Bay giảng Kỹ thuật Vi xử lý

Ngành Điện tử-Viễn thông
Đại học Bách khoa Đà Nẵng
của Hồ Viết Việt, Khoa ĐTVT

Tài liệu tham khảo

[1] Kỹ thuật vi xử lý, Văn Thế Minh, NXB Giáo dục, 1997

[2] Kỹ thuật vi xử lý và Lập trình Assembly cho hệ vi xử lý, Đỗ Xuân Tiến, NXB Khoa học & kỹ thuật, 2001

Chương 3

Vi xử lý 8088-Intel

3.1 Kiến trúc và hoạt động của 8088
- Nguyên lý hoạt động
- Sơ đồ khối chức năng

3.2 Cấu trúc thanh ghi của 8088

3.3 Phương pháp quản lý bộ nhớ

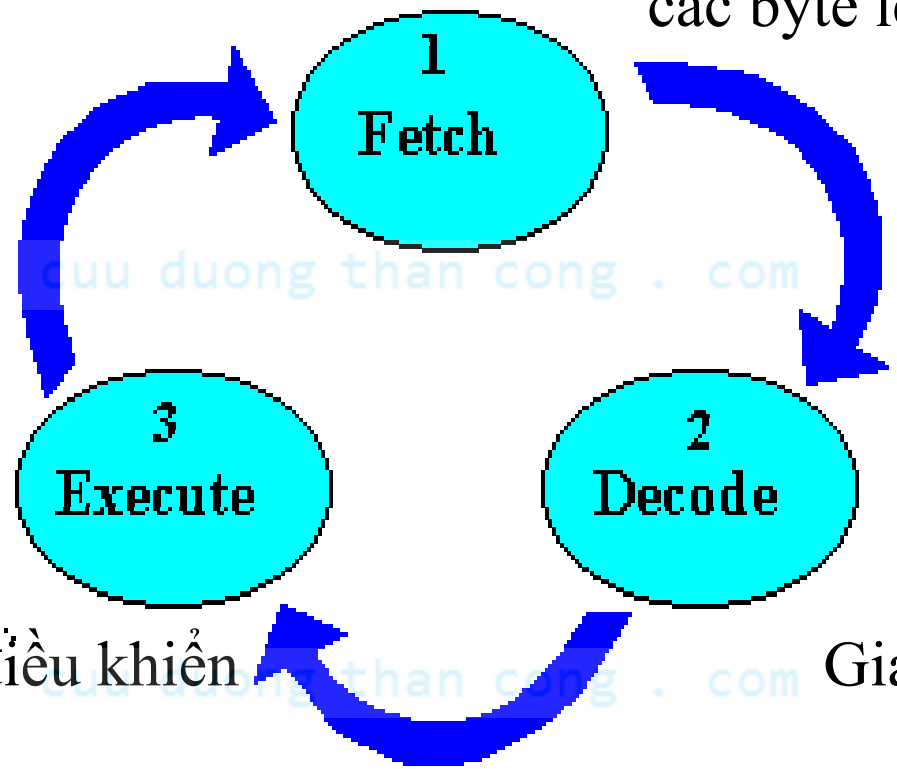
3.4 Mô tả tập lệnh Assembly

Intel® 8088 Chip

Nguyên lý hoạt động của một bộ vi xử lý

Lấy - Giải mã - Thực hiện lệnh

Tìm và copy
các byte lệnh từ bộ nhớ



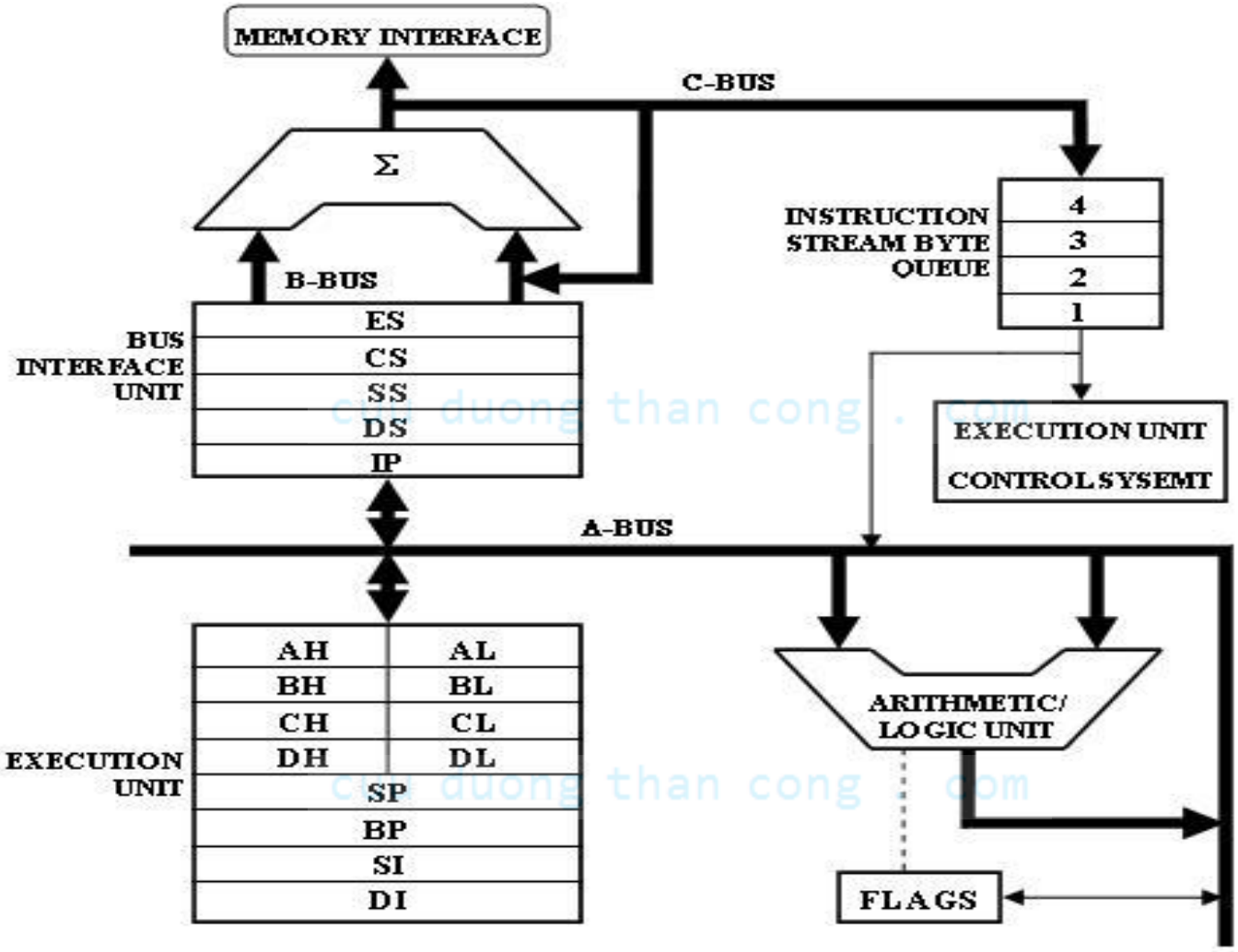
Giải mã lệnh

Tạo ra các tín hiệu điều khiển
để thực hiện lệnh

Chu kỳ lệnh và Chu kỳ máy

- Chu kỳ lệnh: Tổng thời gian tìm lệnh, giải mã lệnh và thực hiện 1 lệnh
- Nói chung, Chu kỳ lệnh của các lệnh khác nhau là khác nhau
- Chu kỳ lệnh bao giờ cũng bằng một số nguyên lần chu kỳ máy
- Chu kỳ máy bằng nghịch đảo của tần số hoạt động (tốc độ đồng hồ) của bộ vi xử lý

3.1 Kiến trúc và Hoạt động của 8088



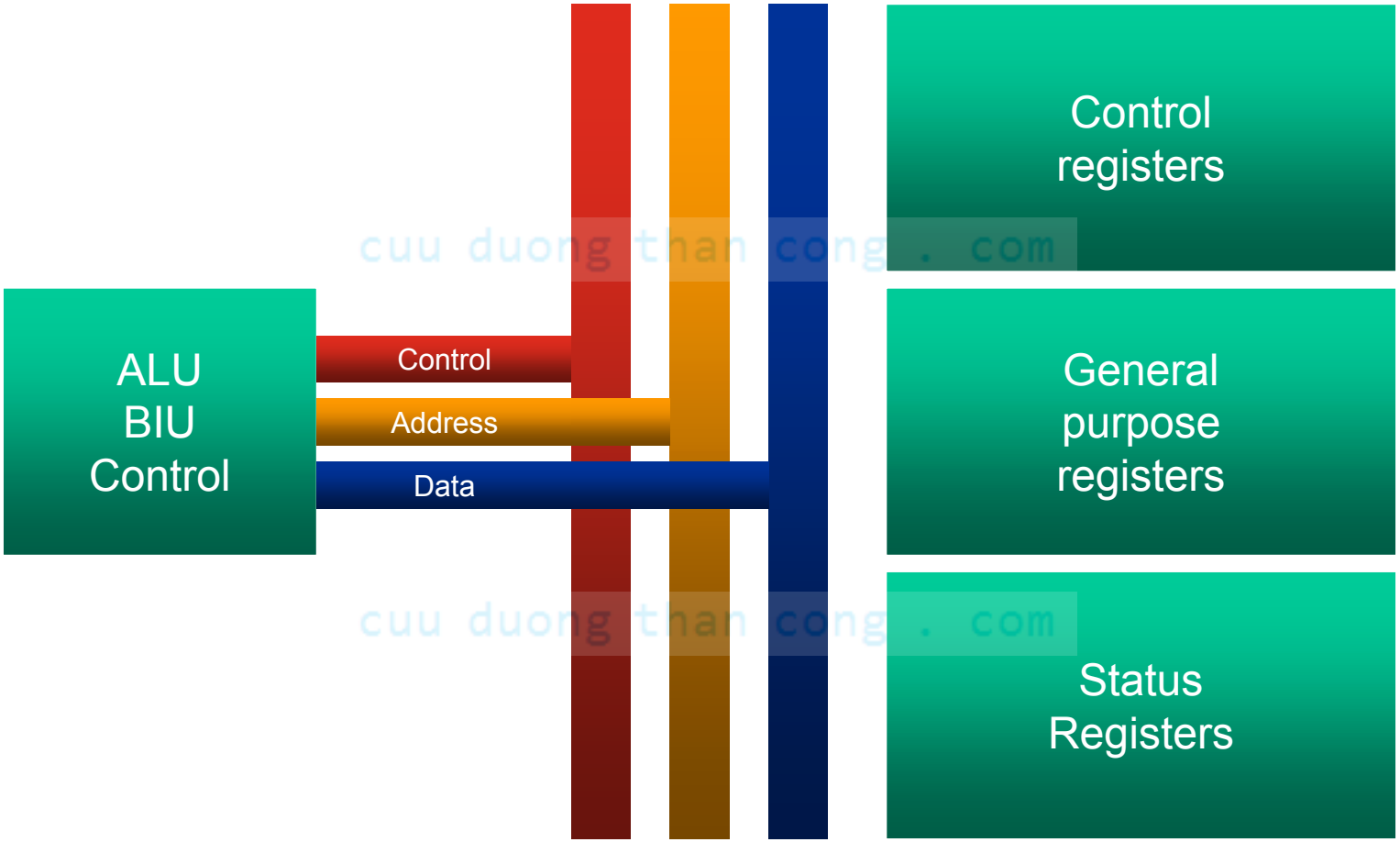
Đơn vị giao tiếp Bus - BIU

- Phát các tín hiệu địa chỉ đến bộ nhớ và các cổng I/O thông qua A-Bus
- Đọc mã lệnh từ bộ nhớ thông qua D-Bus
- Đọc dữ liệu từ bộ nhớ thông qua D-Bus
- Ghi dữ liệu vào bộ nhớ thông qua D-Bus
- Đọc dữ liệu từ các cổng I thông qua D-Bus
- Ghi dữ liệu ra các cổng O thông qua D-Bus

Đơn vị thực hiện - EU

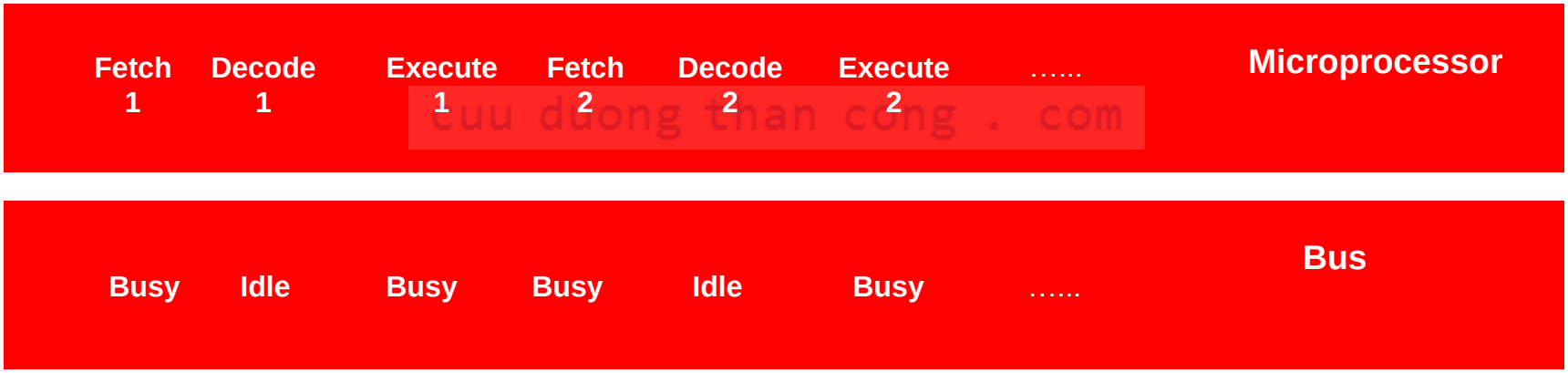
- Bao gồm CU và ALU
- CU : Giải mã lệnh để tạo ra các tín hiệu điều khiển nhằm thực hiện lệnh đã được giải mã
- ALU: thực hiện các thao tác khác nhau đối với các toán hạng của lệnh

