ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

KIẾN TRÚC MÁY TÍNH

ASSEMBLY PROGRAMMING

Mục tiêu:

- Nắm được tổng quan đặc tính ngôn ngữ lập trình Assembly;
- Nắm được cấu trúc chương trình Assemly và các câu lệnh căn bản;
- Lập trình được các bài toán cơ bản.

Chương 6. GIỚI THIỆU HỢP NGỮ

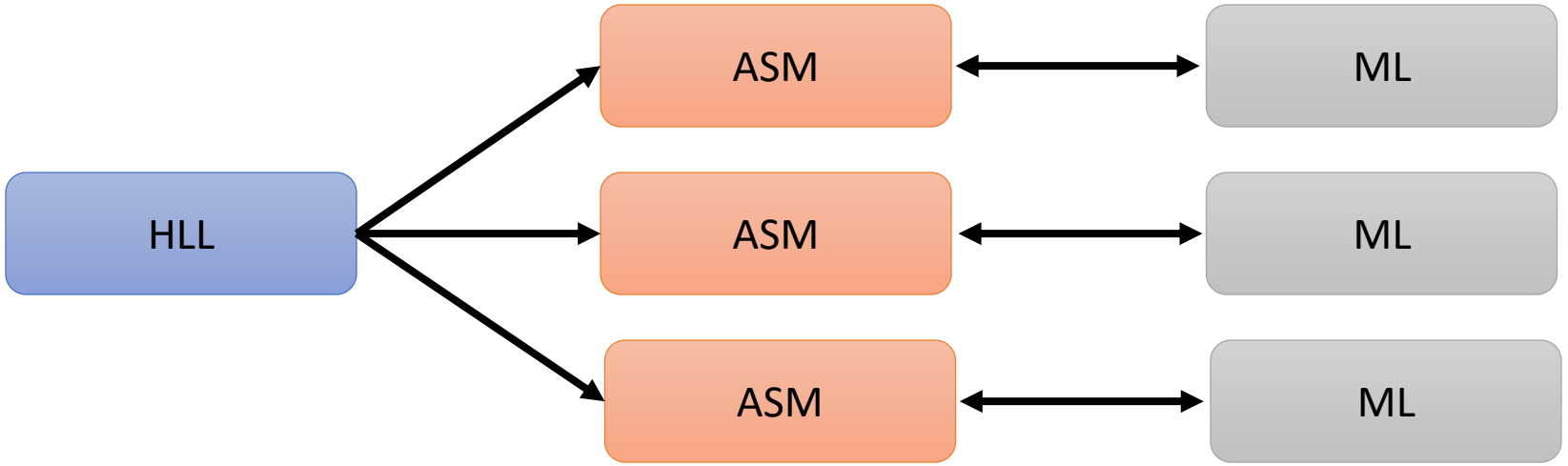
- 6.1 Khái niệm
- 6.2 Cấu trúc chương trình
- 6.3 Khai báo biến
- 6.4 Một số nhóm lệnh cơ bản

6.1 Khái niệm

- **Ngôn ngữ máy (ML - Machine Language)**
  - CPU trực tiếp “hiểu” và thực thi được
  - Dạng nhị phân
  - Độ dài lệnh do CPU qui định
- **Hợp ngữ (Assembly Language)**
  - Giúp lập trình viên dễ viết hơn ngôn ngữ máy
  - Thay thế lệnh dạng nhị phân bằng ký hiệu tượng trưng

- **Ngôn ngữ cấp cao (High Level Language)**
  - Gần gũi với con người
- ASM và HLL phải dịch về ML để thực hiện

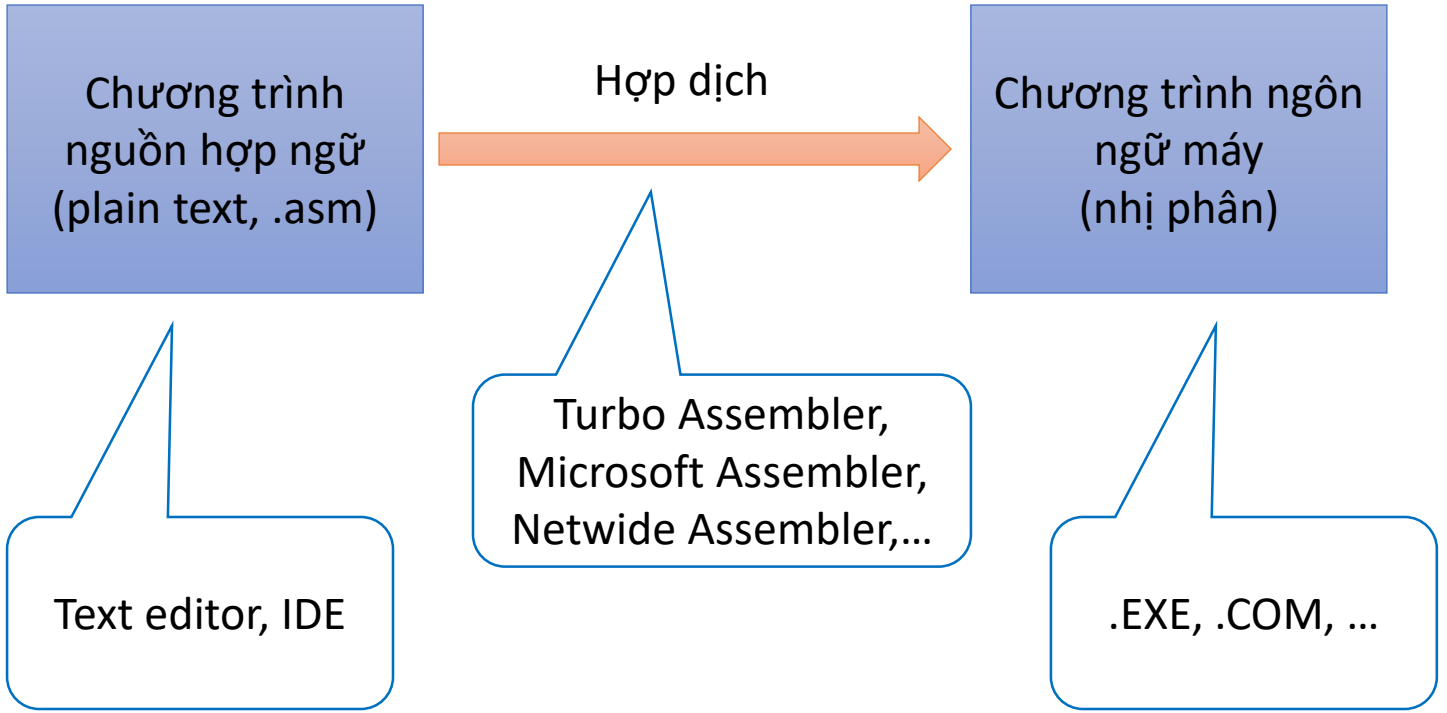
Tương đương logic giữa các ngôn ngữ



Tại **sao** phải lập trình hợp ngữ?

- Chương trình nhanh, nhỏ gọn
- Khai thác trực tiếp thiết bị
- Làm việc ở cấp độ thấp
  - Thiết kế máy tính
  - Viết trình dịch (compiler), trình điều khiển thiết bị (driver)
- Lập trình hệ thống nhúng (embedded system)
  - Tài nguyên hạn chế
  - Tối ưu phần mềm

Viết và chạy chương trình hợp ngữ



Assembler

- Một chương trình viết bằng ngôn ngữ Assembly muốn máy tính thực hiện được ta phải chuyển thành ngôn ngữ máy.
- Chương trình dùng để dịch 1 file viết bằng Assembly ➔ ngôn ngữ máy, gọi là Assembler.
- 2 chương trình dịch thông dụng:

MASM và TASM

Lệnh máy

- Là 1 chuỗi nhị phân có ý nghĩa đặc biệt – nó ra lệnh cho CPU thực hiện tác vụ.
- Tác vụ đó có thể là :
  di chuyển 1 số từ vị trí nhớ này sang vị trí nhớ khác.
  Cộng 2 số hay so sánh 2 số.

