KHOA VÔ TUYẾN ĐIỆN TỬ
BỘ MÔN KỸ THUẬT VI XỬ LÝ

KỸ THUẬT VI XỬ LÝ VÀ LẬP TRÌNH HỢP NGỮ

Giáo viên: Nguyễn Khoa Sang

HỌC VIỆN KỸ THUẬT QUÂN SỰ - 2016

Chương 3. Lập trình hợp ngữ cho hệ vi xử lý

Nội dung

1. Tổng quan về ngôn ngữ Assembly
2. Các thành phần cơ bản của Assembly
3. Chương trình biên dịch Macro Assembler 5.1
4. Tập lệnh của bộ vi xử lý 80X86
5. Tổ chức Macro
6. Xây dựng chương trình Assembly

Tại sao lại dùng Assembly?

-Sử dụng trực tiếp tập lệnh của bộ VXL nên điều hành chức năng sát với phần cứng, khai thác triệt để khả năng của phần cứng mà các ngôn ngữ khác không làm được.

-Tốc độ thực thi nhanh nhất so với các ngôn ngữ khác

Tổng quan về ngôn ngữ Assembly

-Ngôn ngữ máy: các lệnh của chương trình được viết bằng mã nhị phân hay mã Hexa → nếu thêm, xóa 1 mã lệnh thì các mã lệnh có địa chỉ đi kèm như jump, call,… cũng phải tính lại

-Ngôn ngữ Assembly: là ngôn ngữ dưới dạng các ký hiệu hình thức hoặc từ gợi nhớ tuân theo quy tắc nào đó dễ đọc và dễ hiểu, mỗi từ gợi nhớ đó tương đương 1 lệnh CPU

-Assembler là chương trình dịch các chương trình viết bằng Assembly sang mã máy

Các thành phần cơ bản của Assembly

-File nguồn assembly gồm tập hợp các phát biểu hợp ngữ, mỗi phát biểu viết trên 1 dòng, có thể là 1 lệnh hoặc 1 chỉ dẫn

-Các lệnh assembly giống như lệnh CPU. Khi assembler dịch 1 file nguồn assembly thì mỗi lệnh sẽ được dịch sang 1 lệnh mã máy tương ứng

-Các lệnh chỉ dẫn dùng điều khiển cách dịch trình dịch assembler, không phải lệnh của CPU

Các thành phần cơ bản của Assembly (cont)

-Bộ ký tự, từ khóa, tên của Assembly

Các thành phần cơ bản của Assembly (cont)

-Cấu trúc 1 lệnh của assembly gồm 4 phần:

[nhãn:] tên lệnh [toán hạng] [;ghi chú]

+ Mỗi dòng 1 lệnh duy nhất, nằm trên 1 dòng

+ Mỗi phần cách nhau bằng 1 hoặc nhiều khoảng trắng

+ Các phần trong dấu [] có thể có hoặc không

+ Mỗi dòng dài tối đa 128 ký tự

Các thành phần cơ bản của Assembly (cont)

-Các dạng hằng dùng trong assembly: nhị phân, thập phân, hexa, chuỗi ký tự (xâu ký tự)

-Ví dụ:

10001111B;123D hay 123;56FH hay 0f8H;

“A nói: ‘I love you!’”;

Các thành phần cơ bản của Assembly (cont)

-Các chỉ dẫn trong assembly:

[Tên] tên chỉ dẫn [toán hạng] [;chú thích]

+ Mỗi phần cách nhau bởi 1 hoặc nhiều dấu cách

+ Toán hạng: tên tượng trưng, hằng, biến, biểu thức,….

+ Chỉ dẫn: dùng định nghĩa tên tượng trưng, khai báo dữ liệu, biến, …

Các chỉ dẫn trong assembly (cont)

-Nhóm định nghĩa tên tượng trưng: **EQU**

+ name EQU <text> hoặc name EQU expreession

+ Chức năng: định nghĩa tên tượng trưng name và gán giá trị bản text hoặc giá trị biểu thức 16 bit cho name.

+ VD: Table EQU [BX][SI]; 1 toán hạng

AD EQU 86; 1 hằng số

move EQU MOV; 1 từ khóa

Các chỉ dẫn trong assembly (cont)

-Nhóm định nghĩa tên tượng trưng: **=**

+ name = expreession

+ Chức năng: định nghĩa hoặc định nghĩa lại tên tượng trưng name và gán giá trị biểu thức 16 bit cho name nhiều lần.

+ VD: ab = 10;

MOV AL,dong;

ab = 20;

Các chỉ dẫn trong assembly (cont)

-Nhóm khai báo dữ liệu: khai báo vùng nhớ dành cho dữ liệu sử dụng trong chương trình như số, chuỗi, 1 biểu thức có trị xác định.

-Khai báo biến:

+ [name] **DB** data [,…]

+ [name] **DW** data [,…]

+ [name] **DD** data [,…]

VD: table DB 1,2,3,4,5

DB 6,7,8,9

Các chỉ dẫn trong assembly (cont)

-Nhóm khai báo dữ liệu:Toán tử **DUP** dùng để lặp lại các dữ liệu với số lần quy định bởi count.

-Cú pháp: Count **DUP** (data [,…]); toán tử DUP có thể lồng nhau

-Ví dụ:

Mem_500_bytes **DB** 500 **DUP**(0)

Các chỉ dẫn trong assembly (cont)

-Nhóm khai báo dữ liệu: Toán tử **?** dùng khi khai báo 1 biến hay 1 mảng mà không cần khởi tạo giá trị ban đầu.

-Ví dụ:

```
Mem_500bytes   DB 500 DUP (?);
Mem8  DB  ?;
```

Các chỉ dẫn trong assembly (cont)

-Nhóm khai báo dữ liệu: **Khai báo biến con trỏ**

+ Biến con trỏ là biến dùng chứa địa chỉ ô nhớ, có thể là near – chỉ chứa địa chỉ offset hoặc far – chứa cả địa chỉ mảng và offset

+ Ví dụ:

```
Nearnext  DW  offsetnext;
Farnext   DD  next;
```

Các chỉ dẫn trong assembly (cont)

-Nhóm khai báo dữ liệu: **Khai báo biến nhãn**

+ LABEL dùng định nghĩa 1 tên biến

+ Dùng khi muốn truy xuất đến cùng 1 vùng nhớ nhưng dạng dữ liệu khác nhau

+ VD:

```
Warray    LABEL    word;
Darray    LABEL    dword;
Barray    100  DUP (?);
```

Các chỉ dẫn trong assembly (cont)

-Nhóm khai báo mảng:

+ Chỉ dẫn SEGMENT và ENDS dùng để khởi đầu, kết thúc 1 mảng chứa 1 dạng cơ sở dữ liệu nào đó gồm CODE, DATA, EXTRA, STACK

+ Cú pháp:

Name SEGMENT [align_type] [combine_type] [‘class’]

……

Name ENDS

Các chỉ dẫn trong assembly (cont)

-Nhóm khai báo chương trình con:

+ Chỉ dẫn PROC và ENDP dùng để khởi đầu và kết thúc 1 modul chương trình dạng thủ tục

+ Cú pháp:

Name PROC [type]

……

Name ENDP

+ Nếu chương trình con dạng near thì chỉ được gọi trong cùng mảng chương trình con đó, dạng far thì có thể gọi trong mảng khác mảng chứa chương trình con.