Tổ chức của microprocessor

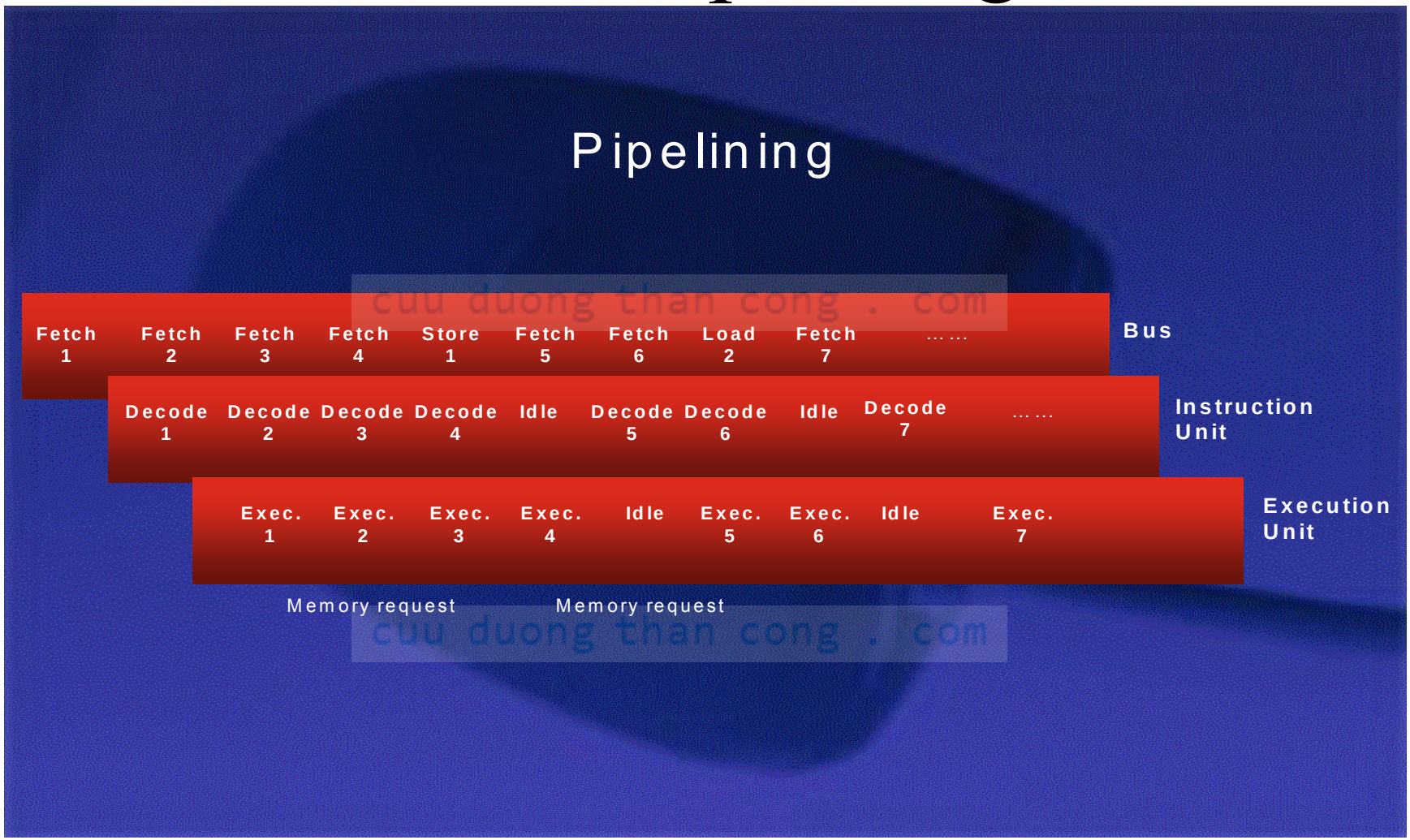


Xử lý lệnh của các vi xử lý trước 8086/8088

- Một thủ tục đơn giản gồm 3 bước:
  - Lấy lệnh từ bộ nhớ
  - Giải mã lệnh
  - Thực hiện lệnh
    - Lấy các toán hạng từ bộ nhớ (nếu có)
    - Lưu trữ kết quả

