



HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI GIẢNG MÔN

KIẾN TRÚC MÁY TÍNH

ThS. Nguyễn Trọng Huân

Khoa Kỹ thuật Điện tử 2

2021

CHƯƠNG 3

INSTRUCTION SET

Nội dung

- 1. Khái niệm lệnh, tập lệnh và các thành phần của lệnh**
- 2. Các dạng toán hạng**
- 3. Các chế độ địa chỉ**
- 4. Một số dạng lệnh thông dụng**
- 5. Cơ chế ống lệnh (pipeline)**

1. LỆNH, TẬP LỆNH VÀ CÁC THÀNH PHẦN CỦA LỆNH

- **Lệnh:** Để thực hiện một chức năng cần cung cấp tín hiệu chọn dữ liệu, chọn nơi lưu trữ dữ liệu, và cấp tín hiệu chọn hàm số cần thực hiện. Tổng hợp các tín hiệu lựa chọn này được gọi là một lệnh.
- **Tập Lệnh:** tập hợp các lệnh của 1 máy tính.
 - Máy tính khác nhau có các tập lệnh khác nhau. Tuy vậy, có thể có nhiều điểm giống nhau giữa các lệnh.
 - Máy tính ở các thế hệ trước thường có tập lệnh rất đơn giản bởi dễ thiết kế và thực hiện.

Các kiến trúc tập lệnh CISC và RISC

- **CISC: Complex Instruction Set Computer**
 - Máy tính với tập lệnh phức tạp (station)
 - Các bộ xử lý: Intel x86, Motorola 680x0
- **RISC: Reduced Instruction Set Computer**
 - Máy tính với tập lệnh thu gọn (PC, Laptop, Cellphone, microcontroller...)
 - SunSPARC, Power PC, MIPS, ARM ...
 - RISC đối nghịch với CISC
 - Kiến trúc tập lệnh tiên tiến

Các đặc trưng của kiến trúc RISC

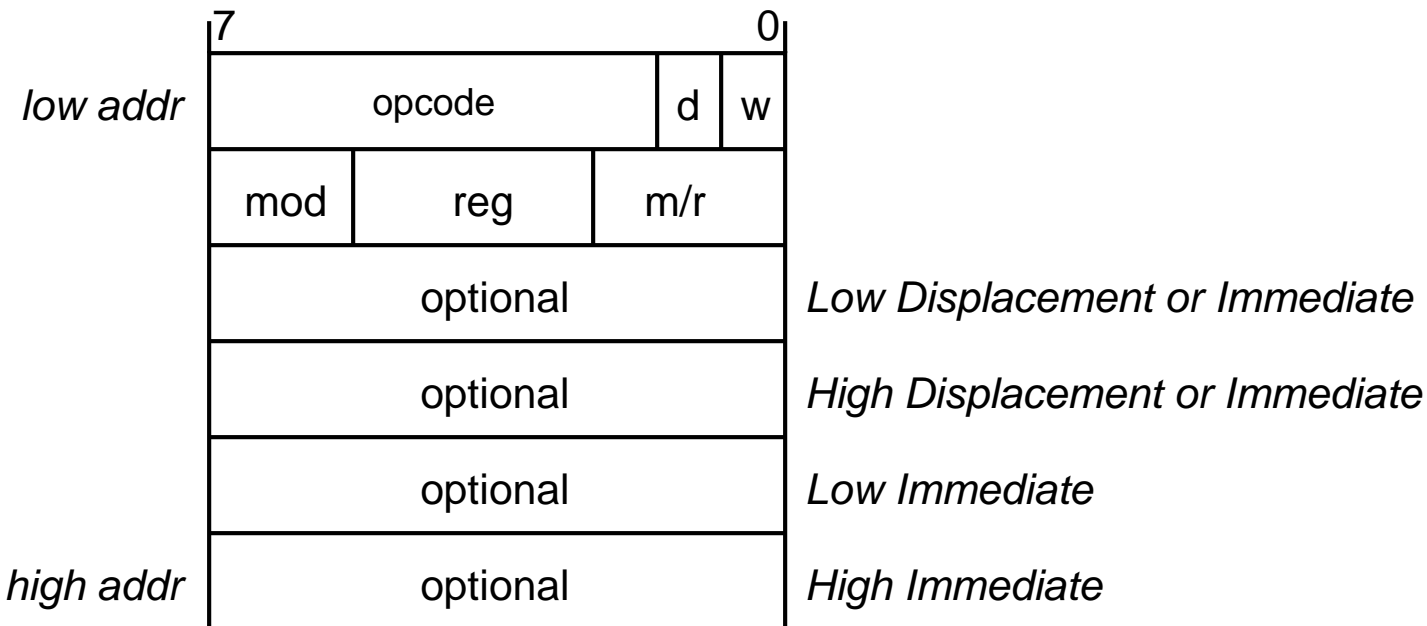
- Số lượng lệnh ít
- Hầu hết các lệnh truy nhập toán hạng ở các thanh ghi
- Truy nhập bộ nhớ bằng các lệnh LOAD/STORE (nạp/lưu)
- Thời gian thực hiện các lệnh là như nhau
- Các lệnh có độ dài cố định (thường là 32 bit)
- Số lượng dạng lệnh ít
- Có ít phương pháp định địa chỉ toán hạng
- Có nhiều thanh ghi
- Hỗ trợ các thao tác của ngôn ngữ bậc cao

