví dụ :

```
.DATA
WORD1 DW 2
WORD2 DW 5
MSG DB 'THIS IS A MESSAGE '
MASK EQU 10010010B
```

1.7.3 Đoạn ngăn xếp

Mục đích của việc khai báo đoạn ngăn xếp là dành một vùng nhớ (vùng satck) để lưu trữ cho stack. Cú pháp của lệnh như sau :

.STACK size

nếu không khai báo size thì 1KB được dành cho vùng stack .

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

1.7.4 Đoạn mã

Đoạn mã chứa các lệnh của chương trình. Bắt đầu đoạn mã bằng chỉ dẫn CODE như sau :

.CODE

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

name PROC
; body of the procedure
name ENDP

Sau đây là câú trúc của một chương trình hợp ngữ mà phần CODE là thủ tục có tên là MAIN

```
.MODEL SMALL
.STACK 100h
.DATA
; định nghĩa số liệu tại đây
.CODE
MAIN PROC
;thân của thủ tục MAIN
MAIN ENDP
; các thủ tục khác nếu có
END MAIN
```

1.8 Các lệnh vào ra

CPU thông tin với các 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 2 lý do:

- các địa chỉ cổng thay đổi tuỳ theo 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 (Basic Input Output System) và các routine của DOS .

Lệnh INT (interrupt)

Để gọi các chương trình con của BIOS và DOS có thể dùng lệnh INT với cú pháp như sau :

INT interrupt_number

ở đây interrupt_number là một số mà nó chỉ định một routine. Ví dụ INT 16h gọi routine thực hiện việc nhập số liệu từ Keyboard .

1.8.1 Lệnh INT 21h

INT 21h được dùng để gọi một số lớn các các hàm (function) của DOS. Tuỳ theo giá trị mà chúng ta đặt vào thanh ghi AH, INT 21h sẽ gọi chạy một routine tương ứng .

Trong phần này chúng ta sẽ quan tâm đến 2 hàm sau đây :

FUNCTION NUMBER	ROUTINE
1	Single key input
2	Single character output

FUNTION 1 : Single key input

Input : AH=1
Output:AL= ASCII code if character key is pressed
AL=0 if non character key is pressed
Để gọi routine này thực hiện các lệnh sau :

```
MOV AH,1 ; input key function
INT 21h ; ASCII code in AL and display character on the screen
```

FUNTION 2 : Display a character or execute a control function

Input : AH=2
DL=ASCII code of the the display character or control character
Output:AL= ASCII code of the the display character or control character
Các lệnh sau sẽ in lên màn hình dấu ?

```
MOV AH,2
MOV DL,'?' ; character is '?'
INT 21H ; display character
```

Hàm 2 cũng có thể dùng để thực hiện chức năng điều khiển .Nếu DL chưá ký tự điều khiển thì khi gọi INT 21h, ký tự điều khiển sẽ được thực hiện .
Các ký tự điều khiển thường dùng là :

ASCII code (Hex)	SYMBOL	FUNCTION
7	BEL	beep
8	BS	backspace
9	HT	tab
A	LF	line feed
D	CR	carriage return

1.9 Chương trình đầu tiên

Chúng ta sẽ viết một chương trình hợp ngữ nhằm đọc một ký tự từ bàn phím và in nó trên đầu dòng mới .

```
TITLE PGM1: ECHO PROGRAM
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
; display dấu nhắc
MOV AH,2
MOV DL,'?'
INT 21H
; nhập 1 ký tự
MOV AH,1 ; hàm đọc ký tự
INT 21H ; ký tự được đưa vào AL
MOV BL,AL ; cất ký tự trong BL
; nhảy đến dòng mới
MOV AH,2 ; hàm xuất 1 ký tự
MOV DL,0DH ; ký tự carriage return
INT 21H, thực hiện carriage return
MOV DL,0AH ; ký tự line feed
INT 21H ; thực hiện line feed
; xuất ký tự
```

```
MOV DL,BL ; đưa ký tự vào DL
INT 21H ; xuất ký tự
; trở về DOS
MOV AH,4CH ; hàm thoát về DOS
INT 21H ; exit to DOS
MAIN ENDP
END MAIN
```

1.10 Tạo ra và chạy một chương trình hợp ngữ

Tham khảo cách sử dụng chương trình Emu8086 trên internet.

1.11 Xuất một chuỗi ký tự

Trong chương trình PGM1 trên đây chúng ta đã dùng INT 21H hàm 2 và 4 để đọc và xuất một ký tự. Hàm 9 ngắt 21H có thể dùng để xuất một chuỗi ký tự .

INT 21H, Function 9 : Display a string

Input : DX=offset address of string
The string must end with a '$' character
Ký tự $ ở cuối chuỗi sẽ không được in lên màn hình. Nếu chuỗi có chứa ký tự điều khiển thì chức năng điều khiển tương ứng sẽ được thực hiện .
Chúng ta sẽ viết 1 chương trình in lên màn hình chuỗi "HELLO!". Thông điệp HELLO được định nghĩa như sau trong đoạn số liệu :

```
MSG DB 'HELLO!$'
Lệnh LEA (Load Effective Address )
LEA destnation, source
```

Ngắt 21h, hàm số 9 sẽ xuất một chuỗi ký tự ra màn hình với điều kiện địa chỉ hiệu dụng của biến chuỗi phải ở trên DX. Có thể thực hiện điều này bởi lệnh :

```
LEA DX,MSG ; đưa địa chỉ offset của biến MSG vào DX
```

Program Segment Prefix (PSP) : Phần đầu của đoạn chương trình

Khi một chương trình được nạp vào bộ nhớ máy tính, DOS dành ra 256 byte cho cái gọi là PSP. PSP chứa một số thông tin về chương trình đang được nạp trong bộ nhớ. Để cho các chương trình có thể truy xuất tới PSP, DOS đặt số phân đoạn của nó (PSP) trong cả DS và ES trước khi thực thi chương trình. Kết qủa là thanh ghi DS không chứa số đoạn của đoạn số liệu của chương trình. Để khắc phục điều này, một chương trình có chứa đoạn số liệu phải được bắt đầu bởi 2 lệnh sau đây :

```
MOV AX,@DATA
MOV DS,AX
```

Ở đây @DATA là tên của đoạn số liệu được định nghĩa bởi DATA . Assembler sẽ chuyển @DATA thành số đoạn .
Sau đây là chương trình hoàn chỉnh để xuất chuỗi ký tự HELLO!

```
TITLE PGM2: PRINT STRING PROGRAM
.MODEL SMALL
.STACK 100H
.DATA
```

```
MSG DB 'HELLO!$'
.CODE
MAIN PROC
; initialize DS
MOV AX,@DATA
MOV DS,AX
; display message
LEA DX,MSG
MOV AH,9
INT 21H
; return to DOS
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN
```

1.12 Chương trình đổi chữ thường sang chữ hoa

Chúng ta sẽ viết 1 chương trình yêu cầu người dùng gõ vào một ký tự bằng chữ thường. Chương trình sẽ đổi nó sang dạng chữ hoa rồi in ra ở dòng tiếp theo .

```
TITLE PGM3: CASE COVERT PROGRAM
.MODEL SMALL
.STACK 100H
.DATA
CR EQU 0DH
LF EQU 0AH
MSG1 DB 'ENTER A LOWER CASE LETTER:$'
MSG2 DB 0DH,0AH,'IN UPPER CASE IT IS :'
CHAR DB ?,'$' ; định nghĩa biến CHAR có giá trị ban đầu chưa
;xác định
.CODE
MAIN PROC
; INITIALIZE DS
MOV AX,@DATA
MOV DS,AX
;PRINT PROMPT USER
LEA DX,MSG1 ; lấy thông điệp số 1
MOV AH,9
INT 21H ; xuất nó ra màn hình
;nhập vào một ký tự thường và đổi nó thành ký tự hoa
MOV AH,1 ; nhập vào 1 ký tự
INT 21H ; cất nó trong AL
SUB AL,20H ; đổi thành chữ hoa và cất nó trong AL
```

```
MOV CHAR, AL ; cất ký tự trong biến CHAR
; xuất ký tự trên dòng tiếp theo
LEA DX, MSG2 ; lấy thông điệp thứ 2
MOV AH,9
INT 21H ; xuất chuỗi ký tự thứ hai, vì MSG2 không kết
;thúc bởi ký tự $ nên nó tiếp tục xuất ký tự có trong biến CHAR
;dos exit
MOV AH,4CH
INT 21H ; dos exit
MAIN ENDP
END MAIN
```

Chương 2 - Trạng thái của vi xử lý và các thanh ghi cờ

Trong chương này chúng ta sẽ xem xét các thanh ghi cờ của vi xử lý và ảnh hưởng của các lệnh máy đến các thanh ghi cờ như thế nào. Trạng thái của các thanh ghi là căn cứ để chương trình có thể thực hiện lệnh nhảy, rẻ nhánh và lặp . Một phần của chương này sẽ giới thiệu chương trình DEBUG của DOS .

2.1 Các thanh ghi cờ (Flags register)

Điểm khác biệt quan trọng của máy tính so với các thiết bị điện tử khác là khả năng cho các quyết định. Một mạch đặc biệt trong CPU có thể làm các quyết định này bằng cách căn cứ vào trạng thái hiện hành của CPU. Có một thanh ghi đặc biệt cho biết trạng thái của CPU đó là thanh ghi cờ .

Bảng 2.1 cho thấy thanh ghi cờ 16 bit của 8086

11	10	9	8	7	6	5	4	3	2	1	0
OF	DF	IF	TF	SF	ZF		AF		PF		CF

Bảng 2.1 :Thanh ghi cờ của 8086

Mục đích của các thanh ghi cờ là chỉ ra trạng thái của CPU .Có hai loại cờ là cờ trạng thái (status flags) và cờ điều khiển (control flags). Cờ trạng thái phản ánh các kết qủa thực hiện lệnh của CPU. Bảng 2.2 chỉ ra tên và

Bit	Name	Symbol
0	Carry flag	CF
2	Parity flag	PF
4	Auxiliary carry flag	AF
6	Zero flag	ZF
7	Sign flag	SF
11	Overflow flag	OF
8	Trap flag	TF
9	Interrrupt flag	IF
10	Direction flag	DF

Bảng 2.2 : Các cờ của 8086

Mỗi bit trên thanh ghi cờ phản ánh 1 trạng thái của CPU .

Các cờ trạng thái (status flags)

Các cờ trạng thái phản ánh kết quả của các phép toán. Ví dụ sau khi thực hiện lệnh SUB AX,AX cờ ZF =1, nghĩa là kết qủa của phép trừ là zero .

Cờ nhớ (Carry Flag - CF) : CF=1 nếu xuất hiện bit nhớ (carry) từ vị trí MSB trong khi thực hiện phép cộng hoặc có bit mượn (borrow) tại MSB trong khi thực hiện phép trừ. Trong các trường hợp khác CF=0. Cờ CF cũng bị ảnh hưởng bởi lệnh dịch (Shift) và quay (Rotate) số liệu .

Cờ chẵn lẻ (Parity Flag - PF) : PF=1 nếu byte thấp của kết qủa có tổng số con số 1 là một số chẵn (even parity). PF=0 nếu byte thấp là chẵn lẻ lẻ (old parity). Ví dụ nếu kết qủa là FFFEh thì PF=0

Cờ nhớ phụ (Auxiliary Carry Flag - AF) :AF =1 nếu có nhớ (mượn) từ bit thứ 3 trong phép cộng (trừ) .

Cờ Zero (Zero Flag -ZF) : ZF=1 nếu kết qủa là số 0 .

Cờ dấu (Sign Flag - SF) : SF=1 nếu MSB của kết qủa là 1 (kết qủa là số âm). SF=0 nếu MSB=0

Cờ tràn (Overflow Flag - OF) : OF=1 nếu xảy ra tràn số trong khi thực hiện các phép toán. Sau đây chúng ta sẽ phân tích các trường hợp xảy ra tràn trong khi thực hiện tính toán. Hiện tượng tràn số liên quan đến việc biểu diễn số trong máy tính với một số hữu hạn các bit. Các số thập phân có dấu biểu diễn bởi 1 byte là -128 đến +127. Nếu biểu diễn bằng 1 từ (16 bit) thì các số thập phân có thể biểu diễn là -32768 đến +32767. Đối với các số không dấu, dải các số có thể biểu diễn trong một từ là 0 đến 65535, trong một byte là 0 đến 255. Nếukết qủa của một phép toán vượt ra ngoài dãi số có thể biểu diễn thì xảy ra sự tràn số. Khi có sự tràn số kết qủa thu được sẽ bị sai .

2.2 Tràn (overflow)

Có 2 loại tràn số : Tràn có dấu (signed overflow) và tràn không dấu (unsigned overflow). Khi thực hiện phép cộng số học chẳng hạn phép cộng, sẽ xảy ra 4 khả năng sau đây :

1) không tràn
2) chỉ tràn dấu
3) chỉ tràn không dấu
4) tràn cả dấu và không dấu

Ví dụ của tràn không dấu là phép cộng ADD AX,BX với AX=0FFFFh, BX=0001h .Kết qủa dưới dạng nhị phân là :

```
1111 1111 1111 1111
0000 0000 0000 0001
10000 0000 0000 0000
```

Nếu diễn giải kết qủa dưới dạng không dấu thì kết qủa là đúng (10000h=65536). Nhưng kết qủa đã vượt quá độ lớn của từ nhớ. Bit 1 (bit nhớ từ vị trí MSB) đã xảy ra và kết qủa trên AX =0000h là sai . Sự tràn như thế là tràn không dấu. Nếu xem rằng phép cộng trên đây là phép cộng hai số có dấu thì kết qủa trên AX = 0000h là đúng, vì FFFFh = -1 , còn 0001h = +1, do đó kết qủa phép cộng là 0. Vậy trong trường hợp này sự tràn dấu không xảy ra .

Ví dụ về sự tràn dấu : giả sử AX = BX = 7FFFh , lệnh ADD AX,BX sẽ cho kết qủa như sau :

```
0111 1111 1111 1111
0111 1111 1111 1111
1111 1111 1111 1110 = FFFE h
```

Biểu diễn có dấu và không dấu của 7FFFh là 3276710. Như vậy là đối với phép cộng có dấu cũng như không dấu thì kết qủa vẫn là 32767 + 32767 = 65534. Số này(65534) đã vượt ngoài dãi giá trị mà 1 số 16 bit có dấu có thể biểu diễn. Hơn nửa FFFEh = -2. Do vậy sự tràn dấu đã xảy ra. Trong trường hợp xảy ra tràn, CPU sẽ biểu thị sự tràn như sau :

- CPU sẽ set OF =1 nếu xảy ra tràn dấu
- CPU sẽ set CF = 1 nếu xảy ra tràn không dấu

Sau khi có tràn, một chương trình hợp lý sẽ được thực hiện để sửa sai kết qủa ngay lập tức. Các lập trình viên sẽ chỉ phải quan tâm tới cờ OF hoặc CF nếu biểu diễn số của họ là có dấu hay không dấu một cách tương ứng .

Vậy thì làm thế nào để CPU biết được có tràn ?

- Tràn không dấu sẽ xảy ra khi có một bit nhớ (hoặc mượn) từ MSB
- Tràn dấu sẽ xảy ra trong các trường hợp sau :

a) Khi cộng hai số cùng dấu, sự tràn dấu xảy ra khi tổng có dấu khác với hai toán hạng ban đầu . Trong ví dụ 2, cộng hai số 7FFFh +7FFFh (hai số dương) nhưng kết qủa là FFFFh (số âm)

b) Khi trừ hai số khác dấu (giống như cộng hai số cùng dấu) kết qủa phải có dấu hợp lý .Nếu kết qủa cho dấu không như mong đợi thì có nghĩa là đã xảy ra sự tràn dấu. Ví dụ 8000h - 0001h = 7FFFh (số dương). Do đó OF=1 .

Vậy làm thế nào để CPU chỉ ra rằng có tràn ?

- OF=1 nếu tràn dấu
- CF=1 nếu tràn không dấu

Làm thế nào để CPU biết là có tràn ?

- Tràn không dấu xảy ra khi có số nhớ (carry) hoặc mượn (borrow) từ MSB
- Tràn dấu xảy ra khi cộng hai số cùng dấu (hoặc trừ 2 số khác dấu) mà kết qủa với dấu khác với dấu mong đợi
- Phép cộng hai số có dấu khác nhau không thể xảy ra sự tràn. Trên thực tế CPU dùng phương pháp sau : cờ OF=1 nếu số nhớ vào và số nhớ ra từ MSB là không phù hợp. Nghĩa là có nhớ vào nhưng không có nhớ ra hoặc có nhớ ra nhưng không có nhớ vào .

Cờ điều khiển (control flags)

Có 3 cở điều khiển trong CPU, đó là :

- Cờ hướng (Direction Flag = DF)
- Cờ bẫy (Trap flag = TF)
- Cờ ngắt (Interrupt Flag = IF)

Các cờ điều khiển được dùng để điều khiển hoạt động của CPU

Cờ hướng (DF) được dùng trong các lệnh xử lý chuỗi của CPU. Mục đích của DF là dùng để điều khiển hướng mà một chuỗi được xử lý. Trong các lệnh xử lý chuỗi hai thanh ghi DI và SI được dùng để địa chỉ bộ nhớ chứa chuỗi. Nếu DF=0 thì lệnh xử lý chuỗi sẽ tăng địa chỉ bộ nhớ sao cho chuỗi được xử lý từ trái sang phải Nếu DF=1 thì địa chỉ bộ nhớ sẽ được xử lý theo hướng từ phải sang trái .