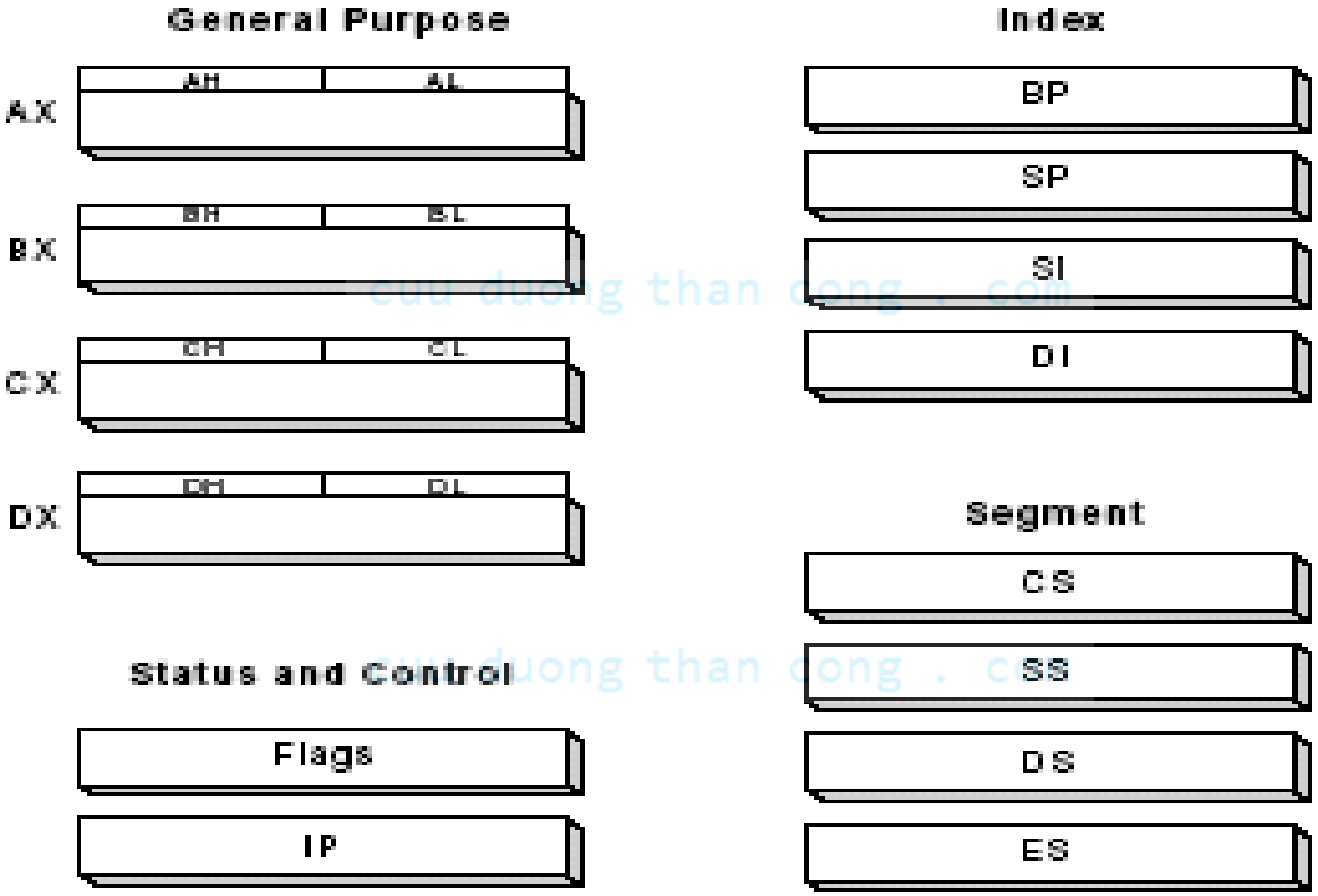


Cơ chế Pipelining

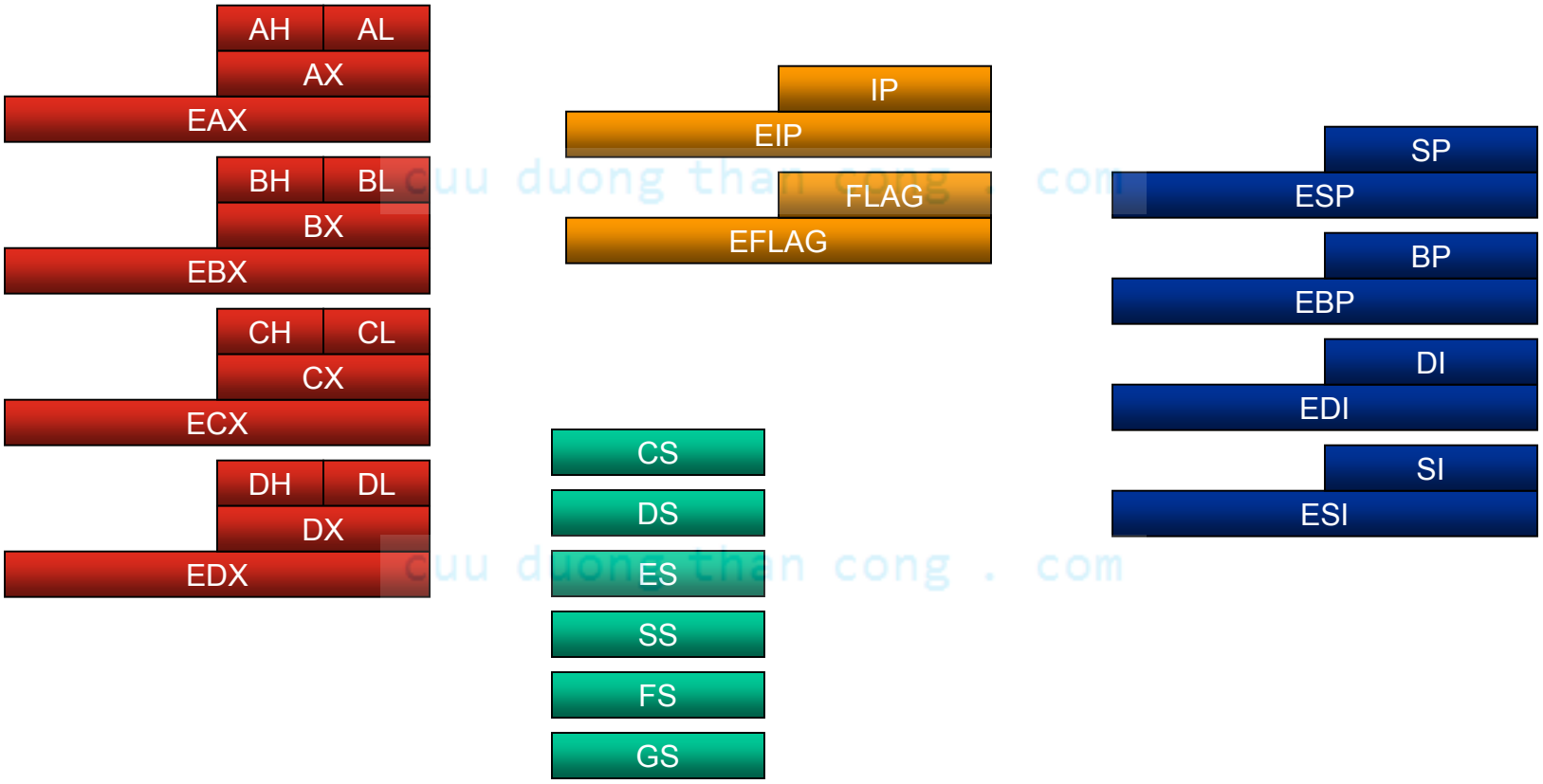


3.2 Cấu trúc thanh ghi của 8088

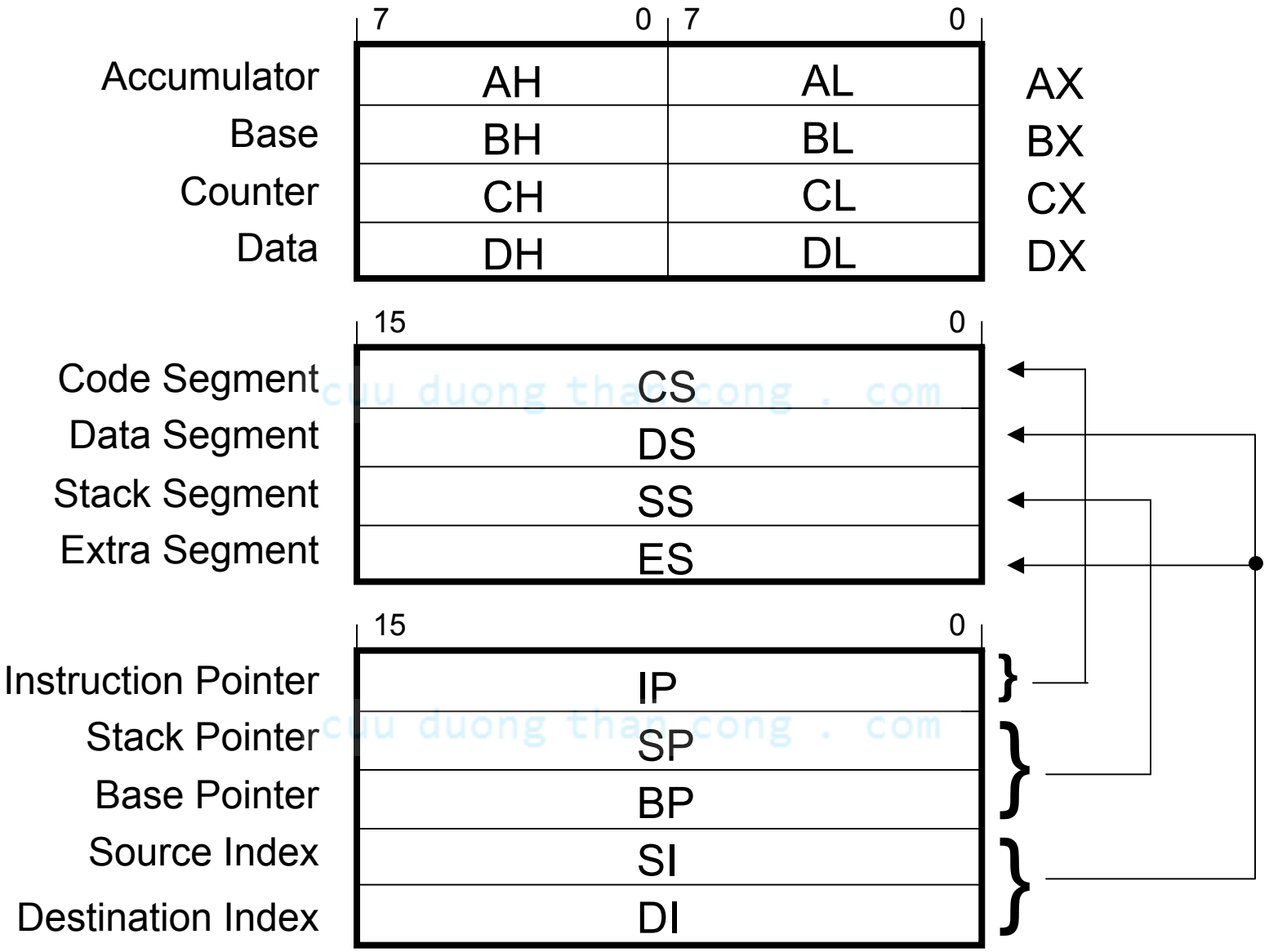
8088 có 14 thanh ghi 16-bit



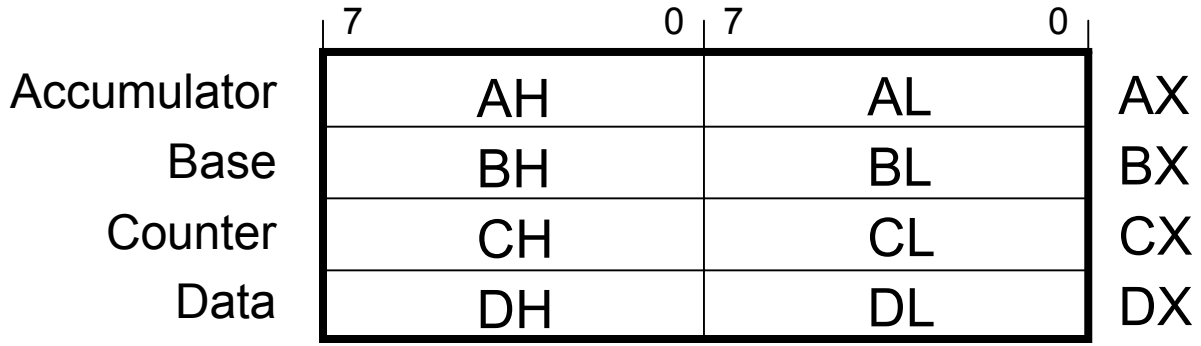
Cấu trúc thanh ghi của họ x86



Cấu trúc thanh ghi 8086/8088

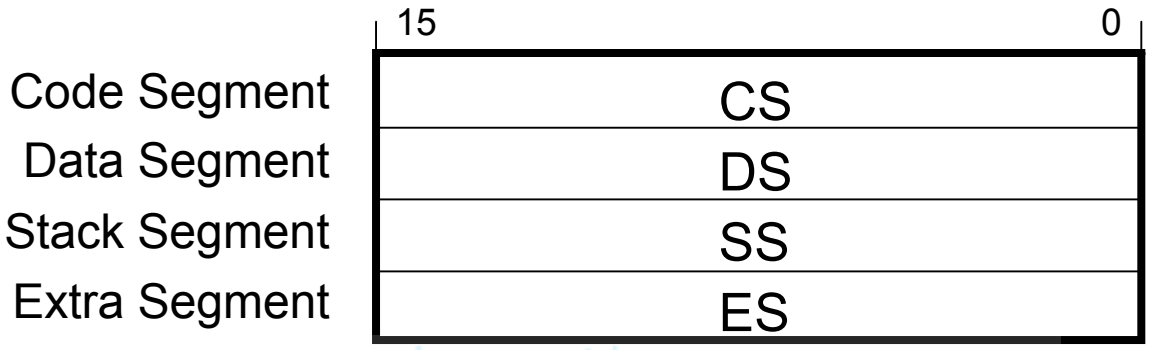


Các thanh ghi đa năng



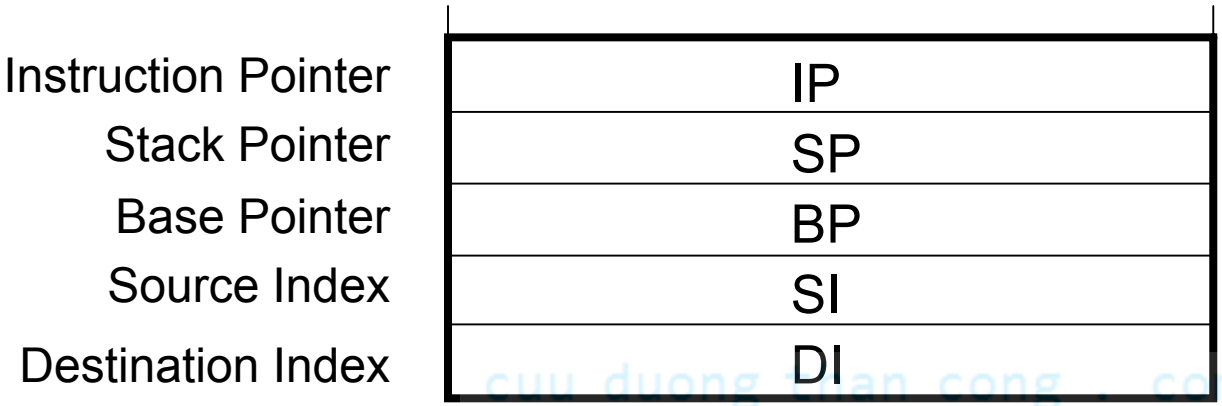
- Có thể truy cập như các thanh ghi 8-bit
- Lưu trữ tạm thời dữ liệu để truy cập nhanh hơn và tránh khỏi phải truy cập bộ nhớ
- Có công dụng đặc biệt đối với một số câu lệnh

Các thanh ghi segment



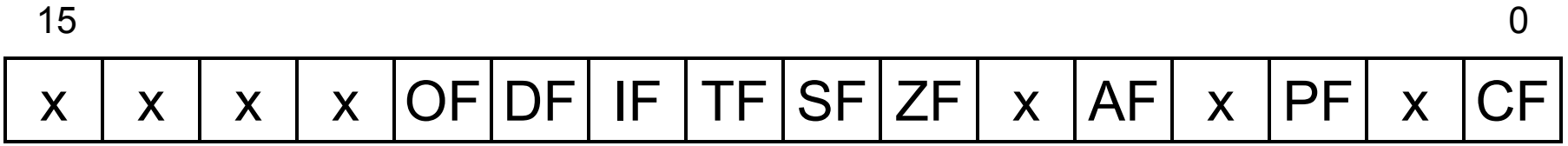
- Lưu trữ địa chỉ segment của một ô nhớ cần truy cập
- Kết hợp với các thanh ghi offset nhất định

Các thanh ghi offset



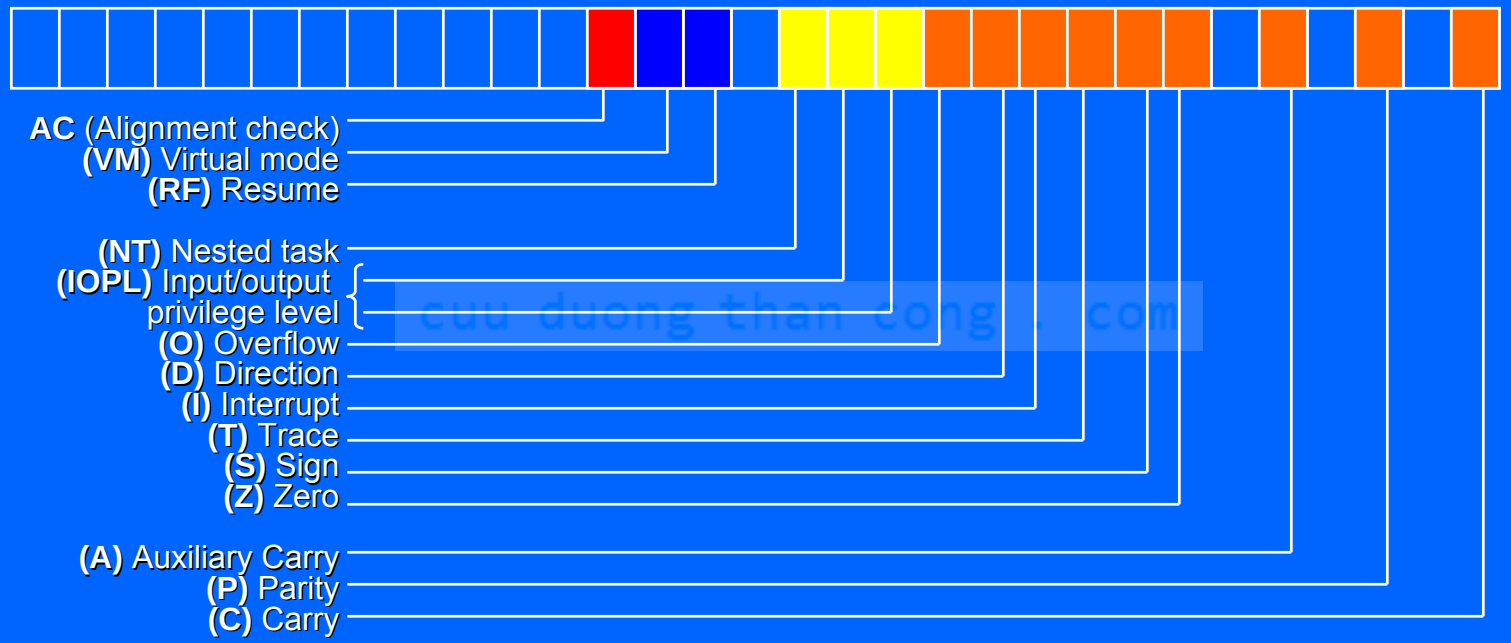
- Lưu trữ địa chỉ offset của một ô nhớ cần truy cập
- Kết hợp với các thanh ghi segment nhất định

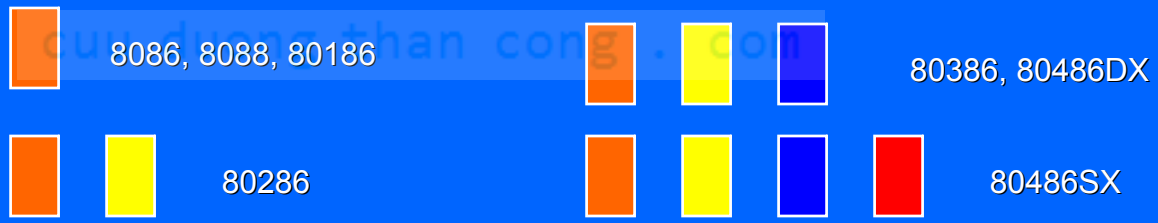
Thanh ghi cờ



- Không phải tất cả các bit đều được sử dụng
- Mỗi bit được sử dụng được gọi là một cờ
- Các cờ đều có tên và có thể được Lập/Xoá riêng lẽ
- Bao gồm các cờ trạng thái và các cờ điều khiển

Flags register

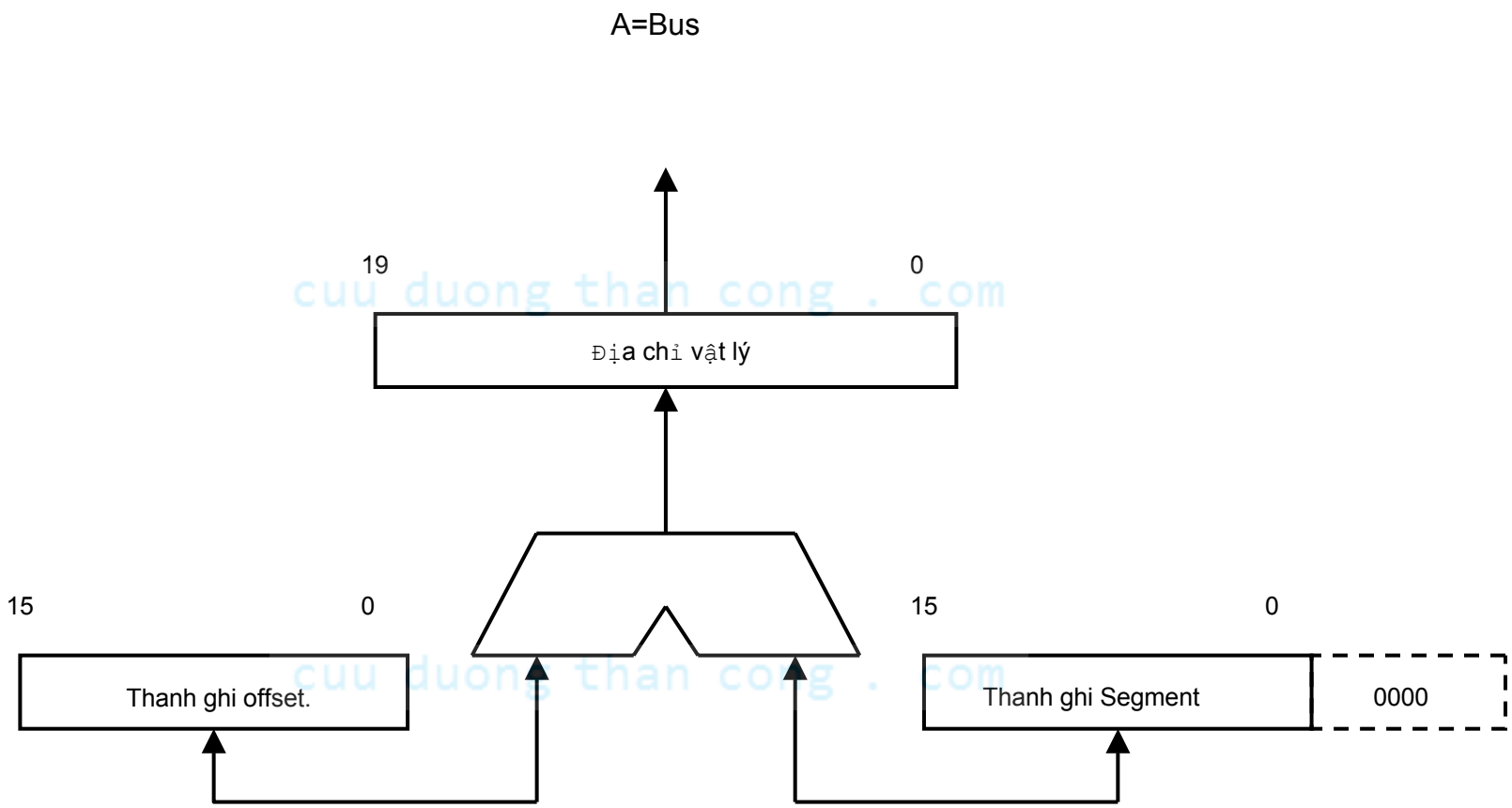




3.3 Phương pháp quản lý bộ nhớ

- Bộ nhớ được xem là một tập hợp các ô nhớ
- Mỗi ô nhớ được nhận dạng bằng một Địa chỉ vật lý duy nhất 20-bit
- Trong hoạt động truy cập một ô nhớ, Địa chỉ vật lý của nó được tạo ra từ hai giá trị 16-bit: Địa chỉ segment và Địa chỉ Offset
- Địa chỉ logic = Địa chỉ segment:Địa chỉ offset

Mối liên hệ giữa ĐCVL và ĐCLG



3.4 Mô tả tập lệnh Assembly của 8086/8088

- Khuôn dạng: Mnemonics Các toán hạng
- Nhóm lệnh chuyển số liệu
- Nhóm lệnh số học
- Nhóm lệnh logic
- Nhóm lệnh Rẽ nhánh
- Nhóm lệnh thao tác string
- Nhóm lệnh hỗn hợp

Nhóm lệnh chuyển số liệu
Data Transfer Instructions

-Chuyển số liệu (sao chép số liệu) từ vị trí này sang vị trí khác

- Nguồn số liệu không thay đổi
- Đích sẽ có giá trị như giá trị của Nguồn
- Các lệnh chuyển số liệu không ảnh hưởng đến các cờ trạng thái trên thanh ghi cờ
- Một số lệnh tiêu biểu: MOV, XCHG

