CHƯƠNG 6
CẤP MÁY QUI ƯỚC

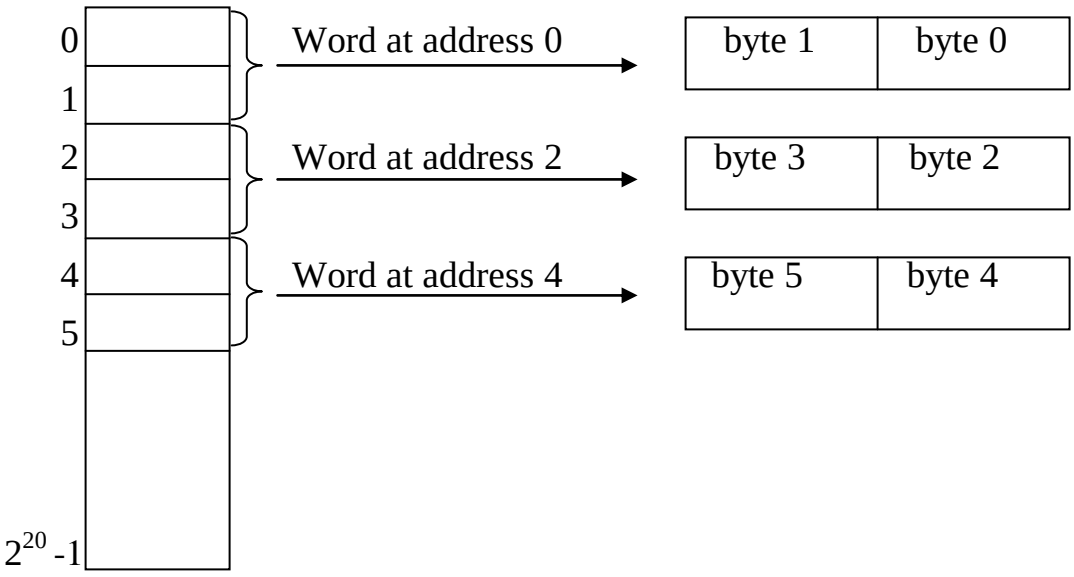
Chương này nghiên cứu về cấp máy qui ước (cấp 2). Chúng ta sẽ khảo sát cấp máy này thông qua các thế hệ CPU của Intel.

1. Họ 8088/8086/80286/80386 của Intel

1.1. 8088/8086

a. Không gian địa chỉ nhớ của 8088

Ở cấp máy qui ước, 8088 và 8086 giống hệt nhau, nên những gì bàn về 8088 đúng cho cả 8086.



Hình 4.1. Cấu trúc bộ nhớ của 8088

8088 có thể địa chỉ hoá được 2^{20} = 1 MB, địa chỉ bộ nhớ chính được đánh bắt đầu từ 0, kết thúc là 2^{20} - 1.

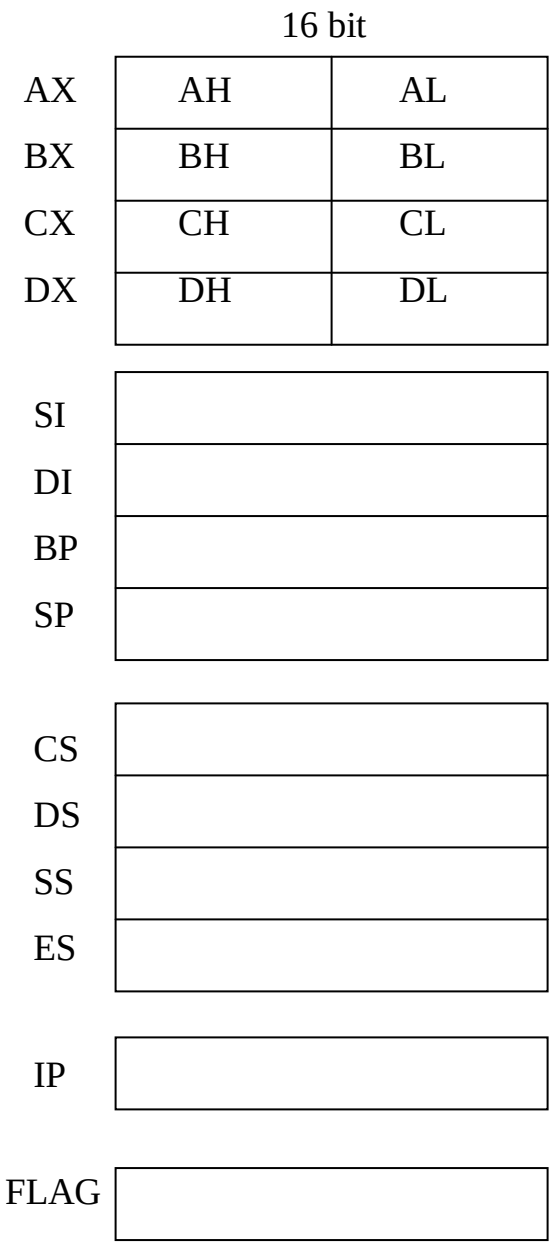
Các thanh ghi của 8088 chỉ có độ rộng 16 bit. Do vậy, các nhà thiết kế phải đưa ra giải pháp sử dụng các thanh ghi đoạn: code segment, data segment, stack segment, extra segment. Các thanh ghi đoạn chỉ chứa 16 bit cao của địa chỉ 20 bit.