2.3 Các lệnh ảnh hưởng đế cờ như thế nào

Tại một thời điểm, CPU thực hiện 1 lệnh, các cờ lần lượt phản ánh kết qủa thực hiện lệnh. Dĩ nhiên có một số lệnh không làm thay đổi một cờ nào cả hoặc thay đổi chỉ 1 vài cờ hoặc làm cho một vài cờ có trạng thái không xác định. Trong phần này chúng ta chỉ xét ảnh hưởng của các lệnh (đã nghiên cứu ở chương trước) lên các cờ như thế nào .

Bảng sau đây cho thấy ảnh hưởng của các lệnh đến các cờ :

INSTRUCTION	**AFFECTS FLAGS**
MOV/XCHG	NONE
ADD/SUB	ALL
INC/DEC	ALL trừ CF
NEG	ALL (CF=1 trừ khi kết qủa bằng 0 , OF=1 nếu kết qủa là 8000H)

Để thấy rỏ ảnh hưởng của các lệnh lên các cờ chúng ta sẽ lấy vài ví dụ .

Ví dụ 1 : ADD AX,AX trong đó AX=BX=FFFFh

```
    FFFFh
+   FFFFh
   1FFFEh
```

Kết qủa chứa trên AX là FFFEh = 1111 1111 1111 1110

SF=1 vì MSB=1

PF=0 vì có 7 (lẻ) số 1 trong byte thấp của kết qủa
ZF=0 vì kết qủa khác 0
CF=1 vì có nhớ 1 từ MSB
OF=0 vì dấu của kết qủa giống như dấu của 2 số hạng ban đầu .

Ví dụ 2 : ADD AL,BL trong đó AL= BL= 80h

```
  80h
+ 80h
 100h
```

Kết qủa trên AL = 00h
SF=0 vì MSB=0
PF=1 vì tất cả các bit đều bằng 0
ZF=1 vì kết qủa bằng 0
CF=1 vì có nhớ 1 từ MSB
OF=1 vì cả 2 toán hạng là số âm nhưng kết qủa là số dương (có nhớ ra từ MSB nhưng không có nhớ vào) .

Ví dụ 3 : SUB AX,BX trong đó AX=8000h và BX= 0001h

```
  8000h
- 0001h
 7FFFh = 0111 1111 1111 1111
```

SF=0 vì MSB=0
PF=1 vì có 8 (chẳn) số 1 trong byte thấp của kết qủa
ZF=0 vì kết qủa khác 0
CF=0 vì không có mượn
OF=1 vì trừ một số âm cho 1 số dương (tức là cộng 2 số âm) mà kết qủa là một số dương .

Ví dụ 4 : INC AL trong đó AL=FFh
Kết qủa trên AL=00h = 0000 0000
SF=0 vì MSB=0
PF=1
ZF=1 vì kết qủa bằng 0
CF không bị ảnh hưởng bởi lệnh INC mặc dù có nhớ 1 từ MSB
OF=0 vì hai số khác dấu được cộng với nhau (có số nhớ vào MSB và cũng có số nhớ ra từ MSB)

Ví dụ 5: MOV AX,-5
Kết quả trên BX = -5 = FFFBh
Không có cờ nào ảnh hưởng bởi lệnh MOV

Ví dụ 6: NEG AX trong đó AX=8000h
8000h =1000 0000 0000 0000
bù 1 =0111 1111 1111 1111
+1
1000 0000 0000 0000 = 8000h
Kết qủa trên AX=8000h
SF=1 vì MSB=1

PF=1 vì có số chẵn con số 1 trong byte thấp của kết qủa
ZF=0 vì kết qủa khác 0
CF=1 vì lệnh NEG làm cho CF=1 trừ khi kết qủa bằng 0
OF=1 vì dấu của kết qủa giống với dấu của toán hạng nguồn .

Chương 3 - CÁC LỆNH ĐIỀU KHIỂN

Một chương trình thông thường sẽ thực hiện lần lượt các lệnh theo thứ thự mà chúng được viết ra. Tuy nhiên trong một vài trường hợp cần phải chuyển điều khiển đến 1 phần khác của chương trình. Trong phần này chúng ta sẽ nghiên cứu các lệnh nhảy và lệnh lặp có tính đến cấu trúc của các lệnh này trong các ngôn ngữ cấp cao .

3.1 Ví dụ về lệnh nhảy

Để hình dung được lệnh nhảy làm việc như thế nào chúng ta hãy viết chương trình in ra toàn bộ tập các ký tự IBM .

```
TITLE PGR3-1:IBM CHARACTER DISPLAY
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
MOV AH,2 ; hàm xuất ký tự
MOV CX,256 ; số ký tự cần xuất
MOV DL,0 ; DL giữ mã ASCII của ký tự NUL
; PRINT_LOOP :
INT 21H ;display character
INC DL
DEC CX
JNZ PRINT_LOOP ;nhảy đến print_loop nếu CX# 0
;DOS EXIT
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN
```

Trong chương trình chúng ta đã dùng lệnh điều khiển ***Jump if not zero (JNZ)*** để quay trở lại đoạn chương trình xuất ký tự có nhãn địa chỉ bộ nhớ làPRINT_LOOP

3.2 Nhảy có điều kiện

Lệnh JNZ là một lệnh nhảy có điều kiện .Cú pháp của một lệnh nhảy có điều kiện là :

Jxxx destination-label

Nếu điều kiện của lệnh được thỏa mãn thì lệnh tại Destination-label sẽ được thực hiện, nếu điều kiện không thỏa thì lệnh tiếp theo lệnh nhảy sẽ được thực hiện.
Đối với lệnh JNZ thì điều kiện là kết qủa của lệnh trước nó phải bằng 0 .

Phạm vi của lệnh nhảy có điều kiện .

Cấu trúc mã máy của lệnh nhảy có điều kiện yêu cầu destination-label đến (precede) lệnh nhảy phải không quá 126 bytes .

Làm thế nào để CPU thực hiện một lệnh nhảy có điều kiện ?

Để thực hiện một lệnh nhảy có điều kiện CPU phải theo dõi thanh ghi cờ.

Nếu điều kiện cho lệnh nhảy (được biểu diễn bởi một tổ hợp trạng thái các cờ) là đúng thì CPU sẽ điều chỉnh IP đến destination-label sao cho lệnh tại địa chỉ destination-label được thực hiện. Nếu điều kiện nhảy không thỏa thì IP sẽ không thay đổi, nghĩa là lệnh tiếp theo lệnh nhảy sẽ được thực hiện .

Trong chương trình trên đây, CPU thực hiện lệnh JNZ PRINT_LOOP bằng cách khám xét các cờ ZF. Nếu ZF=0 điều khiển được chuyển tới PRINT_LOOP.

Nếu ZF=1 lệnh MOV AH,4CH sẽ được thực hiện .

Bảng 3-1 cho thấy các lệnh nhảy có điều kiện. Các lệnh nhảy được chia thành 3 loại :

- nhảy có dấu (dùng cho các diễn dịch có dấu đối với kết quả)
- nhảy không dấu (dùng cho các diễn dịch không dấu đối với kết quả)
- nhảy một cờ (dùng cho các thao tác chỉ ảnh hưởng lên 1 cờ)

Một số lệnh nhảy có 2 Opcode. Chúng ta có thể dùng một trong 2 Opcode, nhưng kết quả thực hiện lệnh là như nhau .

Nhảy có dấu

SYMBOL	DESCRITION CONDITION FOR JUMPS
JG/JNLE	jump if greater than
ZF=0 and SF=OF	jump if not less than or equal to
JGE/JNL	jump if greater than or equal to
SF=OF	jupm if not less or equal to
JL/JNGE	jump if lees than
	jump if not greater or equal SF<>OF
JLE/JNG	jump if less than or equal ZF=1 or SF<>OF
	jump if not greater

Nhảy có điều kiện không dấu

SYMBOL	DESCRITION CONDITION FOR JUMPS
JA/JNBE	jump if above CF=0 and ZF=0
	jump if not below or equal
JAE/JNB	jump if above or equal CF=0
	jump if not below
JB/JNA	jump if below Cf=1
	jump if not above or equal
JBE/JNA	jump if below or equal CF=1 or ZF=1
	jump if not above

Nhảy 1 cờ

SYMBOL	DESCRITION CONDITION FOR JUMPS
JE/JZ	jump if equal ZF=1
	jump if equal to zero
JNE/JNZ	jump if not equal ZF=0
	jump if not zero
JC	jump if carry CF=1
JNC	jump if no carry CF=0
JO	jump if overflow OF=1
JNO	jump if not overflow OF=0
JS	jump if sign negative SF=1
JNS	jump if nonnegative sign SF=0

JP/JPE	jump if parity even PF=1
JNP/JPO	jump if parity odd PF=0

Lệnh CMP (Compare)

Các lệnh nhảy thường lấy kết qủa của lệnh Compare như là điều kiện. Cú pháp của lệnh CMP là :

CMP destination, source

Lệnh này so sánh toán hạng nguồn và toán hạng đích bằng cách tính hiệu Destinaition - Source. Kết qủa sẽ không được cất giữ. Như vậy là lệnh CMP giống như lệnh SUB, chỉ khác là trong lệnh CMP toán hạng đích không thay đổi .

Giả sử chương trình chứa các lệnh sau :

```
CMP AX,BX ;trong đó AX=7FFF và BX=0001h
JG BELOW
```

Kết qủa của lệnh CMP AX,BX là 7FFEh. Lệnh JG được thỏa mãn vì ZF=0=SF=OF do đó điều khiển được chuyển đến nhãn BELOW.

Diễn dịch lệnh nhảy có điều kiện

Ví dụ trên đây về lệnh CMP cho phép lệnh nhảy sau nó chuyển điều khiển đến nhãn BELOW. Đây là ví dụ cho thấy CPU thực hiện lệnh nhảy như thế nào . Chúng thực hiện bằng cách khám xét trạng thaí các cờ .Lập trình viên không cần quan tâm đến các cờ, mà có thể dùng tên của các lệnh nhảy để chuyển điều khiển đến một nhãn nào đó. Các lệnh

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

có nghĩa là nếu AX>BX thì nhảy đến nhãn BELOW

Mặc dù lệnh CMP được thiết kế cho các lệnh nhảy. Nhưng lệnh nhảy có thể đứng trước 1 lệnh khác, chẳng hạn :

```
DEC AX
JL THERE
```

có nghĩa là nếu AX trong diễn dịch có dấu < 0 thì điều khiển được chuyển cho THERE .

Nhảy có dấu so với nhảy không dấu

Một lệnh nhảy có dấu tương ứng với 1 nhảy không dấu. Ví dụ lệnh nhảy có dấu JG và lệnh nhảy không dấu JA. Việc sử dụng JG hay JA là tuỳ thuộc vào diễn dịch có dấu hay không dấu. Bảng 3-1 cho thấy các lệnh nhảy có dấu phụ thuộc vào trạng thái của các cờ ZF,SF,OF .Các lệnh nhảy không dấu phụ thuộc vào trạng thái của các cờ ZF và CF. Sử dụnh lệnh nhảy không hợp lý sẽ tạo ra kết quả sai .

Giả sử rằng chúng ta diễn dịch có dấu .Nếu AX=7FFFh và BX=8000h, các lệnh :

```
CMP AX,BX
JA below
```

sẽ cho kết qủa sai mặc dù 7FFFh > 8000h (lệnh JA không thực hiện được vì 7FFFFh < 8000h trong diễn dịch không dấu)

Sau đây chúng ta sẽ lấy ví dụ để minh họa việc sử dụng các lệnh nhảy Ví dụ : Giả sử rằng AX và BX chứa các số có dấu. Viết đoạn ct để đặt số lớn nhất vào CX .

Giải :

```
MOV CX,AX ; đặt AX vào CX
CMP BX,CX ;BX lớn hơn CX?
```

```
JLE NEXT ; không thì tiếp tục
MOV CX,BX ; yes, đặt BX vào CX
NEXT:
```

3.3 Lệnh JMP

Lệnh JMP (jump) là lệnh nhảy không điều kiện. Cú pháp của JMP là

JMP destination

Trong đó destination là một nhãn ở trong cùng 1 đoạn với lệnh JMP . Lệnh JMP dùng để khắc phục hạn chế của các lệnh nhảy có điều kiện (không quá 126 bytes kể từ vị trí của lệnh nhảy có điều kiện)

Ví dụ chúng ta có đoạn chương trình sau :

```
TOP:
; thân vòng lặp
DEC CX
JNZ TOP ; nếu CX>0 tiếp tục lặp
MOV AX,BX
```

giả sử thân vòng lặp chứa nhiều lệnh mà nó vượt khỏi 126 bytes trước lệnh JNZ TOP. Có thể giải quyết tình trạng này bằng các lệnh sau :

```
TOP:
; thân vòng lặp
DEC CX
JNZ BOTTOM ; nếu CX>0 tiếp tục lặp
JMP EXIT
BOTTOM:
JMP TOP
EXIT:
MOV AX,BX
```

3.4 Cấu trúc của ngôn ngữ cấp cao

Chúng ta sẽ dùng các lệnh nhảy để thực hiện các cấu trúc tương tự như trong ngôn ngữ cấp cao

3.4.1 Cấu trúc rẽ nhánh

Trong ngôn ngữ cấp cao cấu trúc rẽ nhánh cho phép một chương trình rẽ nhánh đến những đoạn khác nhau tuỳ thuộc vào các điều kiện. Trong phần này chúng ta sẽ xem xét 3 cấu trúc

a) IF-THEN

Cấu trúc IF-THEN có thể diễn đạt như sau :

```
IF condition is true
THEN
execute true branch statements
END IF
```

Ví dụ : Thay thế giá trị trên AX bằng giá trị tuyết đối của nó

Thuật toán như sau :

```
IF AX<0
```

```
THEN
replace AX by -AX
END-IF
```

Có thể mã hoá như sau :

```
; if AX<0
CMP AX,0
JNL END_IF ; no, exit
;then
NEG AX, yes, change sign
END_IF :
```

b) IF_THEN_ELSE

```
IF condition is true
THEN
execute true branch statements
ELSE
execute false branch statements
END_IF
```

Ví dụ : giả sử AL và BL chứa ASCII code của 1 ký tự .Hãy xuất ra màn hình ký tự trước (theo thứ tự ký tự)

Thuật toán

```
IF AL<= BL
THEN
display AL
ELSE
display character in BL
END_IF
```

Có thể mã hoá như sau :

```
MOV AH,2 ; chuẩn bị xuất ký tự
;if AL<=BL
CMP AL,BL ;AL<=BL?
JNBE ELSE_ ; no, display character in BL
;then
MOV DL,AL
JMP DISPLAY
ELSE_:
MOV DL,BL
DISPLAY:
INT 21H
END_IF :
```

c) CASE

Case là một cấu trúc rẽ nhánh nhiều hướng. Có thể dùng để test một thanh ghi hay, biến nào đo hay một biểu thức mà giá trị cụ thể nằn trong 1 vùng các giá trị

Cấu trúc của CASE như sau :

```
CASE expression
value_1 : Statements_1
value_2 : Statements_2
.
.
value_n : Statements_n
```

Ví dụ : Nếu AX âm thì đặt -1 vào BX
Nếu AX bằng 0 thì đặt 0 vào BX
Nếu AX dương thì đặt 1 vào BX

Thuật toán :

```
CASE AX
< 0 put -1 in BX
= 0 put 0 in BX
> 0 put 1 in BX
```

Có thể mã hoá như sau :

```
; case AX
CMP AX,0 ;test AX
JL NEGATIVE ;AX<0
JE ZERO ;AX=0
JG positive ;AX>0
NEGATIVE:
MOV BX,-1
JMP END_CASE
ZERO:
MOV BX,0
JMP END_CASE
POSITIVE:
MOV BX,1
JMP END_CASE
END_CASE :
```

Rẻ nhánh với một tổ hợp các điều kiện

Đôi khi tình trạng rẻ nhánh trong các lệnh IF ,CASE cần một tổ hợp các điều kiện dưới dạng :

```
Condition_1 AND Condition_2
Condition_1 OR Condition_2
```

Ví dụ về điều kiện AND : Đọc một ký tự và nếu nó là ký tự hoa thì in nó ra màn hình

Thuật toán :

```
Read a character (into AL)
IF ('A'<= character ) AND (charater <= 'Z')
```

```
THEN
display character
END_IF
```

Sau đây là code

```
;read a character
MOV AH,2
INT 21H ; character in AL
; IF ('A'<= character ) AND (charater <= 'Z')
CMP AL,'A' ; char >='A'?
JNGE END_IF ;no, exit
CMP AL,'Z ; char <='Z'?
JNLE END_IF ; no exit
; then display it
MOV DL,AL
MOV AH,2
INT 21H
END_IF :
```

Ví dụ về điều kiện OR : Đọc một ký tự, nếu ký tự đó là 'Y' hoặc 'y' thì in nó lên màn hình, ngược lại thì kết thúc chương trình .