Data Transfer Instructions - MOV

Khuôn dạng: MOV Đích,Nguồn

- Tác dụng: (Đích) ← (Nguồn)
- Đích: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

- Nguồn: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

3. Một giá trị cụ thể

Một số lưu ý đối với MOV

- Đích và Nguồn phải có cùng kích cỡ
- Đích và Nguồn không thể đồng thời thuộc bộ nhớ
- Nếu Đích là một thanh ghi segment của VXL thì Nguồn không thể là một giá trị cụ thể (nói cách khác, không thể nạp giá trị trực tiếp cho một thanh ghi segment bằng lệnh MOV)

Data Transfer Instructions - XCHG

Khuôn dạng: XCHG T/h1,T/h2

- Tác dụng: (T/h1) ← (T/h2)
- T/h1: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

- T/h2: có thể là:

1. *Một thanh ghi 8 hoặc 16 bit của VXL*
2. *Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)*

Một số lưu ý đối với XCHG

- T/h1 và T/h2 phải có cùng kích cỡ
- T/h1 và T/h2 không thể đồng thời thuộc bộ nhớ
- T/h1 và T/h2 không thể là các thanh ghi segment

Các mode địa chỉ

- Khi thực hiện lệnh, VXL sẽ thực hiện những thao tác nhất định trên số liệu, các số liệu này được gọi chung là các toán hạng.
- Các toán hạng trong một câu lệnh có thể là một phần của câu lệnh (ở dạng mã máy), có thể nằm ở một thanh ghi của VXL hoặc ở Bộ nhớ
-Cách xác định toán hạng trong các câu lệnh được gọi là các mode (định) địa chỉ

Các mode địa chỉ

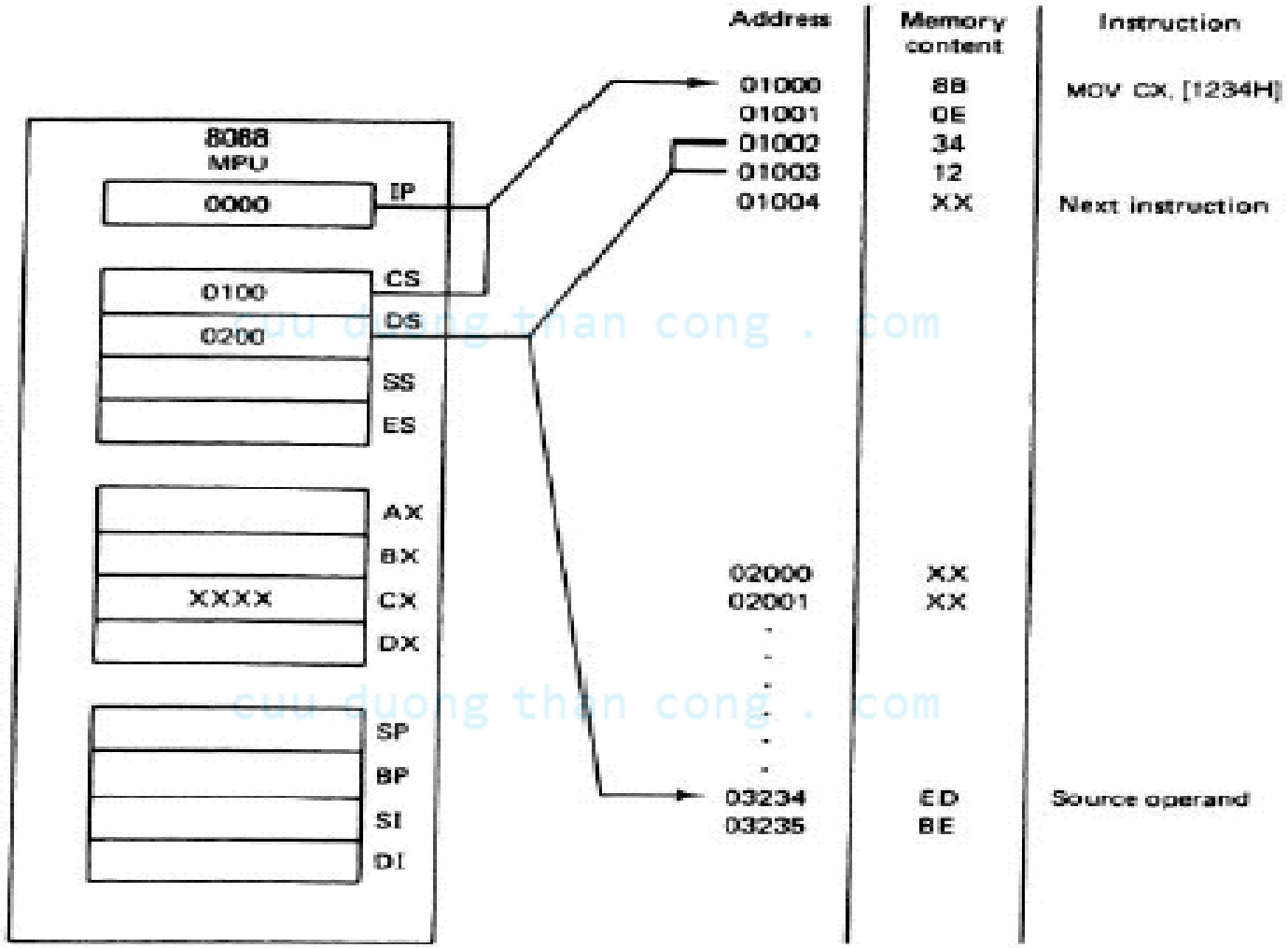
- Mode địa chỉ thanh ghi: MOV AX,BX
- Mode địa chỉ tức thì: MOV AL,55h
- Các mode địa chỉ bộ nhớ: Các cách thức xác định địa chỉ vật lý của toán hạng nằm trong bộ nhớ:

    Mode địa chỉ trực tiếp

    Các mode địa chỉ gián tiếp …

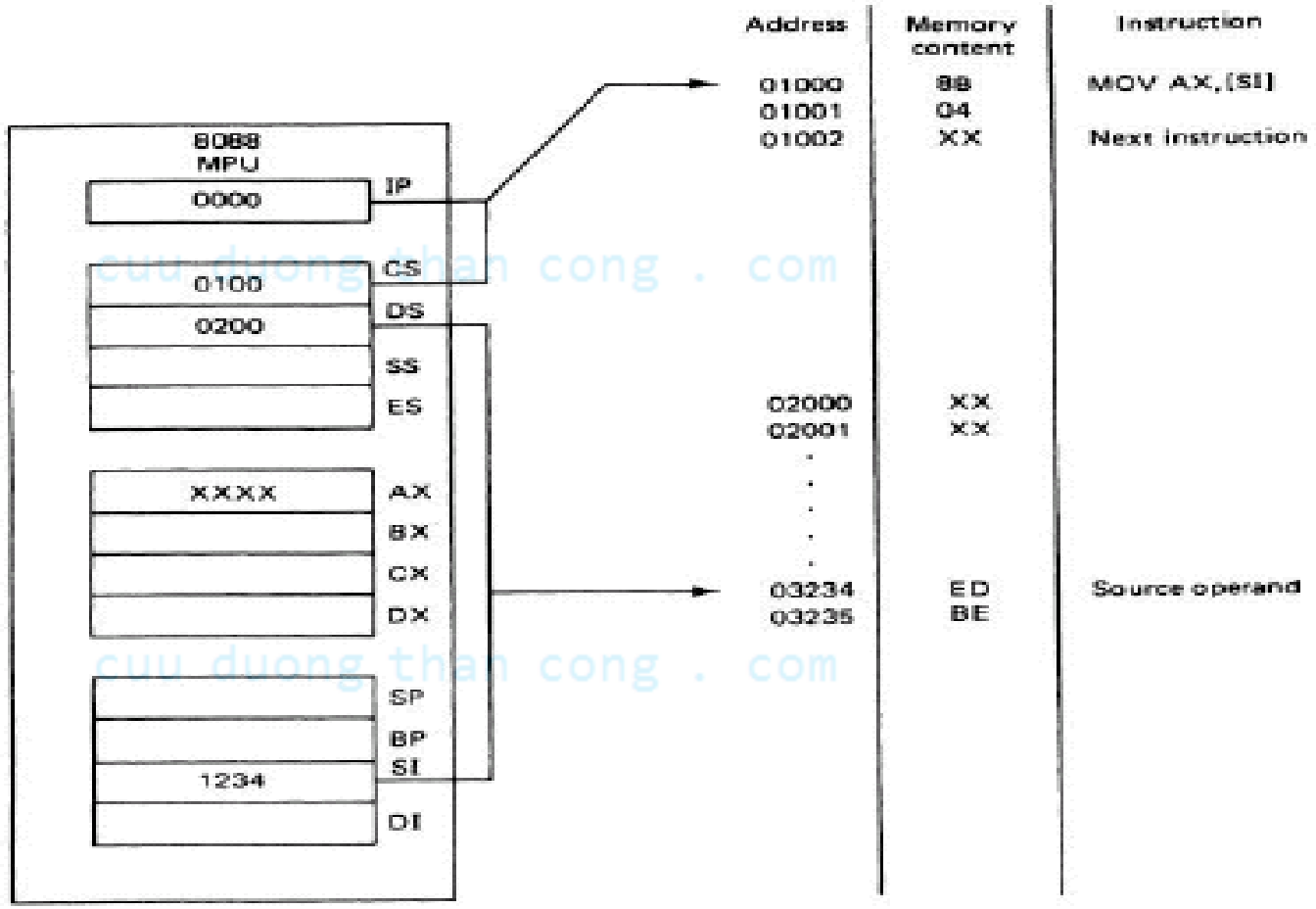
Mode địa chỉ trực tiếp
(**Direct Addressing Mode**)

MOV CX, [1234h]



Mode địa chỉ gián tiếp thanh ghi
(Register Indirect Addressing Mode)

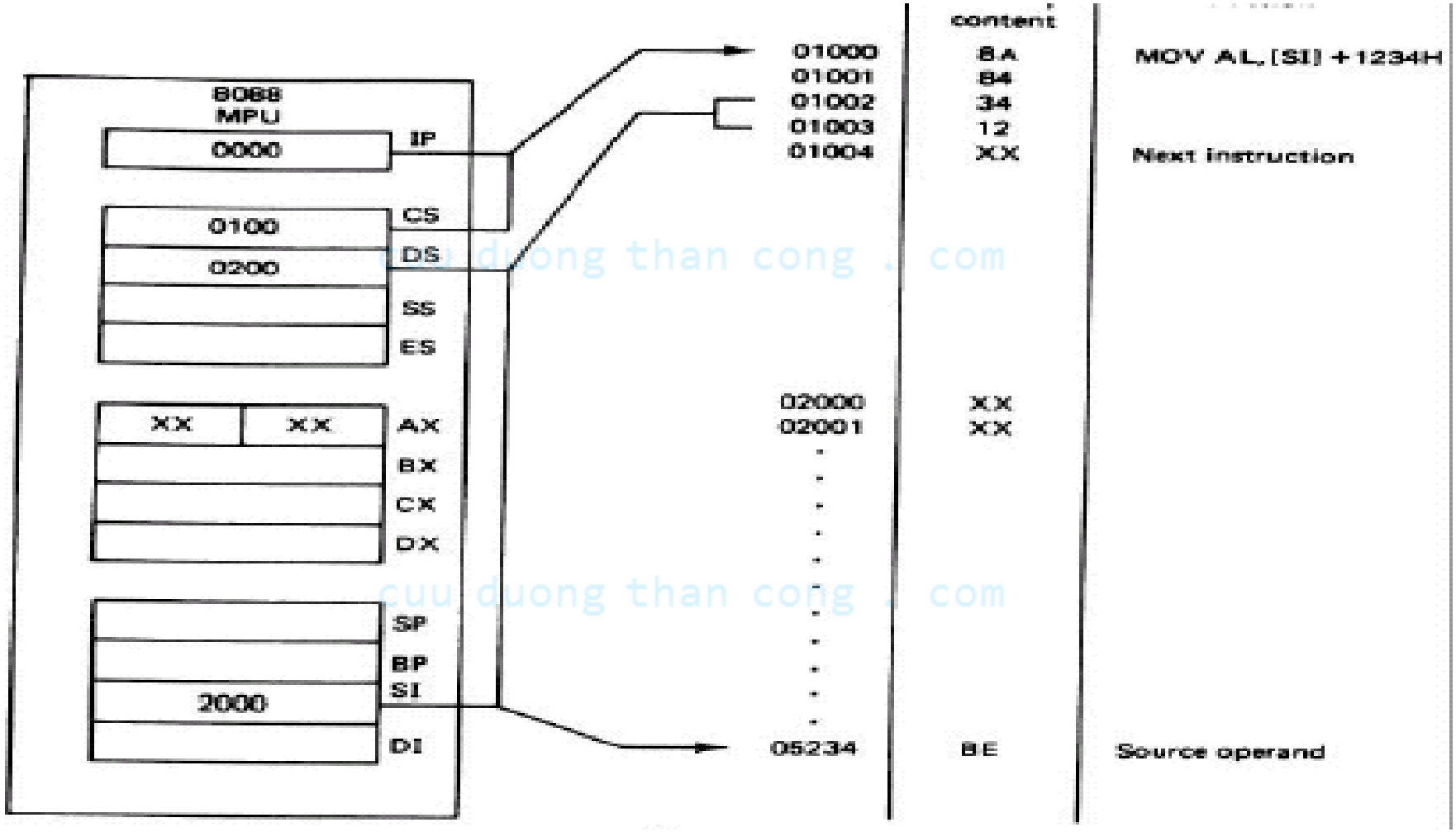
MOV AX, [SI]



Mode địa chỉ cơ sở-chỉ số

(Based-Indexed Addressing Mode)

MOV AH, [BX] [SI] + 1234h



Nhớ các mode địa chỉ bộ nhớ như thế nào?