CHU KỲ LỆNH

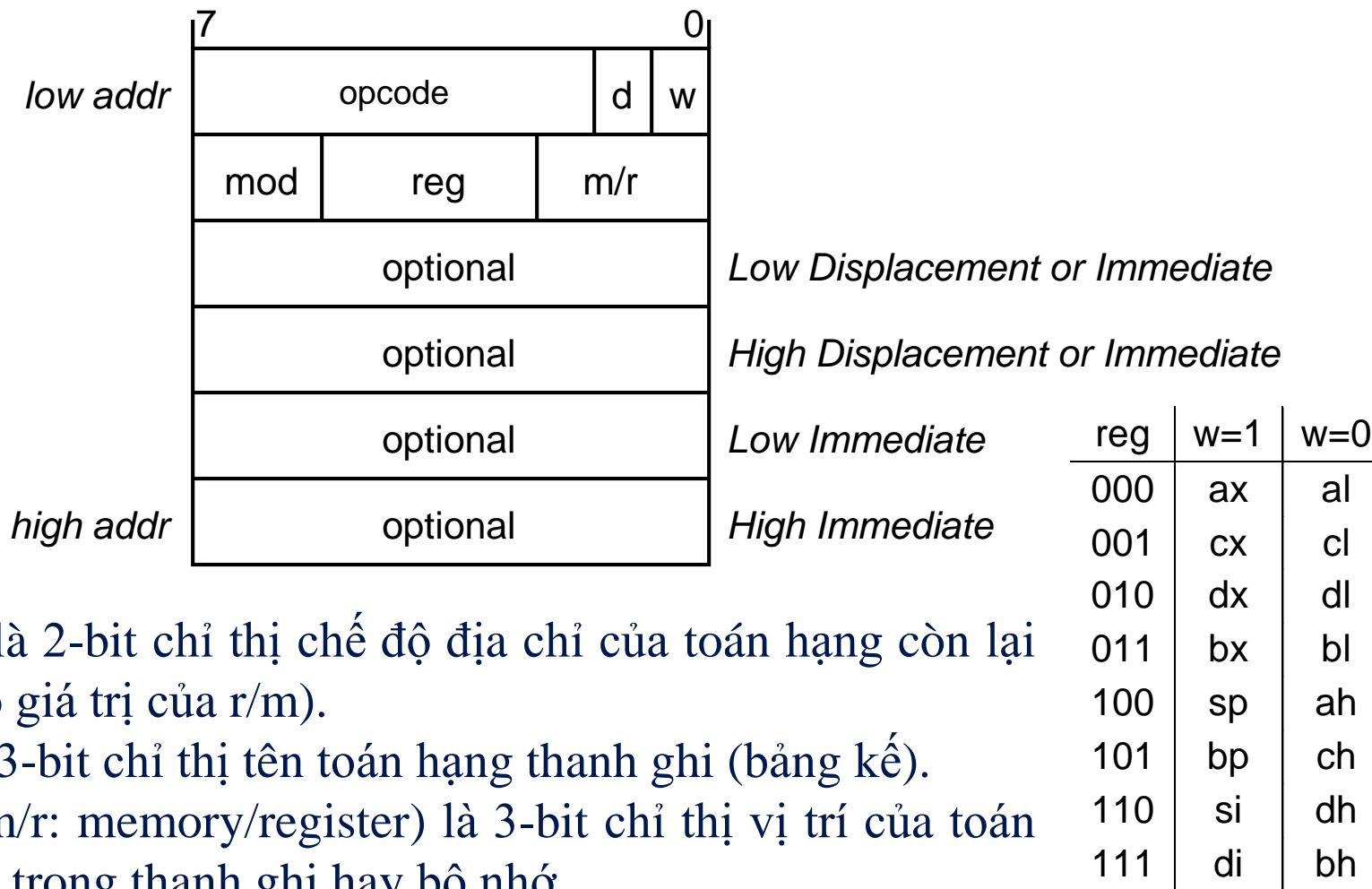
- Nhận lệnh
- Giải mã lệnh
- Nhận toán hạng
- Thực hiện lệnh
- Cát toán hạng
- Ngắt
- **Các phần tiếp theo trong chương này sẽ nghiên cứu kiến trúc tập lệnh CPU INTEL 8086 16BIT**

CÁC THÀNH PHẦN CỦA LỆNH

- Bao gồm hai phần:
 - **OPCODE**: mã lệnh, quy định chức năng lệnh thực hiện.
 - **Operand**: Chọn loại dữ liệu cho lệnh.

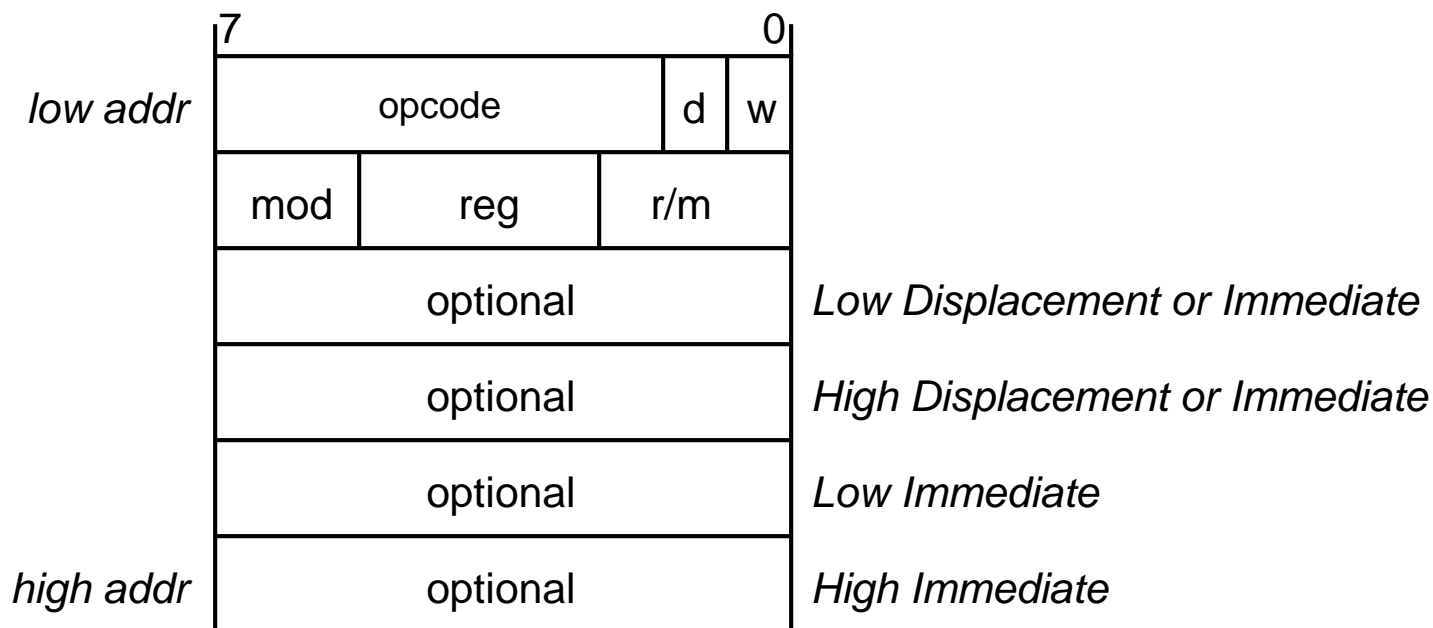


- **opcode** 6-bit quy định loại lệnh sẽ thực hiện (mã hóa 63 lệnh khác nhau)
- **d (destination)**: d=1 toán hạng quy định bởi 3bit reg là toán hạng đích
d=0 toán hạng quy định bởi 3bit reg là toán hạng nguồn
- **w (word)**: w=1 mã hóa toán hạng trong lệnh bao gồm 2 byte
w=0 mã hóa toán hạng trong lệnh bao gồm 1 byte