Thuật toán

```
Read a charcter (into AL)
IF (character ='Y') OR (character='y')
THEN
dispplay it
ELSE
terminate the program
END_IF
```

Code

```
;read a character
MOV AH,2
INT 21H ; character in AL
; IF (character ='y' ) OR (charater = 'Y')
CMP AL,'y' ; char ='y'?
JE THEN ;yes, goto display it
CMP AL,'Y' ; char ='Y'?
JE THEN ; yes, goto display it
JMP ELSE_ ;no, terminate
THEN :
MOV DL,AL
MOV AH,2
INT 21H
JMP END_IF
```

```
ELSE_:
MOV AH,4CH
INT 21h
END_IF :
```

4.3.2 Cấu trúc lặp

Một vòng lặp gồm nhiều lệnh được lặp lại, số lần lặp phụ thuộc điều kiện .

*a) Vòng **FOR***

Lệnh LOOP có thể dùng để thực hiện vòng FOR .Cú pháp của lệnh LOOP như sau :

LOOP destination_label

Số đếm cho vòng lặp là thanh ghi CX mà ban đầu nó được gán 1 giá trị nào đó. Khi lệnh LOOP được thực hiện CX sẽ tự động giảm đi 1. Nếu CX chưa bằng 0 thì vòng lặp được thực hiện tiếp tục. Nếu CX=0 lệnh sau lệnh LOOP được thực hiện

Dùng lệnh LOOP, vòng FOR có thể thực hiện như sau :

```
; gán cho cho CX số lần lặp
TOP:
; thân của vòng lặp
LOOP TOP
```

Ví dụ : Dùng vòng lặp in ra 1 hàng 80 dấu '*'

```
MOV CX,80 ; CX chuá số lần lặp
MOV AH,2 ; hàm xuất ký tự
MOV DL,'*' ;DL chuá ký tự '*'
TOP:
INT 21h ; in dấu '*'
LOOP TOP ; lặp 80 lần
```

Lưu ý rằng vòng FOR cũng như lệnh LOOP thực hiện ít nhất là 1 lần. Do đo nếu ban đầu CX=0 thì vòng lặp sẽ làm cho CX=FFFH ,tức là thực hiện lặp đến 65535 lần. Để tránh tình trạng này, lệnh JCXZ (Jump if CX is zero) phải được dùng trước vòng lặp.

Lệnh JXCZ có cú pháp như sau :

JCXZ destination_label

Nếu CX=0 điều khiển được chuyển cho destination_label. Các lệnh sau đây sẽ đảm bảo vòng lặp không thực hiện nếu CX=0

```
JCXZ SKIP
TOP :
; thân vòng lặp
LOOP TOP
SKIP :
```

b) Vòng WHILE

Vòng WHILE phụ thuộc vào 1 điều kiện .Nếu điều kiện đúng thì thực hiện vòng WHILE. Vì vậy nếu điều kiện sai thì vòng WHILE không thực hiện gì cả .

Ví dụ : Viết đoạn mã để đếm số ký tự được nhập vào trên cùng một 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?
JE END_WHILE ; đúng, thoát
INC DX ;tăng DX lên 1
INT 21h ; đọc ký tự
JMP WHILE_ ; lặp
END_WHILE :
```

c) Vòng REPEAT

Cấu trúc của REPEAT là

repeat statements
until condition

Trong cấu trúc repeat mệnh đề được thi hành đồng thời điều kiện được kiểm tra. Nếu điều kiện đúng thì vòng lặp kết thúc .

Ví dụ : viết đoạn mã để đọc vào các ký tự cho đến khi gặp ký tự trống .

```
MOV AH,1 ; đọc ký tự
REPEAT:
INT 21h ; ký tự trên AL
;until
CMP AL,' ' ; AL=' '?
JNE REPEAT
```

Lưu ý : việc sử dụng REPEAT hay WHILE là tuỳ theo chủ quan của mỗi người. Tuy nhiên có thể thấy rằng REPEAT phải tiến hành ít nhất lần, trong khi đó WHILE có thể không tiến hành lần nào cả nếu ngay từ đầu điều kiện đã bị sai .

3.5 Lập trình với cấu trúc cấp cao

Bài toán : Viết chương trình nhắc người dùng gõ vào một dòng văn bản . Trên 2 dòng tiếp theo in ra ký tự viết hoa đầu tiên và ký tự viết hoa cuối cùng theo thứ tự alphabetical. Nếu người dùng gõ vào một ký tự thường, máy sẽ thông báo 'No capitals'

Kết qủa chạy chương trình sẽ như sau :

Type a line of text :
TRUONG DAi HOC DALAT
First capital = A
Last capital = U

Để giải bài toán này ta dùng kỹ thuật lập trình TOP-DOWN, nghĩa là chia nhỏ bài toán thành nhiều bài toán con. Có thể chia bài toán thành 3 bài toán con như sau :

1. Xuất 1 chuỗi ký tự (lời nhắc)
2. Đọc và xử lý 1 dòng văn bản
3. In kết qủa

Bước 1: Hiện dấu nhắc .

Bước này có thể mã hoá như sau :

```
MOV AH,9 ; hàm xuất chuỗi
LEA DX,PRMOPT ;lấy địa chỉ chuỗi vào DX
```

```
INT 21H ; xuất chuỗi
```

Dấu nhắc có thể mã hoá như sau trong đoạn số liệu .

```
PROMPT DB 'Type a line of text :',0DH,0AH,'$'
```

Bước 2 : Đọc và xử lý một dòng văn bản

Bước này thực hiện hầu hết các công việc của chương trình : đọc các ký tự từ bàn phím, tìm ra ký tự đầu và ký tự cuối, nhắc nhở người dùng nếu ký tự gõ vào không phải là ký tự hoa .
Có thể biểu diễn bước này bởi thuật toán sau :

```
Read a character
WHILE character is not a carrige return DO
IF character is a capital (*)
THEN
IF character precedes first capital
Then
first capital= character
End_if
IF character follows last character
Then
last character = character
End_if
END_IF
Read a character
END_WHILE
```

Trong đó dòng (*) có nghĩa là điều kiện để ký tự là hoa là điều kiện AND

```
IF ('A'<= character ) AND (character <= 'Z')
```

Bước 2 có thể mã hoá như sau :

```
MOV AH,1 ; đọc ký tự
INT 21H ; ký tự trên AL
WHILE :
;trong khi ký tự gõ vào không phải là CR thì thực hiện
CMP AL,0DH ; CR?
JE END_WHILE ;yes, thoát
; nếu ký tự là hoa
CMP AL,'A' ; char >='A'?
JNGE END_IF ;không phải ký tự hoa thì nhảy đến END_IF
CMP AL,'Z' ; char <= 'Z'?
JNLE END_IF ; không phải ký tự hoa thì nhảy đến END_IF
; thì
Chương 3 : Các lệnh lặp và rẽ nhánh 41
; nếu ký tự nằm trước biến FIRST (giá trị ban đầu là'[' : ký tự sau Z )
CMP AL,FISRT ; char < FIRST ?
JNL CHECK_LAST; >=
```

```
; thì ký tự viết hoa đầu tiên = ký tự
MOV FIRST,AL ; FIRST=character
;end_if
CHECK_LAST:
; nếu ký tự là sau biến LAST (giá trị ban đầu là '@': ký tự trước
A)
CMP AL,LAST ; char > LAST ?
JNG END_IF ; <=
;thì ký tự cuối cùng = ký tự
MOV LAST, AL ;LAST = character
;end_if
END_IF :
; đọc một ký tự
INT 21H ; ký tự trên AL
JMP WHILE_ ; lặp
END_WHILE:
```

Các biến FIRST và LAST được định nghĩa như sau trong đoạn số liệu :

FIRST DB '[$' ; '[' là ký tự sau Z

LAST DB '@ $ ' ; '@' là ký tự trước A

Bước 3 : In kết qủa

Thuật toán

```
IF no capital were typed
THEN
display 'No capital'
ELSE
display first capital and last capital
END_IF
Bước 3 sẽ phải in ra các thông báo :
· NOCAP_MSG nếu không phải chữ in
· CAP1_MSG chữ in đầu tiên
· CAP2_MSG chữ in cuối cùng
Chúng được định nghĩa như sau trong đoạn số liệu .
NOCAP_MSG DB 0DH,0AH,'No capitals $'
CAP1_MSG DB 0DH,0AH, 'First capital= '
FIRST DB '[ $ '
CAP2_MSG DB 0DH,0AH,'Last capital='
LAST DB '@ $'
```

Bước 3 có thể mã hoá như sau :

```
;in kết quả
MOV AH,9 ; hàm xuất ký tự
; IF không có chữ hoa nào được nhập thì FIRST ='['
CMP FIRST,'[' ; FIRST='[' ?
```

```
JNE CAPS ; không, in kết qủa
;THEN
LEA DX,NOCAP_MSG
INT 21H
CAPS:
LEA DX,CAP1_MSG
INT 21H
LEA DX,CAP2_MSG
INT 21H
; end_if
```

Chương trình có thể viết như sau :

```
TITLE PGM3-1 : FIRST AND LAST CAPITALS
.MODEL SMALL
.STACK 100h
.DATA
     PROMPT DB 'Type a line of text', 0DH, AH, '$'
     NOCAP_MSG DB 0DH,0AH, 'No capitals $'
     CAP1_MSG DB 0DH,0AH, 'First capital='
     FIRST DB '[ $'
     CAP2_MSG DB 'Last capital = '
     LAST DB '@ $'
.CODE
MAIN PROC
; khởi tạo DS
MOV AX,@DATA
MOV DS,AX
; in dấu nhắc
MOV AH,9 ; hàm xuất chuỗi
LEA DX,PROMPT ;lấy địa chỉ chuỗi vào DX
INT 21H ; xuất chuỗi
;đọc và xử lý 1 dòng văn bản
MOV AH,1 ; đọc ký tự
INT 21H ; ký tự trên AL
WHILE :
;trong khi ký tự gõ vào không phải là CR thì thực hiện
CMP AL,0DH ; CR?
JE END_WHILE ;yes, thoát
; nếu ký tự là hoa
CMP AL,'A' ; char >='A'?
JNGE END_IF ;không phải ký tự hoa thì nhảy đến END_IF
CMP AL,'Z' ; char <= 'Z'?
JNLE END_IF ; không phải ký tự hoa thì nhảy đến END_IF
```

```
; thì
; nếu ký tự nằm trước biến FIRST
CMP AL,FISRT ; char < FIRST ?
JNL CHECK_LAST; >=
; thì ký tự viết hoa đầu tiên = ký tự
MOV FIRST,AL ; FIRST=character
;end_if
CHECK_LAST:
; nếu ký tự là sau biến LAST
CMP AL,LAST ; char > LAST ?
JNG END_IF ; <=
;thì ký tự cuối cùng = ký tự
MOV LAST, AL ;LAST = character
;end_if
END_IF :
; đọc một ký tự
INT 21H ; ký tự trên AL
JMP WHILE_ ; lặp
END_WHILE:
;in kết quả
MOV AH,9 ; hàm xuất ký tự
; IF không có chữ hoa nào được nhập thì FIRST ='['
CMP FIRST,'[' ; FIRST='[' ?
JNE CAPS ; không, in kết qủa
;Then
LEA DX,NOCAP_MSG
INT 21H
CAPS:
LEA DX,CAP1_MSG
INT 21H
LEA DX,CAP2_MSG
INT 21H
; end_if
; dos exit
MOV AH,4CH
INT 21h
MAIN ENDP
END MAIN
```

Chương 4 - CÁC LỆNH LOGIC, DỊCH VÀ QUAY

Trong chương này chúng ta sẽ xem xét các lệnh mà chúng có thể dùng để thay đổi từng bit trên một byte hoặc một từ số liệu. Khả năng quản lý đến từng bit thường là không có trong các ngôn ngữ cấp cao (trừ C) và đây là lý do giải thích tại sao hợp ngữ vẫn đóng vai trò quan trọng trong khi lập trình .

4.1 Các lệnh logic

Chúng ta có thể dùng các lệnh logic để thay đổi từng bit trên byte hoặc trên một từ số liệu . Khi một phép toán logic được áp dụng cho toán hạng 8 hoặc 16 bit thì có thể áp dụng phép toán logic đó trên từng bit để thu được kết qủa cuối cùng . Ví dụ : Thực hiện các phép toán sau :

1. 10101010 AND 1111 0000
2. 10101010 OR 1111 0000
3. 10101010 XOR 1111 0000
4. NOT 10101010

Giải :

1. 10101010
   AND 1111 0000
   = 1010 0000
2. 10101010
   OR 1111 0000
   = 1111 1010
3. 1010 1010
   XOR 1111 0000
   0101 1010
4. NOT 10101010
   = 01010101

4.1.1 Lệnh AND,OR và XOR

Lệnh AND,OR và XOR thực hiện các chức năng đúng như tên gọi của nó .
Cú pháp của chúng là :

AND destination, source
OR destination, source
XOR destination, source

Kết qủa của lệnh được lưu trữ trong toán hạng đích do đó chúng phải là thanh ghi hoặc vị trí nhớ. Toán hạng nguồn là có thể là hằng số, thanh ghi hoặc vị trí nhớ .
Dĩ nhiên hai toán hạng đều là vị trí nhớ là không được phép .

Anh hưởng đến các cờ :

Các cờ SF,ZF và PF phản ánh kết qủa AF không xác định CF=OF=0
Để thay đổi từng bit theo ý muốn chúng ta xây dựng toán hạng nguồn theo kiểu mặt nạ (mask). Để xây dựng mặt nạ chúng ta sử dụng các tính chất sau đây của các phép toán AND ,OR và XOR :

b AND 1 = b	b OR 0 = b	b XOR 0 = b
b AND 0 = 0	b OR 1 = 1	b XOR 1 = not b

- Lệnh AND có thể dùng để xóa (clear) toán hạng đích nếu mặt nạ bằng 0
- Lệnh OR có thể dùng để đặt (set) 1 cho toán hạng đích nếu mặt nạ bằng 1

- Lệnh XOR có thể dùng để lấy đảo toán hạng đích nếu mặt nạ bằng 1. Lệnh XOR cũng có thể dùng để xóa nội dung một thanh ghi (XOR với chính nó)

Ví dụ : Xoá bit dấu của AL trong khi các bit khác không thay đổi

Giải : Dùng lệnh AND với mặt nạ 0111111=7Fh

```
AND AL,7Fh ; xóa bit dấu (dấu + ) của AL
```

Ví dụ : Set 1 cho các bit MSB và LSB của AL, các bit khác không thay đổi .

Giải : Dùng lệnh OR với mặt nạ 10000001 =81h

```
OR AL,81h ; set 1 cho LSB và MSB của AL
```

Ví dụ : Thay đổi bit dấu của DX

```
Giải : Dùng lệnh XOR với mặt nạ 1000000000000000=8000h
XOR DX,8000h
```

Các lệnh logic là đặc biệt có ích khi thực hiện các nhiệm vụ sau :

Đổi một số dưới dạng ASCII thành một số

Giả sử rằng chúng ta đọc một ký tự từ bàn phím bằng hàm 1 ngắt 21h. Khi đó AL chứa mã ASCII của ký tự. Điều này cũng đúng nếu ký tự đó là một số (digital character). Ví dụ nếu chúng ta gỏ số 5 thì AL = 35h (ASCII code for 5)

Để chứa 5 trên AL chúng ta dùng lệnh :

```
SUB AL,30h
```

Có một cách khác để làm việc này là dùng lệnh AND để xóa nửa byte cao (high nibble = 4 bit cao) của AL :

```
AND AL,0Fh
```

Vì các số từ 0-9 có mã ASCII từ 30h-39h, nên cách này dùng để đổi mọi số ASCII ra thập phân . Chương trình hợp ngữ đổi một số thập phân thành mã ASCII của chúng được xem như bài tập .

Đổi chữ thường thành chữ hoa

Mã ASCII của các ký tự thường từ a-z là 61h-7Ah và mã ASCII của các ký tự hoa từ A-Z là 41h -5Ah. Giả sử DL chưá ký tự thường, để đổi nó thành chữ hoa ta dùng lệnh :

```
SUB DL,20h
```

Nếu chúng ta so sánh mã nhị phân tương ứng của ký tự thường và ký tự hoa thì thấy rằng chỉ cần xóa bit thứ 5 thì sẽ đổi ký tự thường sang ký tự hoa .

Character	**Code**	**Character**	**Code**
a(61h)	01100001	A (41h)	01000001
b (62h)	01100010	B (42h)	01000010
.			
.			
z (7Ah)	01111010	Z (5Ah)	01011010

Có thể xóa bit thứ 5 của DL bằng cách dùng lệnh AND với mặt nạ 11011111= DF h

AND DL,0DFh ; đổi ký tự thường trong DL sang ký tự hoa

Xóa một thanh ghi

Chúng ta có thể dùng lệnh sau để xóa thanh ghi AX :

```
MOV AX,0
```

hoặc

```
SUB AX,AX
```

```
XOR AX,AX
```

Lệnh thứ nhất cần 3 bytes trong khi đó 2 lệnh sau chỉ cần 2 bytes. Nhưng lệnh MOV phải được dùng để xoá 1 vị trí nhớ .

Kiểm tra một thanh ghi có bằng 0 ?

Thay cho lệnh

```
CMP AX,0
```

Người ta dùng lệnh

```
OR CX,CX
```

để kiểm tra xem CX có bằng 0 hay không vì nó làm thay đổi cờ ZF (ZF=0 nếu CX=0)

4.1.2 Lệnh NOT

Lệnh NOT dùng để lấy bù 1 (đảo) toán hạng đích. Cú pháp là :

NOT destination

Không có cờ nào bị ảnh hưởng bởi lệnh NOT

Ví dụ : Lấy bù 1 AX