- Tất cả bắt đầu trong bảng sau đây:

- Lấy ra 0 hoặc 1 phần tử từ mỗi cột
- (Không lấy 2 phần tử từ một cột)
- Phải lấy ít nhất 1 phần tử từ bảng

Các ví dụ

Instruction	Comment	Addressing Mode	Memory Contents
MOV AX, BX	Move to AX the 16-bit value in BX	Register	89 D8 OP MODE
MOV AX, DI	Move to AX the 16-bit value in DI	Register	89 F8 OP MODE
MOV AH, AL	Move to AL the 8-bit value in AX	Register	88 C4 OP MODE
MOV AH, 12h	Move to AH the 8-bit value 12H	Immediate	B4 12 OP DATA8
MOV AX, 1234h	Move to AX the value 1234h	Immediate	B8 34 OP DATA16
MOV AX, CONST	Move to AX the constant defined as CONST	Immediate	B8 lsb msb OP DATA16
MOV AX, X	Move to AX the address or offset of the variable X	Immediate	B8 lsb msb OP DATA16
MOV AX, [1234h]	Move to AX the value at memory location 1234h	Direct	A1 34 12 OP DISP16
MOV AX, [X]	Move to AX the value in memory location DS:X	Direct	A1 lsb msb OP DISP16

Các ví dụ

Instruction	Comment	Addressing Mode	Memory Contents
MOV [X], AX	Move to the memory location pointed to by DS:X the value in AX	Direct	A3 lsb msb OP DATA16
MOV AX, [DI]	Move to AX the 16-bit value pointed to by DS:DI	Indexed	8B 05 OP MODE
MOV [DI], AX	Move to address DS:DI the 16-bit value in AX	Indexed	89 05 OP MODE
MOV AX, [BX]	Move to AX the 16-bit value pointed to by DS:BX	Register Indirect	8B 07 OP MODE
MOV [BX], AX	Move to the memory address DS:BX the 16-bit value stored in AX	Register Indirect	89 07 OP MODE
MOV [BP], AX	Move to memory address **SS:BP** the 16-bit value in AX	Register Indirect	89 46 OP MODE
MOV AX, TAB[BX]	Move to AX the value in memory at DS:BX + TAB	Register Relative	8B 87 lsb msb OP MODE DISP16
MOV TAB[BX], AX	Move value in AX to memory address DS:BX + TAB	Register Relative	89 87 lsb msb OP MODE DISP16
MOV AX, [BX + DI]	Move to AX the value in memory at DS:BX + DI	Base Plus Index	8B 01 OP MODE

Các ví dụ

Instruction	Comment	Addressing Mode	Memory Contents
MOV [BX + DI], AX	Move to the memory location pointed to by DS:X the value in AX	Base Plus Index	89 01 OP MODE
MOV AX, [BX + DI + 1234h]	Move word in memory location DS:BX + DI + 1234h to AX register	Base Rel Plus Index	8B 81 34 12 OP MODE DISP16
MOV word [BX + DI + 1234h], 5678h	Move immediate value 5678h to memory location BX + DI + 1234h	Base Rel Plus Index	C7 81 34 12 78 56

Mã máy

Một lệnh có thể dài từ1 đến 6 byte

• Byte 1 gồm:

– Opcode (6 bit) xác định phép toán cần thực hiện

– Bit D xác định toán hạng ở REG của Byte 2 là nguồn hay đích:

1: Đích

0: Nguồn

– Bit W xác định kích cỡ của toán hạng là 8 bit hay 16 bit

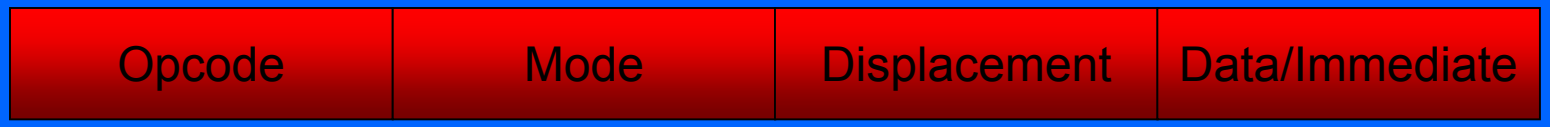
0: 8 bit

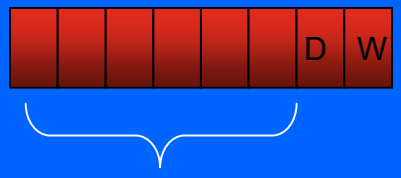
1: 16 bit

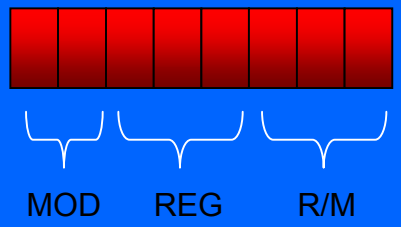
• Byte 2 gồm:Mode field (MOD), Register field (REG)

Register/memory field (R/M field)

Anatomy of an instruction



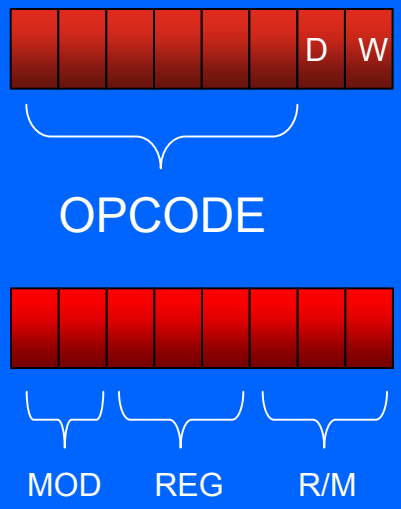




- Opcode contains the type of instruction we execute plus two special bits, D and W
- The mode byte is used only in instructions that use register addressing modes and encodes the source and destination for instructions with two operands
- D stands for direction and defines the data flow of the instruction
  - D=0, data flows from REG to R/M
  - D=1, data flows from R/M to REG
- W stands for the size of data
  - W=0, byte-sized data
  - W=1, word (in real mode) or double-word sized (in protected mode)

Anatomy of an instruction

Opcode	Mode	Displacement	Data/Immediate



- MOD field specifies the addressing mode
- 00 – no displacement
- 01 – 8-bit displacement, sign extended
- 10 – 16-bit displacement
- 11 – R/M is a register, register addressing mode
- If MOD is 00,01, or 10, the R/M field selects one of the memory addressing modes

Registers in the REG and R/M fields

Code	W=0 (Byte)	W=1 (Word)	W=1 (DWord)
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Example

- Consider the instruction 8BECh
- 1000 1011 1110 1100 binary
- Opcode 100010 -> MOV
- D=1 data goes from R/M to REG
- W=1 data is word-sized
- MOD=11, register addressing
- REG=101 destination, R/M=100 source
- MOV BP, SP

Code	W=0	W=1	W=1
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Displacement addressing

- If MOD is 00, 01, or 10 R/M has an entirely different meaning

MOD	FUNCTION
00	No displacement
01	8-bit sign-extended displacement
10	16-bit displacement
11	R/M is a register (register addressing mode)

R/M Code	Function
000	DS:BX+SI
001	DS:BX+DI
010	SS:BP+SI
011	SS:BP+DI
100	DS:SI
101	DS:DI
110	SS:BP
111	DS:BX

Examples:

If MOD=00 and R/M=101 mode is [DI]

If MOD=01 and R/M=101 mode is [DI+33h]

If MODE=10 and R/M=101 modes is [DI+2233h]

Example

- Instruction 8A15h
- 1000 1010 0001 0101
- Opcode 100010 -> MOV
- D=1, data flows from R/M to REG
- W=0, 8-bit argument
- MOD=00 (no displacement)
- REG=010 (DL)
- REG=101 ([DI] addressing mode)
- MOV DL, [DI]

Code	W=0	W=1	W=1
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

R/M Code	Function
000	DS:BX+SI
001	DS:BX+DI
010	SS:BP+SI
011	SS:BP+DI
100	DS:SI
101	DS:DI
110	SS:BP
111	DS:BX

Direct Addressing Mode

- MOD is always 00
- R/M is always 110
- REG encodes the register to/from we take data as usual
- Third byte contains the lower-order bytes of the displacement, fourth byte contains the high order byte of the displacement

Direct Addressing

- Example: 8816 00 10
- 1000 1000 0001 0110 0000 0000 0001 0000
- Opcode 100010 -> MOV
- W=0 (byte-sized data)
- D=0 data flows from REG
- MOD 00, REG=010 (DL), R/M=110
- Low-order byte of displacement 00
- High-order byte of displacement 10
- MOV [1000h], DL

Code	W=0	W=1	W=1
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Segment MOV instructions

- Different opcode 100011
- Segments are selected by setting the REG field

REG Code	Segment reg.
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS

Example MOV BX, CS

Opcode 10001100

MOD=11 (register addressing)

REG=001 (CS)

R/M=011 (BX)

8CCB

Mã máy

REG xác định thanh ghi cho toán hạng thứ nhất

REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Mã máy

MOD và R/M cùng nhau xác định toán hạng thứ hai

CODE	EXPLANATION
00	Memory Mode, no displacement follows*
01	Memory Mode, 8-bit displacement follows
10	Memory Mode, 16-bit displacement follows
11	Register Mode (no displacement)

*Except when R/M = 110, then 16-bit displacement follows

(a)

Mã máy

MOD và R/M cùng nhau xác định toán hạng thứ hai

MOD = 11			EFFECTIVE ADDRESS CALCULATION			
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

(b)

Ví dụ

Mã hoá lệnh MOV BL,AL

- Opcode đối với MOV là 100010
- Ta mã hoá AL sao cho AL là toán hạng nguồn:

– D = 0 (AL là toán hạng nguồn)

- W bit = 0 (8-bit)
- MOD = 11 (register mode)
- REG = 000 (mã của AL)
- R/M = 011 (mã của BL)

Kết quả:: 10001000 11000011 = 88 C3

Nhóm lệnh Số học

- Bên cạnh tác dụng, cần chú ý đến ảnh hưởng của lệnh đối với các cờ trạng thái
- Các lệnh số học th/thường: ADD, SUB, …
- Các lệnh số học khác: CMP. NEG, INC, DEC, …
- Ảnh hưởng đến các cờ trạng thái
  - CF
  - OF Phụ thuộc vào quá trình thực hiện phép toán
  - AF

  - ZF = 1 nếu Kết quả bằng 0
  - SF = 1 nếu MSB của Kết quả = 1
  - PF = 1 nếu byte thấp của kết quả có Parity chẵn

Arithmetic Instructions - ADD

Khuôn dạng: ADD Đích,Nguồn

- Tác dụng: (Đích) ← (Đích)+(Nguồn)
- Đích: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

- Nguồn: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

3. Một giá trị cụ thể

Ảnh hưởng của ADD

- ZF = 1 nếu Kết quả bằng 0
- SF = 1 nếu MSB của Kết quả = 1
- PF = 1 nếu byte thấp của kết quả có Parity chẵn

- CF được lập nếu tràn không dấu (có nhớ từ MSB)
- OF được lập nếu tràn có dấu:
  - Có nhớ từ MSB, Không có nhớ vào MSB
  - Có nhớ vào MSB, Không có nhớ từ MSB
- AF được lập nếu có nhớ từ nibble thấp vào nibble cao (từ bit 3 vào bit 4)

Các cờ trên thanh ghi cờ

- Các bit nhất định trên thanh ghi cờ điều khiển hoạt động hoặc phản ánh trạng thái của vi xử lý
  - Các cờ điều khiển (TF, IF, DF)
    - Quyết định cách đáp ứng của vi xử lý trong các tình huống nhất định
  - Các cờ trạng thái (CF, PF, AF, ZF, SF, OF)
    - Bị ảnh hưởng bởi các phép toán nhất định
    - Phục vụ cho các lệnh có điều kiện

Các cờ điều khiển

- DF - Direction flag (Cờ hướng)
  - DF = 1: hướng xuống
  - DF = 0: hướng lên
- IF – Interrupt flag (Cờ ngắt)
  - IF = 1: cho phép ngắt ngoài
  - IF = 0: cấm ngắt ngoài (đối với ngắt che được)
- TF - Trace flag
  - TF = 1: vi xử lý thực hiện từng lệnh một

Các cờ trạng thái

- Carry
  - carry or borrow at MSB in add or subtract
  - last bit shifted out
- Parity
  - low byte of result has even parity
- Auxiliary
  - carry or borrow at bit 3
- Zero
  - result is 0
- Sign
  - result is negative
- Overflow
  - signed overflow occurred during add or subtract

(Signed) Overflow

- Can only occur when adding numbers of the same sign (subtracting with different signs)
- Detected when carry into MSB is not equal to carry out of MSB
  - Easily detected because this implies the result has a different sign than the sign of the operands
- Programs can ignore the Flags!

Signed Overflow Example

```
  10010110
+ 10100011
  00111001
```

Carry in = 0, Carry out = 1

Neg+Neg=Pos

Signed overflow occurred

OF = 1 (set)

```
  00110110
+ 01100011
  10011001
```

Carry in = 1, Carry out = 0

Pos+Pos=Neg

Signed overflow occurred

OF = 1 (set)

Examples of No Signed Overflow

```
  10010110
+ 01100011
  11111001
```

Carry in = 0, Carry out = 0

Neg+Pos=Neg

No Signed overflow occurred

OF = 0 (clear)

```
  10010110
+ 11110011
  10001001
```

Carry in = 1, Carry out = 1

Neg+Neg=Neg

No Signed overflow occurred

OF = 0 (clear)

Unsigned Overflow

- The carry flag is used to indicate if an unsigned operation overflowed
- The processor only adds or subtracts - it does not care if the data is signed or unsigned!

```
  10010110
+ 11110011
----------
  10001001
```

Carry out = 1

Unsigned overflow occurred

CF = 1 (set)

DEBUG's Register Display

```
-R
…000 SP=0010 BP=0000 SI=0000 DI=0000
…00F IP=004F NV UP DI PL NZ NA PO NC
```


- The state of the Flags are shown in line 2
- **OV/NV: (no)oVerflow DN/UP: direction**
- **EI/DI: En(Dis)abled Interrupts**
- **NG/PL: sign ZR/NZ: (not)Zero**
- **AC/NA: (no)Auxiliary PE/PO: Even/Odd**
- **CY/NC: (no)Carry *(set/clear)***

Arithmetic Instructions - SUB

Khuôn dạng: SUB Đích,Nguồn

- Tác dụng: (Đích) ← (Đích)-(Nguồn)
- Đích: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

- Nguồn: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

3. Một giá trị cụ thể

Ảnh hưởng của SUB

- ZF = 1 nếu Kết quả bằng 0
- SF = 1 nếu MSB của Kết quả = 1
- PF = 1 nếu byte thấp của kết quả có Parity chẵn

- CF được lập nếu tràn không dấu (có mượn vào MSB)
- OF được lập nếu tràn có dấu:
  - Có mượn từ MSB, Không có mượn từ MSB
  - Có mượn từ MSB, Không có mượn vào MSB
- AF được lập nếu có mượn từ nibble cao vào nibble thấp (từ bit 4 vào bit 3)

Arithmetic Instructions - CMP

Khuôn dạng: CMP Đích,Nguồn

- Tác dụng: (Đích)-(Nguồn)
- Đích: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

- Nguồn: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

3. Một giá trị cụ thể

Arithmetic Instructions – INC, DEC, NEG

- INC T/h
- Trong đó: T/h có thể là các thanh ghi hoặc vị trí nhớ
- Tác dụng: (T/h) ← (T/h)+1

- DEC T/h
- Trong đó: T/h có thể là các thanh ghi hoặc vị trí nhớ
- Tác dụng: (T/h) ← (T/h)-1
- Lưu ý: Các lệnh INC và DEC không ảnh hưởng đến cờ CF
- Lệnh NEG T/h: Đảo dấu của T/h (Lấy bù 2)
- *Lệnh NEG sẽ lập cờ OF nếu giá trị của T/h là giá trị âm nhất trong dải giá trị của các số có dấu tương ứng*