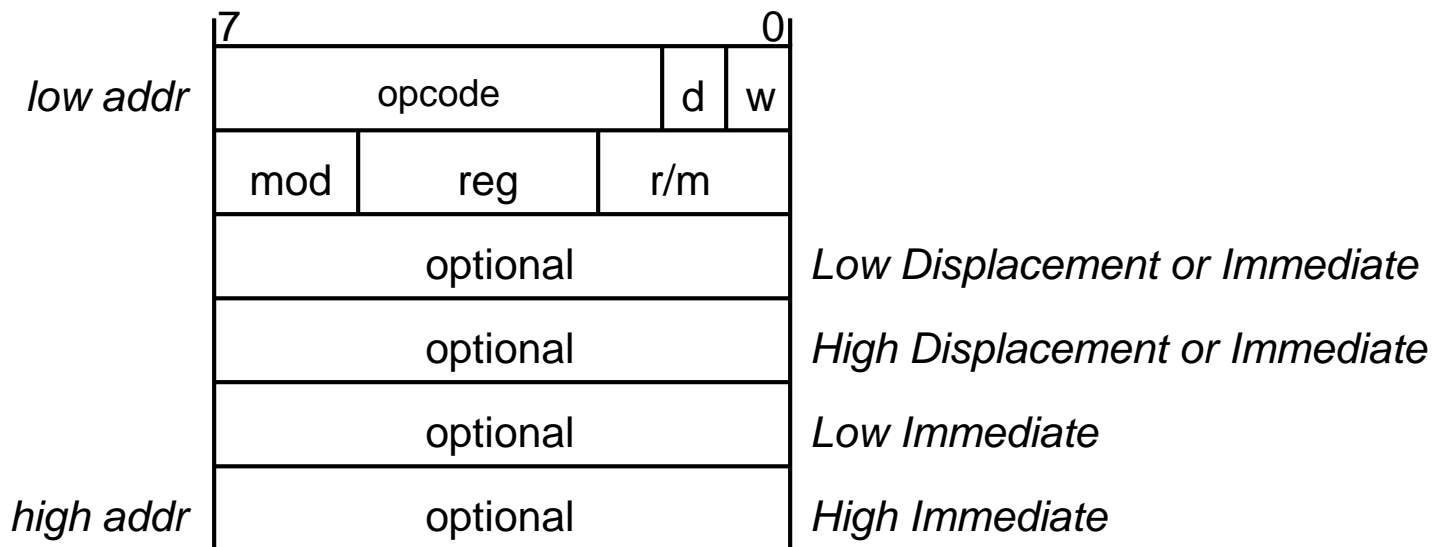


- **mod** là 2-bit chỉ thị chế độ địa chỉ của toán hạng còn lại (tùy theo giá trị của r/m).
- **reg** là 3-bit chỉ thị tên toán hạng thanh ghi (bảng kế).
- **m/r** (m/r: memory/register) là 3-bit chỉ thị vị trí của toán hạng lưu trong thanh ghi hay bộ nhớ.

Mod r/m	00	01	10	11	
				W=0	W=1
000	[BX]+[SI]	[BX]+[SI] + d8	[BX]+[SI] + d16	AL	AX
001	[BX]+[DI]	[BX]+[DI] + d8	[BX]+[DI] + d16	CL	CX
010	[BP]+[SI]	[BP]+[SI] + d8	[BP]+[SI] + d16	DL	DX
011	[BP]+[DI]	[BP]+[DI] + d8	[BP]+[DI] + d16	BL	BX
100	[SI]	[SI] + d8	[SI] + d16	AH	SP
101	[DI]	[DI] + d8	[DI] + d16	CH	BP
110	[BP]	[BP] + d8	[BP] + d16	DH	SI
111	[BX]	[Bx] + d8	[Bx] + d16	BH	DI



- **Displacement** (độ dời) có thể 8 hoặc 16 bit
 - Là giá trị Hexa được mã hoá trong lệnh.
 - Sử dụng để tính toán giá trị địa chỉ của toán hạng
- **Immediate** (dữ liệu tức thời) có thể 8, 16 hoặc 32 bit
 - Là giá trị hexa
 - Sử dụng làm toán hạng trong lệnh



Ví dụ: xét lệnh

Mã lệnh sẽ là:

opcode là:

d là:

w là:

mod là:

reg là:

r/m là:

100010

1

1

11

000

011

mov ax, bx

8B C3 = 100010111000011

mov

thanh ghi là toán hạng đích

toán hạng đích là 1 word

chỉ thị giá trị r/m trở tới địa chỉ thanh ghi

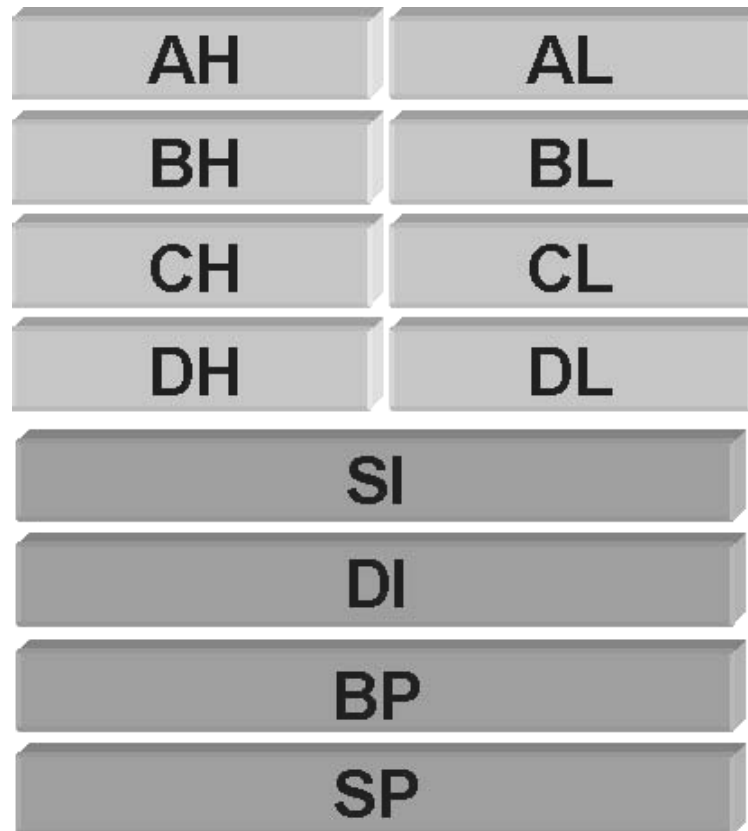
thanh ghi toán hạng đích là **ax**

thanh ghi toán hạng nguồn là **bx**

CÁC THANH GHI CỦA CỦA 8086

- Các thanh ghi đa năng: AX, BX, CX, DX, DI, SI, BP, SP.
- Các thanh ghi đoạn: CS, DS, ES, SS.
- Thanh ghi cờ (Flags).
- Thanh ghi IP

Các thanh ghi đa năng (16 bit)



- **Đều có thể sử dụng chứa dữ liệu 16 bit.**
- **AX, BX, CX và DX có thể chia thành hai phần 8 bit.**

Các thanh ghi đa năng

- **AX (Accumulator)** làm thanh ghi chứa dữ liệu cho lệnh nhân và chia.

Ví dụ: MUL BL ; $AX = AL * BL$

- **BX (Base):** Sử dụng làm thanh ghi địa chỉ cơ sở.

Ví dụ: MOV AL,[BX] ; nội dung BX giữ địa chỉ của ô nhớ.

MOV AH,[BX+03] ; chép nội dung ô nhớ có địa chỉ [BX+3] vào AH.

MOV CH,[BX+DI+9]

- **CX (Counter):** Sử dụng làm bộ đếm số vòng lặp.

Ví dụ: LOOP N ; trừ CX đi 1, lặp lại nhãn N khi CX $\neq 0$.