Các chỉ dẫn trong assembly (cont)

-Nhóm tham chiếu bên ngoài: PUBLIC và EXTRN

-Dùng khi thiết kế chương trình lớn, phải chia chương trình thành nhiều file nguồn nhỏ. Khi assembler dịch từng file nhỏ sau đó ghép nối lại thành chương trình lớn cho VXL chạy. Để 1 tên biến, nhãn hay tên tượng trưng có thể dùng chung cho các phần khác thì ta khai báo dùng chỉ dẫn PUBLIC, khi sử dụng dùng EXTRN.

-Chỉ dẫn INCLUDE đọc file chứa các thành phần dùng chung.

Các chỉ dẫn trong assembly (cont)

-Nhóm chỉ dẫn điều khiển:

+ Chỉ dẫn: END [start_address];

+ Chỉ dẫn: EVEN; làm cho chương trình chạy nhanh hơn

Các chỉ dẫn trong assembly (cont)

-Nhóm chỉ dẫn chế độ: để báo cho assembler biết là dịch cho CPU nào.

.8086 cho 8086/8088, đồng xử lý toán học 8087

.286 cho 8086/8088/80286, đồng xử lý 80287

.386 cho 8086/8088/80386, đồng xử lý 80387

+ Chú ý: chỉ dẫn này đặt bên ngoài mảng

Các chỉ dẫn trong assembly (cont)

-Nhóm chỉ dẫn chú thích:

COMENT * [text]

text

Các toán tử dùng trong assembler

Sự khác nhau toán tử và lệnh:

-Toán tử điều khiển việc tính toán các trị hằng xác định khi dịch

-Lệnh điều khiển sự tính toán các giá trị không xác định được lúc dịch chương trình, khi thực hiện thì các giá trị mới xác định.

Các toán tử số học

Toán tử	*Cú pháp*	*công dụng*
+	+exp	Dấu dương
-	-exp	Dấu âm
*	exp1*exp2	Nhân
/	exp1/exp2	Chia
mod	exp1 mod exp2	Phần dư
+	exp1+ exp2	Cộng
-	exp1- exp2	Trừ
shl	exp shl count	Dịch exp sang trái *count* bit
shr	exp shr count	Dịch exp sang phải *count* bit.

Trong đó exp, exp1, exp2 là các biểu thức hằng, count là số nguyên.

Ví dụ:

```
MOV     BX , (80 * 4 +10)*2
MOV AX , 01110100B shl 2
MOV     dl , 300 mod 8
```

Các toán tử logic

Toán tử	*Cú pháp*
not	not exp
and	exp1 and exp2
or	exp1 or exp2
xor	exp1 xor exp2

Ví dụ:

```
MOV  AL , 8 or 4 and  2 ;
MOV  AL , not (20 xor 0011100B).
```

Nhóm các toán tử quan hệ

Toán tử	*Cú pháp*	*cho trị*
EQ	exp1 EQ exp2	true nếu exp1= exp2
NE	exp1 NE exp2	true nếu exp1<> exp2
LT	exp1 LT exp2	true nếu exp1< exp2
LE	exp1 LE exp2	true nếu exp1<= exp2
GT	exp1 GT exp2	true nếu exp1> exp2
GE	exp1 GE exp2	true nếu exp1>= exp2

Ví dụ:

```
MOV AX , 4 EQ 3      ; cho trị 0
MOV AX , 4 NE 3      ; cho trị -1
MOV AX , 4 LT 3      ; cho trị 0
```

Nhóm cung cấp thông tin về biến và nhãn

-Toán tử SEG: SEG expression; cho địa chỉ mảng của biểu thức

-Ví dụ:

```
table   DB      ?
MOV     AX, SEG table
MOV     DS, AX
MOV     DX, OFFSET table
```

Nhóm cung cấp thông tin về biến và nhãn

-Toán tử (:): segment : expression; quy định cách tính địa chỉ của 1 biến hay 1 nhãn đối với 1 mảng được chỉ ra.

-Lưu ý: nếu toán tử (:) dùng chung với các toán tử chỉ số [] thì segment phải đặt ngoài toán tử [].

-Ví dụ:

Ví dụ: Toán hạng [ES : DI] viết sai, phải viết lại là ES :[DI].

Ví dụ :

```
MOV   AX , ES : [BX +4]
MOV   AX , Data_seg: count
```

Nhóm cung cấp thông tin về biến và nhãn

-Toán tử $: cho địa chỉ offset của phát biểu chứa toán tử $ đó, dùng tính độ dài 1 chuỗi

Ví dụ:

```
str     DB  'To  count every byte in a string'
        DB 'especially if you might change'it later'
lenstr EQU      $ - offset str
```

Nhóm cung cấp thông tin về biến và nhãn

-Toán tử type: TYPE expression; cho biết độ rộng của expression;

Nếu expression là biến thì trị 1 biểu thị dạng byte, 2-word, 4-dword.

Nếu expression là nhãn thì trị OFFFFh biểu thị dạng near và OFFFEH biểu thị dạng far.

Nếu expression là hằng thì TYPE cho trị O.

Ví dụ:

```
var     DW   ?
array   DD   10  DUP  (?)
str     DB 'this is  string'
MOV     AX, TYPE var      ; cho trị 2
MOV     AX, TYPE array    ; cho trị 4
MOV     AX, TYPE  str     ; cho trị 1
```

Nhóm cung cấp thông tin về biến và nhãn

-Toán tử length: LENGTH var; cho số các đơn vị mà biến var xin cấp phát;

-Ví dụ:

```
table  DW  500 DUP (?)
MOV CX, LENGTH  table  ; cho trị 500
```

Nhóm thuộc tính

Toán tử PTR:

type PTR expression

cho phép thay đổi dạng của biểu thức expression

- Nếu expression là một biến hay một toán hạng bộ nhớ thì type có thể là byte, word hay dword.
- Nếu expression là một nhãn thì type có thể là near hay far.

Ví dụ:

```
stuff    DD  ?
table    DW  500 dup  (?)
MOV   AL, BYTE  PTR stuff; nạp byte đầu của mảng table vào AL
MOV BX, WORD  PTR stuff; nạp nội dung 2 byte thấp của biến
                                  Stuff vào thanh ghi BX
MOV   BYTE PTR [BX], 0; nạp trị 00 vào ô nhớ có địa chỉ xác định
                                bởi thanh ghi BX
MOV  WORD PTR [BX], 0 ; nạp trị 0000 vào vùng nhớ 2 byte có
                                  địa chỉ xác định bởi thanh ghi BX.
```

Nhóm thuộc tính

Toán tử high và low:

HIGH expression; Cho trị của byte cao và byte thấp của
LOW expression; biểu thức expression.