Nhóm lệnh Logic

- Cần chú ý đến ảnh hưởng của lệnh đối với các cờ trạng thái
- Các lệnh logic th/thường: NOT, AND, OR, XOR

NOT A: ~A

AND A,B: A &= B

OR A,B : A |= B

XOR A,B: A ^= B

- NOT không ảnh hưởng đến các cờ trạng thái.
- Các lệnh khác:
  - CF = 0
  - OF = 0
  - ZF = 1 nếu Kết quả bằng 0
  - SF = 1 nếu MSB của Kết quả = 1
  - PF = 1 nếu byte thấp của kết quả có Parity chẵn
  - AF không xác định

Một số ví dụ

AL	1100 1010
NOT AL	
AL	0011 0101

AL	0011 0101
BL	0110 1101
AND AL, BL	
AL	0010 0101

AL	0011 0101
BL	0000 1111
AND AL, BL	
AL	0000 0101

AL	0011 0101
BL	0110 1101
OR AL, BL	
AL	0111 1101

AL	0011 0101
BL	0000 1111
OR AL, BL	
AL	0011 1111

AL	0011 0101
BL	0110 1101
XOR AL, BL	
AL	0101 1000

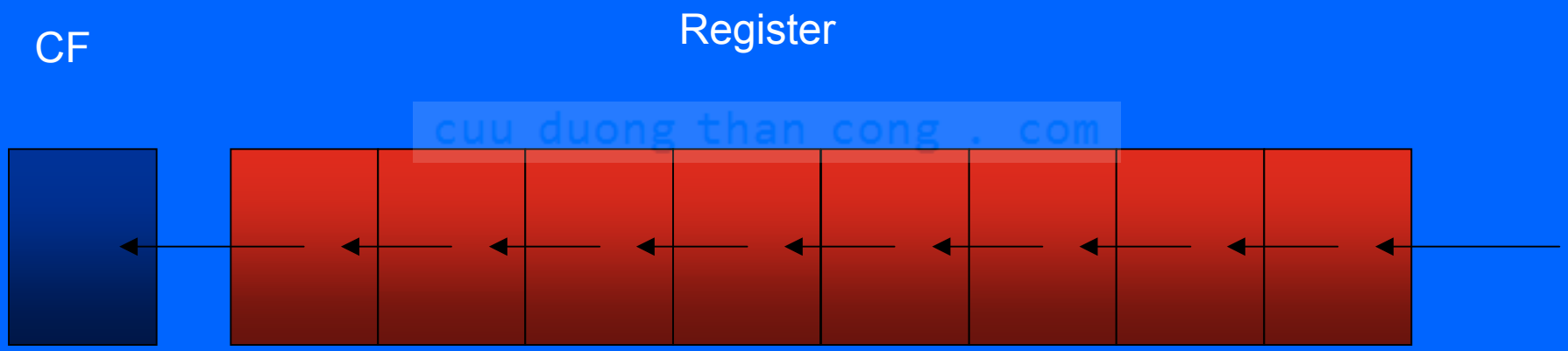
Một số ứng dụng

- Bài toán Xoá bit: Xoá một bit nào đó của một toán hạng mà không làm ảnh hưởng đến các bit còn lại của toán hạng đó
- Bài toán Kiểm tra bit: Xác định một bit nào đó của một toán hạng là bằng 0 hay 1 (thông qua giá trị của một cờ trạng thái)
- Bài toán Lập bit: Lập một bit nào đó của một toán hạng mà không làm ảnh hưởng đến các bit còn lại của toán hạng đó

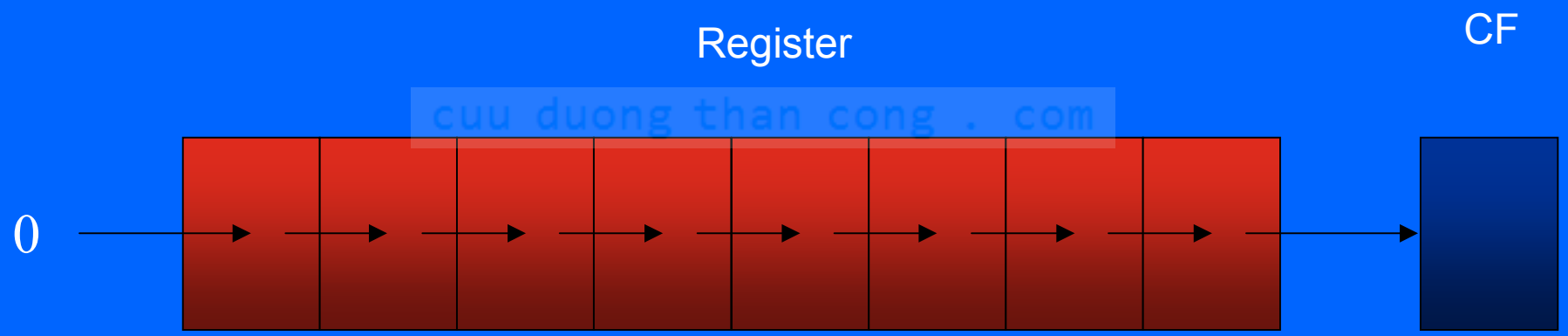
Nhóm lệnh logic

- Các lệnh logic khác: Lệnh TEST, Các lệnh dịch (Shift) và Các lệnh quay (Rotate)
- Lệnh TEST chỉ khác lệnh AND là không giữ lại kết quả của phép toán
- Các lệnh dịch và Các lệnh quay đều có hai khuôn dạng:
  Khuôn dạng 1: Mnemonic Toán hạng,1
  Khuôn dạng 2: Mnemonic Toán hạng,CL
- Tác dụng của một câu lệnh theo khuôn dang 2 giống như tác dụng liên tiếp của N câu lệnh tương ứng theo khuôn dạng 1, với N là giá trị của thanh ghi CL

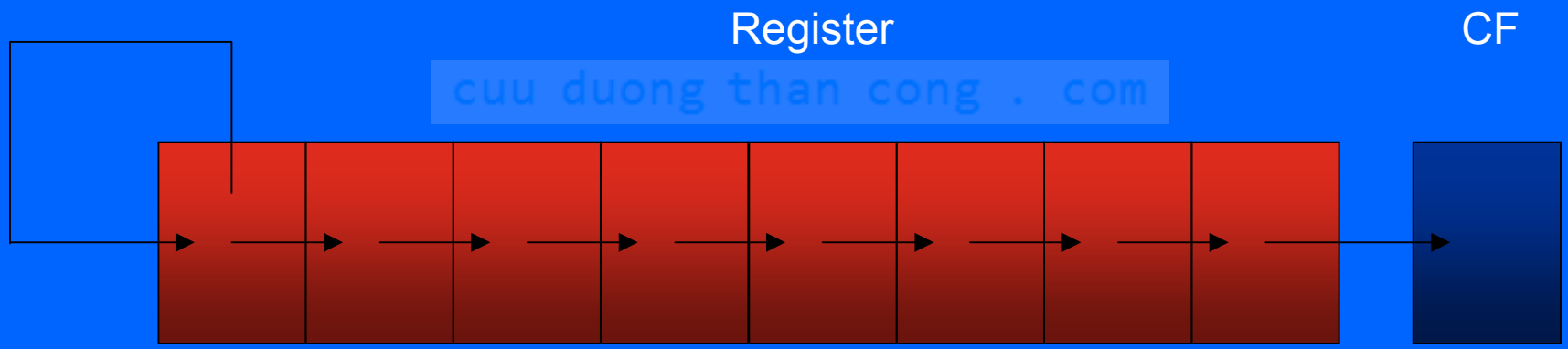
Các lệnh Dịch trái: SHL, SAL



Shift right SHR

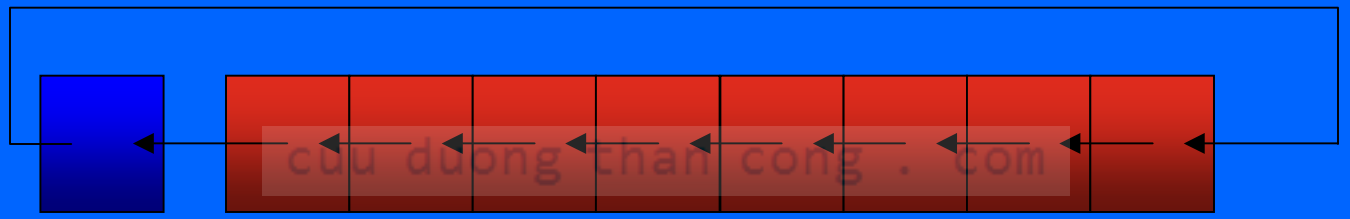


Shift right SAR

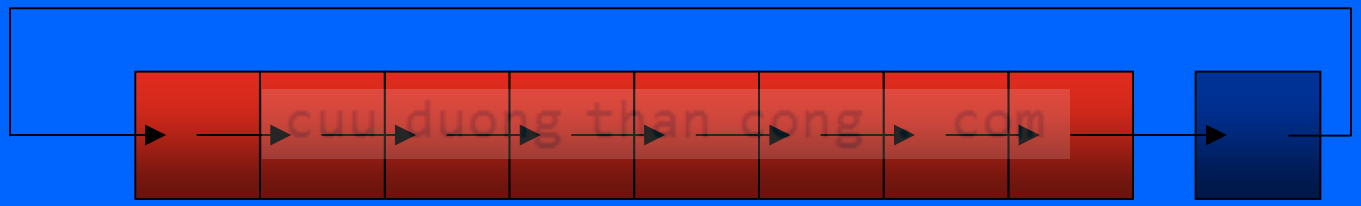


Rotate through Carry L/R
(Quay trái/phải thông qua carry)

RCL

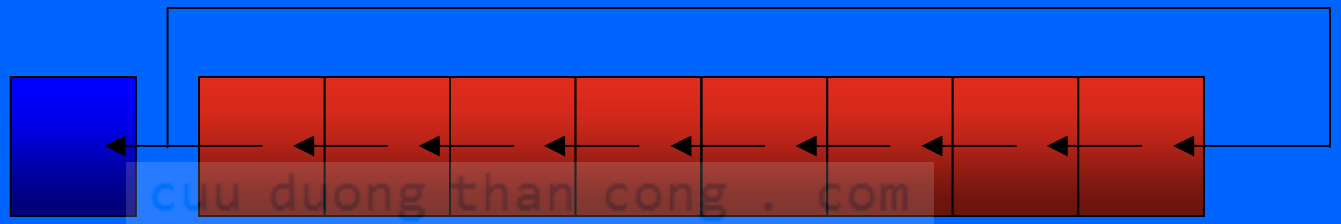


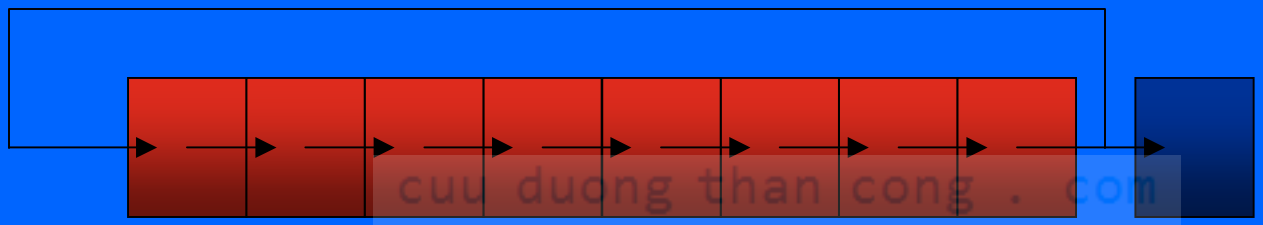
RCR



Rotate left/right
(Quay trái/phải không qua carry)

ROL





ROR

Nhóm lệnh rẽ nhánh

- Làm thay đổi trật tự thực hiện lệnh bình thường của vi xử lý
- *Lệnh nhảy không điều kiện: JMP*
- *Các lệnh nhảy có điều kiện: Jxxx*
- *Lệnh lặp: LOOP và các biến thể của nó*
- *Các lệnh có liên quan đến Chương trình con:*
  - CALL (gọi chương trình con)
  - RET (trở về chương trình gọi)
- *Các lệnh có liên quan đến Chương trình con phục vụ ngắt*
  - INT (gọi chương trình con phục vụ ngắt - Gọi ngắt)
  - IRET (quay về chương trình gọi ngắt)

Lệnh nhảy không điều kiện

- JMP *nhãn*
  - Nhảy gần: E9 xx xx (3 byte)
  - Nhảy ngắn: EB xx (2 byte)
  - Nhảy xa: EA xx xx xx xx (5 byte)
- *Nhãn: tên do người lập trình tự đặt ra theo qui tắc đặt tên của Assembler và có thể đặt vào trước một câu lệnh bất kỳ trong chương trình cùng với dấu :*

  nhãn: Câu lệnh cần thực hiện

- *Nhãn sẽ được dịch thành địa chỉ*
- *Khoảng cách nhảy: Khoảng cách đại số (có dấu) từ lệnh nhảy đến lệnh cần thực hiện*

Cơ chế thực hiện lệnh nhảy

- Các lệnh nhảy ngắn và gần chỉ làm thay đổi giá trị của thanh ghi IP
  - Lệnh nhảy ngắn cộng khoảng cách nhảy 8-bit có dấu vào giá trị hiện thời của IP
  - Lệnh nhảy gần cộng khoảng cách nhảy 16-bit có dấu vào giá trị hiện thời của IP
- Lệnh nhảy xa làm thay đổi cả CS và IP
  - Gán cho CS và IP các giá trị mới

Mã máy của lệnh nhảy

```
1106:0100 EB2A          JMP     012C
• 012C-0102=002A
1106:0102 EBFC          JMP     0100
• 0100-0104=FFFC
1106:0104 E97F00        JMP     0186
• 0186-0106=0080 (too far for short!)
• 0186-0107=007F
1106:0107 E9F5FE        JMP     FFFF
• FFFF-010A=FEF5
```

Các lệnh nhảy có điều kiện

- Jxxx *nhãn*
  - Có gần 40 menmonic khác nhau
- Các lệnh nhảy điều kiện đơn: phụ thuộc vào giá trị của 1 cờ.
- JNZ/JNE - Nhảy nếu cờ ZF = 0, nghĩa là kết quả của phép toán trước đó khác không
- JC - Nhảy nếu CF = 1, nghĩa là câu lệnh trước đó lập cờ carry
- JZ/JE
- JNC

Các lệnh nhảy có điều kiện

- Tất cả các lệnh nhảy có điều kiện phải là nhảy ngắn
  - khoảng cách nhảy: -128 to +127 bytes
- Tổ hợp với lệnh nhảy không điều kiện để có thể vượt qua giới hạn này.
- Các lệnh nhảy điều kiện kép: phụ thuộc vào giá trị của nhiều cờ
- JB/JNAE
- JNL/JGE

ứng dụng của các lệnh nhảy có điều kiện

- Kết hợp với JMP để xây dựng các cấu trúc lập trình cơ bản:

  - Cấu trúc điều kiện
  - Cấu trúc lặp

- Các lệnh nhảy thường theo sau các lệnh làm thay đổi giá trị của các cờ trạng thái:
  - CMP
  - TEST …

Cấu trúc điều kiện

```
mov  ax,n
cmp ax,7
jz nhan1
lệnh 1
jmp nhan2
nhan1:lệnh 2
nhan2:lệnh 3
```

Cấu trúc lặp

```
    mov ax,n

 nhan1: cmp ax,0
    jz nhan2
    lệnhi
    sub ax,2
    jmp nhan1

nhan2: lệnhk
```

Cấu trúc điều kiện - AND

```
char n; int w,x;
if (n>='A' && w==x)
   whatever();
```

```
;if(n>='A'&&w==x)
   mov ah,n
   cmp ah,'A'
   jl nogo
   mov ax,w
   cmp ax,x
   jne no_go
;then-part
   call whatever
nogo:
```

Cấu trúc điều kiện - OR

```
char n,k; unsigned int w;
if (n<>k || w<=10)
   whatever();
```

```
;if(n<>k||w<=10)
   mov ah,n
   cmp ah,k
   jne then_
   cmp w,10
   ja end_if
then_:
   call whatever
end_if:
```

Lệnh LOOP

- LOOP *nhan*
  - Giảm CX đi 1
  - Nếu (CX) <> 0 thì JMP nhan. Nếu không thì tiếp tục thực hiện lệnh theo trật tự bình thường

```
        mov cx,9
nhan:   lệnh 1
        lệnh 2
        lệnh 3
        loop nhan
```

LOOPZ/E và LOOPNZ/E

- Các biến thể của LOOP
- Giá trị của cờ ZF có thể làm kết thúc sớm vòng lặp
- Loop while ZF/equal && CX!=0
- Loop while (NZ/ not equal) && CX!=0
- Lưu ý: LOOP giảm CX nhưng không ảnh hưởng đến các cờ
- LOOPZ == LOOPE
- LOOPNZ==LOOPNE
- Các lệnh trong vòng lặp có thể tác động đến cờ ZF (CMP ?)