- **DX (Data):** Làm thanh ghi chứa dữ liệu trong các lệnh nhân chia 16 bit. Làm thanh ghi giữ địa chỉ cổng vào ra.

Ví dụ: MUL BX ; $DX\ AX = AX * BX$

DIV BX ; DX chứa dư, AX chứa thương

OUT DX,BL ; chuyển nội dung BL ra cổng có địa chỉ DX

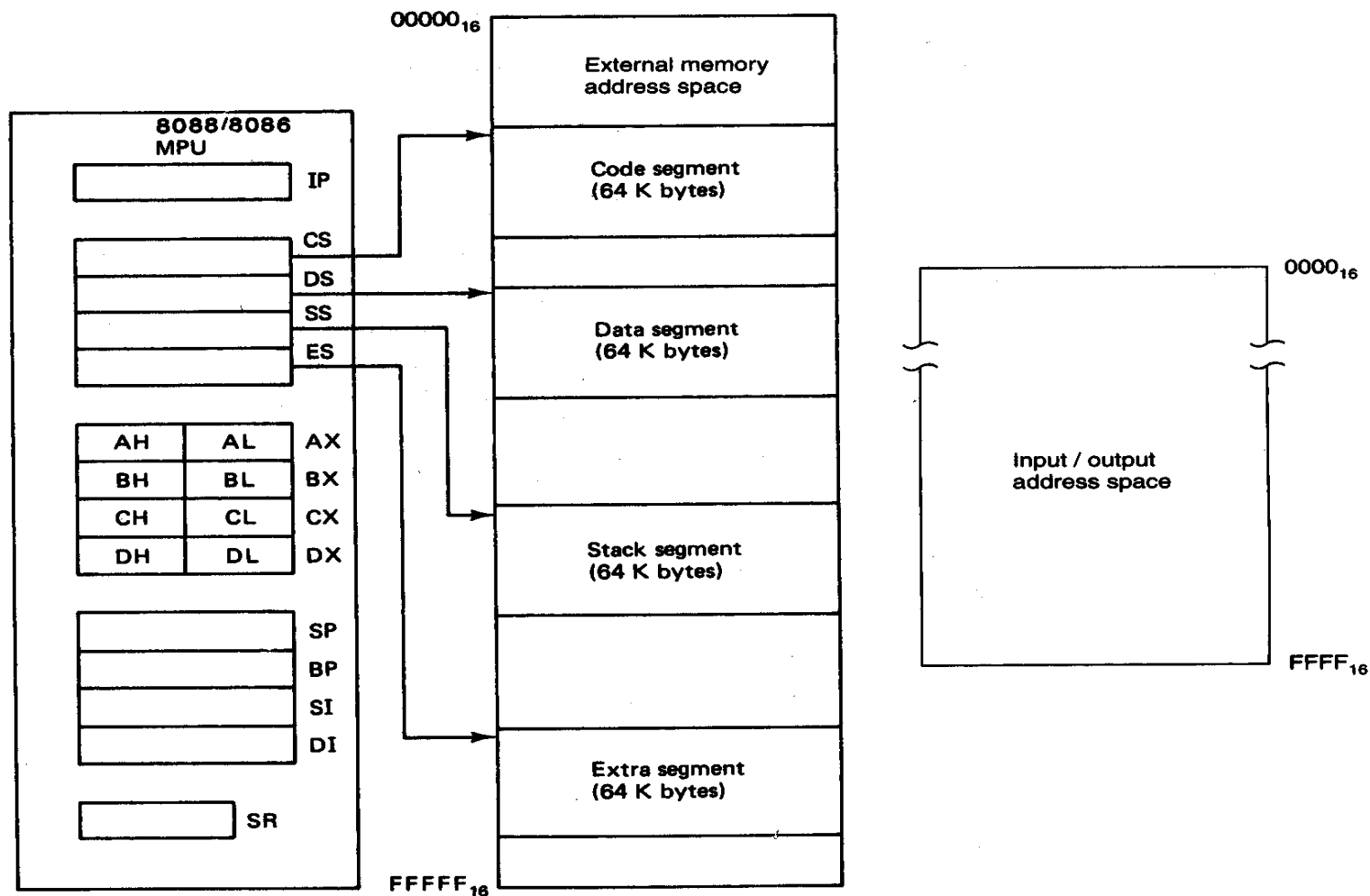
Các thanh ghi đa năng

- **DI (Destination Index):** thanh ghi chỉ số đích – Làm thanh ghi địa chỉ hoặc giữ địa chỉ đích dữ liệu trong các lệnh về chuỗi.
Ví dụ: MOV AH,[DI]
MOV BH,[DI+7]
MOVSb
- **SI (Source Index):** thanh ghi chỉ số nguồn – Làm thanh ghi địa chỉ hoặc giữ địa chỉ nguồn dữ liệu trong các lệnh về chuỗi.
Ví dụ: MOV AH,[SI]
MOV BH,[SI+7]
MOVSw
- **BP (base pointer):** Làm thanh ghi địa chỉ hoặc thanh ghi con trỏ cơ sở.
Ví dụ: MOV AL,[BP]
MOV AH,[BP+03]
MOV CH,[BP+DI+9]

Các thanh ghi đoạn (16 BIT)

- Các thanh ghi đoạn được sử dụng để quản lý địa chỉ của bộ nhớ ngoài được kết nối với VXL:
 - CS : Code Segment – Đoạn mã lệnh.
 - DS : Data Segment – Đoạn dữ liệu.
 - SS : Stack Segment – Đoạn ngăn xếp.
 - ES : Extra Segment – Đoạn mở rộng.

Phân đoạn bộ nhớ



Chế độ địa chỉ theo đoạn

CS

16-bit Segment Base Address

0000

+

IP

16-bit Offset Address

20-bit Physical Address

➤ Chế độ địa chỉ thực (vật lý):
Physical address = Segment * 10H
+ offset

VD: Địa chỉ đoạn và địa chỉ offset
(Segment, Logical):

0FE6:032Bh

• Địa chỉ độ dài, chỉ số (offset, index):

032Bh

• Địa chỉ vật lý:

$$\begin{array}{rcl} 0FE60h & \rightarrow & 65120 \\ + 032Bh & \rightarrow & 811 \\ \hline 1018Bh & \rightarrow & 65931 \end{array}$$

D15

Thanh ghi trạng thái (Flags)

D0

*	NT	IO	PL	O	D	I	T	S	Z	*	A	*	P	*	C
---	----	----	----	---	---	---	---	---	---	---	---	---	---	---	---

- ❖ Chứa các trạng thái của VXL khi thực hiện lệnh. Các cờ phổ biến:
 - **Cờ nhớ CF (Carry Flag).**
CF=1 khi cộng tràn, trừ mượn.
 - **Cờ chẵn lẻ PF (Parity Flag).**
PF = 1 khi số bit 1 là một số chẵn.
 - **Cờ dấu SF (Sign Flag).**
SF =1 khi kết quả có bit cao nhất là 1
 - **Cờ tràn OF (Overflow Flag).**
OF = 1 khi tràn bit 1 từ MSB-1 qua MSB.
 - **Cờ ZERO ZF (Zero Flag).**
ZF = 1 khi kết quả bằng 0.
 - **Cờ nhớ phụ AF (Auxiliary Carry Flag).**
AF=1 khi tràn bit 1 từ D3 qua D4.
 - **Cờ định hướng DF (Direction Flag):**
Xác định hướng địa chỉ của các lệnh xử lý chuỗi (DF=0, xử lý chuỗi theo hướng tăng địa chỉ)
 - **Cờ ngắt IF : (Interrupt Flag).**
IF=1 cho phép ngắt.
 - **Cờ bẫy TF: (Trap Flag):** Sử dụng chạy từng bước để sửa sai chương trình.