```
NOT AX
```

4.1.3 Lệnh TEST

Lệnh TEST thực hiện phép AND giữa toán hạng đích và toán hạng nguồn nhưng không làm thay đổi toán hạng đích. Mục đích của lệnh TEST là để set các cờ trạng thái. Cú pháp của lệnh test là :

TEST destination,source

Các cờ bị ảnh hưởng của lệnh TEST :

SF,ZF và PF phản ánh kết qủa
AF không xác định
CF=OF=0

Lệnh TEST có thể dùng để khám 1 bit trên toán hạng. Mặt nạ phải chứa bit 1 tại vị trí cần khám, các bit khác thì bằng 0. Kết quả của lệnh :

TEST destination,mask

sẽ là 1 tại bit cần test nếu như toán hạng đích chứa 1 tại bit test. Nếu toán hạng đích chứa 0 tại bit test thì kết quả sẽ bằng 0 và do đó ZF=1 .

Ví dụ : Nhảy tới nhãn BELOW nếu AL là một số chẵn

Giải : Số chẵn có bit thứ 0 bằng 0, lệnh

```
TEST AL,1 ; AL chẵn ?
JZ BELOW ; đúng, nhảy đến BELOW
```

4.2 Lệnh SHIFT

Lệnh dịch và quay sẽ dịch các bit trên trên toán hạng đích một hoặc nhiều vị trí sang trái hoặc sang phải. Khác nhau của lệnh dịch và lệnh quay là ở chỗ : các bit bị dịch ra (trong lệnh dịch) sẽ bị mất .Trong khi đó đối với lệnh quay, các bit bị dịch ra từ một đầu của toán hạng sẽ được đưa trở lại đầu kia của nó .

Có 2 khả năng viết đối với lệnh dịch và quay :

OPCODE destination,1
OPCODE destination,CL

trong cách viết thứ hai thanh ghi CL chứa N là số lần dịch hay quay. Toán hạng đích có thể là một thanh ghi 8 hoặc 16 bit, hoặc một vị trí nhớ .

Các lệnh dịch và quay thường dùng để nhân và chia các sô nhị phân. Chúng cũng được dùng cho các hoạt động nhập xuất nhị phân và hex .

4.2.1 Lệnh dịch trái (left shift)

Lệnh SHL dịch toán hạng đích sang trái .Cú pháp của lệnh như sau :

SHL destination ,1 ; dịch trái dest 1 bit

SHL destination, CL ; dịch trái N bit (CL chứa N)

Cứ mỗi lần dịch trái, một số 0 được thêm vào LSB .

Các cờ bị ảnh hưởng :

SF,PF,ZF phản ảnh kết qủa

AF không xác định

CF= bit cuối cùng được dịch ra

OF= 1 nếu kết qủa thay đổi dấu vào lần dịch cuối cùng

Ví dụ : Giả sử DH =8Ah và CL=3. Hỏi giá trị của DH và CF sau khi lệnh

```
SHL DH,CL được thực hiện ?
```

Kết qủa DH=01010000=50h, CF=0

Nhân bằng lệnh SHL

Chúng ta hãy xét số 235decimal. Nếu dịch trái 235 một bit và thêm 0 vào bên phải chúng ta sẽ có 2350. Noí cách khác, khi dịch trái 1 bit chúng ta đã nhân 10.

Đối với số nhị phân, dịch trái 1 bit có nghĩa là nhân nó với 2.Ví dụ AL=00000101=5d

```
SHL AL,1 ; AL=00001010=10d
SHL AL,CL ; nếu CL=2 thì AL=20d sau khi thực hiện lệnh
```

Lệnh dịch trái số học (SAL =Shift Arithmetic Left)

Lệnh SHL có thể dùng để nhân một toán hạng với hệ số 2. Tuy nhiên trong trường hợp người ta muốn nhấn mạnh đến tính chất số học của phép toán thì lệnh SAL sẽ được dùng thay cho SHL. Cả 2 lệnh đều tạo ra cùng một mã máy .

Một số âm cũng có thể được nhân 2 bằng cách dịch trái. Ví dụ : Nếu AX=FFFFh= -1 thì sau khi dịch trái 3 lần AX=FFF8h = -8

Tràn

Khi chúng ta dùng lệnh dịch trái để nhân thì có thể xảy ra sự tràn. Đối với lệnh dịch trái 1 lần, CF và OF phản ánh chính xác sự tràn dấu và tràn không dấu . Tuy nhiên các cờ sẽ không phản ánh chính xác kết qủa nếu dịch trái nhiều lần bởi vì dịch nhiều lần thực chất là một chuỗi các dịch 1 lần liên tiếp và vì vậy các cờ CF và OF chỉ phản ánh kết quả của lần dịch cuối cùng. Ví dụ : BL=80h, CL=2 thì lệnh

```
SHL BL,CL
```

sẽ làm cho CF=OF=0 mặc dù trên thực tế đã xảy ra cả tràn dấu và tràn không dấu .

Ví dụ : viết đoạn mã nhân AX với 8. Giả sử rằng không có tràn .

```
MOV CL,3 ; CL=3
SHL AX,CL ; AX*8
```

4.2.2 Lệnh dịch phải (Right Shift)

Lệnh SHR dịch phải toán hạng đích 1 hoặc N lần .

SHR destination,1

SHR destination,CL

Cứ mỗi lần dịch phải, một số 0 được thêm vào MSB

Các cờ bị ảnh hưởng giống như lệnh SHL
Ví dụ : giả sử DH = 8Ah, CL=2
Lệnh SHR DH,CL ; dịch phải DH 2 lần sẽ cho kết quả như sau :
Kết qủa trên DH=22h, CF=1
Cũng như lệnh SAL, lệnh SAR (dịch phải số học) hoạt động giống như SHR , *chỉ có 1 điều khác là MSB vẫn giữ giá trị nguyên thủy* (bit dấu giữ nguyên) sau khi dịch .

Chia bằng lệnh dịch phải

Lệnh dịch phải sẽ chia 2 giá trị của toán hạng đích. Điều này đúng đối với số chẳn. Đối với số lẻ, lệnh dịch phải sẽ chia 2 và làm tròn xuống số nguyên gần nó nhất. Ví dụ, nếu BL = 00000101=5 thì khi dịch phải BL=00000010 =2 .

Chia có dấu và không dấu

Để thực hiện phép chia bằng lệnh dịch phải, chúng ta phải phân biệt giữa số có dấu và số không dấu. Nếu diễn dịch là không dấu thì dùng lệnh SHR, còn nếu diễn dịch có dấu thì dùng SAR (bit dấu giữ nguyên) .
Ví dụ : dùng lệnh dịch phải để chia số không dấu 65143 cho 4. Thương số đặt trên AX .

```
MOV AX,65134
MOV CL,2
SHR AX,CL
```

Ví dụ : Nếu AL = -15, cho biết AL sau khi lệnh SAR AL,1 được thực hiện
Giải : AL= -15 = 11110001b
Sau khi thực hiện SAR AL ta có AL = 11111000b = -8

4.3 Lệnh quay (Rotate)

Quay trái (rotate left) = ROL sẽ quay các bit sang trái, LSB sẽ được thay bằng MSB. Còn CF=MSB
Cú pháp của ROL như sau :

ROL destination,1
ROL destination,CL

Quay phải (rotate right) = ROR sẽ quay các bit sang phải, MSB sẽ được thay bằng LSB. Còn CF=LSB
Cú pháp của lệnh quay phải là

ROR destination,1
ROR destination,CL

Trong các lệnh quay phải và quay trái CF chứa bit bị quay ra ngoài .
Ví dụ sau đây cho thấy cách để khám các bit trên một byte hoặc 1 từ mà không làm thay đổi nội dung của nó .
Ví dụ : Dùng ROL để đếm số bit 1 trên BX mà không thay đổi nội dung của nó. Kết qủa cất trên AX .
Giải :

```
XOR AX,AX ; xoá AX
MOV CX,16 ; số lần lặp = 16 (một từ )
TOP:
ROL BX,1 ; CF = bit quay ra
JNC NEXT ; nếu CF =0 thì nhảy đến vòng lặp
```

```
INC AX ; ngược lại (CF=1), tăng AX
NEXT:
LOOP TOP
```

Quay trái qua cờ nhớ (rotate through carry left) = RCL. Lệnh này giống như lệnh ROL chỉ khác là cờ nhớ nằm giữa MSB và LSB trong vòng kín của các bit

Cú pháp của của lệnh RCL như sau :

RCL destination,1

RCL destination,CL

Quay phải qua cờ nhớ (rotate through carry right) = RCR. Lệnh này giống như lệnh ROR chỉ khác là cờ nhớ nằm giữa MSB và LSB trong vòng kín của các bit .

Cú pháp của của lệnh RCR như sau :

RCR destination,1

Anh hưởng của lệnh quay lên các cờ

SF,PF và ZF phản ảnh kết quả

CF-bit cuối cùng được dịch ra

OF=1 nếu kết qủa thay đổi dấu vào lần quay cuối cùng

Ưng dụng : Đảo ngược các bit trên một byte hoặc 1 từ .Ví dụ AL =10101111 thì sau khi đảo ngược AL=11110101 .

Có thể lặp 8 lần công việc sau :**Dùng SHL để dịch bit MSB ra CF, Sau đó dùng RCR để đưa nó vào BL .**

Đoạn mã để làm việc này như sau :

```
MOV CX,8 ;số lần lặp
REVERSE :
SHL AL,1 ; dịch MSB ra CF
RCR BL,1 ; đưa CF (MSB) vào BL
LOOP REVERSE
MOV AL,BL ; AL chứa các bit đã đảo ngược
```

4.4 Xuất nhập số nhị phân và số hex

Các lệnh dịch và quay thường được sử dụng trong các hoạt động xuất nhập số nhị phân và số hex.

4.4.1 Nhập số nhị phân

Giả sử cần nhập một số nhị phân từ bàn phím, kết thúc là phím CR. Số nhị phân là một chuỗi các bit 0 và 1. Mỗi một ký tự gõ vào phải được biến đổi thành một bit giá trị (0 hoặc 1) rồi tích luỹ chúng trong 1 thanh ghi. Thuật toán sau đây sẽ đọc một số nhị phân từ bàn phím và cất nó trên thanh ghi BX .

```
Clear BX
input a character (‘0’ or ‘1’)
WHILE character<> CR DO
convert character to binary value
left shift BX
insert value into LSB of BX
input a character
```

```
END_WHILE
```

Đoạn mã thực hiện thuật toán trên như sau :

```
XOR BX,BX ; Xoá BX
MOV AH,1 ; hàm đọc 1 ký tự
INT 21h ; ký tự trên AL
WHILE_:
CMP AL,0DH ; ký tự là CR?
JE END_WHILE ; đúng, kết thúc
AND AL,0Fh ; convert to binary value
SHL BX,1 ; dịch trái BX 1 bit
OR BL,AL ; đặt giá trị vào BX
INT 21h ; đọc ký tự tiếp theo
JMP WHILE_ ; lặp
END_WHILE:
```

4.4.2 Xuất số nhị phân

Giả sử cần xuất số nhị phân trên BX (16 bit). Thuật toán có thể viết như sau

```
FOR 16 times DO
rotate left BX (put MSB into CF)
IF CF=1
Chương 4 : Các lệnh dịch và quay 55
then
output '1'
else
output '0'
END_IF
END_FOR
```

Đoạn mã để xuất số nhị phân có thể xem như bài tập .

4.4.3 Nhập số HEX

Nhập số hex bao gồm các số từ 0 đến 9 và các ký tự A đến F. Kết qủa chứa trong BX .
Để cho đơn giản chúng ta giả sử rằng :

- chỉ có ký tự hoa được dùng
- người dùng nhập vào không qúa 4 ký tự hex

Thuật toán như sau :

```
Clear BX
input character
WHILE character<> CR DO
convert character to binary value(4 bit)
left shift BX 4 times
insert value into lower 4 bits of BX
input character
END_WHILE
```

Đoạn mã có thể viết như sau :

```
XOR BX,BX ; clear BX
MOV CL,4 ; counter for 4 shift
MOV AH,1 ; input character
; function
INT 21h ; input a chracter AL
WHILE_:
CMP AL,0Dh ; character <>CR?
JE END_WHILE_ ; yes, exit
; convert character to binary value
CMP AL,39H ; a character?
JG LETTER ; no, a letter
; input is a digit
AND AL,0Fh ; convert digit to binary
value
JMP SHIFT ; go to insert BX
LETTER:
SUB AL,37h ; convert letter to binary
value
SHIFT:
SHL BX,CL ; make room for new value
; insert value into BX
OR BL,AL ; put value into low 4 bits of
BX
INT 21H ; input a character
JMP WHILE_
END_WHILE:
```

4.4.4 Xuất số HEX

Để xuất số hex trên BX (16 bit = 4 digit hex) có thể bắt đầu từ 4 bit bên trái, chuyển chúng thành một số hex rồi xuất ra màn hình .

Thuật toán như sau :

```
FOR 4 times DO
move BH to DL
Shift DL 4 times to right
IF DL < 10
then
convert to character in ‘0’ ...’9’
else
convert to character in ‘A’..’F’
END_IF
output character (HAM 2 NGAT 21H)
```

```
rotate BX left 4 times
END_FOR
```

Phần code cho thuật toán này xem như bài tập .

Chương 5 - NGĂN XẾP VÀ THỦ TỤC

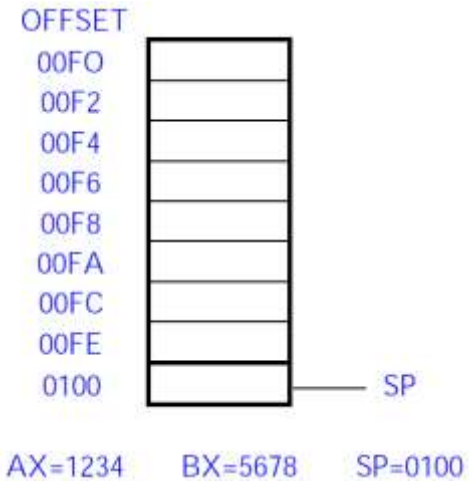
Đoạn ngăn xếp (stack segment) trong chương trình được dùng để cất giữ tạm thời số liệu và địa chỉ. Trong chương này chúng ta sẽ xem xét cách tổ chức stack và sử dụng nó để thực hiện các thủ tục (procedure) .

5.1 Ngăn xếp

Ngăn xếp là cấu trúc dữ liệu 1 chiều. Điều đó có nghĩa là số liệu được đưa vào và lấy ra khỏi stack tại đầu cuối của stack theo nguyên tắc LIFO (last in first out). Vị trí tại đó số liệu được đưa vào hay lấy ra gọi là đỉnh của ngăn xếp (top of stack) .Có thể hình dung satck như một chồng đĩa. Đĩa đưa vào sau cùng nằm tại đỉnh của chồng đĩa. Khi lấy ra, đĩa trên cùng sẽ được lấy ra trước. Một chương trình phải dành ra một khối nhớ cho ngăn xếp. Chúng ta dùng chỉ dẫn

```
.STACK 100h
```

để khai báo kích thước vùng stack là 256 bytes .
Khi chương trình được dịch và nạp vào bộ nhớ thanh ghi SS (stack segment) sẽ chứa địa chỉ đoạn stack. Còn SP (stack pointer) chứa địa chỉ đỉnh của ngăn xếp .
Trong khai báo stack 100h trên đây, SP nhận giá trị 100h. Điều này có nghĩa là stack trống rỗng (empty) như hình 4-1.



Hình 4.1 : STACK EMPTY

Lệnh PUSH và PUSHF

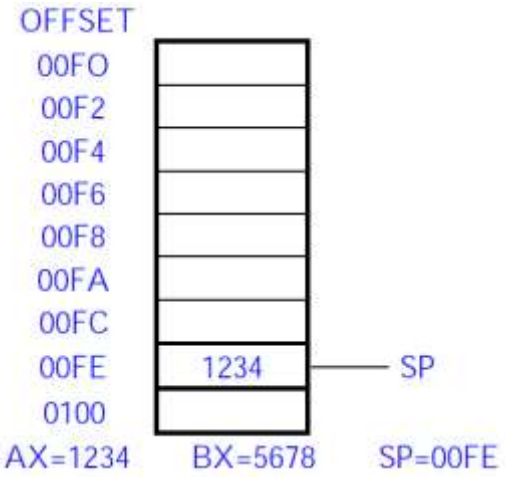
Để thêm một từ mới vào stack chúng ta dùng lệnh :

```
PUSH source ; đưa một thanh ghi hoặc từ nhớ 16 bit vào stack
```

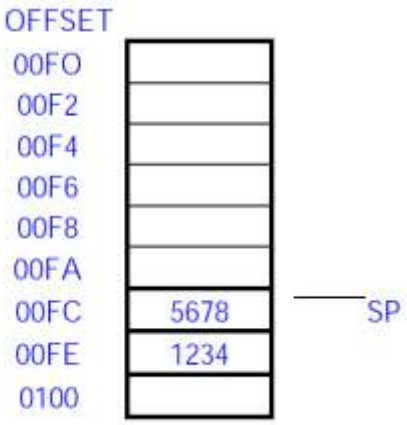
Ví dụ PUSH AX. Khi lệnh này được thực hiện thì :

- SP giảm đi 2
- một bản copy của toán hạng nguồn được chuyển đến địa chỉ SS:SP còn toán hạng nguồn không thay đổi .

Lệnh PUSHF không có toán hạng .Nó dùng để đẩy nội dung thanh ghi cờ vào stack . Sau khi thực hiện lệnh PUSH thì SP sẽ giảm 2. Hình 5-2 và 5-3 cho thấy lệnh PUSH làm thay đổi trạng thái stack như thế nào .



Hình 5-2 : STACK sau khi thực hiện lệnh PUSH AX



Hình 5-3 : STACK sau khi thực hiện lệnh PUSH BX

Lệnh POP và POPF

Để lấy số liệu tại đỉnh stack ra khỏi stack ,chúng ta dùng lệnh :