0 0 0 0 0 1 0 0	Add a number to the AL register
1 0 0 0 0 1 0 1	Add a number to a variable
1 0 1 0 0 0 1 1	Move the AX reg to another reg

Lệnh máy (2)

- Tập lệnh máy được định nghĩa trước, khi CPU được sản xuất và nó đặc trưng cho kiểu CPU .
- Ex : B5 05 là 1 lệnh máy viết dạng số hex, dài 2 byte.
- Byte đầu B5 gọi là Opcode
- Byte sau 05 gọi là toán hạng Operand
  - Ý nghĩa của lệnh B5 05 : chép giá trị 5 vào reg AL

Biểu diễn dữ liệu trong bộ nhớ

- Byte (8 bit)
  - Giá trị: 0..255 hoặc -128..127
- Word (16 bit)
  - Giá trị: 0..65535 hoặc -32768..32767
  - Lưu trữ trong bộ nhớ: byte cao lưu địa chỉ cao, byte thấp lưu địa chỉ thấp
  - Ví dụ: $(20400)_{10} = (4FB0)_{16}$

Địa chỉ	Giá trị ô nhớ
i	B0
i + 1	4F

Biểu diễn dữ liệu trong bộ nhớ (2)

- Ký tự
  - Cách viết: ‘a’, “a”
  - Lưu trong bộ nhớ dạng mã ASCII 8 bit
  - Ví dụ: ‘a’ →

97

- Chuỗi
  - Cách viết: ‘abc’, “abc”
  - Lưu trong bộ nhớ dạng dãy ký tự
  - Ví dụ: “abc”

  →

97	98	99

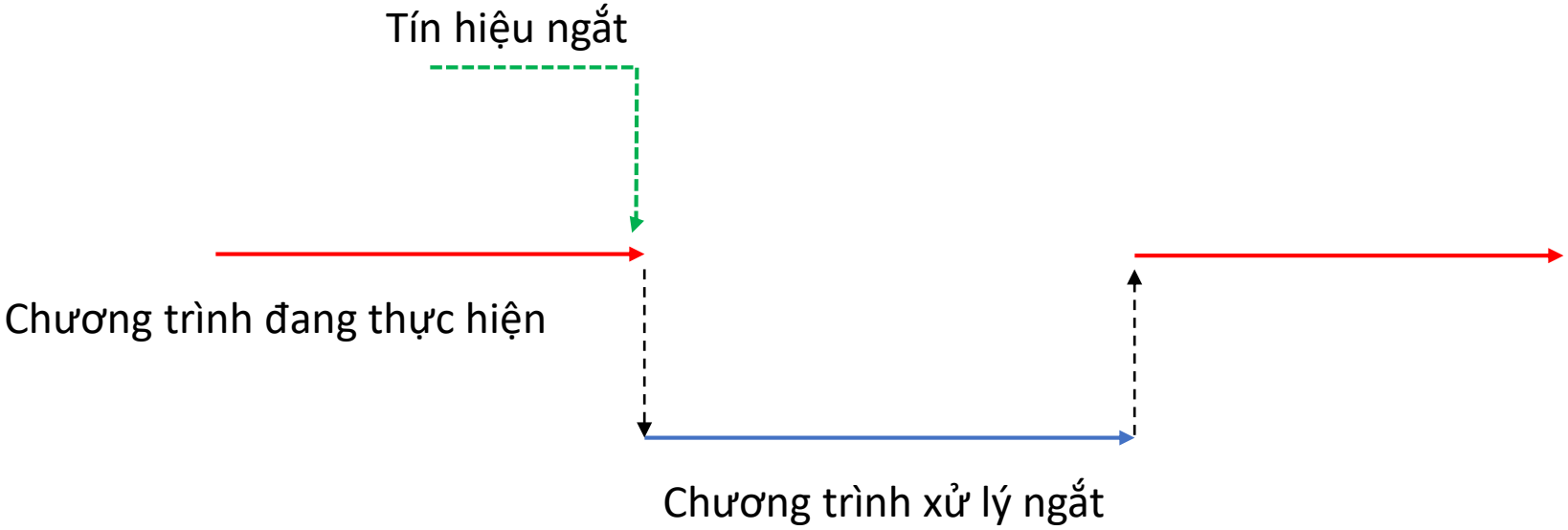
Biểu diễn dữ liệu trong bộ nhớ (3)

- Số BCD
  - Dùng 8 bit hoặc 4 bit biểu diễn cho 1 ký số hệ 10
  - Ví dụ: $(15)_{10}$ → $(0001\ 0101)_{BCD\text{-}4bit}$
    hoặc $(00000001\ 00000101)_{BCD\text{-}8bit}$
- Số nguyên có dấu biểu diễn theo dạng bù 2
- Qui ước

Hệ	Cách viết
10	255, 255d, 255D
2	10011b, 10011B
16	3Fh, 3FH, 0C8h

Ngắt (Interrupt)

- Là tín hiệu gởi đến CPU để tạm ngưng chương trình đang hoạt động để thực hiện một nhiệm vụ khác, sau khi thực hiện xong, chương trình bị tạm ngưng sẽ được tiếp tục thực hiện.



Ngắt (2)

- Ngắt cứng (Hard Interrupt)
  - Do thiết bị phần cứng phát sinh để đáp ứng sự kiện nào đó (nhấn phím, di chuyển chuột, …)
  - Dùng điều khiển các thiết bị nhập xuất
  - Các kênh ngắt cứng của CPU ký hiệu IRQ0, IRQ1, …
- Ngắt không chắn được (NMI – Non Maskable Interrupt)
  - Không chắn được bằng phần mềm, độ ưu tiên cao nhất
  - Báo hiệu sự cố (hư hỏng bộ nhớ, điện áp, …)

Ngắt (3)

- Ngắt nội bộ (Internal Interrupt)
  - Do CPU tự phát sinh
  - Divide by zero: khi CPU thực hiện chia một số cho 0
  - Overflow: khi tính toán bị tràn số
  - Trap: phát sinh sau khi thực hiện một lệnh (TF = 1)
- Ngắt mềm (Soft Interrupt)
  - Do chương trình đang thực hiện tạo ra
  - Dùng gọi một chương trình con trong ROM/RAM
  - Tạo ra bằng lệnh **INT <số hiệu ngắt>**

Ngắt (4)

- Bảng véc-tơ ngắt (Interrupt Vector Table)
  - Lưu trữ địa chỉ của các chương trình xử lý ngắt (Interrupt Handler)
  - Gồm 256 phần tử, mỗi phần tử 4 byte, chứa địa chỉ segment và offset của chương trình xử lý ngắt tương ứng.
  - Khi có tín hiệu ngắt, CPU lấy số hiệu ngắt tra trong IVT để tìm địa chỉ chương trình xử lý ngắt. Sau đó sẽ nạp vào CS:IP để chuyển đến thực hiện.
  - Địa chỉ IVT trong bộ nhớ: **0000:0000 .. 0000:03FF**

Cách viết lệnh hợp ngữ

- Cú pháp

  [Nhãn:] <Tên lệnh> [toán hạng] [;Ghi chú]

  [Nhãn:] : đánh dấu vị trí của lệnh trong bộ nhớ

  <Tên lệnh> : từ gợi nhớ của lệnh

  [toán hạng] : các toán hạng mà lệnh yêu cầu

  [;Ghi chú] : giải thích cho lệnh

  Chú ý: Mỗi lệnh viết trên một dòng

- Ví dụ: L1: MOV AX, BX ; AX = BX

Cách đặt tên, nhãn

- Bao gồm
  - Chữ cái: a..z, A..Z
  - Chữ số: 0..9
  - Ký tự đặc biệt: . ? @ _ $
- Phải bắt đầu bằng chữ cái hoặc ký tự đặc biệt.
- Dấu chấm không được dùng trong tên
- Hợp ngữ không phân biệt chữ hoa/thường

Khai báo dữ liệu

- Cú pháp

[tên] DB | DW <danh sách trị>

[tên] : đánh dấu địa chỉ offset bắt đầu của dữ liệu trong bộ nhớ

DB : khai báo dữ liệu dạng Byte (8 bit)

DW : khai báo dữ liệu dạng Word (16 bit)

<danh sách trị> : liệt kê các giá trị ban đầu của từng phần tử trong vùng nhớ

Ghi chú: Dữ liệu trong đoạn nếu khai báo liên tiếp nhau sẽ cấp phát liên tục trong bộ nhớ.