Thanh ghi IP

- Thanh ghi con trỏ lệnh IP (IP - Instruction Pointer).
 - Giữ địa chỉ độ dời của vùng nhớ chứa mã lệnh.
 - Tự động tăng lên 1 khi đọc xong một ô nhớ lệnh.
 - Khi điều khiển chương trình IP được thay đổi giá trị mới.

2. TOÁN HẠNG TRONG LỆNH

➤ Register

- Được mã hoá trong lệnh.
- Lệnh thực hiện nhanh.
- Không có truy cập Bus
- Chiều dài lệnh ngắn.
- Ví dụ: lệnh cộng **ADD AX, BX**

➤ Immediate

- Hằng số mã hoá trong lệnh
- 8 hoặc 16 bit
- Không truy cập Bus
- Chỉ có thể là toán hạng nguồn
- Qui ước giá trị trong trình biên dịch Assembler: thập phân (10), nhị phân (0000 1111B), hexa (0FH)
- Ví dụ: lệnh trừ **SUB AX, 10**

➤ Memory

- Nằm trong bộ nhớ, yêu cầu truy cập Bus
- Có thể cần tính toán địa chỉ.
- Địa chỉ của toán hạng trong lệnh là địa chỉ
- Ví dụ: lệnh sao chép dữ liệu ***MOV AX, [50h]***
MOV BX, var

3. CÁC CHẾ ĐỘ ĐỊA CHỈ TRONG LỆNH

- Tức thời (Immediate)
- Thanh ghi (Register)
- Trực tiếp (Direct)
- Gián tiếp thanh ghi (Indirect register)
- Tương đối thanh ghi (Register relative)
- Chỉ số (indexed)
- Cơ sở chỉ số (Based Indexed)
- Cơ sở chỉ số tương đối (Based Indexed relative)
- Địa chỉ cổng (I/O Port)

Chế độ địa chỉ tức thời – Immediate Addressing

- Dữ liệu là một số mã hóa (nằm trực tiếp) trong lệnh, dữ liệu có thể là 8bit hoặc 16bit.

Ví dụ: MOV BL,44

MOV SI,0

MOV AL,'a'

MOV AX,'AB'

MOV CL,11001110B

MOV AX,1000H

MOV BP,3000H

Chú ý: kích thước của dữ liệu phải tương ứng với dung lượng của thanh ghi chứa.

Chế độ địa chỉ thanh ghi – Register Addressing

- Dữ liệu nằm trong thanh ghi.

Ví dụ: MOV AX,BX

MOV AL,AH

MOV SP,BP

MOV SI,DI

MOV DS,AX

- ❖ Các thanh ghi được sử dụng:

- 8bit: AH, AL, BH, BL, CH, CL, DH, DL.
- 16bit: AX, BX, CX, DX, SP, BP, SI, DI, CS, DS, SS, ES.

Chú ý:

MOV ES,DS ; không trao đổi dữ liệu giữa 2 thanh ghi đoạn

MOV BL,DX ; dung lượng 2 thanh ghi không tương thích nhau

Chế độ địa chỉ trực tiếp (Direct Addressing)

- Dữ liệu nằm trong bộ nhớ, địa chỉ offset của ô nhớ nằm trực tiếp trong lệnh.

MOV CH, **number**

MOV CX, **beta**

MOV AX, **[5000h]**

→ Number, beta là biến được khai báo lưu trong bộ nhớ

Chế độ địa chỉ gián tiếp thanh ghi (Register Indirect Addressing)

- Địa chỉ được lưu giữ trong thanh ghi.
- Các thanh ghi dùng trong chế độ địa chỉ này: BP (SS), BX, SI, DI (DS).

Ví dụ 1: `MOV AL, [BP]` ; dữ liệu 8 bits có địa chỉ SS:BP
`MOV CX, [BX]` ; dữ liệu 16 bits có địa chỉ DS:BX
`MOV [DI], AH` ; đ/c đích ES:DI

Chú ý: `MOV [DI], [BX]`; không trao đổi dữ liệu trực tiếp giữa 2 ô nhớ

Ví dụ 2: cho vùng nhớ DS bắt đầu tại địa chỉ 0100h, Bx=1000h, nội dung của các ô nhớ như sau: 01000h=11; 01001h=22; 02000h=32; 02001h=53 .Cho biết kết quả sau khi thực hiện lệnh: `MOV AX,[BX]`

Chế độ địa chỉ tương đối thanh ghi (Register Relative Addressing)

- Địa chỉ được tính bằng nội dung của thanh ghi BP (SS), BX, SI, DI (DS,ES) cộng với độ dời 8bit hoặc 16bit.
- Địa chỉ này cũng được sử dụng để truy cập dữ liệu mảng.

Ví dụ 1: MOV AL, [BP+2]; dữ liệu 8 bits có địa chỉ SS:BP+2
MOV AH, [BX-4]; dữ liệu 8 bits có địa chỉ DS:BX-4

Ví dụ 2: cho vùng nhớ DS bắt đầu tại địa chỉ 0200h,
Bx=0100h, nội dung của các ô nhớ như sau: 01100h=10;
01101h=20; 02100h=32; 02101h=53, 03100h=45; 03101h=23
Cho biết kết quả sau khi thực hiện lệnh:

MOV AX, [BX+1000H];

Chế độ địa chỉ chỉ số (Indexed Addressing)

- Địa chỉ được lưu giữ trong các thanh ghi chỉ số SI (DS), DI (ES).

Ví dụ:

MOV AX, **[SI]** ; dữ liệu 16 bits có địa chỉ DS:SI

MOV **[DI]**, AX ; dữ liệu 16 bits có địa chỉ ES:DI

Chế độ địa chỉ cơ sở chỉ số (Based Indexed Addressing)

- Sử dụng thanh ghi cơ sở (BP, BX) và thanh ghi chỉ số (SI, DI) để lưu giữ địa chỉ. Địa chỉ bằng tổng nội dung của 2 thanh ghi này.

Ví dụ 1: MOV DX, **[BX+DI]** ;

Ví dụ 2: cho vùng nhớ DS bắt đầu tại địa chỉ 0100h, nội dung Bx=1000h, DI=0010h nội dung của các ô nhớ như sau: 01010h=66; 01011h=12; 02010h=1fh; 02011h=00, 03010h=ffh; 03011h=05. Cho biết kết quả sau khi thực hiện lệnh:

MOV AX, [BX+DI];

Chế độ địa chỉ cơ sở chỉ số tương đối (Based Indexed Relative Addressing)

- Tương tự chế độ địa chỉ cơ sở chỉ số nhưng có cộng thêm độ dời 8bit hoặc 16bit.