Expression phải là một trị hằng xác định khi dịch. Ví dụ:

```
Const OABCDH
MOV   AL, LOW const  ; trị CD →AL
MOV  AH ,   HIGH const  ; trị AB→AH
```

Chương trình biên dịch assembler

<Xem tài liệu>

Tập lệnh của bộ VXL 80X86

Quy ước khi viết cú pháp lệnh Assembly:

[ ] : Những tham số đặt trong 2 dấu [ ] là tuỳ chọn (có thể có hoặc không).
Reg : Chỉ toán hạng thanh ghi (8 bit hay 16 bit).
Reg8 : Chỉ toán hạng thanh ghi 8 bít.
Reg16 : Chỉ toán hạng thanh ghi 16 bít.
Mem : Chỉ toán hạng bộ nhớ 8 bít hay 16 bít
Mem8 : Chỉ toán hạng bộ nhớ 8 bít
Mem16 : Chỉ toán hạng bộ nhớ 16 bít
Mem32 : Chỉ toán hạng bộ nhớ 32 bít
Immed : Chỉ toán hạng trực tiếp (8 bit hay 16 bit).
Immed8 : Chỉ toán hạng trực tiếp 8 bit .
Immed16 : Chỉ toán hạng trực tiếp 16 bit.
SegReg : Chỉ toán hạng thanh ghi màng.

Nhóm lệnh chuyển dữ liệu

MOV dest, source;

-Chuyển dữ liệu giữa 2 thanh ghi

-Chuyển dữ liệu giữa thanh ghi và bộ nhớ

-Gán giá trị hằng vào thanh ghi hay bộ nhớ

-Chuyển dữ liệu giữa thanh ghi mảng và thanh ghi hay bộ nhớ

Nhóm lệnh chuyển dữ liệu

MOV dest, source;

-Chuyển dữ liệu giữa 2 thanh ghi

Reg8 <-- reg8 . Ví dụ:

MOV AL, AH; chuyển nội dung thanh ghi 8 bit AH vào AL

Reg16 <-- reg16. Ví dụ:

MOV BX, SI; chuyển nội dung thanh ghi 16 bit SI vào BX.

Nhóm lệnh chuyển dữ liệu

MOV dest, source;

-Chuyển dữ liệu giữa thanh ghi và bộ nhớ

Mem8 <-- *reg8*. Ví dụ:

```
        MOV   [BX], AL;chuyển AL vào ngăn nhớ có địa chỉ là nội
;dung của thanh ghi BX
        MOV   DS : [150h], AL
        MOV   ES : [SI], DL
```

Reg8 <-- *mem8*. Ví dụ:

```
        MOV    AL, mem [BX]
        MOV   AH, ds: [10H]
```

men16 <-- *reg16*.Ví dụ:

```
        MOV   [BX], AX
        MOV   varw,AX; varw là một biến dạng word
        MOV   CS : [SI+BX], DX
```

Nhóm lệnh chuyển dữ liệu

MOV dest, source;

-Gán giá trị hằng vào thanh ghi hay bộ nhớ

reg8 <-- *immed8*. Ví dụ:

```
MOV   AL, 0B5h
```

Mem8 <-- *imed8*. Ví dụ:

```
MOV   mem[BX],-1
MOV   byte ptr ds: [150h],10h
```

Reg16 <-- *immed16*.Ví dụ:

```
MOV AX,0B800H
```

mem16 <-- *immed16*.Ví dụ:

```
MOV  count,2000
MOV word ptr [BX],5AB7H
```

Nhóm lệnh chuyển dữ liệu

MOV dest, source;

-Chuyển dữ liệu giữa thanh ghi mảng và thanh ghi hay bộ nhớ

segreg <-- *reg16*. Ví dụ:

```
MOV  DS, AX
```

Segreg <-- *mem16*. Ví dụ:

```
MOV ES , screen
```

reg16 <-- *segreg*. Ví dụ:

```
MOV AX  , CS
```

mem16 <-- *segreg*. Ví dụ :

```
MOV  [BX + DI]  , ES
```

Nhóm lệnh chuyển dữ liệu

MOV dest, source; ***lưu ý***

Lệnh MOV không ảnh hưởng thanh ghi cờ. MOV không thể chuyển dữ liệu trực tiếp giữa hai toán hạng bộ nhớ với nhau, muốn chuyển ta phải dùng 1 thanh ghi trung gian.Ví dụ: Muốn chuyển dữ liệu 16 bit từ mem1 vào mem2, ta phải

```
MOV    AX, mem1
MOV    mem2, AX
```

MOV không thể chuyển trực tiếp một hằng vào một thanh ghi mảng, muốn chuyển ta phải dùng 1 thanh ghi trung gian. Ví dụ: Chuyển giá trị B800h vào thanh ghi DS, ta phải dùng một thanh ghi trung gian 16 bit

```
MOV     AX, 0B800H
MOV    DS  , AX
```

MOV không thể chuyển trực tiếp theo giữa hai thanh ghi mảng. Nó cũng không thể dùng thanh ghi mảng CS làm toán hạng đích trong lệnh MOV.

Nhóm lệnh chuyển dữ liệu

XCHG dest, Source

Toán hạng source và dest chỉ có thể là Reg hay Mem.

Chức năng: Hoán chuyển nội dung của 2 toán hạng nguồn và đích. Ta chỉ có thể hoán chuyển giữa thanh ghi và thanh ghi hoặc giữa thanh ghi và bộ nhớ. Ví dụ:

```
XCHG  AX , BX      ; Hoán chuyển giữa 2 thanh ghi 16 bít
XCHG  AL , CH      ; Hoán chuyển giữa 2 thanh ghi 8 bít
XCHG  AX , [BX]    ; Hoán chuyển giữa  thanh ghi và bộ nhớ
XCHG   BX , mem    ; Hoán chuyển giữa  thanh ghi và bộ nhớ
XCHG mem , AX
```

Chú ý: Lệnh này không ảnh hưởng thanh ghi cờ. Không thể dùng lệnh này với các thanh ghi mảng.

Nhóm lệnh chuyển dữ liệu

PUSH source (reg16 hay mem16)

PUSH immed16 (chỉ dùng với CPU 80286/386/486)

Chức năng: Dùng để cất một hằng hay nội dung một thanh ghi 16 bit hay nội dung một toán hạng bộ nhớ 16 bit vào STACK.

Lệnh PUSH giảm thanh ghi SP 2 đơn vị và chuyển nội dung 16 bit của toán hạng nguồn vào đỉnh của STACK. Đỉnh STACK được xác định bởi cặp thanh ghi SS:SP. Ví dụ:

```
PUSH  SI
PUSH  DS
PUSH  CS
PUSH  SP
PUSH  mem [BX][SI]
PUSH  [BX + SI]
PUSH  147EH (chỉ dùng với CPU 80286/80386/80486/80586 ...)
```

Nhóm lệnh chuyển dữ liệu

POP dest (reg16 hay mem16)

Chức năng : POP dùng để lấy dữ liệu 16 bit trong STACK (được xác định bởi cặp thanh ghi SS:SP) vào toán hạng dest.