Khai báo dữ liệu (2)

- Ví dụ

  B1 db 10

  W1 dw 4906

  B2 db 32, 60

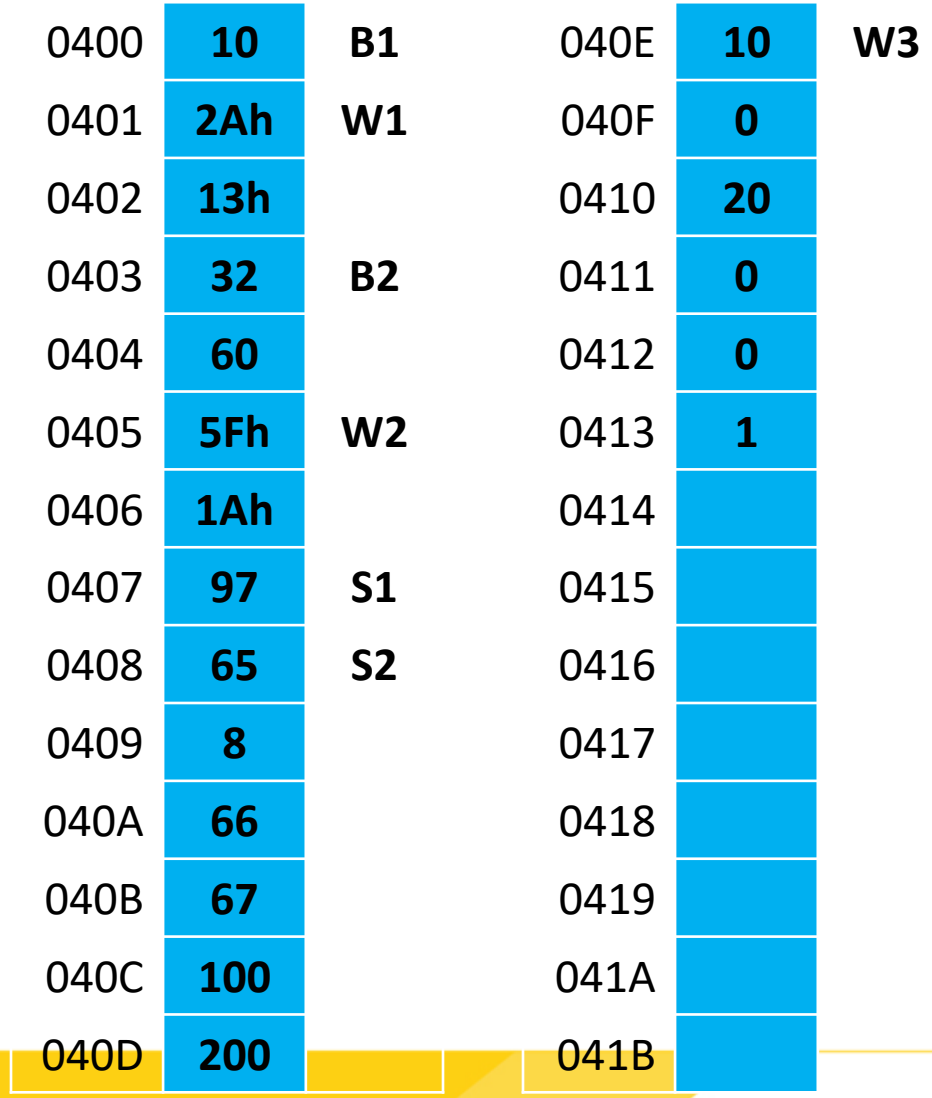
  W2 dw 1A5Fh

  S1 db “a”

  S2 db “A”, 8, “BC”

  db 100, 200

  W3 dw 10, 20, 256



Khai báo dữ liệu (3)

- Sử dụng ký tự ? để khai báo giá trị bất kỳ
- Sử dụng toán tử DUP để khai báo nhiều phần tử có giá trị giống nhau

  count DUP(exp)

→ Khai báo count lần giá trị exp

- Ví dụ

  S1 db 5 DUP(10) → 10, 10, 10, 10, 10

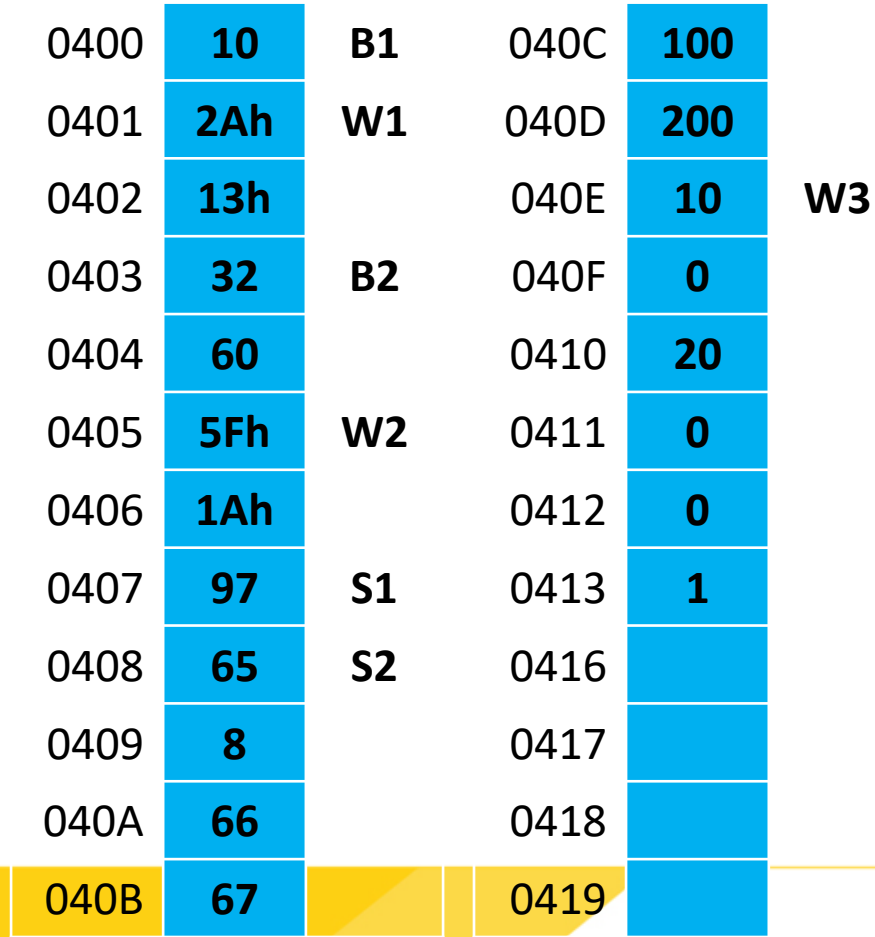
  S2 db 3 DUP(2 DUP ('a')) → 'a', 'a', 'a', 'a', 'a', 'a'

  B1 db ? → B1 nhận giá trị bất kỳ có sẵn trong bộ nhớ

Khai báo dữ liệu (4)

- Truy xuất giá trị trong bộ nhớ: tùy vào kiểu của vùng nhớ
- Ví dụ

[0401h] = 2Ah (8 bit)
[B1] = 10 (8 bit)
[W1] = 132Ah (16 bit)
[B1 + 2] = 13h (8 bit)
[W1 + 1] = 3C20h (16 bit)
[S2 + 6] = 10 (8 bit)