Ví dụ 1:

MOV AX, **[BX+SI+10H]** ; dữ liệu 16bit có đ/c DS:BX+SI+10h

MOV CL, **[BP+SI-7]** ; dữ liệu 8bit có đ/c SS:BP+SI-7

Ví dụ 2: cho vùng nhớ DS bắt đầu tại địa chỉ 1000h, nội dung Ax=a9h, Bx=0020h, DI=0010h nội dung một các ô nhớ như sau: 00130h=66; 00131h=12; 01300h=1fh; 01310h=00, 10130h=ffh; 10131h=05. Cho biết kết quả sau khi thực hiện lệnh:

MOV AX, [BX+DI+100H];

Chế độ địa chỉ cổng (I/O Port Addressing)

- CPU Intel x86 có thể quản lý tới 65,536 địa chỉ cổng vào ra
- Mỗi địa chỉ cổng (giống như bộ nhớ), truy cập tới một cổng duy nhất
- Địa chỉ I/O có thể 1 byte hoặc 2 byte
- Có hai chế độ địa chỉ cổng
 - 1) Chế độ địa chỉ cổng tức thời
 - Chỉ có thể là địa chỉ 1 byte
 - Giá trị trong khoảng từ 00 tới FFh
 - 2) Địa chỉ cổng giữ trong DX
 - Có thể mang các giá trị trong khoảng từ 0000h tới FFFFh
- Chỉ có thể sử dụng thanh ghi DX để giữ địa chỉ cổng
- Chỉ có thể sử dụng các thanh ghi AL, AX để chứa dữ liệu cổng.

4. MỘT SỐ LỆNH THÔNG DỤNG

- Tập lệnh của VXL 8086
 - Các lệnh truyền dữ liệu.
 - Các lệnh xử lý dữ liệu.
 - ✓ Các lệnh số học
 - ✓ Các lệnh logic
 - ✓ Các lệnh quay dịch
 - Các lệnh điều khiển chương trình.
 - ✓ Các lệnh nhảy, rẽ nhánh, lặp.
 - ✓ Các lệnh chương trình con.
 - Các lệnh xử lý chuỗi dữ liệu.

Các lệnh truyền dữ liệu

MOV

- Cú pháp: **MOV Dest, Source**
- Thực hiện: $\text{Dest} \leftarrow \text{Source}$: Dữ liệu được chép từ toán hạng nguồn (Source) tới toán hạng đích (Dest). Chú ý lệnh không làm thay đổi toán hạng nguồn, chỉ làm thay đổi toán hạng đích.
- Chú ý một số trường hợp không sử dụng được trong lệnh này bao gồm:
 - Không chuyển dữ liệu trực tiếp giữa hai ô nhớ.
 - Không chuyển giá trị tức thời (Imm) vào thanh ghi đoạn.
 - Không chuyển dữ liệu giữa hai thanh ghi đoạn.
 - Không dùng thanh ghi CS làm toán hạng đích.

Các toán hạng sử dụng trong lệnh MOV

Dest	Source	Ví dụ
Reg	Reg	MOV AH,BH
Reg	Mem	MOV CL,[1000H]
Reg	Imm	MOV DH,10
Mem	Reg	MOV X1,BX
Mem	Imm	MOV X2,25H
Seg.Reg	Reg16	MOV DS,AX
Seg.Reg	Mem16	MOV ES,X2

XCHG, LEA

- **XCHG**: Dữ liệu được chuyển đổi giữa hai toán hạng đích và nguồn, sau khi thực hiện lệnh cả toán hạng đích và toán hạng nguồn đều thay đổi.
 - Cú pháp: **XCHG Dest,Source**
 - Thực hiện: Dest ↔ Source
 - VD: XCHG [5000h],AX

- **LEA**: Lệnh lấy địa chỉ offset (Effective Address).
 - Lấy địa chỉ offset của biến Var đưa vào một thanh ghi 16 bit.
 - Cú pháp: **LEA Reg16,Var** ;Var là tên biến cần lấy địa chỉ.
 - Thực hiện: Reg16 ← Địa chỉ offset của biến Var.
 - VD: LEA BX, [Si]
LEA CS, [BX+SI+5000h]

PUSH, POP

➤ PUSH: Lệnh cất dữ liệu vào ngăn xếp.

- Cú pháp: **PUSH Source**
- Thực hiện: Cất dữ liệu chỉ thị bởi toán hạng nguồn vào đỉnh ngăn xếp và tự động giảm con trỏ ngăn xếp SP tùy theo số byte của toán hạng nguồn, toán hạng nguồn trong lệnh có thể là một ô nhớ hoặc một thanh ghi.
- Ví dụ : PUSH AX; lấy nội dung chứa trong AX cất vào ngăn xếp
 $[SS*10H+(SP-1)] \leftarrow AH$
 $[SS*10H+(SP-2)] \leftarrow AL$

➤ POP: Lệnh lấy dữ liệu ra khỏi ngăn xếp.

- Cú pháp: **POP Dest**
- Thực hiện: Lấy dữ liệu từ đỉnh ngăn xếp ra toán hạng đích và tăng giá trị của con trỏ ngăn xếp SP lên tương ứng với số byte chứa trong toán hạng đích, toán hạng đích trong lệnh cũng có thể là một ô nhớ hoặc một thanh ghi.
- Ví dụ: POP AX; lấy nội dung trong ngăn xếp tại vị trí con trỏ SP lưu vào AX

CÁC LỆNH XỬ LÝ DỮ LIỆU

- Các lệnh số học
- Các lệnh logic

Các lệnh số học (ADD, SUB, INC, DEC, MUL, DIV, CMP)

add	ax, bx	;ax \leftarrow ax+bx, set flags
adc	ax, bx	;ax \leftarrow ax+bx+CF(lsb), set flags
inc	ax	;ax \leftarrow ax+1, set flags
sub	ax, bx	;ax \leftarrow ax-bx, set flags
sbb	ax, bx	;ax \leftarrow (ax-CF)-bx, set flags
dec	ax	;ax \leftarrow ax-1, set flags
cmp	ax, bx	;Flags phụ thuộc kết quả so sánh ax,bx
mul	cx	;dx:ax \leftarrow ax * cx (unsigned)
div	cl	;thương al \leftarrow ax/cl, dư ah \leftarrow ax/cl

BÀI GIẢNG MÔN KIẾN TRÚC MÁY TÍNH

ADD, ADC, INC

➤ Add

- Cú pháp: **ADD Dest,Source**
- Thực hiện: $\text{Dest} \leftarrow \text{Dest} + \text{Source}$
- $\text{CF} \leftarrow$ Bit tràn của kết quả.
- Các toán hạng sử dụng trong lệnh:

Dest	Source	Ví dụ
Reg	Reg	ADD AX,BX
Reg	Mem	ADD BX,VAR
Mem	Reg	ADD SUM,DX
Reg	Imm	ADD CL,09
Mem	Imm	ADD SUM,10

➤ **ADC** (Add with Carry):

- Cú pháp: **ADC Dest,Source**
- Thực hiện: $\text{Dest} \leftarrow \text{Dest} + \text{Source} + \text{CF}$
- $\text{CF} \leftarrow \text{Bit tràn của kết quả.}$
- Ví dụ: `ADC bx, dx` ; $\text{bx} \leftarrow \text{bx} + \text{dx} + \text{CF}$

➤ **INC**: Lệnh tăng (Increment).