Lệnh POP tăng thanh ghi SP 2 đơn vị để chi đến đỉnh mới của STACK.Ví dụ:

```
POP   SI
POP   DS
POP   CS
POP   SP
POP   mem[BX][SI]
POP   [BX + SI]
```

Nhóm lệnh chuyển dữ liệu

Lệnh PUSHA (PUSH ALL) (chỉ dùng với CPU 80286/386/486)

Chức năng: PUSH tất cả các thanh ghi theo thứ tự AX, CX, DX,BX,SP,BP,SI,DI lần lượt vào STACK.

Chú ý: Lệnh PUSHA không cất nội dung các thanh ghi mảng và thanh ghi cờ vào STACK.

Lệnh POPA (POP ALL) (chỉ dùng với CPU 80286/386)

Chức năng: POPA khôi phục nội dung tất cả các thanh ghi theo thứ tự ngược lại DI, SI, BP, SP, SP, BX, DX, CX, AX lần lượt từ STACK.

Nhóm lệnh chuyển dữ liệu

XLAT (source-cable)

Chức năng: Chuyển nội dung của ô nhớ nằm trong một bảng các ô nhớ 8 bit vào thanh ghi AL. Trong đó bảng có địa chỉ bắt đầu xác định bởi cặp thanh ghi DS:BX, Vị trí offset của byte nhớ trong bảng được xác định bởi thanh ghi AL

XLAT Tương đương với các lệnh sau:

```
MOV     AH, 0
MOV     SI,AX
MOV     AL,[BX+SI]
```

Nhóm lệnh chuyển địa chỉ

Lệnh LEA: (Load Effective address)

LEA reg16,men16

Chức năng: Chuyển địa chỉ offset của một toán hạng bộ nhớ mem16 vào thanh ghi đích reg16. Ví dụ:

LEA BX , ds:[500H] ; chuyển 500H vào BX.

LEA SI , table ; chuyển địa chỉ offset của table vào thanh ghi SI.

LEA BX ; table [SI] ; chuyển địa chỉ offset của bảng table + nội

; dung của thanh ghi SI vào thanh ghi BX.

Nhóm lệnh chuyển địa chỉ

Lệnh LDS: (Load Point er using DS)

LDS reg16, mem32

Chức năng: Chuyển nội dung toán hạng bộ nhớ mem32 vào cặp thanh ghi 16 bit, phần cao 16 bit của mem32 được nạp vào thanh ghi DS, còn phần thấp được nạp vào thanh ghi reg16 trong lệnh.

Lệnh LDS BX , MEM [SI] tương đương với:

MOV BX , MEM [SI]

MOV AX , MEM [SI +2]

MOV DS , AX

Nhóm lệnh chuyển địa chỉ

Lệnh LES : (Load pointer using ES)

LES reg16 , mem32

Chức năng: giống như lệnh LDS nhưng dùng thanh ghi mảng ES thay cho thanh ghi mảng DS.

Nhóm lệnh chuyển thanh ghi cờ

Lệnh LAHF: (Load AH from flag)

LAHF

Chức năng: Chuyển phần thấp của thanh ghi cờ gồm cac cờ SF, ZF, AF, PF và CF tương ứng vào các bit 7,6,4,2 và 0 của thanh ghi AH, các bit còn lại 5,3 và 1 không thay đổi.

Lệnh SAHP: (Store AH into Flag)

SAHF

Chức năng: Chuyển các bit 7,6,4,2 và 0 của thanh ghi AH tương ứng vào các cờ SF, ZF,AF,PF,CF của thanh ghi cờ. Các cờ còn lại OF,DF,IF,TF không bị ảnh hưởng.

Lệnh PUSHF:

PUSHF

Chức năng: SUSHF giảm thanh ghi SP 2 đơn vị và chuyển nội dung của phần tử trên đỉnh STAK vào thanh ghi cờ.

Nhóm lệnh chuyển dữ liệu qua cổng

Lệnh IN:

IN AL , port8

Hay IN AL , DX

Chức năng: Đọc một lượng 8 bit từ một cổng nhập vào thanh ghi AL. Nếu địa chỉ của cổng có giá trị trong khoảng từ 0 đến FF thì địa chỉ đó có thể viết trực tiếp trong câu lệnh, còn trong trường hợp địa chỉ của cổng có giá trị > FF thì ta phải dùng thanh ghi DX để định địa chỉ cổng.

Ví dụ:

```
IN    AL , 61H    ; đọc một byte từ cổng 61h
MOV DX , 03f8H
IN    AL , DX     ; đọc một byte từ cổng 03f8H.
```

Nhóm lệnh chuyển dữ liệu qua cổng

Lệnh OUT:

OUT port8 , AL

Hay OUT DX , AL

Chức năng : xuất một lượng 8 bit từ thanh ghi AL ra cổng xuất. Nếu địa chỉ của cổng có giá trị trong khoảng từ 0 đến FF thì địa chỉ đó có thể viết trực tiếp trong câu lệnh, còn trong trường hợp địa chỉ của cổng có giá trị > FF thì ta phải dùng thanh ghi DX để định địa chỉ cổng.

Ví dụ:

OUT 61H , AL ; gởi một byte ra cổng 61h

MOV DX , 03f8H

OUT DX , AL ; gởi một byte ra cổng 03f8H.

Nhóm lệnh chuyển điều khiển

Lệnh nhảy không điều kiện JMP(jump) có cú pháp:

JMP Label hoặc

JMP Target (reg hoặc mem)

Chức năng: chuyển điều khiển (CS **:** IP) tới vị trí mới được xác định bởi Label hay nội dung của Target.

Dạng1: JMP Label.

JMP near Label (chiếm 3 byte),

JMP short Label (chiếm 2 byte)

JMP far Label (chiếm 5 byte)

Dạng2: JMP Target.

JMP AX ; IP←AX

JMP word PTR [AX]; IP←[AX]

JMP dword PTR [AX]; IP←[AX] CS←[AX+2]

Nhóm lệnh chuyển điều khiển

Lệnh nhảy có điều kiện J<Điều kiện> có cú pháp:

J<Điều kiện> short_label

Lệnh nhảy có điều kiện trước tiên kiểm tra điều kiện. Nếu điều kiện thoả mãn thì nhảy tới nhãn ***short_label,*** còn không thoả mãn thì lệnh nhảy bị bỏ qua. Lệnh nhảy có điều kiện là lệnh 2 byte, byte đầu là mã lệnh, byte sau là địa chỉ tương đối. Do vậy khoảng cực đại mà nó nhảy được là -128 đến +128. Muốn nhảy xa hơn phải dùng lệnh nhảy không điều kiên.

Lệnh so sánh cú pháp

CMP left, right.