6.2 Cấu trúc chương trình

- **Tập tin dạng COM**
  - Chỉ có duy nhất 1 đoạn (segment)
  - Kích thước tối đa: 64KB - 256 - 2
  - Sử dụng chương trình con dạng gần (cùng đoạn)
  - Áp dụng cho các chương trình nhỏ
  - Khi nạp:
    - Định vị vùng nhớ trống
    - Tạo PSP (Program Segment Prefix) kích thước 256 byte ở đầu vùng nhớ
    - Nạp chương trình từ vị trí 100h
    - PUSH 0 vào Stack

Cấu trúc chương trình (2)

- **Tập tin dạng EXE**
  - Có nhiều đoạn
  - Kích thước tùy ý
  - Sử dụng chương trình con dạng gần và xa
  - Áp dụng cho các chương trình lớn (>64KB)
  - Có header ở đầu tập tin để điều khiển nạp chương trình
  - Kích thước header là bội số của 512

Cấu trúc chương trình (3)

COM	EXE

.MODEL TINY
.CODE
ORG 100h
Begin:
;
; Các lệnh của chương trình
;
 	
INT 20h
 	
; Khai báo chương trình con
; Khai báo dữ liệu
 	
END Begin
	.MODEL SMALL
	.STACK 200h
	.DATA
	; Khai báo dữ liệu
	.CODE
	Begin:
	MOV AX, @DATA
	MOV DS, AX
	;
	; Các lệnh của chương trình
	;
	MOV AX, 4C00H
	INT 21H
	
	; Khai báo chương trình con
	
	END Begin

Khung chương trình hợp ngữ (EXE)

```
.MODEL Small  ; kiểu bộ nhớ
.STACK  100    ; kích thước
.DATA
        ; khai báo biến và hằng
.CODE
  MAIN PROC
     ; khởi đầu cho DOS
      MOV  AX, @DATA
      MOV  DS,AX
          các lệnh chương trình chính
     MOV AH,4CH  ; thoát khỏi chương trình
     INT 21H
  MAIN ENDP
     các chương trình con (nếu có) để ở tại đây
END MAIN
```

Khai báo quy mô sử dụng bộ nhớ

Kiểu	Mô tả
TINY	Mã lệnh và dữ liệu gói gọn trong một đoạn
SMALL	Mã lệnh trong 1 đoạn. Dữ liệu trong 1 đoạn
MEDIUM	Mã lệnh nhiều hơn 1 đoạn. Dữ liệu trong 1 đoạn
COMPACT	Mã lệnh trong 1 đoạn. Dữ liệu trong nhiều hơn 1 đoạn
LARGE	Mã lệnh nhiều hơn 1 đoạn. Dữ liệu nhiều hơn 1 đoạn, không có mảng nào lớn hơn 64K
HUGE	Mã lệnh nhiều hơn 1 đoạn. Dữ liệu nhiều hơn 1 đoạn, các mảng có thể lớn hơn 64K

Khai báo đoạn ngăn xếp

- Khai báo vùng nhớ dùng làm ngăn xếp phục vụ cho chương trình
  - Stack kích thước
- Nếu không khai báo thì mặc định là 1KB

Khai báo đoạn dữ liệu

- Dùng định nghĩa các biến và hằng
- Ví dụ

```
.DATA
        MSG   DB  ‘Hello!$’
        CR      DB  13
        LF      EQU 10
```

Khai báo đoạn mã

```
.Code
        Main Proc
          ; Các lệnh thân chương trình chính
           Call ten_chuong_trinh_con  ;goi ctc
         Main Endp
        Ten_chuong_trinh_con  Proc
          ; Các lệnh thân chương trình con
          RET
        Ten_chuong_trinh_con Endp
END MAIN
```

INT 21H

- Lệnh INT số hiệu ngắt được dùng để gọi chương trình ngắt của DOS và BIOS.
- Muốn sử dụng hàm nào của INT 21h ta đặt function_number vào thanh ghi AH, sau đó gọi INT 21h

AH=1: nhập 1 ký tự từ bàn phím

AH=2: xuất 1 ký tự ra mà hình.

AH=9: xuất 1 chuỗi ký tự ra màn hình

AH=4CH: kết thúc chương trình .EXE

INT 21h

- Hàm 1 : Nhập 1 ký tự

  Input : AH =1

  Output : AL = mã ASCCI của phím ấn

  = 0 nếu 1 phím điều khiển được ấn

- Hàm 2 : Hiển thị 1 ký tự ra màn hình

  Input : AH =2

  DL = Mã ASCII của ký tự hiển thị hay ký tự điều khiển

6.3.KHAI BÁO BIẾN TRONG HỢP NGỮ

Khai báo biến

- **Cú pháp : [tên biến] DB | DW |.... [trị khởi tạo]**
- **Là một tên ký hiệu dành riêng cho 1 vị trí trong bộ nhớ nơi lưu trữ dữ liệu.**
- **Offset của biến là khoảng cách từ đầu phân đoạn đến biến đó.**
- **VD : khai báo 1 danh sách aList ở địa chỉ 100 với nội dung sau :**

```
.data
aList  db “ABCD”
```

Khai báo biến

Từ gợi nhớ	Mô tả	Số byte	Thuộc tính
DB	Định nghĩa byte	1	Byte
DW	Từ	2	Word
DD	Từ kép	4	Doubleword
DQ	Từ tứ	8	Quardword
DT	10 bytes	10	tenbyte

Minh họa khai báo biến

- Char db ‘A’
- Num db 41h
- Mes db “Hello Word”,’$’
- Array_1 db 10, 32, 41h, 00100101b
- Array_2 db 2,3,4,6,9
- Myvar db ? ; Biến không khởi tạo
- Btable db 1,2,3,4,5
  db 6,7,8,9,10

Toán tử DUP

Lặp lại 1 hay nhiều giá trị khởi tạo.

- VD :

```
Bmem   DB  50 Dup(?)  ; khai báo vùng nhớ gồm 50 bytes.
db  4 dup (“ABC”)  ;12 bytes “ABCABCABCABC”
db 4096 dup (0) ; Vùng đệm 4096 bytes tất cả bằng 0
```

Khởi tạo biến

- Lưu ý : Khi khởi tạo trị là 1 số hex thì giá trị số luôn luôn bắt đầu bằng 1 ký số từ 0 đến 9. Nếu ký số bắt đầu là A.. F thì phải thêm số 0 ở đầu.
- VD :

```
Db  A6H   ; sai
Db  0A6h  ; đúng
```

6.4. MỘT SỐ NHÓM LỆNH CĂN BẢN

- Nhóm lệnh chuyển dữ liệu
- Nhóm lệnh chuyển địa chỉ
- Nhóm lệnh chuyển cờ hiệu
- Nhóm lệnh chuyển dữ liệu qua cổng
- Lệnh so sánh, vòng lặp
- Nhóm lệnh xử lý số học.

Hoạt động của VXL đối với từng loại hành động

Truyền dữ liệu	Truyền dữ liệu từ vị trí này sang vị trí khác
	Nếu có liên quan đến bộ nhớ: Xác định địa chỉ bộ nhớ Thực hiện chuyển đổi địa chỉ bộ nhớ từ địa chỉ bộ nhớ ảo sang thực Kiểm tra bộ nhớ cache Khởi tạo lệnh đọc/ghi bộ nhớ
Số học	Có thể liên quan đến việc truyền dữ liệu (trước và/hoặc sau truyền dữ liệu)
	Thực hiện hàm trong ALU
	Thiết lập mã điều kiện và cờ
Logic	Tương tự như số học
Truyền điều khiển	Cập nhật PC. Đối với lệnh gọi/trả về chương trình con (Call/Return), quản lý việc liên kết và truyền tham số
Điều khiển vào/ra	Ra mệnh lệnh cho mô-đun I/O
	Nếu sử dụng I/O ánh xạ bộ nhớ, xác định địa chỉ ánh xạ bộ nhớ