```
POP destination ; lấy số liệu tại đỉnh stack ra destination
```

Destination có thể là 1 thanh ghi hoặc từ nhớ 16 bit. Ví dụ :

```
POP BX ; Lấy số liệu trong stack ra thanh ghi BX .
```

Khi thực hiện lệnh POP :

- nội dung của đỉnh stack (địa chỉ SS:SP) được di chuyển đến đích .
- SP tăng 2

Lệnh POPF sẽ lấy đỉnh stack đưa vào thanh ghi cờ .

Các lệnh PUSH,PUSHF,POP,POPF không ảnh hưởng đến các cờ .

Lưu ý : Lệnh PUSH, POP là lệnh 2 bytes vì vậy các lệnh 1 byte như :

```
PUSH DL ; lệnh không hợp lệ
PUSH 2 ; lệnh không hợp lệ
```

Ngoài chức năng lưu trữ số liệu và địa chỉ của chương trình do người sử dụng viết, stack còn được dùng bởi hệ điều hành để lưu trữ trạng thái của chương trình chính khi có ngắt .

5.2 Ưng dụng của stack

Bởi vì nguyên tắc làm việc của stack là LIFO nên các đối tượng được lấy ra khỏi stack có trật tự ngược lại với trật tự mà chúng được đưa vào stack. Chương trình sau đây sẽ đọc một chuỗi ký tự rồi in chúng trên dòng mới với trật tự ngược lại .

Thuật toán cho chương trình như sau :

```
Display a '? '
Initialize count to 0
Read a character
WHILE character is not CR DO
PUSH chracter onto stack
Incremet count
Read a character
END_WHILE ;
Goto a new line
FOR count times DO
POP a chracter from the stack
Display it ;
END_FOR
```

Sau đây là chương trình :

```
TITLE PGM5-1 : REVERSE INPUT
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
; in dấu nhắc
MOV AH,2
MOV DL,'?'
INT 21H
; xoá biến đếm CX
XOR CX,CX
;đọc 1 ký tự
MOV AH,1
INT 21H
;Trong khi character không phải là CR
WHILE_:
CMP AL,0DH
JE END_WHILE
;cất AL vào stack tăng biến đếm
PUSH AX ; đẩy AX vào stack
INC CX ; tăng CX
; đọc 1 ký tự
INT 21h
```

```
JMP WHILE_
END_WHILE:
; Xuống dòng mới
MOV AH,2
MOV DL,0DH
INT 21H
MOV DL,0AH
INT 21H
JCXZ EXIT ; thoát nếu CX=0 (không có ký tự nào được nhập)
; lặp CX lần
TOP:
;lấy ký tự từ stack
POP DX
;xuất nó
INT 21H
LOOP TOP ; lặp nếu CX>0
; end_for
EXIT:
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN
```

Giải thích thêm về chương trình : vì số ký tự nhập là không biết vì vậy dung thanh ghi CX để đếm số ký tự nhập. CX cũng dùng cho vòng FOR để xuất các ký tự theo thứ tự ngược lại. Mặc dù ký tự chỉ giữ trên AL nhưng phải đẩy cả thanh ghi AX vào stack. Khi xuất ký tự chúng ta dùng lệnh POP DX để lấy nội dung trên stack ra.
Mã ASCII của ký tự ở trên DL, sau đó gọi INT 21h để xuất ký tự .

5.3 Thủ tục (Procedure)

Trong chương 3 chúng ta đã đề cập đến ý tưởng lập trình top-down. Ý tưởng này có nghĩa là một bài toán nguyên thuỷ được chia thành các bài toán con mà chúng dễ giải quyết hơn bài toán nguyên thuỷ. Trong các ngôn ngữ cấp cao người ta dùng thủ tục để giải các bài toán con, và chúng ta cũng làm như vậy trong hợp ngữ. Như vậy là một chương trình hợp ngữ có thể được xây dựng bằng các thủ tục .
Một thủ tục gọi là thủ tục chính sẽ chứa nội dung chủ yếu của chương trình .
Để thực hiện một công việc nào đó, thủ tục chính gọi (CALL) một thủ tục con . Thủ tục con cũng có thể gọi một thủ tục con khác . Khi một thủ tục gọi một thủ tục khác, điều khiển được chuyển tới (control transfer) thủ tục được gọi và các lệnh của thủ tục được gọi sẽ được thi hành. Sau khi thi hành hết các lệnh trong nó, thủ tục được gọi sẽ trả điều khiển (return control) cho thủ tục gọi nó. Trong ngôn ngữ cấp cao, lập trình viên không biết và không thể biết cơ cấu của việc chuyển và trả điều khiển giữa thủ tục chính và thủ tục con. Nhưng trong hợp ngữ có thể thấy rỏ cơ cấu này (xem phần 5.4) .
Cú pháp của lệnh tạo một thủ tục như sau :

```
name PROC type
```

```
; body of procedure
RET
name ENDP
```

Name do người dùng định nghĩa là tên của thủ tục .
Type có thể là NEAR (có thể không khai báo) hoặc FAR .
NEAR có nghĩa là thủ tục được gọi nằm cùng một đoạn với thủ tục gọi. FAR có nghĩa là thủ tục được gọi và thủ tục gọi nằm khác đoạn. Trong phần này chúng ta sẽ chỉ mô tả thủ tục NEAR .
Lệnh RET trả điều khiển cho thủ tục gọi. Tất cả các thủ tục phải kết thúc bởi RET trừ thủ tục chính .
Chú thích cho thủ tục : Để người đọc dễ hiểu thủ tục người ta thường sử dụng chú thích cho thủ tục dưới dạng sau :
; (mô tả các công việc mà thủ tục thi hành)
; input: (mô tả các tham số có tham gia trong chương trình)
; output : (cho biết kết qủa sau khi chạy thủ tục)
; uses : (liệt kê danh sách các thủ tục mà nó gọi)

5.4 CALL & RETURN

Lệnh CALL được dùng để gọi một thủ tục. Có 2 cách gọi một thủ tục là gọi trực tiếp và gọi gián tiếp .

CALL name ; gọi trực tiếp thủ tục có tên là name
CALL address-expression ; gọi gián tiếp thủ tục

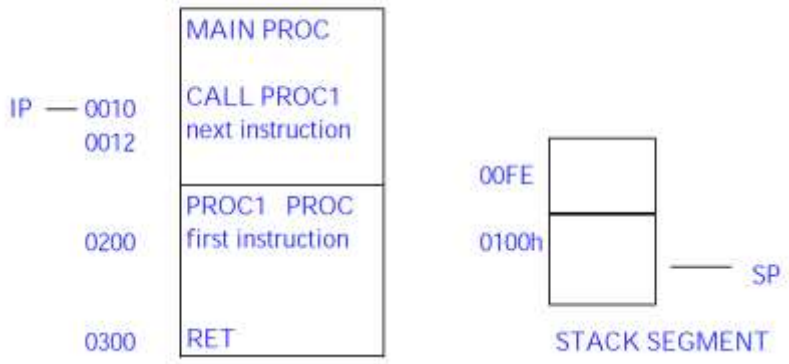
trong đó address-expression chỉ định một thanh ghi hoặc một vị trí nhớ mà nó chứa địa chỉ của thủ tục .
Khi lệnh CALL được thi hành thì :

- Địa chỉ quay về của thủ tục gọi được cất vào stack. Địa chỉ này chính là offset của lệnh tiếp theo sau lệnh CALL .
- IP lấy địa chỉ offset của lệnh đầu tiên trên thủ tục được gọi, có nghĩa là điều khiển được chuyển đến thủ tục .

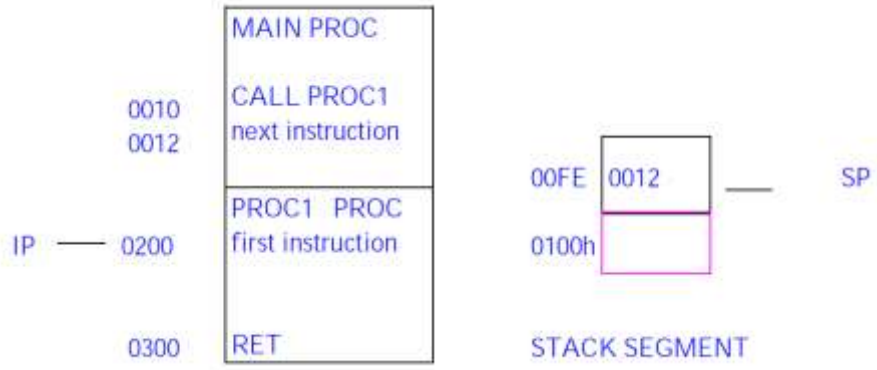
Để trả điều khiển cho thủ tục chính, lệnh

```
RET pop-value
MAIN PROC
CALL PROC1
next instruction
PROC1 PROC
first instruction
RET
```

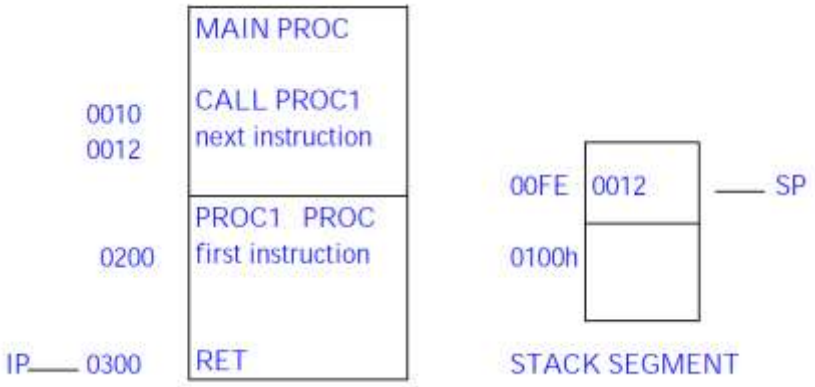
được sử dụng. Pop-value (một số nguyên N) là tùy chọn. Đối với thủ tục NEAR , lệnh RET sẽ lấy giá trị trong SP đưa vào IP. Nếu pop-value là ra một số N thì IP=SP+N Trong cả 2 trường hợp thì CS:IP chứa địa chỉ trở về chương trình gọi và điều
khiển được trả cho chương trình gọi (xem hình 5-2)



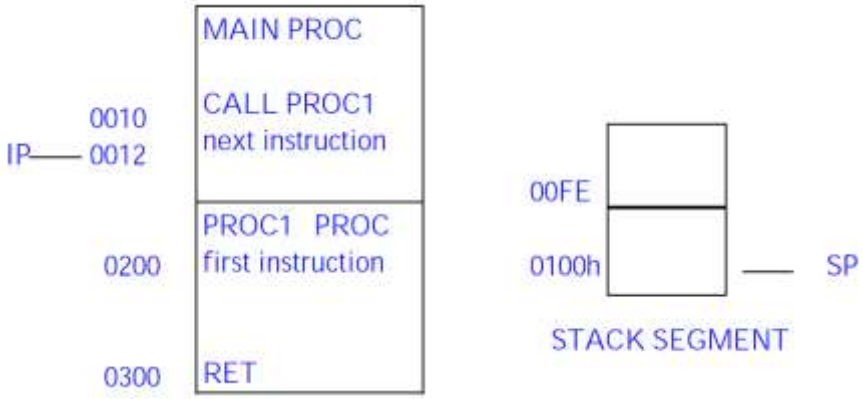
Hình 5-2 a : Trước khi CALL



Hình 5-2 b : Sau khi CALL



Hình 5-2 c : Trước khi RET



Hình 5-2 d : Sau khi RET

5.5 Ví dụ về thủ tục

Chúng ta sẽ viết chương trình tính tích của 2 số dương A và B bằng thuật toán cộng (ADD) và dịch (SHIFT)

Thuật toán như sau :

```
MAIN PROC
CALL PROC1
next instruction
PROC1 PROC
first instruction
RET
MAIN PROC
CALL PROC1
next instruction
PROC1 PROC
first instruction
RET
Product = 0
REPEAT
IF lsb of B is 1
THEN
product=product+A
END_IF
shift left A
shift right B
UNTIL B=0
```

Trong chương trình sau đây chúng ta sẽ mã hoá thủ tục nhân với tên là MULTIPLY. Chương trình chính không có nhập xuất, thay vào đó chúng ta dùng kĩ thuật DEBUG để nhập xuất .

```
TITLE PGM5-1: MULTIPLICATION BY ADD AND SHIFT
.MODEL SMALL
.STACK 100H
```

```
.CODE
MAIN PROC
; thực hiện bằng DEBUG. Đặt A = AX, B=BX
CALL MULTIPLY
;DX chứa kết qủa
MOV AH,4CH
INT 21H
MAIN ENDP
MULTIPY PROC
; input : AX=A, BX=B, AX và BX có giá trị trong khoảng 0...FFH
; output : DX= kết qủa
PUSH AX
PUSH BX
XOR DX,DX
REPEAT:
; Nếu lsb của B =1
TEST BX,1 ;lsb=1?
JZ END_IF ; không, nhảy đến END_IF
; thì
ADD DX,AX ; DX=DX+AX
END_IF :
SHL AX,1 ; dịch trái AX 1 bit
SHR BX,1 ;dịch phải BX 1 bit
; cho đến khi BX=0
JNZ REPEAT ; nếu BX chưa bằng 0 thì lặp
POP BX ; lấy lại BX
POP AX ; lấy lại AX
RET ; trả điều khiển cho chương trình chính
MULTIPLY ENDP
END MAIN
```

Trong quá trình chạy DEBUG có thể kiểm tra nội dung các thanh ghi. Lưu ý đặc biệt đến IP để xem cách chuyển và trả điều khiển khi gọi và thực hiện một thủ tục .

Chương 6 - LỆNH NHÂN VÀ CHIA

Trong chương 5 chúng ta đã nói đến các lệnh dịch mà chúng có thể dùng để nhân và chia với hệ số 2. Trong chương này chúng ta sẽ nói đến các lệnh nhân và chia một số bất kỳ . Quá trình xử lý của lệnh nhân và chia đối với số có dấu và số không dấu là khác nhau do đó có lệnh nhân có dấu và lệnh nhân không dấu . Một trong những ứng dụng thường dùng nhất của lệnh nhân và chia là thực hiện các thao tác nhập xuất thập phân. Trong chương này chúng ta sẽ viết thủ tục cho nhập xuất thập phân mà chúng được sử dụng nhiều trong các hoạt động xuất nhập từ ngoại vi .

6.1 Lệnh MUL và IMUL

Nhân có dấu và nhân không dấu

Trong phép nhân nhị phân số có dấu và số không dấu phải được phân biệt một cách rõ ràng. Ví dụ chúng ta muốn nhân hai số 8 bit 1000000 và 1111111 . Trong diễn dịch không dấu, chúng là 128 và 255. Tích số của chúng là 32640 = 0111111110000000b. Trong diễn dịch có dấu, chúng là -128 và -1. Do đó tích của chúng là 128 = 0000000010000000b . Vì nhân có dấu và không dấu dẫn đến các kết qủa khác nhau nên có 2 lệnh nhân :

MUL (multiply) nhân không dấu

IMUL (integer multiply) nhân có dấu

Các lệnh này nhân 2 toán hạng byte hoặc từ. Nếu 2 toán hạng byte được nhân với nhau thì kết qủa là một từ 16 bit .Nếu 2 toán hạng từ được nhân với nhau thì kết qủa là một double từ 32 bit. Cú pháp của chúng là :

MUL source ;

IMUL source ;

Toán hạng nguồn là thanh ghi hoặc vị trí nhớ nhưng không được là một hằng

Phép nhân kiểu byte

Đối với phép nhân mà toán hạng là kiểu byte thì

AX=AL*SOURCE ;

Phép nhân kiểu từ

Đối với phép nhân mà toán hạng là kiểu từ thì

DX:AX=AX*SOURCE

Ảnh hưởng của các lệnh nhân lên các cờ .

SF,ZF ,AF,PF : không xác định

sau lệnh MUL CF/OF = 0 nếu nửa trên của kết qủa(DX) bằng 0

=1 trong các trường hợp khác

sau lệnh IMUL CF/OF = 0 nếu nửa trên của kết qủa có bit dấu giống như bit dấu của nửa thấp .

= 1 trong các trường hợp khác

Sau đây chúng ta sẽ lấy vài ví dụ .

Ví dụ 1 : Giả sử rằng AX=1 và BX=FFFFh

INSTRUCTION	Dec product	Hex Product	DX	AX	CF/OF
MUL BX	65535	0000FFFF	0000	FFFF	0
IMUL BX	-1	FFFFFFFF	FFFF	FFFF	0

Ví dụ 2 : Giả sử rằng AX=FFFFh và BX=FFFFh

INSTRUCTION	Dec product	Hex Product	DX	AX	CF/OF

MUL BX	4294836225	FFFE0001	FFFE	0001	1
IMUL BX	1	00000001	00000	0001	0

Ví dụ 3 : Giả sử rằng AX=0FFFh

INSTRUCTION	**Dec product**	**Hex Product**	**DX**	**AX**	**CF/OF**
MUL BX	16769025	00FFE001	00FF	E001	1
IMUL BX	16769025	00FFE001	00FF	E001	1

Ví dụ 4 : Giả sử rằng AX=0100h và CX=FFFFh

INSTRUCTION	**Dec product**	**Hex Product**	**DX**	**AX**	**CF/OF**
MUL BX	16776960	00FFFF00	00FF	FF00	1
IMUL BX	-256	FFFFFF00	FFFF	FF00	0

Ví dụ 5 : Giả sử rằng AL=80h và BL=FFh

INSTRUCTION	**Dec product**	**Hex Product**	**DX**	**AX**	**CF/OF**
MUL BL	128	7F80	7F	80	1
IMUL BL	128	00080	00	80	1

6.2 Ứng dụng đơn giản của lệnh MUL và IMUL

Sau đây chúng ta sẽ lấy một số ví dụ minh họa việc sử dụng lệnh MUL và IMUL trong chương trình .

Ví dụ 1 : Chuyển đoạn chương trình sau trong ngôn ngữ cấp cao thành mã hợp ngữ : A = 5xA - 2xB. Giả sử rằng A và B là 2 biến từ và không xảy ra sự tràn .

Code :