Left có thể là thanh ghi hay bộ nhớ còn right có thể là thanh ghi hay bộ nhớ hay hằng số. Có chức năng so sánh toán hạng ***left*** và ***right.*** Kết quả phản ánh trong các cờ trạng thái nhưng không làm thay đổi nội dung toán hạng ***left.***

Nhóm lệnh lặp

Lệnh LOOP có cú pháp:

LOOP short_label

Chức năng: Giảm nội dung CX đi 1 đơn vị và nhảy đến nhãn short_label nếu thoả mãn điều kiện CX<>0. Nếu CX=0 thì lệnh này bị bỏ qua.

Lệnh LOOPE có cú pháp:

LOOPE short_label

Chức năng: Giảm nội dung CX đi 1 đơn vị và nhảy đến nhãn short_label nếu thoả mãn điều kiện cờ ZF=1 và CX<>0. Nếu không thoả mãn thì lệnh này bị bỏ qua.

Nhóm lệnh lặp

Lệnh LOOPZ có cú pháp:

LOOPZ short_label

Chức năng: Tương đương lệnh LOOPE.

Lệnh LOOPNE có cú pháp:

LOOPNE short_label

Chức năng: Giảm nội dung CX đi 1 đơn vị và nhảy đến nhãn short_label nếu thoả mãn điều kiện cờ ZF=0 và CX<>0. Nếu không thoả mãn thì lệnh này bị bỏ qua.

Nhóm lệnh lặp

Lệnh LOOPNZ có cú pháp:

LOOPNZ short_label

Chức năng: Chức năng: Tương đương lệnh LOOPNE.

Thí dụ minh hoạ: Cho vùng nhớ BufferRAM có dung lượng 200 byte trong mảng dữ liệu do DS quản lý. Viết đoạn chương trình tìm giá trị ký tự $ trong vùng đó. Nếu tìm thấy thì thoát khỏi vòng lặp.

Giải:

```
MOV CX,200; bộ đếm lùi CX được gán 200
MOV BX,OFFSET BufferRAM-1; BX trỏ tới byte đầu của buffer-1
        Count:
INC BX;                 tăng con trỏ BX để trỏ tới byte tếp theo
CMP byte ptr [BX], '$'; so sánh byte đó với $
LOOPNZ count; nếu khác nhau thì lặp lại
JZ found        ; nếu bằng thì  thoát ra (nhảy tới nhãn found)
        found: ...
```

Lệnh gọi chương trình con

CALL Label

hoặc ***CALL target*** (reg/mem)

Dạng CALL near chỉ cần 3 byte còn CALL far chiếm 5 byte. Khi thực hiện lệnh gọi chương trình con thì trình tự tác động sau được thực hiện:

CALL Label	***CALL far ptr Label***
((SP))← (IP)	((SP))← (CS)
(IP) ← offset Label	((SP)-2)← (IP)
(SP)← (SP)-2	(IP) ← offset Label
	(CS) ← seg Label
	(SP)← (SP)-4

Lệnh quay lại từ chương trình con có cú pháp:

RET (constant) hay

RETN (constant) hay

RETF (constant) .

RETN dùng để kết thúc chương trình con kiểu near, RETF dùng để kết thúc chương trình con kiểu far. Nếu dùng RET thì phụ thuộc vào cách khai báo thủ tục là near thì nó tương đương RETN còn khai báo thủ tục là far thì nó tương đương RETF.

Khi thực hiện lệnh quay trở về thì trình tự tác động sau được thực hiện:

RETN	RETF
(IP)←((SP))	(IP)←((SP))
(SP)← (SP)+2	(CS)←((SP)+2)
	(SP)← (SP)+4

Nhóm lệnh tính toán số học

Lệnh cộng không nhớ ADD (addition) có cú pháp:

ADD dest, source

Dest←source+ Dest

Chức năng: là phép cộng không nhớ giữa số hạng dest và source, kết quả đặt trong dest. Chú ý là phép cộng không được thực hiện giữa hai thanh ghi mảng và dest chỉ có thể là dạng thanh ghi hay mem.

Lệnh cộng có nhớ ADC (add with carry) có cú pháp:

ADC dest, source

Dest←source+ Dest +(CF)

Chức năng: là phép cộng có nhớ giữa số hạng dest và source, tức là phải cộng với nội dung CF. Kết quả đặt trong dest. Chú ý là phép cộng không được thực hiện giữa hai thanh ghi mảng và dest chỉ có thể là dạng thanh ghi hay mem.

Nhóm lệnh tính toán số học

Lệnh tăng INC (increment) có cú pháp:

INC dest (reg/mem)

Dest← Dest +1

Lệnh chỉnh thập phân AAD (ascii adjust for addition) dùng để cộng hai số mã hoá ASCII mà kết quả cho ở dạng thập phân. AL là toán hạng đích.

Lệnh chỉnh thập phân DAA (decimal adjust for addition) dùng để cộng hai số mã hoá nhị phân mà kết quả cho ở dạng thập phân. AL là toán hạng đích.

Lệnh trừ không nhớ SUB (substract) có cú pháp:

SUB dest, source

Dest← Dest- source

Chức năng: là phép trừ không nhớ giữa số bị trừ dest và số trừ source, kết quả đặt trong dest. Chú ý là phép trừ không được thực hiện giữa hai thanh ghi mảng và dest chỉ có thể là dạng thanh ghi hay mem.

Nhóm lệnh tính toán số học

Lệnh trừ có nhớ SBB có cú pháp:

SBB dest, source

Dest← Dest - source - (CF)

Chức năng: là phép trừ có nhớ giữa số số bị trừ dest và số trừ source và trừ di nội dung CF. Kết quả đặt trong dest. Chú ý là phép trừ có nhớ không được thực hiện giữa hai thanh ghi mảng và dest chỉ có thể là dạng thanh ghi hay mem.

Lệnh giảm DEC (decrement) có cú pháp:

DEC dest (reg/mem)

Dest← Dest –1

Lệnh này chỉ ảnh hưởng tới cờ AF, OF, PF, SF, ZF mà không ảnh hưởng tới cờ CF.

Nhóm lệnh tính toán số học

Lệnh NEG có cú pháp:

NEG dest (reg/mem)

Dest← - Dest

Lệnh này dùng để đổi dấu toán hạng đích.

Lệnh chỉnh thập phân AAS (ascii adjust for substract) dùng để trừ hai số mã hoá ASCII mà kết quả cho ở dạng thập phân. AL là toán hạng đích.

Lệnh chỉnh thập phân DAS (decimal adjust for substract) dùng để trừ hai số mã hoá nhị phân mà kết quả cho ở dạng thập phân. AL là toán hạng đích.

Lệnh nhân số không dấu ***MUL*** (multiple) có cú pháp:

MUL source

Chức năng: là phép nhân không dấu. Nếu source là toán hạng 8 bit thì AL được ngầm định chứa thừa số thứ nhất còn source là thừa số thứ hai. Kết quả 16 bit chứa trong AX. Nghĩa là:

MUL source_8bit

(AX) ←(AL)*(source_8bit)

Nếu source là toán hạng 16 bit thì AX được ngầm định chứa thừa số thứ nhất còn source là thừa số thứ hai. Kết quả 32 bit chứa trong cặp DX:AX. Nghĩa là:

MUL source_16bit

(DX:AX) ←(AX)*(source_16bit)

Nhóm lệnh tính toán số học

Lệnh nhân số có dấu ***IMUL*** có cú pháp:

IMUL source

Chức năng: Giống như nhân số không dấu MUL nhưng là với toán hạng có dấu. Kết quả cũng là số có dấu.