6.4.1.Các lệnh chuyển dữ liệu

MOVE	Copy dữ liệu từ nguồn đến đích
LOAD	Nạp dữ liệu từ bộ nhớ đến bộ xử lý
STORE	Cất dữ liệu từ bộ xử lý đến bộ nhớ
EXCHANGE	Trao đổi nội dung của nguồn và đích
CLEAR	Chuyển các bit 0 vào toán hạng đích
SET	Chuyển các bit 1 vào toán hạng đích
PUSH	Cất nội dung toán hạng nguồn vào ngăn xếp
POP	Lấy nội dung đỉnh ngăn xếp đưa đến toán hạng đích

Lệnh MOV (Move)

- Cú pháp

  MOV *dest, source*

- Ý nghĩa
  - Gán giá trị của toán hạng source cho dest
  - dest có thể là Reg, Mem
  - source có thể là Reg, Mem hoặc Immed
  - dest và source phải cùng kích thước (8/16 bit)
- Ví dụ

  MOV AX, BX

  MOV AL, DS:[100h]

Lệnh MOV (2)

- Chú ý
  - Lệnh MOV không ảnh hưởng thanh ghi cờ hiệu
  - Không thể chuyển hằng trực tiếp vào thanh ghi đoạn
  - Không thể chuyển dữ liệu trực tiếp giữa hai thanh ghi đoạn
- Ví dụ

  MOV DS, 0B800h → Sai

  MOV DS, ES → Sai

  → *Sử dụng thanh ghi trung gian, thường là thanh ghi đa dụng*

Lệnh XCHG

- Cú pháp

  XCHG *dest, source*

- Ý nghĩa
  - Hoán đổi giá trị của toán hạng source với dest
  - dest, source có thể là Reg, Mem
  - dest và source phải cùng kích thước (8/16 bit)
  - Không được hoán đổi 2 thanh ghi đoạn với nhau
- Ví dụ

  XCHG AX, BX

  XCHG AL, DS:[100h]

Lệnh PUSH

- Cú pháp

  PUSH *source*

- Ý nghĩa
  - Thêm một phần tử có giá trị là source vào stack
  - source có thể là Reg16, Mem16
  - SP sẽ giảm đi 2 để trỏ đến offset của phần tử mới
- Ví dụ

  PUSH AX

  PUSH [BX + SI]

Lệnh POP

- Cú pháp

  POP *dest*

- Ý nghĩa
  - Lấy phần tử trên đỉnh stack đưa vào dest
  - dest có thể là Reg16, Mem16
  - SP tăng 2 đơn vị do lấy đi bớt 1 phần tử
- Ví dụ

  POP BX

  POP [BX + SI]

6.4. MỘT SỐ NHÓM LỆNH CĂN BẢN

6.4.2.Nhóm lệnh chuyển địa chỉ

- LEA
- LDS, LES

Lệnh LEA

- Cú pháp

  LEA *Reg16, Mem*

- Ý nghĩa
  - Lấy địa chỉ offset của toán hạng bộ nhớ Mem lưu vào Reg16
- Ví dụ

```
LEA   BX, B1                ; BX = địa chỉ offset của B1
LEA   SI, DS:[0800h]        ; SI = 0800h
```

6.4. MỘT SỐ NHÓM LỆNH CĂN BẢN

- 6.4.3.Nhóm lệnh chuyển cờ hiệu

  LAHF, SAHF, PUSHF, POPF

6.4. MỘT SỐ NHÓM LỆNH CĂN BẢN

❑6.4.4.Nhóm lệnh chuyển dữ liệu qua cổng

- INPUT Copy dữ liệu từ một cổng xác định đưa đến đích
- OUTPUT Copy dữ liệu từ nguồn đến một cổng xác định

Truyền chương trình và dữ liệu vào bộ nhớ và đưa kết quả tính toán ngược lại cho người sử dụng.

- IN AL/AX, địa chỉ cổng
- OUT địa chỉ cổng, AL/AX

6.4. MỘT SỐ NHÓM LỆNH CĂN BẢN

❑6.4.5.Lệnh so sánh, vòng lặp

- Lệnh nhảy không điều kiện: JMP
- Lệnh so sánh: CMP
- Lệnh nhảy có điều kiện: JB, JNAE, JBE, JNA, JA, JNBE, JAE, JNB, JE, JZ, JNE, JNZ, JL, JNGE, JLE, JNG, JG, JNLE, JGE, JNL, JP, JNP, JS, JNS, JO, JNO, JC, JNC, JCXZ
- Lệnh lặp: LOOP, LOOPE, LOOPZ, LOOPNE, LOOPNZ
- Lệnh gọi chương trình con: CALL, RET, RETN, RETF

- Vì sao cần hành động chuyển điều khiển?
  - Cần thiết để có thể thực thi 1 lệnh hơn một lần, thậm chí nhiều nghìn lần
  - Hầu như mọi chương trình đều có sự ra quyết định
  - Cần có cơ chế phân tách các tác vụ ra thành các công việc nhỏ hơn để có thể thực hiện từng công việc một.
- Các hành động truyền điều khiển:
  - Rẽ nhánh
  - Lệnh nhảy – skip
  - Gọi hàm

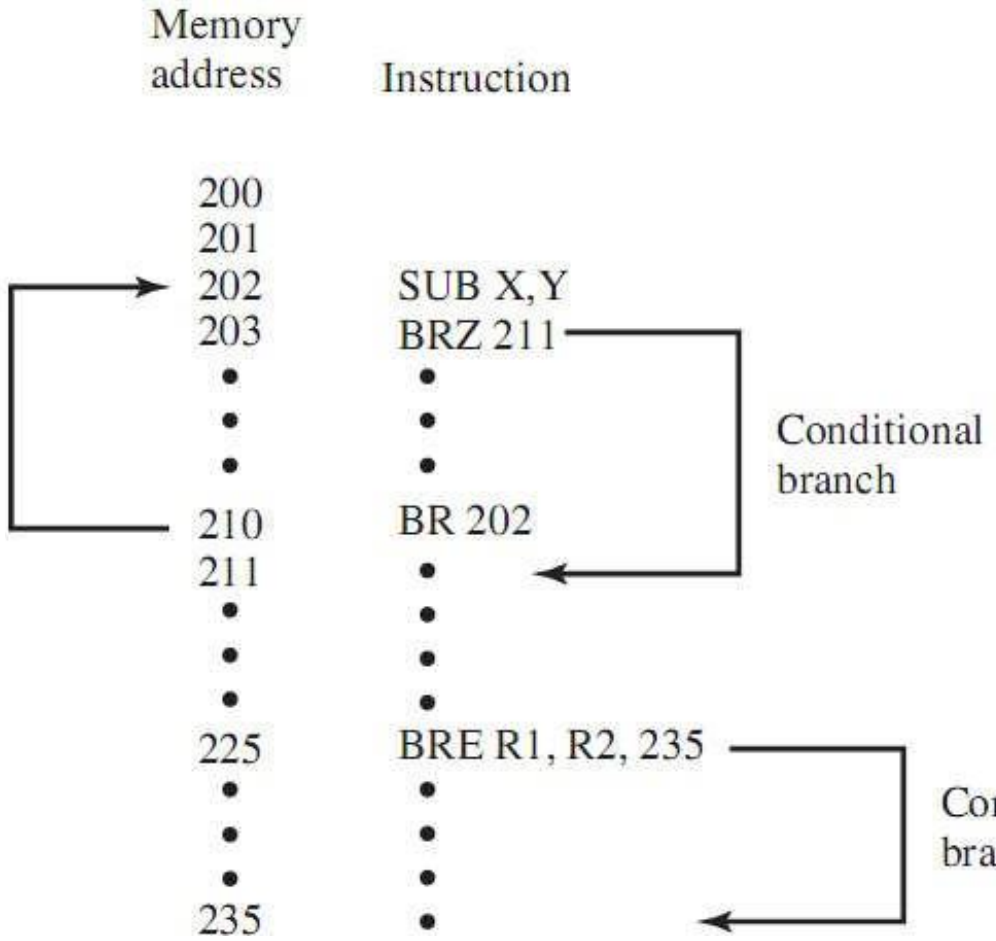
Các lệnh chuyển điều khiển

- JUMP (BRANCH) -Lệnh nhảy không điều kiện (nạp vào PC một địa chỉ xác định)
- JUMP CONDITIONAL - Lệnh nhảy có điều kiện:
  - điều kiện đúng → nạp vào PC một địa chỉ xác định
  - điều kiện sai → không làm gì cả
- CALL - Lệnh gọi chương trình con:
  - Cất nội dung của PC (địa chỉ trở về) ra một vị trí xác định (thường ở Stack)
  - Nạp vào PC địa chỉ của lệnh đầu tiên của chương trình con
- RETURN - Lệnh trở về từ chương trình con: Khôi phục địa chỉ trở về trả lại cho PC để trở về chương trình chính