- Cú pháp: **INC Dest**
- Thực hiện: $\text{Dest} \leftarrow \text{Dest} + 1$
- Toán hạng đích sử dụng trong lệnh có thể là một thanh ghi hoặc một ô nhớ.
- Ví dụ: `inc bl`; $\text{bl} = \text{bl} + 1$
`inc [bx]`

SUB, SBB, DEC, CMP

- **SUB AX, BX** ; $ax \leftarrow ax - bx$ tác động tới các cờ
- **SBB AX, BX** ; $ax \leftarrow (ax - CF) - bx$ và tác động tới các cờ
- **DEC AX** ; $ax \leftarrow ax - 1$
- **CMP AX, BX** ; so sánh ax và bx, tác động tới các cờ tùy thuộc vào kết quả

Thực hiện: $ax - bx$, chú ý nội dung ax không thay đổi

- $ax < bx \rightarrow CF=1, ZF=0$
- $ax = bx \rightarrow ZF=1, CF=0$
- $ax > bx \rightarrow CF=0, ZF=0$

MUL

➤ Nhân 8 bit: **MUL Reg8**

- Thực hiện: $AX = AL * \text{Reg8}$

- Ví dụ:

MUL BL ; $AX \leftarrow AL * BL$

➤ Nhân 16bit: **MUL Reg16**

- Thực hiện: $DX:AX = AX * \text{Reg16}$

- Ví dụ:

MUL BX ; $DX:AX \leftarrow BX * AX$

DIV

➤ Chia 8 bit: **DIV Reg8**

- Thực hiện: AX/Reg8
- Ví dụ:

DIV CL ;giá trị ax chia cho giá trị cl
 ;thương nằm trong al
 ;phần dư nằm trong ah

➤ Chia 16bit: **DIV Reg16**

- Thực hiện: DX:AX/Reg16
- Ví dụ:

DIV CX ;giá trị dx:ax chia cho giá trị cx
 ;thương nằm trong ax
 ;phần dư nằm trong dx

Các lệnh logic (NOT, AND, OR, XOR)

- Ảnh hưởng tới các cờ trạng thái:
- Luôn xoá CF và OF
 - SF, ZF, AF, PF thay đổi tùy theo kết quả

Ví dụ 1:

NOT AX ; NOT

AND AX, BX ; AND


OR AX, BX ; OR

XOR AX, BX ; XOR


Ví dụ 2: MOV AL, 01011010B
AND AL, 00001111B
-> AL= 00001010B

Các lệnh dịch


shl - Logical Shift Left



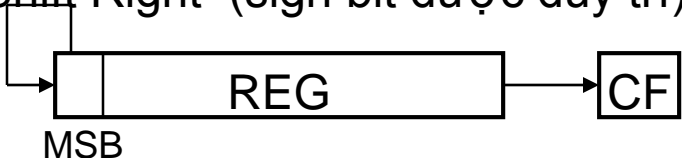
shr - Logical Shift Right



sal - Arithmetic Shift Left (giống dịch trái logic)



sar - Arithmetic Shift Right (sign bit được duy trì)



Các lệnh xoay

rol - Rotate Left



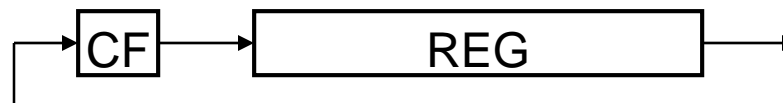
rcl - Rotate Left có cờ CF



ror - Rotate Right



rcr - Rotate Right có cờ CF



Các lệnh điều khiển chương trình

- Thông thường làm thay đổi giá trị của **CS:IP**
- Thay đổi vị trí thực hiện các lệnh.
- Các lệnh:
 - Lệnh nhảy không điều kiện
 - Lệnh nhảy có điều kiện
 - Lệnh lặp
 - Chuyển điều khiển chương trình không điều kiện (gọi chương trình)

Lệnh nhảy không điều kiện JMP

- **Cú pháp: `JMP LABEL`** ;Nhãn LABEL được mã hoá bằng địa chỉ Offset của nơi chuyển tới trong mã lệnh.
- **Phạm vi của nhãn Label:**
 - SHORT - 1 bytes, cho phép nhảy trong khoảng ± 127 từ vị trí hiện tại
 - NEAR- 2 bytes, cho phép nhảy trong khoảng $\pm 32K$ từ vị trí hiện tại → Nhảy trong cùng segment
 - FAR - 4 bytes nhảy tới mọi vị trí trong bộ nhớ → Nhảy đến segment khác

Lệnh nhảy không điều kiện JMP

Ví dụ: Viết chương trình cho BX chứa tổng các số từ 0 tới 65535.

	XOR	BX,	BX	;Xoá bx và khởi động các cờ
start:	MOV	AX,	1	;ax \leftarrow 1
	ADD	AX,	BX	;ax \leftarrow ax+bx
	JMP	NEXT		;cộng số độ dời vào IP ; (+2 từ xor tới mov)
	XOR	BX,	BX	;Xoá bx và khởi động các cờ
	XOR	AX,	AX	;Xoá ax và khởi động các cờ
next:	MOV	BX,	AX	;bx \leftarrow ax
	JMP	start		;cộng giá trị độ dời vào IP ;(là một giá trị âm – 2 giá trị IP hiện tại)

Các lệnh nhảy có điều kiện

- Nhảy theo trạng thái các cờ (dựa theo các cờ C và Z) sau khi thực hiện lệnh so sánh (số không dấu)

JA/JNBE label ;jump if cf=zf=0 - jump above-jump not below/equal

JAE/JNB label ;jump if cf=0 or zf=1

- jump above/equal-jump not below

JB/JNAE label ;jump if cf=1 - jump below-jump not above/equal

JBE/JNA label ;jump if cf=1 or zf=1

- jump below/equal-jump not above

JE label ;jump if zf=1 - jump equal

JNE label ; - jump not equal

a=above=lớn hơn; b=below=nhỏ hơn; e=equal=bằng; n=not

Ví Dụ:

CMP AL,BL

JB there ; nhảy nếu al nhỏ hơn bl
; so sánh không dấu

Các lệnh nhảy có điều kiện

- Nhảy theo trạng thái các cờ (dựa theo các cờ Z, S và V) sau khi thực hiện lệnh so sánh (số có dấu)

JG/JNLE	LABEL ;jump if ZF=0 and (SF=OF) - jump greater/not less nor equal
JGE/JNL	LABEL ;jump if SF=OF - jump greater-equal/not less than
JL/JNGE	LABEL ;jump if SF \neq OF - jump less than/not greater nor equal
JLE/JNG	LABEL ;jump if ZF=1 or SF \neq OF - jump less or equal/not greater than

g=greater=lớn hơn; l=less=nhỏ hơn; e=equal=bằng; n=not

Ví dụ:

```
CMP AL,BL
JL  there    ; nhảy nếu al nhỏ hơn bl
                ; so sánh có dấu
```

Lệnh lặp LOOP

➤ Cú pháp:

LABEL:

.....