Lệnh hiệu chỉnh thập phân AAM (ascii adjust for multiple) có cú pháp:

AAM

Chức năng: Hiệu chỉnh phép nhân hai số ASCII. AAM chia AL cho 10 và lưu phần thương số vào AL còn phần dư vào AH.

Lệnh chia số không dấu ***DIV*** (Division) có cú pháp:

DIV source

Chức năng: là phép chia không dấu. Nếu source là toán hạng 8 bit thì AX được ngầm định chứa số bị chia còn source là số chia. Thương số chứa trong AL còn số dư trong AH. Nghĩa là:

DIV source_8bit

(AL) ←(AX) DIV (source_8bit)

(AH) ←(AX) MOD (source_8bit)

Nếu source là toán hạng 16 bit thì DX:AX được ngầm định chứa số bị chia còn source là số chia. Thương số chứa trong AX còn số dư trong DX. Nghĩa là:

(AX) ←(DX:AX) DIV (source_16bit)

(DX) ←(DX:AX) MOD (source_16bit)

Lệnh DIV không ảnh hưởng tới các cờ.

Nhóm lệnh tính toán số học

Lệnh chia số có dấu ***IDIV*** có cú pháp:

IDIV source

Chức năng: Giống như chia số không dấu DIV nhưng là với toán hạng có dấu. Kết quả cũng là số có dấu. Lệnh IDIV không ảnh hưởng tới các cờ.

Lệnh hiệu chỉnh thập phân AAD (ascii adjust for division) có cú pháp:

AAD

Chức năng: Hiệu chỉnh phép chia hai số ASCII

Nhóm lệnh dịch chuyển và quay vòng

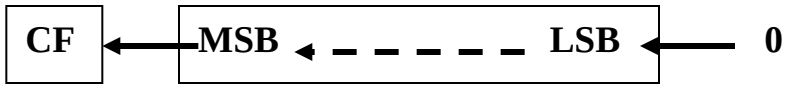
Lệnh SHL (shift logical left) có cú pháp:

SHL dest,1 hay SHL dest,CL

Chức năng: Dịch sang trái toán hạng dest 1 bit (trường hợp 1) hay dịch sang trái toán hạng dest với số bước bằng CL bit (trường hợp 2). Bit có trọng số lớn nhất hiện hành →CF còn bit có trọng số nhỏ nhất hiện hành được gán bằng 0 (hình 3.3).

Chú ý: Đối với số không dấu thì dich trái 1 bit tương đương nhân nó với 2. Các cờ CF, OF, PF, SF, ZF bị tác động khi thực hiện lệnh này.

Hình 3.3. Phương thức dịch chuyển của lệnh SHL



Lệnh SHR (shift logical right) có cú pháp:

SHR dest,1 hay SHR dest,CL

Chức năng: giống như lệnh SHL nhưng diễn ra theo chiều ngược lại.

<Còn lại tham khảo giáo trình>

Nhóm lệnh thực hiện phép tính logic

Lệnh nhân logic AND có cú pháp:

AND dest, source

Dest← dest AND source

CF← 0; OF← 0

Chức năng: Nhân logic giữa toán hạng dest với toán hạng source, kết quả đặt trong dest. Dest phải là dạng reg hay mem. Lệnh này có tác động tới các cờ CF, OF, PF, SF, SF, ZF.

Lệnh cộng logic OR có cú pháp:

OR dest, source

Dest← dest OR source

CF← 0; OF← 0

Chức năng: Cộng logic giữa toán hạng dest với toán hạng source, kết quả đặt trong dest. Dest phải là dạng reg hay mem. Lệnh này có tác động tới các cờ CF, OF, PF, SF, SF, ZF.

<Còn lại đọc giáo trình>

Nhóm lệnh xử lý xâu chuỗi

Lệnh MOVS(move string) có các dạng

MOVSB (chuyển byte),

MOVSW(chuyển word),

MOVS [ES:] dest, [seg:] source

Chức năng: chuyển dữ liệu vùng nhớ với điều kiện SI trỏ tới vùng source DI trỏ tới vùng dest. Mỗi lần chuyển SI, DI tự động tăng 1 đơn vị (nếu là MOVSB và cờ hướng DF=0) hoặc SI, DI tự động tăng 2 đơn vị (nếu là MOVSW và cờ hướng DF=0). Ngược lại, Mỗi lần chuyển SI, DI tự động giảm 1 đơn vị (nếu là MOVSB và cờ hướng DF=1) hoặc SI,DI tự động giảm 2 đơn vị (nếu là MOVSW và cờ hướng DF=1).

Nhóm lệnh xử lý xâu chuỗi

Lệnh so sành xâu chuỗi CMPS(compare string) có các dạng

CMPSB (so sành xâu chuỗi kiểu byte),

CMPSW(so sành xâu chuỗi kiểu word),

CMPS [ES:] dest, [seg:] source

Chức năng: so sành xâu chuỗi dữ liệu 2 vùng nhớ với điều kiện SI trỏ tới vùng source DI trỏ tới vùng dest. Mỗi lần so sánh SI, DI tự động tăng 1 đơn vị (nếu là CMPSB và cờ hướng DF=0) hoặc SI, DI tự động tăng 2 đơn vị (nếu là CMPSW và cờ hướng DF=0). Ngược lại, mỗi lần so sánh SI, DI tự động giảm 1 đơn vị (nếu là CMPSB và cờ hướng DF=1) hoặc SI,DI tự động giảm 2 đơn vị (nếu là CMPSW và cờ hướng DF=1).

<Còn lại xem giáo trình>

Tổ chức macro

- Định nghĩa một macro: Macro là tập hợp các lệnh assembly có thể xuất hiện nhiều lần trong một chương trình. Mỗi Macro có một tên riềng để chương trình gọi tới được. Đây là cách xây dựng các lệnh có chức năng rộng lớn hơn so với chức năng của 1 lệnh assembly. Mỗi Macro có thể hình dung như lệnh của ngôn ngữ bậc cao.

Tổ chức macro

- Định nghĩa một macro: Name MACRO [dummy parameter-list]
(Các lệnh assembly nằm ở đây)
ENDM

Như vậy MACRO gồm 3 phần:

Phần header dùng để khai báo một macro: Name là tên Macro cần định nghĩa, MACRO là chỉ dẫn cho trình biên dịch biết đây là cấu trúc macro, dummy là các tham số cần cho macro (có thể có cũng có thể không có).

Phần thân macro gồm các lệnh assembly quy định chức năng của macro.

Phần cuối là lệnh directive ENDM báo cho trình biên dịch biết đây là điểm kết thúc của macro.

Ưu điểm của macro:

Có thể truyền tham số cho macro một cách dễ dàng.