b. Các thanh ghi của của 8088

Các thanh ghi của của 8088 được trình bày trong hình 4.2.

+ AX (Accumulator High / Low): thanh ghi tích luỹ, thường được dùng cho các

phép toán số học, lô gic, chuyển dữ liệu và lưu trữ kết quả. Dùng AX sẽ tạo ra mã máy ngắn nhất so với dùng các thanh ghi khác.



Hình 4.2. Các thanh ghi của 8088

+ BX (Base): thanh ghi cơ sở, thường được dùng để trỏ đến địa chỉ bộ nhớ chính.

+ CX (Counter): thường được dùng để chứa số lần lặp trong các vòng lặp, lệnh dịch bit.

+ DX (Data): thanh ghi dữ liệu, thường được dùng trong các chỉ thi nhân, chia. DX chứa 16 bit cao của tích 32 bit hoặc của số bị chia 32 bit.

Các thanh ghi trên có thể được dùng như 8 thanh ghi 8 bit AH, AL, BH, BL,...

+ SI (Source Index): thanh ghi chỉ số nguồn, thường được dùng trong các chỉ thị thao tác xâu ký tự. Nó trỏ đến địa chỉ offset của bộ nhớ, và thường đi với thanh ghi DS.

+ DI (Destination Index): thanh ghi chỉ số đích, thường được dùng trong các chỉ thị thao tác xâu ký tự. Nó trỏ đến địa chỉ offset của bộ nhớ, và thường đi với thanh ghi ES.

+ BP (Base Pointer): thanh ghi con trỏ cơ sở, thường được dùng để chứa địa chỉ đáy của khung stack hiện tại.

+ SP (Stack Pointer): thanh ghi con trỏ ngăn xếp, thường được dùng để chứa địa chỉ của đỉnh stack.

+ CS (Code Segment), DS (Data Segment), SS (Stack Segment), ES (Extra Segment): là các thanh ghi đoạn, được dùng để chứa địa chỉ đoạn (16 bit cao) của của các địa chỉ nhớ.

+ IP (Instruction Pointer): thanh ghi con trỏ lênh (bộ đếm chương trình), chứa

địa chỉ của chỉ thị kế tiếp sẽ được thực thi.

+ FLAG: thanh ghi cờ, hay từ trạng thái chương trình.

15				11	10	9	8	7	6		4		2		0
				O	D	I	T	S	Z		A		P		C

- O = Overflow: cờ tràn, bit O = 1 khi kết quả của phép toán số học bị tràn.
- D = Direction: cờ hướng, bit D xác định hướng của các thao tác chuỗi.
- I = Interrupt: cờ ngắt, bit I dùng để cho phép / không cho phép các ngắt của chương trình.
- T = Trap: cờ bẫy, bit T cho phép chạy chương trình từng bước để theo dõi.
- S = Sign: cờ dấu, bit S được xác lập bởi các chỉ thị số học, S = 1 cho kết quả âm, S = 0 cho kết quả âm.
- Z = Zero: cờ zê rô, Z = 1 cho kết quả 0, Z = 0 cho kết quả khác 0.
- A = Auxiliary: cờ phụ, bit A xác lập khi số nhớ xuất hiện ở giữa toán hạng.
- P = Parity: cờ chẵn lẻ, bit P được dùng để kiểm tra chẵn lẽ trong các chỉ thị số học.
- C = Carry: cờ nhớ, bit C xác lập khi số nhớ xuất hiện ở cuối toán hạng.

Có 7 bit cờ không được sử dụng bởi 8088.

c. Tập lệnh của 8088

Tập lệnh của 8088 được liệt kê trong hình 4.3.