LOOP

- Sử dụng CX làm số đếm của vòng lặp
- Kết hợp giảm CX và nhảy có điều kiện
- Giảm CX và nếu $CX \neq 0$ thì nhảy tới LABEL

Ví dụ: Tính tổng các số từ 0 đến 100

```
MOV    CX,100          ;  
MOV    AX,0            ;  
AGAIN:ADD  AX,CX        ;  
LOOP   AGAIN           ;
```

5. ỐNG LỆNH (INSTRUCTION PIPELINING)

- **Instruction Pipelining:** Chia chu trình lệnh thành các công đoạn và cho phép thực hiện gối chồng lên nhau (như dây chuyền lắp ráp).
- Ví dụ: 1 ống lệnh bao gồm 6 công đoạn
 - Nhận lệnh (Fetch Instruction – FI) từ bộ nhớ
 - Giải mã lệnh (Decode Instruction – DI), đọc các toán hạng
 - Tính địa chỉ toán hạng (Calculate Operand Address – CO)
 - Nhận toán hạng (Fetch Operands – FO)
 - Thực hiện lệnh (Execute Instruction – EI)
 - Ghi toán hạng (Write Operands – WO)

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

- Với cơ chế thực hiện không pipeline, tại mỗi thời điểm chỉ có một lệnh được thực hiện và chỉ có một đơn vị chức năng của CPU làm việc, các đơn vị chức năng khác trong trạng thái chờ.
- Với cơ chế thực hiện có pipeline, có nhiều lệnh đồng thời được thực hiện gối nhau trong CPU và hầu hết các đơn vị chức năng của CPU liên tục tham gia vào quá trình xử lý lệnh. Số lượng lệnh được xử lý đồng thời đúng bằng số giai đoạn thực hiện lệnh.
- Với 6 giai đoạn thực hiện lệnh, để xử lý 5 lệnh, CPU cần 10 nhịp đồng hồ với cơ chế thực hiện có pipeline, trong khi CPU cần đến 30 nhịp đồng hồ để thực hiện 5 lệnh với cơ chế thực hiện không pipeline.

Các vấn đề khi sử dụng ống lệnh

- Xung đột cấu trúc: do nhiều công đoạn dùng chung một tài nguyên
- Xung đột dữ liệu: lệnh sau sử dụng dữ liệu kết quả của lệnh trước
- Xung đột điều khiển: do rẽ nhánh gây ra

Xung đột cấu trúc

- Xung đột truy cập bộ nhớ
- Xung đột truy cập các thanh ghi

Ví dụ: giai đoạn FI và DI

- **Giải pháp:** nâng cao năng lực phục vụ của các tài nguyên phần cứng
 - Với xung đột truy cập bộ nhớ có thể sử dụng hệ thống nhớ hỗ trợ nhiều lệnh đọc ghi đồng thời, hoặc sử dụng các bộ nhớ tiên tiến như bộ nhớ cache.
 - Với xung đột truy cập các thanh ghi, giải pháp là tăng số lượng thanh ghi vật lý và có cơ chế cấp phát thanh ghi linh hoạt khi thực hiện các lệnh

Xung đột dữ liệu

- Tranh chấp dữ liệu kiểu đọc sau khi ghi (RAW – Read After Write) là dạng xung đột dữ liệu hay gặp nhất

Ví dụ: thực hiện 2 lệnh

```
ADD AX,10
```

```
SUB BX,AX
```

- Có thể thấy lệnh SUB sử dụng kết quả của lệnh ADD (thanh ghi AX là kết quả của ADD và là đầu vào cho SUB) và như vậy hai lệnh có sự phụ thuộc dữ liệu.
- Tuy nhiên, lệnh SUB đọc thanh ghi AX tại giai đoạn giải mã (ID), trước khi lệnh ADD ghi kết quả vào thanh ghi AX ở giai đoạn lưu kết quả (WO). Như vậy, giá trị SUB đọc được từ thanh ghi R1 là giá trị cũ, không phải là kết quả tạo ra bởi ADD

ADD AX,10
SUB BX,AX

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

➤ Giải pháp:

- Nhận dạng tranh chấp RAW khi nó diễn ra;
- Khi tranh chấp RAW xảy ra, tạm dừng (stall) ống lệnh cho đến khi lệnh phía trước hoàn tất giai đoạn WB;
- Có thể sử dụng trình biên dịch (compiler) để nhận dạng tranh chấp RAW và thực hiện:
 - ✓ Chèn thêm các lệnh NO-OP vào giữa các lệnh có thể gây ra tranh chấp RAW. NO-OP là lệnh rỗng, không thực hiện tác vụ hữu ích mà chỉ tiêu tốn thời gian CPU.
 - ✓ Thay đổi trật tự các lệnh trong chương trình và chèn các lệnh độc lập vào giữa các lệnh có thể gây ra tranh chấp RAW;
- Sử dụng phần cứng để nhận dạng tranh chấp RAW và dự đoán trước giá trị dữ liệu phụ thuộc

Xung đột điều khiển

- Do việc sử dụng các lệnh rẽ nhánh, theo thống kê tỷ lệ các lệnh rẽ nhánh trong chương trình khoảng 10-30%.
- Lệnh rẽ nhánh thay đổi nội dung của bộ đếm chương trình, chúng có thể phá vỡ tiến trình thực hiện tuần tự các lệnh trong ống lệnh vì lệnh được thực hiện sau lệnh rẽ nhánh có thể không phải là lệnh liền sau nó mà là một lệnh ở vị trí khác
- Ví dụ:

JMP CONG

MOV AX,5

ADD AX,BX

CONG: ADD CX,10

MOV DX,CX

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

➤ **Giải pháp:**

- Sử dụng đích rẽ nhánh (branch targets)
- Làm chậm rẽ nhánh (delayed branching)
- Dự đoán rẽ nhánh (branch prediction)

- **Làm chậm rẽ nhánh (delayed branching)**
 - Lệnh rẽ nhánh sẽ không gây ra sự rẽ nhánh tức thì mà được làm “trễ” một số chu kỳ, phụ thuộc vào chiều dài của ống lệnh.
 - Phương pháp này cho hiệu quả khá tốt với các ống lệnh ngắn, thường là 2 giai đoạn và với ràng buộc lệnh ngay sau lệnh rẽ nhánh luôn được thực hiện, không phụ thuộc vào kết quả của lệnh rẽ nhánh.
 - Cách thực hiện của phương pháp chậm rẽ nhánh là chèn thêm một lệnh NO-OP hoặc một lệnh độc lập vào ngay sau lệnh rẽ nhánh

HẾT CHƯƠNG 3