```
MOV AX,5 ; AX=5
IMUL A ; AX=5xA
MOV A,AX ; A=5xA
MOV AX,12 ; AX=12
IMUL B ; AX=12xB
SUB A,AX ; A=5xA-12xB
```

Ví Dụ 2 : viết thủ tục FACTORIAL để tính N! cho một số nguyên dương. Thủ tục phải chứa N trên CX và trả về N! trên AX. Giả sử không có tràn .

Giải : Định nghĩa của N! là

N! = 1 nếu N=1

= N x (N-1)x (N-2) x...x 1 nếu N>1

Thuật toán để tính N! như sau :

```
Product =1
Term = N
FOR N times DO
Product = product x term
term=term -1
ENDFOR
```

Code :

```
FACTORIAL PROC
; computes N!
```

```
; input : CX=N
; output : AX=N!
MOV AX,1 ; AX=1
MOV CX,N ; CX=N
TOP:
MUL CX ; Product = product x term
LOOP TOP ;
RET
FACTORIAL ENDP
```

6.3 Lệnh DIV và IDIV

Cũng như lệnh nhân, có 2 lệnh chia DIV và IDIV cho số không dấu và cho số có dấu. Cú pháp của chúng là :

DIV divisor
IDIV divisor

Toán hạng byte

Lệnh chia toán hạng byte sẽ chia số bị chia 16 bit (dividend) trên AX cho số chia (divisor) là 1 byte. Divisor phải là 1 thanh ghi 8 bit hoặc 1 byte nhớ . Thương số ở trên AL còn số dư trên AH .

Toán hạng từ

Lệnh chia toán hạng từ sẽ chia số bị chia 32 bit (dividend) trên DX:AX cho số chia (divisor) là 1 từ. Divisor phải là 1 thanh ghi 16 bit hoặc 1 từ nhớ. Thương số ở trên AX còn số dư trên DX .

Ảnh hưởng của các cờ :

Các cờ có trạng thái không xác định .

Divide Overflow

Khi thực hiện phép chia kết qủa cóthể không chứa hết trên AL hoặc AX nếu số chia bé hơn rất nhiều so với số bị chia. Trong trường hợp này trên màn hình sẽ xuất hiện thông báo : **" *Divide overflow*"**

Ví dụ 1 : Giả sử DX = 0000h, AX = 0005h và BX = 0002h

Instruction	**Dec Quotient**	**Dec Remainder**	**AX**	**DX**
DIV BX	2	1	0002	0001
IDIV BX	2	1	0002	0001

Ví dụ 2 : Giả sử DX = 0000h, AX = 0005h và BX = FFFEh

Instruction	**Dec Quotient**	**Dec Remainder**	**AX**	**DX**
DIV BX	0	5	0000	0005
IDIV BX	-2	1	FFFE	0001

Ví dụ 3 : Giả sử DX = FFFFh, AX = FFFBh và BX = 0002h

Instruction	**Dec Quotient**	**Dec Remainder**	**AX**	**DX**
DIV BX	-2	-1	FFFE	FFFF
IDIV BX	OVERFLOW			

Ví dụ 4 : Giả sử AX = 00FBh và BL = FFh

Instruction	**Dec Quotient**	**Dec Remainder**	**AX**	**DX**

DIV BL	0	251	FB	00
IDIV BL	OVERFLOW			

6.4 Mở rộng dấu của số bị chia

Phép chia với toán hạng từ

Trong phép chia với toán hạng từ, số bị chia phải đặt trên DX:AX ngay cả khi số bị chia có thể đặt trên AX. Trong trường hợp này, cần phải sửa soạn như sau

- Đối với lệnh DIV, DX phải bị xoá
- Đối với lệnh IDIV, DX phải được mở rộng dấu của AX. Lệnh CWD (Convert Word to Doubleword) sẽ thực hiện việc này .

Ví dụ : Chia -1250 cho 7

```
MOV AX,-1250 ; AX= -1250
CWD ; mở rộng dấu của AX vào DX
MOV BX,7 ; BX=7
IDIV BX ; chia DX:AX cho BX, kết qủa trên AX, số dư
; trên DX
```

Phép chia với toán hạng byte

Trong phép chia với toán hạng byte, số bị chia phải đặt trên AX ngay cả khi số bị chia có thể đặt trên AL. Trong trường hợp này, cần phải sửa soạn như sau

- Đối với lệnh DIV, AH phải bị xoá
- Đối với lệnh IDIV, AH phải được mở rộng dấu của AL. Lệnh CBW (Convert Byte to Doublebyte) sẽ thực hiện việc này .

Ví dụ : Chia một số có dấu trong biến byte XBYTE cho -7

```
MOV AL, XBYTE ; AL giữ số bị chia
CBW ; mở rộng dấu của AL vào AH
MOV BL,-7 ; BX= -7
IDIV BL ; chia AX cho BL, kết qủa trên AL, số dư
; trên AH
```

Không có cờ nào bị ảnh hưởng bởi lệnh CWD và CBW .

6.5 Thủ tục nhập xuất số thập phân

Mặc dù trong PC tất cả số liệu được biểu diễn dưới dạng binary. Nhưng việc biểu diễn dưới dạng thập phân sẽ thuận tiện hơn cho người dùng. Trong phần này chúng ta sẽ viết các thủ tục nhập xuất số thập phân . Khi nhập số liệu, nếu chúng ta gõ 21543 chẳng hạn thì thực chất là chúng ta gõ vào một chuỗi ký tự, bên trong PC, chúng được biến đổi thành các giá trị nhị phân tương đương của 21543. Ngược lại khi xuất số liệu, nội dung nhị phân của thanh ghi hoặc vị trí nhớ phải được biến đổi thành một chuỗi ký tự biểu diễn một số thập phân trước khi chúng được in ra.

Xuất số thập phân (Decimal Output)

Chúng ta sẽ viết một thủ tục OUTDEC để in nội dung của thanh ghi AX như là một số nguyên thập phân có dấu. Nếu AX>0 ,OUTDEC sẽ in nội dung của AX dưới dạng thập phân. Nếu AX<0, OUTDEC sẽ in dấu trừ (-), thay AX = -AX (đổi thànb số dương) rồi in số dương này sau dấu trừ (-). Như vậy là trong cả 2 trường hợp, OUTDEC sẽ in giá trị thập phân tương đương của một số dương. Sau đây là thuật toán :

Algorithm for Decimal Output

```
1. IF AX < 0 / AX hold output value /
2. THEN
3. PRINT a minus sign
4. Replace AX by its two's complement
5. END_IF
6. Get the digits in AX's decimal representation
7. Convert these digits to characters and print them .
```

Để hiểu chi tiết bước 6 cần phải làm việc gì, chúng ta giả sử rằng nội dung của AX là một số thập phân, ví dụ 24618 thập phân. Có thể lấy các digits thập phân của 24618 bằng cách chia lặp lại cho 10d theo thủ tục như sau :

Divide 24618 by 10. Qoutient = 2461, remainder = 8
Divide 2461 by 10. Qoutient = 246, remainder = 1
Divide 246 by 10. Qoutient = 24, remainder = 6
Divide 24 by 10. Qoutient = 2, remainder = 4
Divide 2 by 10. Qoutient = 0, remainder = 2

Các digits thu được bằng cách lấy các số dư theo trật tự ngược lại .
Bước 7 của thuật toán có thể thực hiện bằng vòng FOR như sau :

```
FOR count times DO
pop a digit from the stack
convert it to a character
output the character
END_FOR
Code cho thủ tục OUTDEC như sau :
OUTDEC PROC
; Print AX as a signed decimal integer
; input : AX
; output : none
PUSH AX ; save registers
PUSH BX
PUSH CX
PUSH DX
; IF AX<0
OR AX,AX ; AX < 0 ?
JGE @END_IF1 ; NO, AX>0
; THEN
PUSH AX ; save AX
MOV DL,'-' ; GET '-'
MOV AH,2
INT 21H ; print '-'
POP AX ; get AX back
NEG AX ; AX = -AX
```

```
@END_IF1:
; get decimal digits
XOR CX,CX ; clear CX for counts digit
MOV BX,10d ; BX has divisor
@REPEAT1:
XOR DX,DX ; clear DX
DIV BX ; AX:BX ; AX = qoutient, DX=
remainder
PUSH DX ; push remainder onto stack
INC CX ; increment count
;until
OR AX,AX ; qoutient = 0?
JNE @REPEAT1 ; no keep going
; convert digits to characters and print
MOV AH,2 ; print character function
; for count times do
@PRINT_LOOP:
POP DX ; digits in DL
OR DL,30h ; convert digit to character
INT 21H ; print digit
LOOP @PRINT_LOOP
;end_for
POP DX ; restore registers
POP CX
POP BX
POP AX
RET
OUTDEC ENDP
```

Toán tử giả INCLUDE

Chúng ta có thể thay đổi OUTDEC bằng cách đặt nó bên trong một chương trình ngắn và chạy chương trình trong DEBUG. Để đưa thủ tục OUTDEC vào trong chương trình mà không cần gõ nó, chúng ta dùng toán tử giả INCLUDE với cú pháp như sau :

INCLUDE filespec

ở đây filespec dùng để nhận dạng tập tin (bao gồm cả đường dẫn của nó) .

Ví dụ tập tin chứa OUTDEC là PGM6_1.ASM ở ổ A:. Chúng ta có thể viết :

INCLUDE A:\PGM6_1.ASM

Sau đây là chương trình để test thủ tục OUTDEC

```
TITLE PGM6_2 : DECIMAL OUTPUT
.MODEL SMALL
.STACK 100h
.CODE
MAIN PROC
```

```
CALL OUTDEC
MOV AH,4CH
INT 21H
MAIN ENDP
INCLUDE A:\PGM6_1.ASM
END MAIN
```

Sau khi dịch, chúng ta dùng DEBUG nhập số liệu và chạy chương trình .

Nhập Thập phân (Decimal input)

Để nhập số thập phân chúng ta cần biến đổi một chuỗi các digits ASCII thành biểu diễn nhị phân của một số nguyên thập phân. Chúng ta sẽ viết thủ tục INDEC để làm việc này .
Trong thủ tục OUTDEC chúng ta chia lặp cho 10d. Trong thủ tục INDEC chúng ta sẽ nhân lặp với 10d .

Decimal Input Algorithm

```
Total = 0
read an ASCII digit
REPEAT
convert character to a binary value
total = 10x total +value
read a chracter
UNTIL chracter is a carriage return
Ví dụ : nếu nhập 123 thì xử lý như sau :
total = 0
read '1'
convert '1' to 1
total = 10x 0 +1 =1
read '2'
convert '2' to 2
total = 10x1 +2 =12
read '3'
convert '3' to 3
total = 10x12 +3 =123
```

Sau đây chúng ta sẽ xây dựng thủ tục INDEC sao cho nó chấp nhận được các số thập phân có dấu trong vùng - 32768 đến +32767 (một từ). Chương trình sẽ in ra một dấu "?" để nhắc người dùng gõ vào dấu + hoặc -, theo sau đólà một chuỗi các digit và kết thúc là ký tự CR. Nếu người dùng gõ vào một ký tự không phải là 0 đến 9 thì thủ tục sẽ nhảy xuống dòng mới và bắt đầu lại từ đầu. Với những yêu cầu như trên đây thủ tục nhập thập phân phải viết lại như sau :

```
Print a question mask
Total = 0
negative = false
Read a character
CASE character OF
'-' : negative = true
```

```
read a chracter
'+';
read a charcter
END_CASE
REPEAT
IF character not between '0' and '9'
THEN
goto beginning
ELSE
convert character to a binary value
total = 10xtotal + value
IND_IF
read acharacter
UNTIL character is a carriage return
IF negative = true
then
total = - total
END_IF
```

Thủ tục có thể mã hoá như sau (ghi vào đĩa A : với tên là PGM6_2.ASM)

```
INDEC PROC
; read a number in range -32768 to +32767
; input : none
; output : AX = binary equvalent of number
PUSH BX ; Save regiter
PUSH CX
PUSH DX
; print prompt
@BEGIN:
MOV AH,2
MOV DL,'?'
INT 21h ; print '?'
; total = 0
XOR BX,BX ; CX holds total
; negative = false
XOR CX,CX ; cx holds sign
; read a character
MOV AH,1
INT 21h ; character in AL
; CASE character of
CMP AL,'-' ; minus sign
JE @MINUS
CMP AL,'+' ; Plus sign
```

```
JE @PLUS
JMP @REPEAT2 ; start processing characters
@MINUS:
MOV CX,1
@PLUS:
INT 21H
@REPEAT2:
; if character is between '0' to '9'
CMP AL,'0'
JNGE @NOT_DIGIT
CMP Al,'9'
JNLE @NOT_DIGIT
; THEN convert character to digit
AND AL,000FH ; convert to digit
PUSH AX ; save digit on stack
; total =10x total + digit
MOV AX,10
MUL BX ; AX= total x10
POP BX ; Retrieve digit
ADD BX,AX ; TOTAL = 10XTOTAL + DIGIT
;read a character
MOV AH,1
INT 21h
CMP AL,0DH
JNE @REPEAT
; until CR
MOV AX,BX ; restore total in AX
; if negative
OR CX,CX ; negative number
JE @EXIT ; no exit
;then
NEG AX
; end_if
@EXIT:
POP DX
POP CX
POP BX
RET
; HERE if illegal character entered
@NOT_DIGIT
MOV AH,2
MOV DL,0DH
```

```
INT 21h
MOV DL,0Ah
INT 21h
JMP @BEGIN
INDEC ENDP
TEST INDEC
```

Có thể test thủ tục INDEC bằng cách tạo ra một chương trình dùng INDEC cho nhập thập phân và OUTDEC cho xuất thập phân như sau :

```
TITLE PGM6_4.ASM
.MODEL SMALL
.STACK 100h
.CODE
MAIN PROC
; input a number
CALL INDEC
PUSH AX ; save number
; move cursor to a new line
MOV AH,2
MOV DL,0DH
INT 21h
MOV DL,0Ah
INT 21H
;output a number
POP AX
CALL OUTDEC
; dos exit
MOV AH,4CH
INT 21H
MAIN ENDP
INCLUDE A:\PGM6_1.ASM ; include outdec
INCLUDE A:\PGM6-2.ASM ; include indec
END MAIN
```

Chương 7 - MẢNG VÀ CÁC CHẾ ĐỘ ĐỊA CHỈ

Trong chương này chúng ta sẽ đề cập đến mảng một chiều và các kỹ thuật xử lý mảng trong Assembly. Phần còn lại củachương này sẽ trình bày các chế độ địa chỉ.

7.1 Mảng một chiều

Mảng một chiều là một danh sách các phần tử cùng loại và có trật tự. Có trật tự có nghĩa là có phần tử thứ nhất, phần tử thứ hai, phần tử thứ ba ... Trong toán học, nếu A là một mảng thì các phần tử của mảng được định nghĩa làA[1}, A[2], A[3} ... Hình vẽ là dưới đây là mảng A có 6 phần tử .

Index	
1	A[1]
2	A[2]
3	A[3]
4	A[4]
5	A[5]
6	A[6]

Trong chương 1 chúng ta đã dùng toán tử giả DB và DW để khai báo mảng byte và mảng từ. Ví dụ, một chuỗi 5 ký tự có tên là MSG

```
MSG DB 'abcde'
```

hoặc một mảng từ W gồm 6 số nguyên mà giá trị ban đâù của chúng là 10,20,30,40.50 và 60

```
W DW 10,20,30,40,50,60
```

Địa chỉ của biến mảng gọi là **địa chỉ cơ sở của mảng** (base address of the array). Trong mảng W thì địa chỉ cơ sở là 10 .Nếu địa chỉ offset của W là 0200h thì trong bộ nhớ mảng 6 phần tử nói trên sẽ như sau :

Offset address	**Symbolic address**	**Decimal content**
0200h	W	10
0202h	W + 2h	20
0204h	W + 4h	30
0206h	W + 6h	40
0208h	W + 8h	50
020Ah	W + Ah	60

Toán tử DUP (Duplicate)

Có thể định nghĩa một mảng mà các phần tử của nó có cùng một giá trị ban đầu bằng phép DUP như sau :

repeat_count DUP (value)

lặp lại một số (VALUE) n lần (n = repeat_count)

Ví dụ :