Lệnh rẽ nhánh

Rẽ nhánh không điều kiện

Chuyển tới thực hiện lệnh ở vị trí có địa chỉ XX:
PC ← XX



Rẽ nhánh có điều kiện

Kiểm tra điều kiện trong lệnh:

- Nếu đúng → chuyển tới thực hiện lệnh ở vị trí có địa chỉ XX
  PC ← XXX
- Nếu sai → chuyển sang thực hiện lệnh kế tiếp

Lệnh skip

- Chứa địa chỉ ngầm định
- Lệnh nhảy ngầm định rằng 1 lệnh sẽ bị bỏ qua

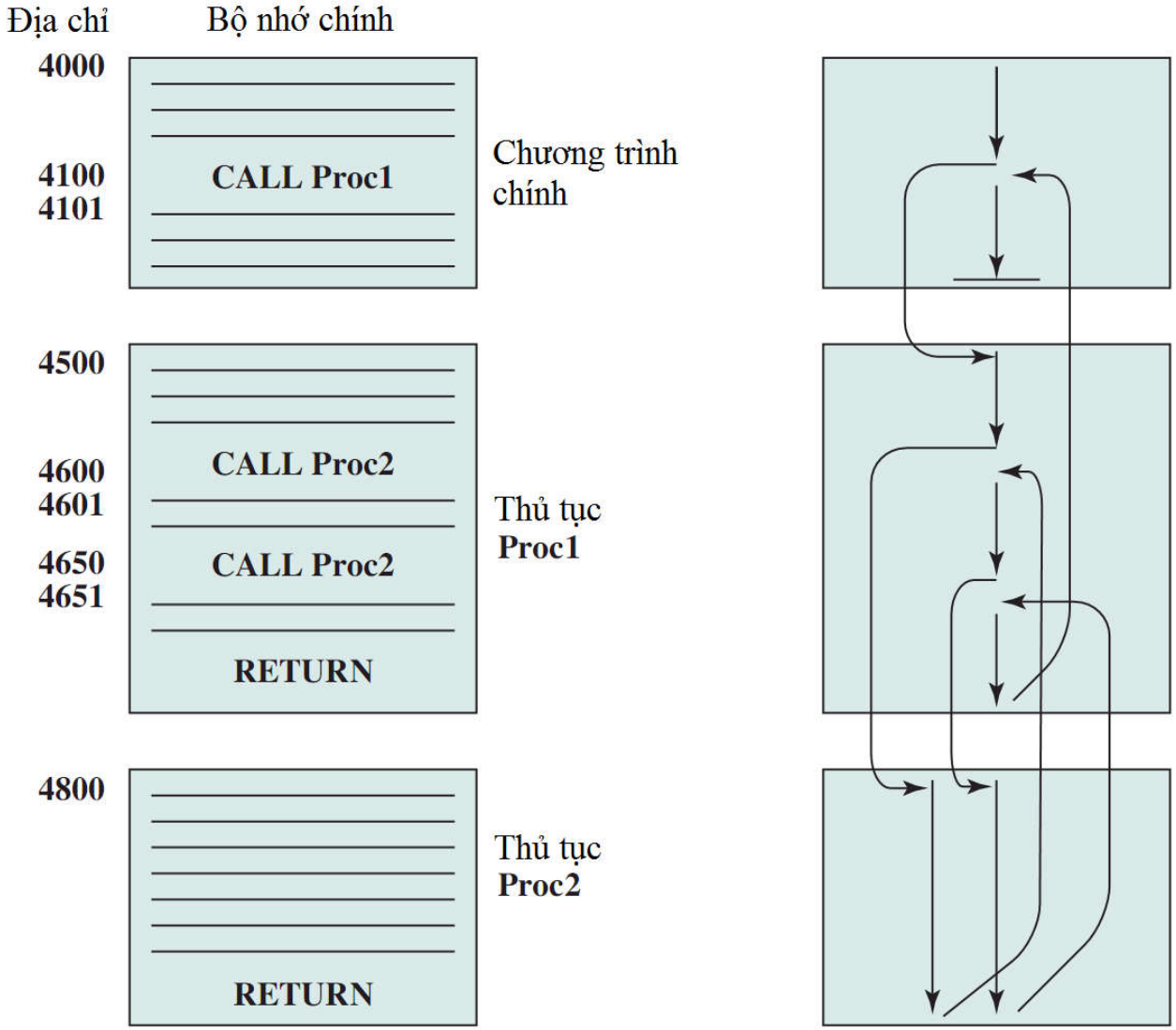
 Địa chỉ ngầm định là địa chỉ của lệnh tiếp theo cộng với chiều dài 1 lệnh

- Không cần trường địa chỉ đích
- Ví dụ: lệnh ISZ - increment-and-skip-if-zero

Lệnh CALL và RETURN

- Chương trình con (CTC) là chương trình máy tính khép kín được kết hợp để tạo thành một chương trình lớn hơn
  - Tại bất kỳ điểm nào trong chương trình, CTC có thể được gọi
  - VXL đi đến và thực thi toàn bộ CTC rồi quay lại điểm mà lệnh CALL được tạo ra
- Lý do chính sử dụng CTC:
  - Tính kinh tế: một thủ tục có thể được sử dụng nhiều lần
  - Tính modul: tác vụ lớn chia thành nhiều công việc nhỏ hơn
- Gồm hai lệnh cơ bản:
  - Lệnh CALL: rẽ nhánh từ vị trí hiện tại đến CTC
  - Lệnh RETURN: trả từ CTC về vị trí mà nó được gọi

Các thủ tục lồng nhau



(a) Lệnh Call và Return

(b) Chuỗi thực thi

Sử dụng ngăn xếp để thực hiện các thủ tục lồng nhau

(a) Nội dung ban đầu của ngăn xếp

(b) Sau lệnh **CALL Proc1**

(c) Bắt đầu **CALL Proc2**

(d) Sau lệnh **RETURN**

(e) Sau lệnh **CALL Proc2**

(f) Sau lệnh **RETURN**

(g) Sau lệnh **RETURN**

Lệnh JMP (Jump)

- Cú pháp

  JMP *label* *hay* JMP *target*

- Ý nghĩa:
  - Chuyển điều khiển chương trình từ vị trí này đến vị trí khác
  - Toán hạng target có thể là Reg hay Mem
  - Lệnh JMP có thể short, near hay far
  - Làm thay đổi nội dung CS:IP hay chỉ IP

Lệnh CMP (Compare)

- Cú pháp

  CMP *left, right*

- Ý nghĩa:
  - So sánh nội dung tóan hạng left và toán hạng right, kết quả so sánh lưu trong thanh ghi cờ hiệu
  - Left là Reg, Mem
  - Right là Reg, Mem, Immed
  - Thường dùng với các lệnh nhảy

Lệnh CMP (2)

- Ví dụ: chương trình so sánh nội dung AX và BX (số không dấu)
  - Nếu Ax > Bx: Ax = Ax – Bx
  - Nếu Ax < Bx: Bx = Bx – Ax
  - Nếu Ax = Bx: Ax = Bx =0

Lệnh CMP (3)

```
CMP AX, BX
JA Label1
JB Label 2
Mov AX, 0
Mov BX, 0
JMP Label3
Label1:
        SUB AX, BX
        JMP Label3
Label2:
        SUB BX,AX
Label3:
……
```