Tốc độ thực hiện nhanh hơn chương trình con vì các thao tác cất giữ và khôi phục thông tin trạng thái không cần thực hiện.

Có thể tạo thư viện các macro một cách dễ dàng.

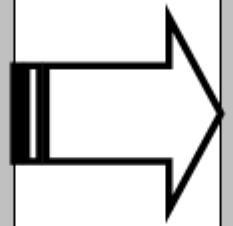
Tổ chức macro (cont)

- Các chỉ dẫn cho macro: ***Chỉ dẫn LOCAL*** có cú pháp

LOCAL localName [,localName] ...

Chức năng: dùng để định nghĩa các ký hiệu chỉ sử dụng riêng cho macro đó. Xét ví dụ tạo macro giữ chậm Delay:

```
Delay MACRO count
        PUSH CX
        MOV CX, count
next: LOOP next
        POP CX
```



```
Delay MACRO count
        LOCAL next
        PUSH CX
        MOV CX, count
next: LOOP next
        POP CX
```

Hình 3.9. Sử dụng chỉ dẫn LOCAL để tạo nhãn next nhiều lần

Đối với macro trên chương trình chính chỉ gọi được một lần vì ta đã biết là nhãn ***next*** chỉ được định nghĩa một lần. Muốn gọi được nhiều lần phải dùng chỉ dẫn ***local.*** Chỉ đãn này báo cho assembly biết để đổi nhãn thành nhãn mới mỗi khi chương trình gọi tới macro (hình 3.9).

Tổ chức macro (cont)

- Các chỉ dẫn cho macro: ***Chỉ dẫn điều kiện*** bao gồm

IFB(if blank) dùng để kiểm tra các tham số truyền cho macro có thiếu không. IFB có cú pháp

IFB <parameter>

(Tại đây các lệnh assemly nếu parameter là trống)

[ELSE + Tại đây các lệnh assemly nếu parameter là không trống]

ENDIF

IFNB(if not blank) dùng để kiểm tra các tham số truyền cho macro có tồn tại không. IFNB có cú pháp tương tự như IFB.

Chỉ dẫn lặp bao gồm

REPL expression định nghĩa một khối cần lặp với số lần chứa trong ***expression.*** Cấu trúc của nó là:

REPL expression

(Tại đây các lệnh assemly)

ENDM

Tổ chức macro (cont)

-Các toán tử cho macro:

Toán tử EQ có cú pháp exp1 EQ exp2 sẽ cho kết quả TRUE (=-1) nếu exp1 = exp2.

Toán tử NE có cú pháp exp1 NE exp2 sẽ cho kết quả TRUE (=-1) nếu exp1 <> exp2.

Toán tử LE có cú pháp exp1 LE exp2 sẽ cho kết quả TRUE (=-1) nếu exp1 <= exp2.

Toán tử GT có cú pháp exp1 GT exp2 sẽ cho kết quả TRUE (=-1) nếu exp1 > exp2.

Toán tử GE có cú pháp exp1 GE exp2 sẽ cho kết quả TRUE (=-1) nếu exp1 >= exp2.

Xây dựng chương trình assembly

-Các bước xây dựng chương trình:

Bước 1: Xây dựng lưu đồ thuật toán tổng quát. Tại đây các yếu tố của nhiệm vụ chương trình được xem xét và phân tích nhằm đưa ra giải pháp và phương thức thực hiện tối ưu. Nếu nhiệm vụ của chương trình là phức tạp thì nó sẽ được tách ra thành các nhiệm vụ con với mối quan hệ được xác định trong thuật toán tổng quát. Mỗi nhiệm vụ con phải có lưu đồ thuật toán riêng để giải quyết trọn vẹn chức năng của nhiệm vụ con đó.

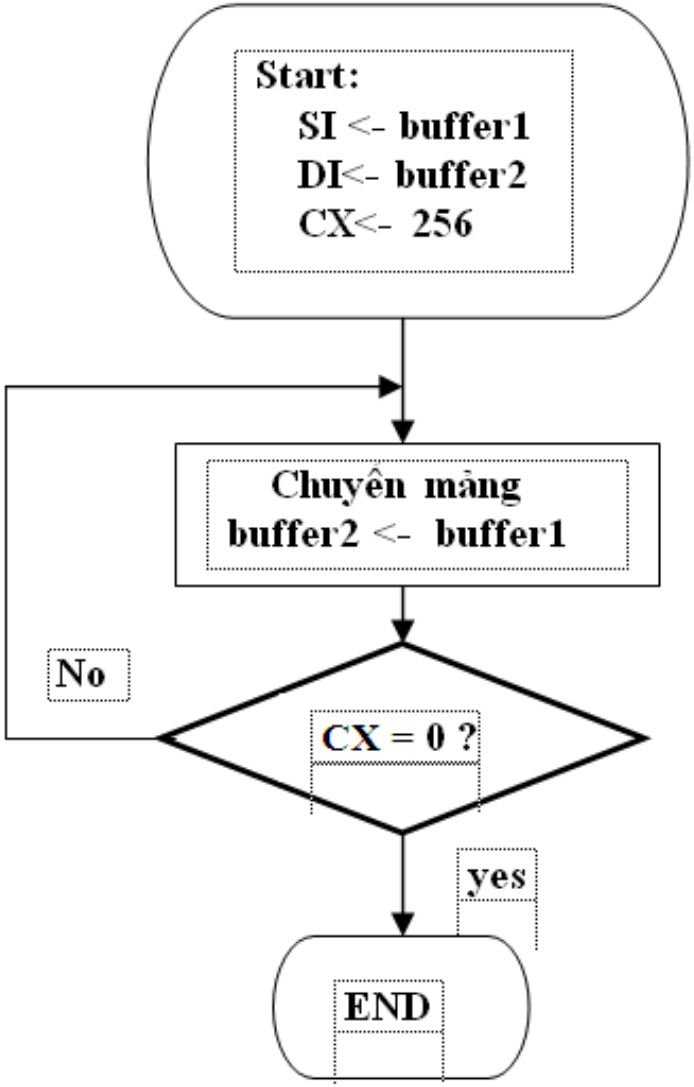
Bước2: Xây dựng chương trình nguồn assembly. Nếu chức năng không lớn thì chương trình nguồn assembly được viết ở dạng đơn giản, chỉ cần một thủ tục chính. Nếu chức năng lớn hay rất lớn thì chương trình nguồn assembly được viết ở dạng phức tạp hơn về cấu trúc. Có thể là một file gồm nhiều thủ tục, nhiều macro hay chương trình gồm nhiều file khác nhau.

Xây dựng chương trình assembly (cont)

- Ví dụ1:Viết chương trình chuyển mảng dữ liệu từ vùng nhớ Buffer1 có địa chỉ bắt đầu là 700h tới vùng nhớ Buffer2 có địa chỉ bắt đầu là 1000h. Biết rằng cả hai vùng nhớ đều nằm trong mảng do thanh ghi DS quản lý và kích thước 2 vùng nhớ bằng nhau và bẳng 256 byte.