Chương trình con

- Chương trình con trong ngôn ngữ Assembly được gọi là Thủ tục (Procedure)
- Một thủ tục có thể được thực hiện nhiều lần
- Có liên quan đến stack:
  - lưu giữ Địa chỉ quay về
  - lưu giữ giá trị của các thanh ghi của vi xử lý

Stack ?

- Cấu trúc dữ liệu LIFO ở RWM
  - PUSH : ghi dữ liệu vào stack,
  - POP: đọc dữ liệu từ stack
- (SS:SP) trỏ đến đỉnh của stack
- (SS:BP) truy cập stack ngẫu nhiên (không theo LIFO)

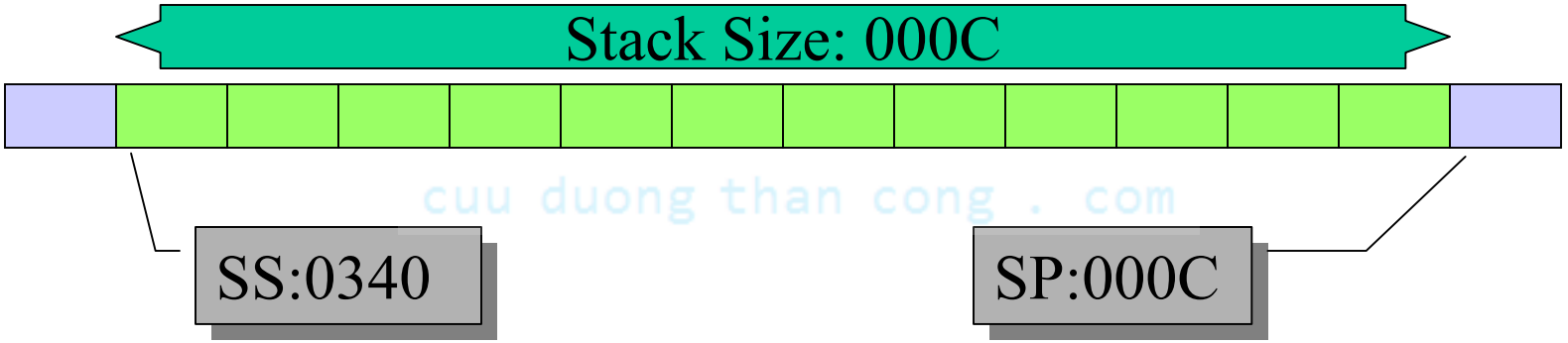
Stack Initialization

- The .stack directive hides an array allocation statement that looks like this
  - `The_Stack  DB  Stack_Size dup (?)`
- On program load…
  - SS is set to a segment address containing this array (usually `The_Stack` starts at offset 0)
  - SP is set to the offset of `The_Stack+Stack_Size` which is one byte past the end of the stack array
    - This is the condition for an empty stack

Initial Stack Configuration

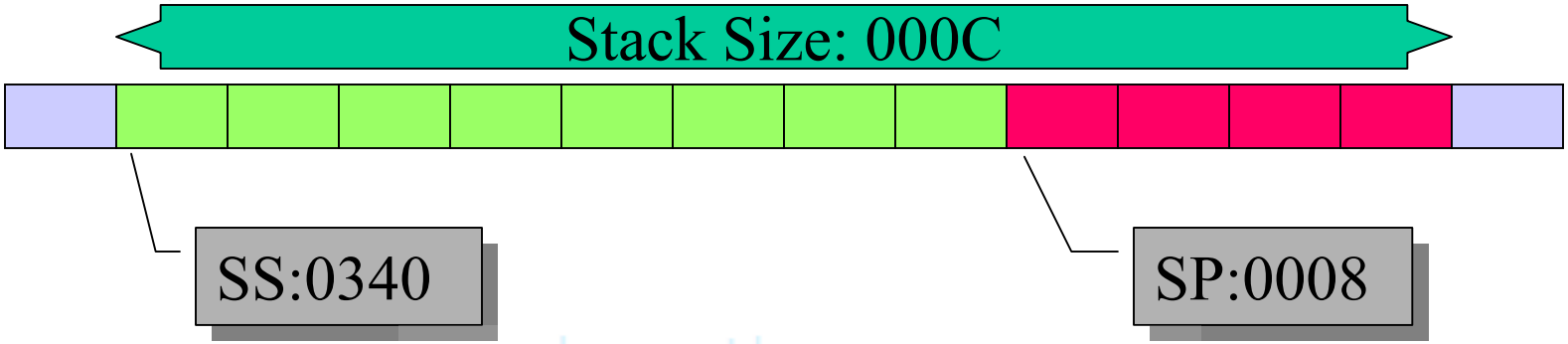
.stack 12 ;Reserve space for the stack

- Loader determines actual segment address for the start of the stack
  - This is an empty stack

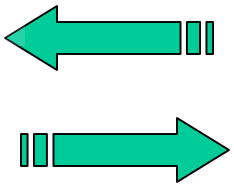


How Does The Stack Work?

- The stack grows backwards through memory towards the start of the stack segment



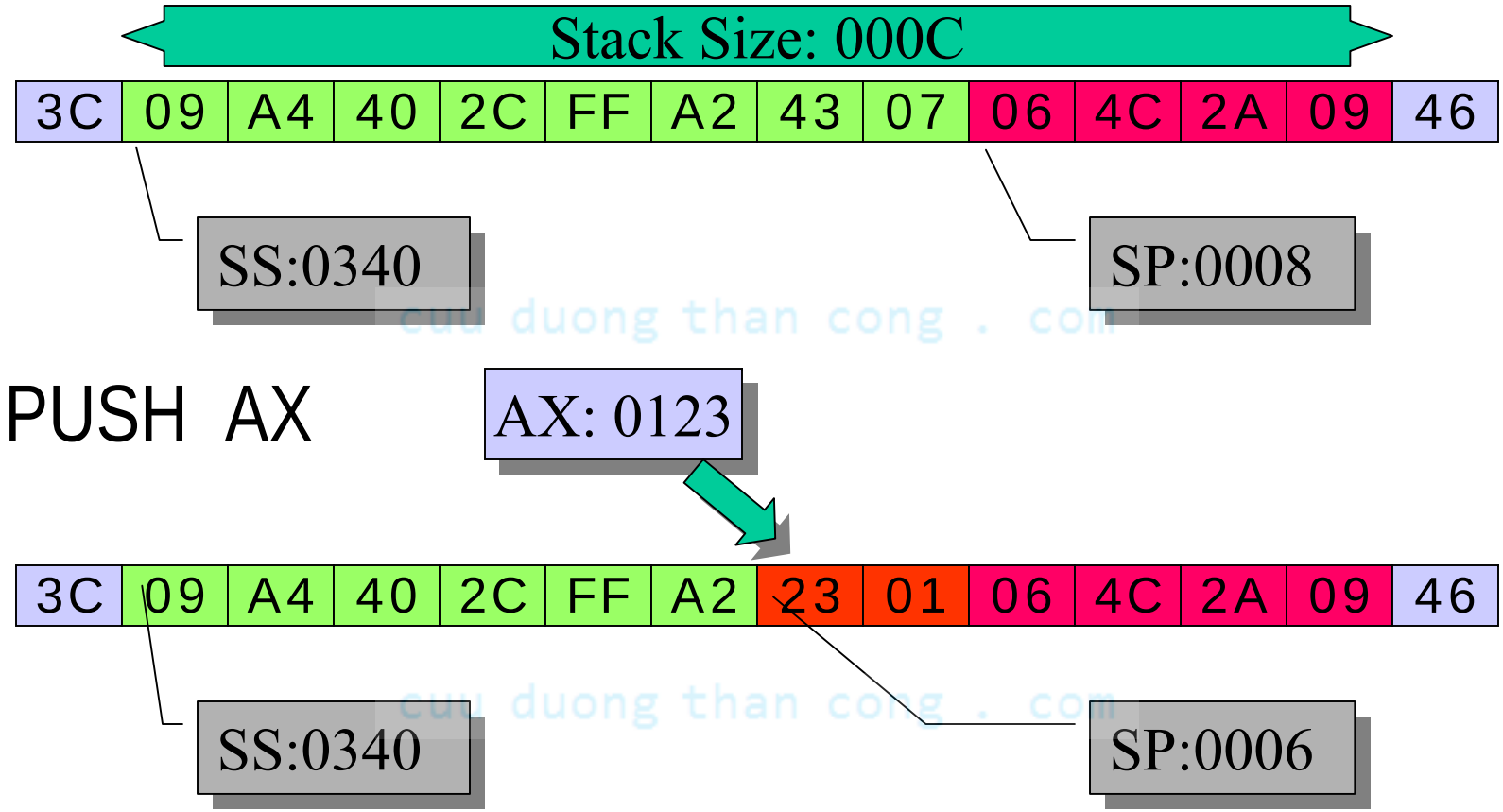
- Push decrements stack pointer
  Pop increments stack pointer



PUSH

- PUSH *nguồn*
  - Push nguồn vào stack
- PUSHF
  - Push thanh ghi cờ vào stack
- Lệnh PUSH trước hết sẽ giảm SP đi 2 rồi lưu giá trị của nguồn vào vị trí nhớ được trỏ bởi (SS:SP)

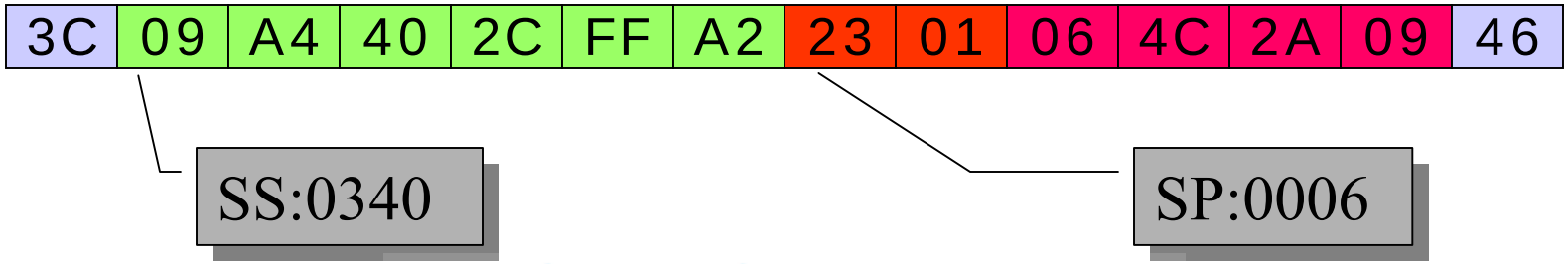
Ví dụ PUSH



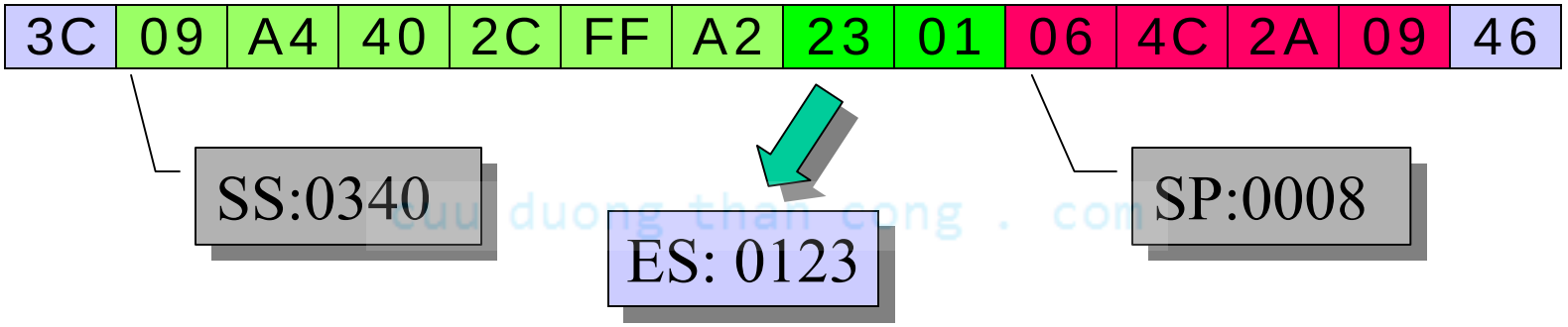
POP

- POP *đích*
  - Pop dữ liệu từ đỉnh stack vào đích
- POPF
  - Pop dữ liệu từ đỉnh stack vào thanh ghi cờ
- Lệnh POP trước hết copy dữ liệu được trỏ bởi (SS:SP) đến đích rồi tăng SP lên 2