Lệnh LOOP

- Cú pháp

  LOOP short_label

- Ý nghĩa:
  - Giảm CX xuống 1 đơn vị, nếu CX <> 0 thì chuyển đến short_label, ngược lại, các lệnh sau LOOP được thực hiện
  - Số lần thực hiện tối đa bằng (Cx-1)

Lệnh LOOP (2)

- Ví dụ: dung lệnh INC tăng AX lên 5 đơn vị

```
Mov Cx, 5
Label:
        INC Ax
        LOOP Label
```

Lệnh LOOPE (Loop while Equal)

- Cú pháp

  LOOPE short_label

- Ý nghĩa:
  - Giảm CX xuống 1 đơn vị, nếu ZF=1 và CX <> 0 thì chuyển đến short_label, ngược lại, các lệnh sau LOOP được thực hiện
  - Lệnh LOOPZ tương tự LOOPE

Lệnh LOOPNE (Loop while Not Equal)

- Cú pháp

  LOOPNE short_label

- Ý nghĩa:
  - Giảm CX xuống 1 đơn vị, nếu ZF=0 và CX <> 0 thì chuyển đến short_label, ngược lại, các lệnh sau LOOP được thực hiện
  - Lệnh LOOPNZ tương tự LOOPNE

6.4. MỘT SỐ NHÓM LỆNH CĂN BẢN

❑6.4.6.Nhóm lệnh xử lý số học

- Nhóm lệnh tính toán số học
  - ADD, SUB, INC, DEC, MUL, DIV, NEG
  - ADC, AAA, DAA, SBB, AAS, DAS, IMUL, AAM, IDIV, AAD, CBW, CWD

Lệnh ADD

- Cú pháp

  ADD *dest, source*

- Ý nghĩa
  - dest = dest + source
  - dest có thể là Reg, Mem
  - source có thể là Reg, Mem, Immed
- Ví dụ

```
ADD   AX, BX          ; AX = AX + BX
ADD   DX, 200h        ; DX = DX + 200h
```

Lệnh ADD (2)

- Chú ý
  - Không thể cộng trực tiếp 2 thanh ghi đoạn
  - Lệnh ADD ảnh hưởng 6 cờ: AF, CF, OF, PF, SF, ZF
    - AF: bật 1 nếu kết quả phép cộng có nhớ/mượn với 4 bit thấp
    - CF: bật 1 nếu kết quả phép cộng có nhớ/mượn
    - PF: bật 1 nếu kết quả có tổng 8 bit thấp là số chẵn
    - ZF: bật 1 nếu kết quả là 0
    - SF: bật 1 nếu kết quả là số âm (bit cao nhất là 1)
    - OF: bật 1 nếu kết quả là số có dấu bị sai (không đủ chứa số có dấu trong dest)

Lệnh INC

- Cú pháp

  INC *dest*

- Ý nghĩa
  - dest = dest + 1
  - dest có thể là Reg, Mem
- Ví dụ

```
INC   AX          ; AX = AX + 1
INC   W1          ; W1 = W1 + 1
```

Lệnh SUB

- Cú pháp

  SUB *dest, source*

- Ý nghĩa
  - dest = dest - source
  - dest có thể là Reg, Mem
  - source có thể là Reg, Mem, Immed
- Ví dụ

```
SUB   AX, BX          ; AX = AX - BX
SUB   DX, 200h        ; DX = DX - 200h
```

Lệnh SUB (2)

- Chú ý
  - Không thể trừ trực tiếp 2 thanh ghi đoạn
  - Lệnh SUB ảnh hưởng 6 cờ: AF, CF, OF, PF, SF, ZF
    - AF: bật 1 nếu kết quả phép trừ có nhớ/mượn với 4 bit thấp
    - CF: bật 1 nếu kết quả phép trừ có nhớ/mượn
    - PF: bật 1 nếu kết quả có tổng 8 bit thấp là số chẵn
    - ZF: bật 1 nếu kết quả là 0
    - SF: bật 1 nếu kết quả là số âm (bit cao nhất là 1)
    - OF: bật 1 nếu kết quả là số có dấu bị sai (không đủ chứa số có dấu trong dest)

Lệnh DEC

- Cú pháp

  DEC *dest*

- Ý nghĩa
  - dest = dest - 1
  - dest có thể là Reg, Mem
- Ví dụ

```
DEC   AX          ; AX = AX - 1
DEC   W1          ; W1 = W1 - 1
```

Lệnh NEG

- Cú pháp

  NEG *dest*

- Ý nghĩa
  - dest = - dest
  - dest có thể là Reg, Mem
  - Lệnh NEG ảnh hưởng 6 cờ AF, CF, SF, PF, ZF, OF
- Ví dụ

```
NEG   AX          ; AX = - AX
DEC   W1          ; W1 = - W1
```

Lệnh MUL

- Cú pháp

  MUL *source*

- Ý nghĩa
  - Nhân số nguyên không dấu, source có thể là Reg, Mem
  - Nếu source là **8 bit** thì **AX = AL * source**
  - Nếu source là **16 bit** thì **DX:AX = AX * source**
    Trong đó DX là 16 bit cao và AX là 16 bit thấp của kết quả (**32 bit**)
  - Cờ CF và OF bật 1 nếu phần cao của kết quả (AH/DX) khác 0

Lệnh MUL (2)

- Ví dụ
  - Nhân 8 bit

```
MOV  AL, 212
MOV  BL, 45
MUL   BL                    ; AX = AL * BL = 9540
```

  - Nhân 16 bit

```
MOV   AX, 1990
MOV   BX, 2013
MUL    BX
```

; AX * BX = 003D1FEEh → DX = 003Dh , AX = 1FEEh

Lệnh DIV

- Cú pháp

  DIV *source*

- Ý nghĩa
  - Chia số nguyên không dấu, source có thể là Reg, Mem
  - Nếu source 8 bit thì lấy AX chia cho source, phần nguyên lưu vào AL, phần dư lưu vào AH
  - Nếu source là 16 bit thì ghép DX:AX thành toán hạng 32 bit chia cho source, phần nguyên lưu vào AX, phần dư lưu vào DX

Lệnh DIV (2)

- Ví dụ
  - Chia 8 bit

```
MOV  AX, 1981
MOV  BL, 10
DIV    BL                ; AL = 198 , AH = 1
```

  - Chia 16 bit

```
MOV  AX, 2013
MOV  DX, 0
MOV  BX, 10
DIV    BX                ; AX = 201 , DX = 3
```

Lệnh DIV (3)

- Chú ý
  - Lệnh DIV không ảnh hưởng các cờ hiệu
  - CPU sẽ phát sinh ngắt nội bộ “divide overflow” và chấm dứt chương trình khi:
    - Chia số cho 0
    - Phần nguyên kết quả lớn hơn 255 đối với chia 8 bit
    - Phần nguyên kết quả lớn hơn 65535 đối với chia 16 bit

6.4. MỘT SỐ NHÓM LỆNH CĂN BẢN

- 6.4.6.Nhóm lệnh xử lý số học.

AND	Thực hiện phép AND hai toán hạng
OR	Thực hiện phép OR hai toán hạng
XOR	Thực hiện phép XOR hai toán hạng
NOT	Đảo bit của toán hạng (lấy bù 1)
TEST	Thực hiện phép AND hai toán hạng để lập cờ
SHIFT	Dịch trái (phải) toán hạng
ROTATE	Quay vòng trái (phải) toán hạng

Nhóm lệnh dịch chuyển và quay

SHL, SHR, ROL, ROR
SAL, SAR, RCL, RCR

Phép toán logic

P	Q	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- Giả sử có 2 thanh ghi chứa dữ liệu:

  (R1) = 1010 1010

  (R2) = 0000 **1 1**

- R1← (R1) AND (R2) = 0000 1010

Phép toán AND dùng để xoá một số bit và giữ nguyên một số bit còn lại của toán hạng.