```
GAMMA DW 100 DUP (0) ; tạo một mảng 100 từ mà giá trị ban đâù là 0
DELTA DB 212 DUP (?) ; tạo một mảng 212 byte giá trị chưa xác định
```

DUP có thể lồng nhau, ví dụ :

```
LINE DB 5,4,3 DUP (2, 3 DUP (0) ,1)
```

tương đương với :

```
LINE DB 5,4,2,0,0,0,1,2,0,0,0,1,2,0,0,0,1
```

Vị trí các phần tử của một mảng

Địa chỉ của một phần tử của mảng có thể được xác định bằng cách cộng một hằng số với địa chỉ cơ sở. Giả sử A là một mảng và S chỉ ra số byte của một phần tử của mảng (S=1 đối với mảng byte và S=2 đối với mảng từ). Vị trí của các phần tử của mảng A có thể tính như sau :

Position	**Location**
1	A
2	A + 1xS
3	A + 2xS
.	..
.	..
N	A + (N-1)xS

Ví dụ : Trao đổi phần tử thứ 10 và thứ 25 của mảng từ W .
Phần tử thứ 10 là W[10] có địa chỉ là W+9x2=W+18
Phần tử thứ 25 là W[25] có địa chỉ là W+24x2=W+48
Vì vậy có thể trao đổi chúng như sau :

```
MOV AX,W+18 ; AX = W[10]
XCHG W+48,AX ; AX= W[25]
MOV W+18, AX ; complete exchange
```

7.2 Các chế độ địa chỉ (addressing modes)

Cách thức chỉ ra toán hạng trong lệnh gọi là chế độ địa chỉ. Các chế độ địa chỉ thường dùng là :

- Chế độ địa chỉ bằng thanh ghi (register mode) : toán hạng là thanh ghi
- Chế độ địa chỉ tức thời (immediate mode) : toán hạng là hằng số
- Chế độ địa chỉ trực tiếp (direct mode) : toán hạng là biến

Ví dụ :

```
MOV AX,0 ; AX là register mode còn 0 là immediate mode
ADD ALPHA,AX ; ALPHA là direct mode
```

Ngoài ra còn có 4 chế độ địa chỉ khác là :

- Chế độ địa chỉ gián tiếp bằng thanh ghi (register indirect mode)
- Chế độ địa chỉ cơ sở (based mode)
- Chế độ địa chỉ chỉ số (indexed mode)
- Chế độ địa chỉ chỉ số sơ sở (based indexed mode)

7.2.1 Chế độ địa chỉ gián tiếp bằng thanh ghi

Trong chế độ địa chỉ gián tiếp bằng thanh ghi, địa chỉ offset của toán hạng được chứa trong 1 thanh ghi. Chúng ta nói rằng thanh ghi là con trỏ (pointer) của vị trí nhớ. Dạng toán hạng là [register].

Trong đó register là các thanh ghi BX, SI, DI , BP. Đối với các thanh ghi BX, SI, DI thì thanh ghi đoạn là DS. Còn thanh ghi đoạn của BP là SS .

Ví dụ : giả sử rằng SI = 100h và từ nhớ tại địa chỉ DS:0100h có nội dung là 1234h. Lệnh MOV AX,[SI] sẽ copy 1234h vào AX .

Giả sử rằng nội dung các thanh ghi và nội dung của bộ nhớ tương ứng là như sau :

Thanh ghi **Nội dung** **Offset** **Nội dung bộ nhớ**

AX	1000h	1000h	1BACh
SI	2000h	2000h	20FFh
DI	3000h	3000h	031Dh

Ví dụ 1:
Hãy cho biết lệnh nào sau đây là hợp lý, offset nguồn và kết qủa của các lệnh hợp lý .
a. MOV BX,[BX]
b. MOV CX,[SI]
c. MOV BX,[AX]
d. ADD [SI],[DI]
e. INC [DI]
Lời giải :

	Source offset	**Result**
a	1000h	1BACh
b	2000h	20FFh
c	illegal source register (must be BX,SI,DI)	
d	illegal memory-memory add	
e	3000h	031Eh

Ví dụ 2 : Viết đoạn mã để cộng vào AX 10 phần tử của một mảng W định nghĩa như sau :
W DW 10,20,30,40,50,60,70,80,90,100
Giải :

```
XOR AX,AX ; xoá AX
LEA SI,W ; SI trỏ tới địa chỉ cơ sở (base) của mảmg W .
MOV CX,10 ; CX chuá số phần tử của mảng
ADDITION:
ADD AX,[SI] ; AX=AX + phần tử thứ nhất
ADD SI,2 ; tăng con trỏ lên 2
LOOP ADDITION ; lặp
```

Ví dụ 3 : Viết thủ tục để đảo ngược một mảng n từ. Điều này có nghĩa là phần tử thứ nhất sẽ đổi thành phần tử thứ n, phần tử thứ hai sẽ thành phần tử thứ n-1 ... Chúng ta sẽ dùng SI như là con trỏ của mảng còn BX chứa số phần tử của mảng (n từ) .
Giải : Số lần trao đổi là N/2 lần. Nhớ rằng phần tử thứ N của mảng có địa chỉ A+2x(N-1)
Đoạn mã như sau :

```
REVERSE PROC
; input: SI= offset of array
; BX= number of elements
; output : reverse array
PUSH AX ; cất các thanh ghi
PUSH BX
PUSH CX
PUSH SI
PUSH DI
; DI chỉ tới phần tử thứ n
```

```
MOV DI,SI ; DI trỏ tới từ thứ nhất
MOV CX,BX ; CX=BX=n : số phần tử
DEC BX ; BX=n-1
SHL BX,1 ;BX=2x(n-1)
ADD DI,BX ;DI = 2x(n-1) + offset của mảng : chỉ tới phần tử
; thứ n
SHR CX,1 ;CX=n/2 : số lần trao đổi
; trao đổi các phần tử
XCHG_LOOP:
MOV AX,[SI] ; lấy 1 phần tử ở nửa thấp của mảng
XCHG AX,[DI] ; đưa nó lên nửa cao của mảng
MOV [SI],AX ; hoàn thành trao đổi
ADD SI,2 ; SI chỉ tới phần tử tiếp theo của mảng
SUB DI,2 ; DI chỉ tới phần tử thứ n-1
LOOP XCHG_LOOP
POP DI
POP SI
POP CX
POP BX
POP AX
RET
REVERSE ENDP
```

7.2.2 Chế độ địa chỉ chỉ số và cơ sở

Trong các chế độ địa chỉ này, địa chỉ offset của toán hạng có được bằng cách cộng một số gọi là displacement với nội dung của một thanh ghi .

Displacement có thể là :

- địa chỉ offset của một biến, ví dụ A
- một hằng (âm hoặc dương), ví dụ -2
- địa chỉ offset của một biến cộng với một hằng số, ví dụ A+4

Cú pháp của một toán hạng có thể là một trong các kiểu tương đương sau :

[register + displacement]
[displacement + register]
[register]+ displacement
[displacement]+ register
displacement[register]

Các thanh ghi phải là BX, SI, DI (địa chỉ đoạn phải là thanh ghi DS) và BP (thanh ghi SS chứa địa chỉ đoạn)

Chế độ địa chỉ được gọi là cơ sở (based) nếu thanh ghi BX(base register) hoặc BP (base pointer) được dùng .

Chế độ địa chỉ được gọi là chỉ số (indexed) nếu thanh ghi SI(source index) hoặc DI (destination index) được dùng .

Ví dụ : Giả sử rằng W là mảng từ và BX chứa 4. Trong lệnh

```
MOV AX,W[BX}
```

displacement là địa chỉ offset của biến W. Lệnh này sẽ di chuyển phần tử có địa chỉ W+4 vào thanh ghi AX. Lệnh này cũng có thể viết dưới các dạng tương đương sau :

```
MOV AX, [W+BX]
MOV AX, [BX+W]
MOV AX, W+[BX]
MOV AX, [BX]+W
```

Lấy ví dụ khác, giả sử rằng SI chứa địa chỉ của mảng từ W. Trong lệnh

```
MOV AX,[SI+2]
```

displacement là 2 .Lệnh này sẽ di chuyển nội dung của từ nhớ W+2 tới AX . Lệnh này cũng có thể viết dưới các dạng khác :

```
MOV AX,[2+SI]
MOV AX,2+[SI]
MOV AX,[SI]+2
MOV AX,2[SI]
```

Với chế độ địa chỉ cơ sở có thể viết lại code cho bài toán tính tổng 10 phần tử của mảng như sau:

```
XOR AX,AX ; xoá AX
XOR BX,BX ; xoá BX (thanh ghi cơ sở )
MOV CX,10 ; CX= số phần tử =10
ADDITION:
ADD AX,W[BX} ; sum=sum+element
ADD BX,2 ; trỏ tới phần tử thứ hai
LOOP ADDITION
```

Ví dụ : Giả sử rằng ALPHA được khai báo như sau :

ALPHA DW 0123h,0456h,0789h,0ADCDH

trong đoạn được địa chỉ bởi DS và giả sử rằng :

BX =2 [0002]= 1084h
SI=4 [0004]= 2BACh
DI=1

Chỉ ra các lệnh nào sau đây là hợp lệ, địa chỉ offset nguồn và số được chuyển .

a. MOV AX,[ALPHA+BX]
b. MOV BX,[BX+2]
c. MOV CX,ALPHA[SI}
d. MOV AX,-2[SI]
e. MOV BX,[ALPHA+3+DI]
f. MOV AX,[BX]2
g. MOV BX,[ALPHA+AX]

Giải :

	Source offset	**Number moved**
a	ALPHA+2	0456h
b	2+2 = 4	2BACh
c	ALPHA+4	0789h
d	-2+4=+2	1084h
e	ALPHA+3+1=ALPHA+4	0789h
f	illegal form source operand ...[BX]2	

```
g       illegal ; thanh ghi AX là không được
        phép
```

Ví dụ sau đây cho thấy một mảng được xử lý như thế nào bởi chế độ địa chỉ chỉ số và cơ sở .
Ví dụ : Đổi các ký tự viết thường trong chuỗi sau thành ký tự viết hoa .

MSG DB 'co ty lo lo ti ca '

Giải :

```
MOV CX,17 ; số ký tự chứa trong CX=17
XOR SI,SI ; SI chỉ số cho ký tự
TOP:
CMP MSG[SI], ' ' ; blank?
JE NEXT ; yes, skip
AND MSG[SI],0DFH ; đổi thành chữ hoa
NEXT:
INC SI ; chỉ số ký tự tiếp theo
LOOP TOP ; lặp
```

Trong các chương trước chúng ta đã biết rằng các toán hạng của một lệnh phải cùng loại, tức là cùng là byte hoặc cùng là từ .Nếu một toán hạng là hằng số thì ASM sẽ chuyển chúng thành loại tương ứng với toán hạng kia. Ví dụ, ASM sẽ thực hiện lệnh MOV AX,1 như là lệnh toán hạng từ. Tương tự, ASM sẽ thực hiện lệnh MOV BH,5 như là lệnh byte. Tuy nhiên, lệnh MOV [BX],1 là không hợp lệ vì ASM không biết toán hạng chỉ bởi thanh ghi BX là toán hạng byte hay toán hạng từ. Có thể khắc phục điều này bằng toán tử PTR như sau :

```
MOV BYTE PTR [BX],1 ; toán hạng đích là toán hạng byte
MOV WORD PTR [BX],1 ; toán hạng đích là toán hạng từ
```

Ví dụ : Thay ký tự t thành T trong chuỗi được định nghĩa bởi :

```
MSG DB 'this is a message'
```

Cách 1: Dùng chế độ địa chỉ gián tiếp thanh ghi :

```
LEA SI,MSG ; SI trỏ tới MSG
MOV BYTE PTR [SI],'T' ; thay t bằng T
```

Cách 2 : Dùng chế độ địa chỉ chỉ số :

```
XOR SI,SI ; xoá SI
MOV MSG[SI],'T ' ; thay t bởi T
```

Ở đây không cần dùng PTR vì MSG là biến byte .
Nói chung toán tử PTR được dùng để khai báo loại (type) của toán hạng. Cú pháp chung của nó như sau:

Type PTR address_expression

Trong đó Type : byte, word, Dword
Addres_expression : là các biến đã được khai báo bởi DB,DW, DD .
Ví dụ chúng ta có 2 khai báo biến như sau :

```
DOLLARS DB 1AH
CENTS DB 52H
```

và chúng ta muốn di chuyển DOLLARS vào AL, di chuyển CENTS vào AH chỉ bằng một lệnh MOV duy nhất. Có thể dùng lệnh sau :

```
MOV AX, WORD PTR DOLLARS ; AL=DOLLARS và AH=CENTS
```

Toán tử giả LABEL

Có một cách khác để giải quyết vấn đề xung đột về loại toán hạng như trên bằng cách dùng toán tử giả LABEL như sau đây :

```
MONEY LABEL WORD
DOLLARS DB 1AH
CENTS DB 52H
```

Các lệnh trên đây khai báo biến MONEY là biến từ với 2 thành phần là DOLLARS và CENTS. Trong đó DOLLRAS có cùng địa chỉ với MONEY. Lệnh

```
MOV AX, MONEY
```

Tương đương với 2 lệnh :

```
MOV AL, DOLLARS
MOV AH, CENTS
```

Ví dụ : Giả sử rằng số liệu được khai báo như sau :

```
.DATA
A DW 1234h
B LABEL BYTE
DW 5678h
C LABEL WORD
C1 DB 9Ah
C2 DB 0bch
```

Hãy cho biết các lệnh nào sau đây là hợp lệ và kết qủa của lệnh .

a. MOV AX,B
b. MOV AH,B
c. MOV CX,C
d. MOV BX,WORD PTR B
e. MOV DL,WORD PTR C
f. MOV AX, WORD PTR C1

Giải :

a. không hợp lệ
b. hợp lệ, 78h
c. hợp lệ, 0BC9Ah
d. hợp lệ, 5678h
e. hợp lệ, 9Ah
f. hợp lệ, 0BC9Ah

Trong chế độ địa chỉ gián tiếp bằng thanh ghi, các thanh ghi con trỏ BX,SI hoặc DI chỉ ra địa chỉ offset còn thanh ghi đoạn là DS. Cũng có thể chỉ ra một thanh ghi đoạn khác theo cú pháp sau :

segment_register : [pointer_register]

Ví dụ : MOV AX, ES:[SI]

nếu SI=0100h thì địa chỉ của toán hạng nguồn là ES:0100h

Việc chiếm đoạn cũng có thể dùng với chế độ địa chỉ chỉ số và chế độ địa chỉ cơ sở .

7.2.5 Truy xuất đoạn stack

Như chúng ta đã nói trên đây khi BP chỉ ra một địa chỉ offset trong chế độ địa chỉ gián tiếp bằng thanh ghi, SS sẽ cung cấp số đoạn. Điều này có nghĩa là có thể dùng dùng BP để truy xuất stack .
Ví dụ : Di chuyển 3 từ tại đỉnh stack vào AX,BX,CX mà không làm thay đổi nội dung của stack .

```
MOV BP,SP ; BP chỉ tới đỉnh stack
MOV AX,[BP] ; copy đỉnh stack vào AX
MOV BX,[BP+2] ; copy từ thứ hai trên stack vào BX
MOV CX,[BP+4] ; copy từ thứ ba vào CX
```

7.3 Sắp xếp số liệu trên mảng

Việc tìm kiếm một phần tử trên mảng sẽ dễ dàng nếu như mảng được sắp xếp (sort). Để sort mảng A gồm N phần tử có thể tiến hành qua N-1 bước như sau :
Bước 1: Tìm số lớn nhất trong số các phần tử A[1]...A[N]. Gán số lớn nhất cho A[N] .
Bước 2 : Tìm số lớn nhất trong các số A[1]...A[N-1]. Gán số lớn nhất cho A[N-1}
.
.
.
Bước N-1 : Tìm só lớn nhất trong 2 só A[1] và A[2}. Gán só lớn nhất cho A[2}
Ví dụ : giả sử rằng mảng A chứa 5 phần tử là các số nguyên như sau :

Position	**1**	**2**	**3**	**4**	**5**
initial	21	5	16	40	7
Bước 1	21	5	16	7	40
Bước 2	7	5	16	21	40
Bước 3	7	5	16	21	40
Bước 4	5	7	16	21	40

Thuật toán

```
i =N
FOR N-1 times DO
find the position k of the largest element among A[1]..A[i]
Swap A[i] and A[k] (uses procedure SWAP )
i=i-1
END_FOR
```

Sau đây là chương trình để sort các phần trong mộ mảng. Chúng ta sẽ dùng thủ tục SELECT để chọn phần tử trên mảng. Thủ tục SELECT sẽ goị thủ tục SWAP để sắp xếp. Chương trình chính sẽ như sau :

```
TITLE PGM7_3: TEST SELECT
.MODEL SMALL
.STACK 100H
.DATA
A DB 5,2,,1,3,4
.CODE
MAIN PROC
MOV AX,@DATA
```

```
MOV DS,AX
LEA SI,A
MOV BX,5 ; số phần tử của mảng chứa trong BX
CALL SELECT
MOV AH,4CH
INT 21H
MAIN ENDP
INCLUDE C:\ASM\SELECT.ASM
END MAIN
```

Tập tin SELECT.ASM chứa thủ tục SELECT vàthủ tục SWAP được viết như
sau tại C:\ASM .