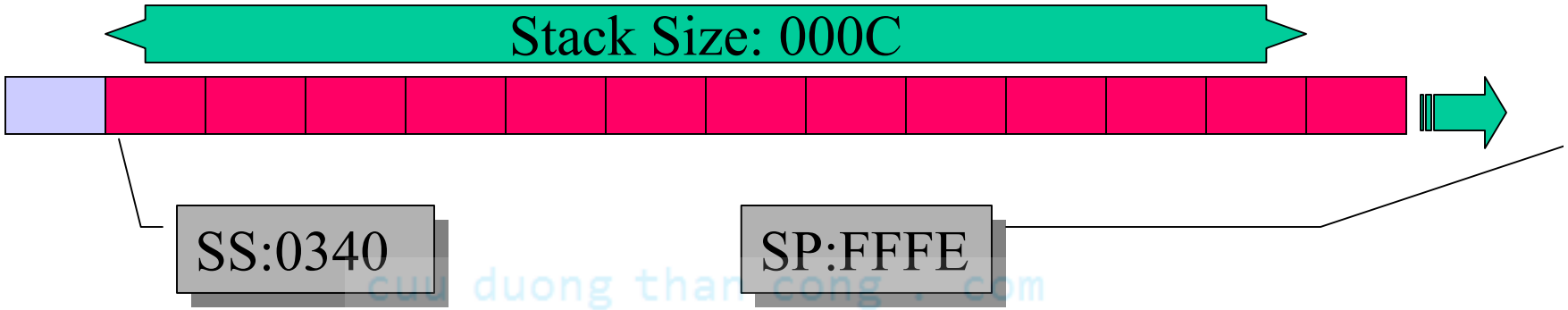
Ví dụ POP



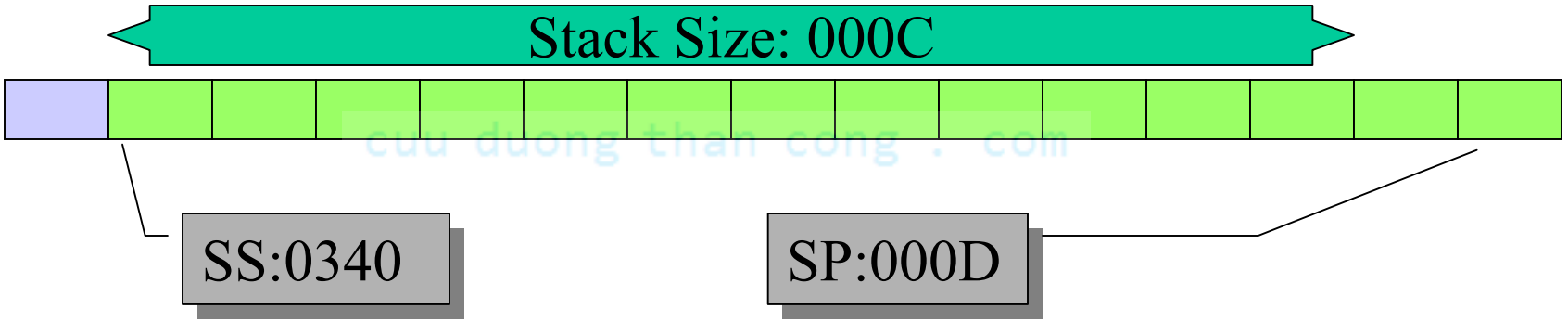
POP ES



Tràn stack!



- Stack Overflow
- Stack Underflow



Thủ tục

```
Tên_Thủ_tục  PROC      kiểu
   ;thân của thủ tục
   RET     ;quay về chương trình gọi
Tên_Thủ_tục  ENDP
```

- *kiểu* là NEAR hoặc FAR
  - ngầm định là NEAR
- Một thủ tục có thể có nhiều lệnh RET

Lệnh CALL và RET

- Gọi một thủ tục (NEAR)

  CALL *Tên_Thủ_tục*

  – push IP vào stack

  – copy địa chỉ của Tên_Thủ_tục vào IP
- Trở về từ một thủ tục (NEAR)

  RET

  – pop giá trị ở đỉnh stack vào IP

Thủ tục Far

- Gọi thủ tục (FAR)

  CALL *Tên_thủ_tục*

  – lần lượt push CS và IP vào stack

  – copy địa chỉ của Tên_thủ_tục vào CS và IP
- Trở về từ thủ tục (FAR)

  RET

  – pop giá trị từ đỉnh stack lần lượt vào IP và CS

Gọi ngắt

- Gọi ngắt là một lời gọi thủ tục đặc biêt
  - FAR
  - Thanh ghi cờ phải được bảo toàn
- INT Số ngắt
  - Thanh ghi cờ được push, TF và IF bị xoá
  - CS và rồI IP được push
  - Địa chỉ của một chương trình con phục vụ ngắt (Vector ngắt) tương ứng với Số ngắt được copy vào CS và IP

Trở về từ ngắt

- IRET
- Tác dụng của lệnh:
  - Giá trị ở đỉnh của stack được pop vào IP
  - Giá trị ở đỉnh của stack được pop vào CS
  - Giá trị ở đỉnh của stack được pop vào thanh ghi cờ

- Chương trình bị ngắt tiếp tục thực hiện dường như không có chuyện gì xảy ra

Xuất ký tự ra màn hình PC

- Ngắt 21h
  - Ngắt này hỗ trợ rất nhiều dịch vụ trên PC
  - Nhận dạng dịch vụ bằng số dịch vụ (số hàm). Số dịch vụ cần được nạp voà thanh ghi AH
  - Tuỳ theo từng dịch vụ, có thể cần thêm một số đối số khác được nạp vào các thanh ghi xác định
- AH = 2, DL = Mã ASCII của ký tự cần xuất
  - Ký tự được hiển thị tại vị trí hiện thờI của con trỏ

Xuất xâu ký tự ra màn hình PC

- Dịch vụ 09h của ngắt 21h
  - DX = Địa chỉ Offset của xâu (trong đoạn dữ liệu)
  - DS = Địa chỉ segment của xâu
  - Xâu ký tự phải kết thúc bằng ký tự '$'
- Để nạp địa chỉ offset của xâu vào DX, có thể:
  - LEA DX, Tênxâu
  - MOV DX, OFFSET Tên xâu

Nhập 1 ký tự từ bàn phím PC

- Dịch vụ 01h của ngắt 21h
- Khi NSD gõ một ký tự từ bàn phím:
  - Ký tự sẽ hiện trên màn hình
  - AL sẽ chứa mã ASCII của ký tự đó
    - AL=0 nếu ký tự được nhập là ký tự điều khiển

Nhóm lệnh thao tác string

- Chúng ta hiểu: string là một mảng byte hoặc từ nằm trong bộ nhớ
- Các thao tác string:
  - Sao chép
  - Tìm kiếm
  - Lưu trữ
  - So sánh

Các đặc điểm

- Nguồn: (DS:SI), Đích: (ES:DI)
  - DS, ES chứa Địa chỉ Segment của string
  - SI, DI chứa Địa chỉ Offset của string
- Cờ hướng DF (0 = Up, 1 = Down)
  - DF = 0 - Tăng địa chỉ (trái qua phải)
  - DF = 1 - Giảm địa chỉ (phảI qua trái)

Chuyển (Sao chép)

- MOVSB, MOVSW
  - Chuyển 1 byte hoặc 1 word từ vị trí nhớ này sang vị trí nhớ khác
  - Tác dụng của lệnh:
    - Sao chép byte/word từ (DS:SI) đến (ES:DI)
    - Tăng/Giảm SI và DI 1 hoặc 2 giá trị
  - Nếu CX chứa một giá trị khác không:
    - REP MOVSB hoặc REP MOVSW sẽ tự động sao chép (CX) lần và CX sẽ về không

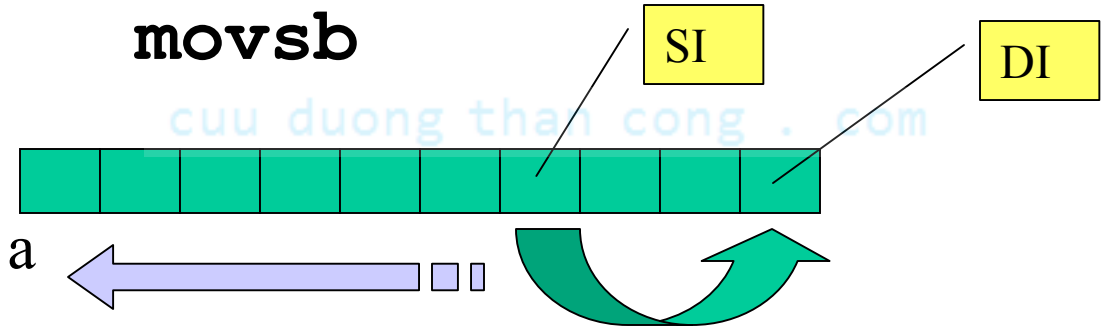
Ví dụ:Sao chép mảng

; Sao chép 10 byte từ mảng a sang mảng b, giả sử (DS) = (ES)

```
mov     cx, 10
mov     di, offset b
mov     si, offset a
cld     ;xoá cờ DF
rep     movsb
```

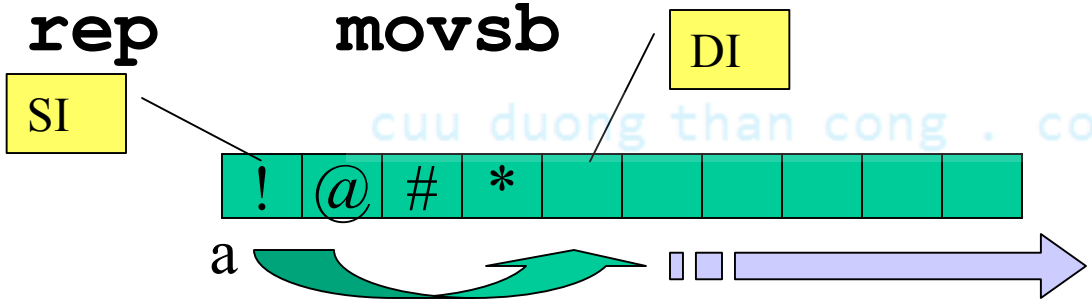
Ví dụ: Tịnh tiến các ô nhớ

```
mov     cx, 7
mov     di, offset a+9
mov     si, offset a+6
std     ;lập cờ DF
rep     movsb
```



Ví dụ

```
pattern     db   "!@#*"
            db   96 dup (?)
mov     cx,96
mov     si, offset pattern
mov     di, offset pattern+4
cld
rep     movsb
```



Lưu trữ string

STOSB, STOSW

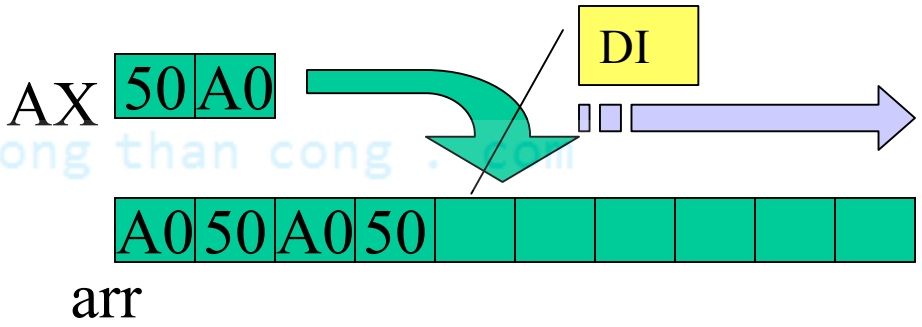
- Copy AL hoặc AX vào một mảng byte hoặc word
  - Đích (ES:DI)
- Tăng hoặc Giảm DI
  - phụ thuộc DF

- Thường được sử dụng có tiền tố REP và số lần lặp trong CX

Ví dụ:

```
arr  dw  200 dup (?)

 mov     ax,50A0h
 mov     di,offset arr
 mov     cx,200
 cld
 rep stosw
```



Nạp String

- LODSB, LODSW
  - Byte hoặc word tại (DS:SI) được copy vào AL hoặc AX
  - SI tăng hoặc giảm 1 hoặc 2 giá trị phụ thuộc DF
- Thường được dùng với STOSx trong một vòng lặp để xử lý từng phần tử trong một mảng

cuu duong than cong . com

Ví dụ:

```
  mov di, offset b
  mov si, offset a
  mov cx,30
  cld
lp:
  lodsb
  and al,0DFh
  stosb
  loop    lp
```

Quét String

SCASB, SCASW

- So sánh AL hoặc AX với byte hoặc word tạI (ES:DI) và tự động tăng hoặc giảm DI
- Lệnh này ảnh hưởng đến các cờ trạng thái
  - Tuỳ theo kết quả so sánh
  - Dùng trong một vòng lặp REPs
    - REPZ, REPE, REPNZ, REPNE

Ví dụ

```
arr  db 'abcdefghijklmnopqrstuvwxyz'
  mov          di, offset arr
  mov          cx,26
  cld
  mov          al,target
  repne  scasb

  jne          nomatch
```

So sánh String

CMPSB, CMPSW

- So sánh byte hoặc word tại (DS:SI) với byte hoặc word tạI (ES:DI), tác động đến các cờ và tăng hoặc giảm SI và DI
- Thường dùng để so sánh hai mảng với nhau

Ví dụ

```
  mov si, offset str1
  mov di, offset str2
  cld
  mov cx, 12
  repe    cmpsb
  jl      str1smaller
  jg      str2smaller
;the strings are equal - so far
;if sizes different, shorter string is
  less
```

Nhóm lệnh hỗn hợp

- Các lệnh Lập/Xoá trực tiếp các cờ:

STC, CLC
STD, CLD
STI, CLI

- Lệnh NOP (No Operation): Không làm gì!!!
- Lệnh NOP thường được dùng trong các vòng lặp tạo trễ (delay)bằng phần mềm
- Các lệnh Nhập/Xuất dữ liệu đối với các cổng I/O
  - IN
  - OUT

Lệnh IN

- Nếu Địa chỉ của cổng Nhỏ hơn hoặc bằng FFh:

```
IN Acc, Địa chỉ cổng
```

- Trong đó: Acc có thể là AL hoặc AX
- Nhập dữ liệu từ cổng vào Acc
- Nếu Địa chỉ của cổng Lớn hơn FFh:

```
MOV DX, Địa chỉ cổng
IN Acc, DX
```

- Trong đó: Acc có thể là AL hoặc AX
- Nhập dữ liệu từ cổng vào Acc

Lệnh OUT

- Nếu Địa chỉ của cổng Nhỏ hơn hoặc bằng FFh:

```
OUT Địa chỉ cổng, Acc
```

- Trong đó: Acc có thể là AL hoặc AX
- Xuất dữ liệu từ Acc ra cổng
- Nếu Địa chỉ của cổng Lớn hơn FFh:

```
MOV DX, Địa chỉ cổng
OUT DX, Acc
```

- Trong đó: Acc có thể là AL hoặc AX
- Xuất dữ liệu từ Acc ra cổng

Tóm tắt chương

- Tính tương thích về Cấu trúc thanh ghi của các vi xử lý họ x86
- Tính tương thích về Tập lệnh của các vi xử lý họ x86