Xây dựng chương trình assembly (cont)

Bước1: Xây dựng lưu đồ thuật toán.



Xây dựng chương trình assembly (cont)

***Bước2:* Xây dựng chương trình nguồn.**

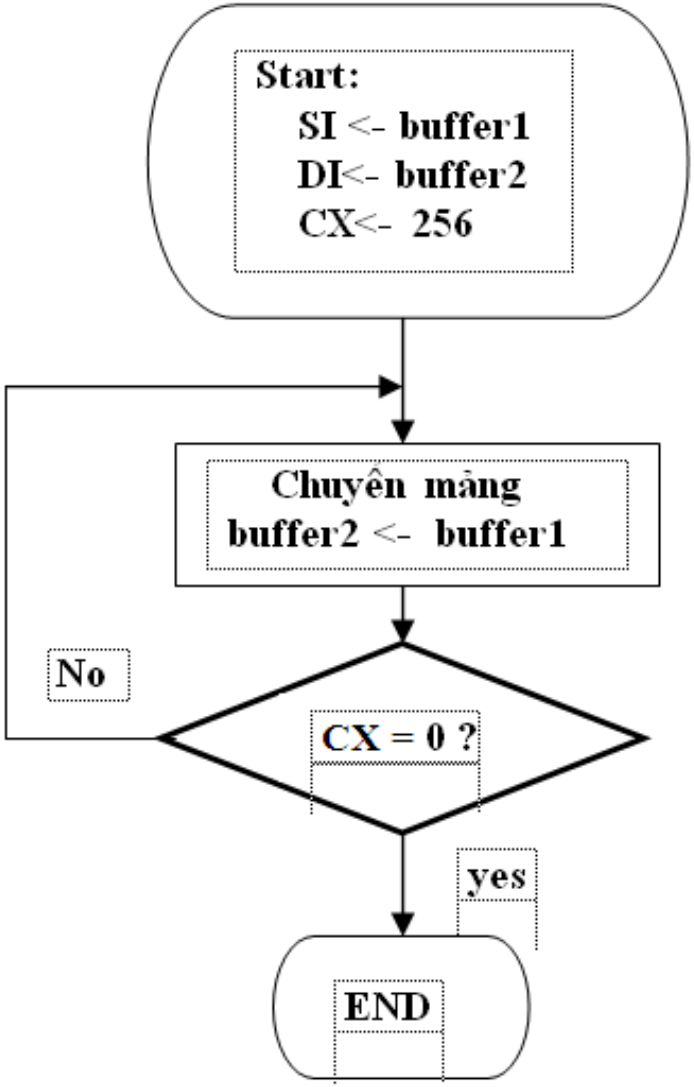
```
Code_Seg          SEGMENT; mở mảng lệnh
ASSUME CS: Code_Seg
          ORG 100h;  Tạo file dạng COM
    ThuTucChinh PROC; thủ tục chính
          CALL Chuyen_Mang; gọi chương trình con Chuyển_Mảng
  ThuTucChinh EndP;
  Chuyen_mang PROC ; thủ tục được đặt tên là chuyển_mảng
MOV SI, 700h; SI trỏ tới offset của Buffer1
MOV DI, 1000h; SI trỏ tới offset của Buffer2
MOV CX, 256; CX là bộ đếm 256 byte cần chuyển
 Loop256:
          MOV AL,[SI]; chuyển nội dung  ngăn nhớ do SI trỏ tới vào AL
          MOV [DI],AL; chuyển nội dung AL vào ngăn nhớ do DI trỏ tới
          INC SI;
          INC DI;
          LOOP Loop256; lặp lại 256 lần
 Chuyen_mang EndP
Code_Seg EndS ; đóng mảng lệnh
END ThuTucChinh
```

Xây dựng chương trình assembly (cont)

- Ví dụ2: Viết chương trình chuyển mảng dữ liệu từ vùng nhớ DISPLAY_VIDEO (địa chỉ 700h) có địa chỉ mảng là DISPLAY_BASE tới vùng dữ lệu BUFFER_SAVE có địa chỉ mảng do DS quản lý. Biết rằng vùng nhớ DISPLAY_VIDEO chứa 1 Kb ký tự (mã hoá ASCII).

Xây dựng chương trình assembly (cont)

Bước1: Xây dựng lưu đồ thuật toán.



Xây dựng chương trình assembly (cont)

Bước2: **Xây dựng chương trình nguồn.**

```
Code_Seg        SEGMENT; mở mảng lệnh
ASSUME CS: Code_Seg, DS:Data_Seg
  ORG 100h;                          Tạo file dạng COM
;-----------------------------------------
;--------Định nghĩa các hằng số----------
;-----------------------------------------
        DISPLAY_BASE  EQU 0B800h
        DISPLAY_VIDEO EQU  700h


;-----------------------------------------
    ThuTucChinh PROC              ; thủ tục chính
            CALL Chuyen_Mang       ; gọi chương trình con Chuyển_Mảng
  ThuTucChinh EndP;
;-----------------------------------------
```

Xây dựng chương trình assembly (cont)

***Bước2:* Xây dựng chương trình nguồn.**

```
Chuyen_Mang PROC NEAR
PUSH AX
            PUSH BX
            PUSH CX
            MOV SI, DISPLAY_VIDEO; SI trỏ tới vùng chứa ký tự cần chuyển
            LEA DI,BUFFER_SAVE; DI trỏ tới vùng sẽ chuyển ký tự tới
 MOV AX,DISPLAY_BASE
 MOV DS,AX ;            DS:DI trỏ tới vùng chứa ký tự cần chuyển
 MOV AX,CS
MOV ES,AX;            ES:SI trỏ tới vùng sẽ chuyển ký tự tới CLD
MOV CX, 1024  ; Bộ đếm 1024
   LOOP_1Kb:
MOVSB       ; dùng lệnh chuyển xâu ký tự kiểu byte
    LOOP LOOP_1Kb ; lặp lại cho tới hết 1024 byte
POP CX
            POP BX
            POP AX
            RET
Chuyen_Mang ENDP
```

Xây dựng chương trình assembly (cont)

***Bước2:* Xây dựng chương trình nguồn.**

```
;-----------------------------------------
Code_Seg EndS                          ; đóng mảng lệnh
Data_Seg SEGMENT                       ; mở mảng dữ liệu
  BUFFER_SAVE  DB 1024 DUP(?); khai báo Buffer để chứa dữ liệu
Data_Seg EndS                          ; đóng mảng dữ liệu
;-----------------------------------------
END ThuTucChinh
```

Điểm chú ý ở đây là hai vùng nhớ cần chuyển dứ liệu nằm ở hai mảng khác nhau nên mối vùng phải sử dụng con trỏ đầy đủ (seg:offset). Mặt khác, ta muốn dùng lệnh MOVSB thay vì lệnh MOV để chuyển ký tự nên phải chuẩn bị trước cặp DS:SI cho trỏ tới vùng dữ liệu nguồn DISPLAY_VIDEO còn cặp ES:SI cho trỏ tới vùng dữ liệu đích BUFFER_SAVE.