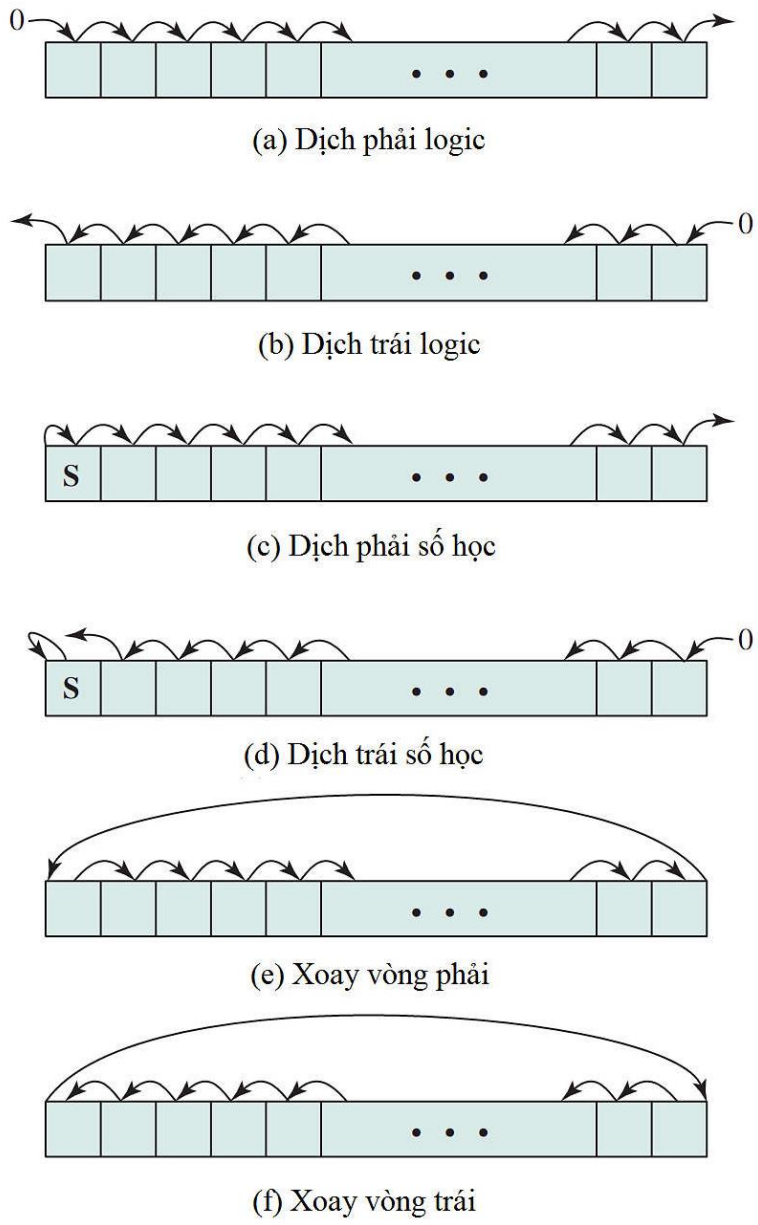
- R1← (R1) OR (R2) = 1010 1 1

Phép toán OR dùng để thiết lập một số bit và giữ nguyên một số bit còn lại của toán hạng.

- R1← (R1) XOR (R2) = 1010 0101

Phép toán XOR dùng để đảo một số bit và giữ nguyên một số bit còn lại của toán hạng.

Hành động dịch và xoay vòng



Dịch logic:

- 1 đầu, bit dịch bị mất đi
- Đầu kia, 0 được dịch vào

Dịch số học:

- Phải: 1 đầu, bit dịch bị mất đi; đầu kia, bit có trọng số cao nhất (LSB) giữ nguyên giá trị cũ
- Trái: 1 đầu bit 0 được dịch vào; đầu kia: bit LSB giữ nguyên giá trị, bit cạnh LSB mất đi

Ví dụ về phép toán dịch và xoay vòng

Đầu vào	Hành động	Kết quả
10100110	Dịch phải logic (3 bit)	00010100
10100110	Dịch trái logic (3 bit)	00110000
10100110	Dịch phải số học (3 bit)	**1** 110100
10100110	Dịch trái số học (3 bit)	10110000
10100110	Xoay vòng phải (3 bit)	11010100
10100110	Xoay vòng trái (3 bit)	00110101

Lệnh AND

- Cú pháp

  AND *dest, source*

- Ý nghĩa:
  - dest = dest And source (thực hiện and từng bit 1)
  - Dest có thể là Reg, Mem
  - Source có thể là Reg, Mem, Immed
  - Ảnh hưởng các cờ CF, OF, PF, SF, ZF
  - Thường dùng để xóa 1 bit nào đó

Lệnh AND

- Ví dụ

```
MOV  AL, 10010110b
AND AL, 00111111b
```

kết quả: AL=00010110b

Lệnh OR

- Cú pháp

  OR *dest, source*

- Ý nghĩa:
  - dest = dest OR source (thực hiện or từng bit 1)
  - Dest có thể là Reg, Mem
  - Source có thể là Reg, Mem, Immed
  - Ảnh hưởng các cờ CF, OF, PF, SF, ZF
  - Thường dùng để bật 1 bit nào đó

Lệnh OR

- Ví dụ

MOV AL, 10010110b

OR AL, 11000000b

kết quả: AL=11010110b

Lệnh XOR

- Cú pháp

  XOR *dest, source*

- Ý nghĩa:
  - dest = dest XOR source (thực hiện xor từng bit 1)
  - Dest có thể là Reg, Mem
  - Source có thể là Reg, Mem, Immed
  - Ảnh hưởng các cờ CF, OF, PF, SF, ZF
  - Dùng xác định các it khác nhau giữa 2 toán hạng

Lệnh OR

- Ví dụ

```
MOV  AL, 10010110b
XOR  AL, 11000000b
```

kết quả: AL=01010110b

Lệnh NOT

- Cú pháp

  NOT *dest*

- Ý nghĩa:
  - Đảo ngược từng bit của toan hạng dest
  - Không ảnh hưởng các cờ
- Ví dụ

```
MOV  AL, 10010110b
NOT  AL
```

kết quả: AL=01101001b

Lệnh TEST

- Cú pháp

  TEST *dest, source*

- Ý nghĩa:
  - Tương tự toan hạng AND nhưng không lưu kết quả
  - Ảnh hưởng các cờ: CF, OF, PF, ZF, SF

Lệnh SHL (Shift logical Left)

- Cú pháp

  SHL *dest, 1*

  Hay SHL *dest, CL*

- Ý nghĩa:
  - Dịch chuyển nội dung dest sang trái 1 bit, bit cao nhất (trái cùng) được đưa vào CF, thêm bit 0 vào phía phải
  - SHL dest, CL: dịch chuyển sang trái với số lần nằm trong CL

Lệnh SHL (Shift logical Left) (2)

- Ví dụ

```
MOV AL, 10010110b
SHL AL, 1
```

kết quả: AL=00101100b, CF=1

- Chú ý:
  - Nếu dest là số nguyên không dấu, SHL dest, 1 sẽ nhân dest với 2
  - SHL ảnh hưởng các cờ CF, OF, PF, SF, ZF
  - Lệnh SHR Tương tự như SHL nhưng sẽ dịch chuyển về bên phải

Lệnh ROL (Rotate Left)

- Cú pháp

  ROL *dest, 1*

  Hay ROL *dest, CL*

- Ý nghĩa:
  - Quay nội dung dest sang trái 1 bit, bit cao nhất (trái cùng) thành bit thấp nhất (phải cùng)
  - ROL dest, CL: quay trái với số lần nằm trong CL

Lệnh ROL (Rotate Left) (2)

- Ví dụ

```
MOV  AL, 10010110b
ROL  AL, 1
```

kết quả: AL=00101101b

- Chú ý:
  - ROL ảnh hưởng các cờ CF, OF
  - Lệnh ROR (rotate Right) tương tự như ROL nhưng quay về phải

- **Bài tập:**
- **[Programmer to Programmer] Richard Blum - Professional Assembly Language (2005, Wiley)**