```
SELECT PROC
; sắp xếp mảng byte
; input: SI = địa chỉ offset của mảng
BX= số phần tử (n) của mảng
; output: SI = địa chỉ offset của mảng đã sắp xếp .
; uses : SWAP
PUSH BX
PUSH CX
PUSH DX
PUSH SI
DEC BX ; N = N-1
JE END_SORT ; Nếu N=1 thì thoát
MOV DX,SI ; cất địa chỉ offfset của mảng vào DX
; lặp N-1 lần
SORT_LOOP:
MOV SI,DX ; SI trỏ tới mảng A
MOV CX,BX ; CX = N -1 số lần lặp
MOV DI,SI ; DI chỉ tới phần tử thứ nhất
MOV AL,[DI] ; AL chứa phần tử thứ nhất
; tìm phần tử lớn nhất
FIND_BIG:
INC SI ; SI trỏ tới phần tử tiếp theo
CMP [SI],AL ; phần tử tiếp theo > phần tử thứ nhất
ING NEXT ; không, tiếp tục
MOV DI,SI ; DI chứa địa chỉ của phần tử lớn nhất
MOV AL,[DI] ; AL chứa phần tử lớn nhất
NEXT:
LOOP FIND_BIG
; swap phần tử lớn nhất với phần tử cuối cùng
CALL SWAP
```

```
DEC BX ; N= N-1
JNE SORT_LOOP ; lặp nếu N<>0
END_SORT:
POP SI
POP DX
POP CX
POP BX
RET
SELECT ENDP
SWAP PROC
; đổi chỗ 2 phần tử của mảng
; input : SI= phần tử thứ nhất
; DI = phần tử thứ hai
; output : các phần tử đã trao đổi
PUSH AX ; cất AX
MOV AL,[SI] ; lấy phần tử A[i]
XCHG AL,[DI] ; đặt nó trên A[k]
MOV [SI],AL ; đặt A[k] trên A[i]
POP AX ; lấy lại AX
RET
SWAP ENDP
```

Sau khi dịch chương trình, có thể dùng DEBUG để chạy thử và test kết qủa .

7.4 Mảng 2 chiều

Mảng 2 chiều là một mảng của một mảng, nghĩa là một mảng 1 chiều mà các phần tử của nó là một mảng 1 chiều khác. Có thể hình dung mảng 2 chiều như một ma trận chữ nhật. Ví dụ mảng B gồm có 3 hàng và 4 cột (mảng 3x4) như sau :

ROW\COLUMN	**1**	**2**	**3**	**4**
1	B[1,1]	B[1,2]	B[1,3]	B[1,4]
2	B[2,1]	B[2,2]	B[2,3]	B[2,4]
3	B[3,1]	B[3,2]	B[3,3]	B[3,4]

Bởi vì bộ nhớ là 1 chiều vì vậy các phần tử của mảng 2 chiều phải được lưu trữ trên bộ nhớ theo kiểu lần lượt. Có 2 cách được dùng :

- Cách 1 là lưu trữ theo thứ tự dòng : trên mảng lưu trữ các phần tử của dòng 1 rồi đến các phần tử của dòng 2 ...
- Cách 2 là lưu trữ theo thứ tự cột : trên mảng lưu trữ các phần tử của cột 1 rồi đến các phần tử của cột 2...

Giả sử mảng B chứa 10,20,30,40 trên dòng 1
chứa 50,60,70,80 trên dòng 2
chứá 90,100,110,120 trên dòng 3

Theo trật tự hàng chúng được lưu trữ như sau :

B DW 10,20,30,40
DW 50,60,70,80

DW 90,100,110,120

Theo trật tự cột chúng được lưu trữ như sau :

B DW 10,50,90
DW 20,60,100
DW 30,70,110
DW 40,80,120

Hầu hết các ngôn ngữ cấp cao biên dịch mảng 2chiều theo trật tự dòng . Trong ASM, chúng ta có thể dùng một trong 2 cách : nếu các thành phần của một hàng được xử lý lần lượt thì cách lưu trữ theo trật tự hàng được dùng. Ngược lại thì dùng cách lưu trữ theo trật tự cột .

Xác định một phần tử trên mảng 2 chiều :

Giả sử rằng mảng A gồm MxN phần tử lưu trữ theo trật tự dòng. Goị S là độ lớn của một phần tử : S=1 nếu phần tử là byte, S=2 nếu phần tử là từ. Để tìm phần tử thứ A[i,j] thì cần tìm : hàng i và tìm phần tử thứ j trên hàng này. Như vậy phải tiến hành qua 2 bước :

Bước 1: Hàng 1 bắt đầu tại vị trí A. Vì mỗi hàng có N phần tử, do đó

Hàng 2 bắt đầu tại A+ NxS .

Hàng 3 bắt đầu tại A+2xNxS .

Hàng thứ i bắt đầu tại A+(i-1)xSxN .

Bước 2: Phần tử thứ j trên một hàng cách vị trí đầu hàng (j-1)xS byte

Từ 2 bước trên suy ra rằng trong mảng 2 chiều NxM phần tử mà chúng được lưu trữ theo trật tự hàng thì phần tử A[i,j] có địa chỉ được xác định như sau :

A+((i-1)xN + (j-1))x S (1)

Tương tự nếu lưu trữ theo trật tự cột thì phần tử A[i,j] có địa chỉ như sau :

A+(i-1)+(j--)xM)xS (2)

Ví dụ : Giả sử A là mảng MxN phần tử kiểu từ (S=2) được lưu trữ theo kiểu trật tự hàng. Hỏi :

Hàng i bắt đầu tại địa chỉ nào ?

Cột j bắt đầu tại địa chỉ nào ?

Hai phần tử trên một cốt cách nhau bao nhiêu bytes

Giải :

Hàng i bắt đầu tại A[i,1] theo công thức (1) thì nó có địa chỉ là : A+(i-1)xNx2

Cột j bắt đầu tại A[1,j] theo công thức (1) thì nó có địa chỉ : A+(j-1)x2

Vì có N cột nên 2 phần tử trên cùng một cột cách nhau 2xN byte .

Trong chế độ này, địa chỉ offset của toán hạng là tổng của :

1. nội dung của thanh ghi cơ sở (BX or BP)
2. nội dung của thanh ghi chỉ số (SI or DI)
3. địa chỉ offset của 1 biến (tuỳ chọn)
4. một hằng âm hoặc dương (tuỳ chọn)

Nếu thanh ghi BX được dùng thì DS chứa số đoạn của địa chỉ toán hạng .Nếu BP được dùng thì SS chuá số đoạn. Toán hạng được viết theo 4 cách dưới đây:

1. variable[base_register][index_register]
2. [base_register + index_register + variable + constant]
3. variable [base_register + index_register + constant]
4. constant [base _ register + index_register + variable]

Trật tự của các thành phần trong dấu ngoặc là tuỳ ý .

Ví dụ, giả sử W là biến từ, BX=2 và SI =4. Lệnh

```
MOV AX, W[BX][SI]
```

sẽ di chuyển nội dung của mảng tại địa chỉ W+2+4 = W+6 vào thanh ghi AX
Lệnh này cũng có thể viết theo 2 cách sau :

```
MOV AX,[W+BX+SI]
MOV AX,W[BX+SI]
```

Chế độ địa chỉ chỉ số cơ sở thường được dùng để xử lý mảng 2 chiều như ví dụ sau : Giả sử rằng A là mảng 5x7 từ được lưu trữ theo trật tự dòng. Viết đoạn mã dùng chế độ địa chỉ chỉ số để :

1) xóa dòng 3
2) xoá cột 4

Giải :
1) Dòng i bắt đầu tại A+(i-1)xNx2. Như vậy dòng 3 bắt đầu tại A+(2-1)x7x2= A + 28. Có thể xóa dòng 3 như sau :

```
MOV BX,28 ; BX chỉ đến đầu dòng 3
XOR SI,SI ; SI sẽ chỉ mục cột
MOV CX,7 ; CX= số phần tử của một hàng
CLEAR:
MOV A[BX][SI],0 ; xoá A[3,1]
ADD SI,2 ; đến cột tiếp theo
LOOP CLEAR
```

2) Cột j bắt đầu tại địa chỉ A + (j-1)x2. Vậy cột 4 bắt đầu tại địa chỉ A+(4-1)x2 = A+ 6. Hai phần tử trên một cột cách nhau Nx2 byte, ở đây N=7, vậy 2 phần tử cách nhau 14 byte. Có thể xóa cột 4 như sau :

```
MOV SI,6 ; SI chỉ đến cột 4
XOR BX,BX ; BX chỉ đến hàng
MOV CX,5 ; CX= 5 : số phần tử trên một cột
CLEAR:
MOV A[BX][SI],0 ; Xoá A[i,4]
ADD BX,1 ; đến dòng tiếp theo
LOOP CLEAR
```

7.6 Ưng dụng để tính trung bình

Giả sử một lớp gồm 5 sinh viên và có 4 môn thi. Kết qủa cho bởi mảng 2 chiều như sau :

Tên SV	Bài thi 1	Bài thi 2	Bài thi 3	Bài thi 4
Mary	67	45	98	33
Scott	70	56	87	44
George	82	72	89	40
Beth	80	67	95	50
Scam	78	76	92	60

Chúng ta sẽ viết 1 chương trình tính điểm trung bình cho mỗi bài thi. Để làm điều này có thể tổng theo cột rồi chia cho 5 .
Thuật toán :

```
1. j = 4
2. repeat
3. Sum the scores in column j
```

```
4. divide sum by 5 to get average in column j
5. j = j - 1
5. Until j = 0
```

Trong đó bước 3 có thể làm như sau :

```
Sum[j]= 0
i = 1
FOR 5 times DO
Sum[j]= Sum[j]+ Score[i, j]
i = i + 1
END_FOR
```

Chương trình có thể viết như sau :

```
TITLE PGM7_4 : CLASS AVERAGE
.MODEL SMALL
.STACK 100H
.DATA
FIVE DB 5
SCORES DW 67,45,98,33 ; MARY
DW 70,56,87,44 ;SCOTT
DW 82,72,89,40 ;GEORGE
DW 80,67,,95,50 ; BETH
DW 78,76,92,60 ;SAM
AVG DW 5 DUP (0)
.CODE
MAIN PROC
MOV AX,@DATA
MOV DS,AX
;J=4
REPEAT:
MOV SI,6 ; SI chỉ đến cột thứ 4
XOR BX,BX ; BX chỉ hàng thứ nhất
XOR AX,AX ; AX chứa tổng theo cột
; Tổng điểm trên cột j
FOR:
ADD AX, SCORES[BX+SI]
ADD BX,8 ; BX chỉ đến hàng thứ 2
LOOP FOR
; end_for
; tính trung bình cột j
XOR DX,DX ; xoá phần cao của số bị chia (DX:AX)
DIV FIVE ; AX = AX/5
MOV AVG[SI],AX ; cất kết qủa trên mảng AVG
```

```
SUB SI,2 ; đến cột tiếp
; un til j=0
JNL REPEAT
;DOS EXIT
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN
```

Sau khi biên dịch chương teình có thể dùng DEBUG để chạy và xem kết qủa bằng lệnh DUMP.

7.7 Lệnh XLAT

Trong một số ứng dụng cần phải chuyển số liệu từ dạng này sang dạng khác. Ví dụ IBM PC dùng ASCII code cho các ký tự nhưng IBM Mainframes dùng EBCDIC (Extended Binary Coded Decimal Interchange Code). Để chuyển một chuỗi ký tự đã được mã hoá bằng ASCII thành EBCDIC, một chương trình phải thay mã ASCII của từng ký tự trong chuỗi thành mã EBCDIC tương ứng .

Lệnh XLAT (không có toán hạng) được dùng để đổi một giá trị byte thành một giá trị khác chứa trong một bảng .

AL phải chứa byte cần biến đổi

DX chứa địa chỉ offset của bảng cần biến đổi

Lệnh XLAT sẽ :

1) cộng nội dung của AL với địa chỉ trên BX để tạo ra địa chỉ trong bảng
2) thay thế giá trị của AL với giá trị tìm thấy trong bảng

Ví dụ, giả sử rằng nội dung của AL là trong vùng 0 đến Fh và chúng ta muốn thay nó bằng mã ASCII của số hex tương đương nó, tức là thay 6h bằng 036h=‘6’ , thay Bh bằng 042h=“B”. Bảng biến đổi là :

```
TABLE    DB    030h ,031h, 032h ,033h ,034h, 035h, 036h, 037h,038h,039h
         DB    041h, 042h ,043h, 044h, 045h, 046h
```

Ví dụ, để đổi 0Ch thành “C”, chúng ta thực hiện các lệnh sau :

```
MOV AL,0Ch ; số cần biến đổi
LEA BX,TABLE ; BX chuá địa chỉ offset của bảng
XLAT ; AL chứa “C”
```

Ở đây XLAT tính TABLE + Ch = TABLE +12 và thay thế AL bởi 043h. Nếu AL chứa một số không ở trong khỏang 0 đến 15 thì XLAT sẽ cho một giá trị sai .

Ví dụ : Mã hoá và giải mã một thông điệp mật

Chương trình này sẽ :

- Nhắc nhở người dùng nhập vào một thông điệp
- Mã hoá nó dưới dạng không nhận biết được ,
- In chúng ra ở dòng tiếp theo
- Dịch chúng trở lại dạng ban đầu rồi in chúng ở dòng tiếp theo

Khi chạy ct màn hình sẽ có dạng sau :

ENTER A MESSAGE :

DAI HOC DA LAT ; input

OXC BUC OX EXK ; encode
DAI HOC DA LAT ; translated

Thuật toán như sau :

```
Print prompt
Read and encode message
Go to anew line
Print encoded message
go to a new line
translate and print message
```

Code:

```
TITLE PGM7_5 : SECRET MESSAGE
.MODEL SMALL
.STACK 100H
.DATA
;ALPHABET ABCDEFGHIJKLMNOPQRSTUVWXYZ
CODE_KEY DB 65 DUP (' '), '
XQPOGHZBCADEIJUVFMNKLRSTWY'
DB 37 DUP (' ') ; 128 ký tự của bảng mã ASCII
CODED DB 80 dup ('$') ; 80 ký tự được gõ vào
DECODE_KEY DB 65 DUP (' '),
'JHIKLQEFMNTURSDCBVWXOPYAZG'
DB 37 DUP (' ')
PROMPT DB 'ENTER A MESSAGE :',0DH,0AH,'$'
CRLF DB 0DH,0AH,'$'
.CODE
MAIN PROC
MOV AX,@DATA
MOV DS, AX
; in dấu nhắc
MOV AH,9
LEA DX,PROMPT
INT 21H
; đọc và mã hoá ký tự
MOV AH,1
LEA BX,CODE_KEY ; BX chỉ tới CODE_KEY
LEA DI, CODED ; DI chỉ tới thông điệp đã mã hoá
WHILE_:
INT 21h ; đọc ký tự vào AL
CMP AL,0DH ; có phải là ký tự CR
JE ENDWHILE ; đúng, đến phần in thông điệp đã mã hoá
XLAT ; mã hoá ký tự
MOV [DI],AL ; cất ký tự trong CODE
```

```
JMP WHILE_ ; xử lý ký tự tiếp theo
; xuống hàng
MOV AH,9
LEA DX,CRLF
INT 21H
; in thông điệp đã mã hoá
LEA DX,CODED
INT 21H
; xuống hàng
LEA DX,CRLF
INT 21H
; giải mã thông điệp và in nó
MOV AH,2
LEA BX,DECODE_KEY ; BX chứa địa chỉ bảng giải mã
LEA SI,CODED ; SI chỉ tới thông điệp đã mã hoá
WHILE1:
MOV AL,[SI] ; lấy ký tự từ thông điệp đã mã hoá
CMP AL.'$' ; có phải cuối thông điệp
JE ENDWHILE1 ; kết thúc
XLAT ; giải mã
MOV DL,AL ;đặt ký tự vào DL
INT 21H ; in ký tự
INC SI ; SI=SI+1
JMP WHILE1 ; tiếp tục
ENDWHILE1:
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN
```

Trong chương trình có đoạn số liệu với các khai báo sau :

; ALPHABET ABCDEFGHIJKLMNOPQRSTUVWXYZ

Cho biết bảng chữa cái tiếng Anh

CODE_KEY DB 65 DUP (' '), ' XQPOGHZBCADEIJUVFMNKLRSTWY'

DB 37 DUP (' ')

Khai báo 128 ký tự của bảng mã ASCII, trong đó thứ tự các ký tự hoa là tuỳ ý .

CODED DB 80 dup ('$')

80 ký tự được gõ vào, giá trị ban đầu là $ để có thể in bằng hàm 9 ngắt 21h

DECODE_KEY DB 65 DUP (' '), 'JHIKLQEFMNTURSDCBVWXOPYAZG'

DB 37 DUP (' ')

Bảng giải mã được thiết lập theo cách mã hoá, nghĩa là trong phần mã hoá chúng ta đã mã hoá ‘A’ thành ‘X’ vì vậy khi giải mã ‘X’ phải giải mã thành ‘A’ ...
Các ký tự gõ vào không phải là ký tự hoa đều được chuyển thành ký tự trống.

KẾT THÚC