Moves	MOV dst, src	Chép ND của nguồn sang đích, DL ở nguồn giữ nguyên, DL ở đích bị thay thế bởi DL nguồn.
	PUSH src	Đẩy dữ liệu từ nguồn vào đỉnh stack.
	POP dst	Lấy dữ liệu từ đỉnh stack đưa vào dst
	XCHG dst1, dst2	Hoán đổi nội dung của dst1 và dst2.
	LEA dst, scr	Nạp địa chỉ của nguồn vào đích (nạp đc offset của một ô nhớ vào thanh ghi).
	LDS dst, src	Nạp vào DS địa chỉ đoạn, vào thanh ghi công dụng chung địa chỉ offset. scr là từ nhớ 32 bit, 16 bit cao vào DS, 16 bit thấp vào dst.
	LES dst, scr	Nạp vào ES địa chỉ đoạn, vào thanh ghi công dụng chung địa chỉ offset. scr là từ nhớ 32 bit, 16 bit cao vào DS, 16 bit thấp vào dst.

- MOV = MOVe, src = source, dst = destilation, XCHG = eXCHanGe, LEA = Load Effective Address, LDS = Load Data Segment, LES = Load Extra Segment,

Arithmetic	ADD dst, scr	Cộng nguồn với đích, kết quả chứa vào đích. (dst) = (scr) + (dst).
	SUB dst, scr	Đích trừ nguồn, kết quả chứa vào đích. (dst) = (dst) - (scr).
	MUL scr	Phép nhân không dấu. Trong phép nhân với byte, toán hạng nguồn (số nhân - 8 bit) là ô nhớ hay thanh ghi, toán hạng đích (số bị nhân - 8 bit) là AL, kết quả chứa trong AX. Trong phép nhân với Word, toán hạng nguồn (số nhân - 16 bit), toán hạng đích (số bị nhân - 16 bit) là AX, kết quả chứa trong DX:AX (DX chứa 16 bit cao). Cờ C và O xác lập khi ND của 16 bit cao khác 0.
	IMUL scr	Phép nhân có dấu. Trong phép nhân với byte, toán hạng nguồn (số nhân - 8 bit) là ô nhớ hay thanh ghi, toán hạng đích (số bị nhân - 8 bit) là AL, kết quả chứa trong AX. Trong phép nhân với Word, toán hạng nguồn (số nhân - 16 bit), toán hạng đích (số bị nhân - 16 bit) là AX, kết quả chứa trong DX:AX (DX chứa 16 bit cao). Cờ C và O xác lập khi ND của 16 bit cao khác 0.
	DIV src	Phép chia không dấu. Trong phép chia cho byte, toán hạng nguồn (số chia - 8 bit) là ô nhớ hay thanh ghi, toán hạng đích (số bị chia) chứa trong AX, thương chứa trong AL, dư chứa trong AH. Trong phép chia cho Word, toán hạng nguồn (số chia - 16 bit), toán hạng đích (số bị chia - 32 bit) chứa trong DX:AX, thương chứa trong AX, dư chứa trong DX.
	IDIV src	Phép chia có dấu. Trong phép chia cho byte, toán hạng nguồn (số chia - 8 bit) là ô nhớ hay thanh ghi, toán hạng đích (số bị chia) chứa trong AX, thương chứa trong AL, dư chứa trong AH. Trong phép chia cho Word, toán hạng nguồn (số chia - 16 bit), toán hạng đích (số bị chia - 32 bit) chứa trong DX:AX, thương chứa trong AX, dư chứa trong DX.

	ADC dst, src	Cờ nhớ được cộng vào tổng của toán hạng nguồn và đích. (dst) = (scr) + (dst) + (C).
	SBB dst, src	(dst) = (dst) - (scr) - (C).
	INC dst	(dst) = (dst) + 1.
	DEC dst	(dst) = (dst) - 1.
	NEG dst	Lấy phần bù 2 của dst, kết quả chứa vào dst.

- ADD = ADDition, SUB = SUBtract, IMUL = Integer MULtiplycation, IDIV = Integer DIVide, ADC = ADd with Carry, SBB = SuBtract with Borrow, INC = INCrement, DEC = DECrement, NEG = NEGate,

BCD	DAA	Hiệu chỉnh kế quả trong phép cộng 2 số mã BCD nén.
	DAS	Hiệu chỉnh kết quả trong phép trừ mã BCD nén.
	AAA	Hiệu chỉnh ASCII đối với phép cộng.
	AAS	Hiệu chỉnh ASCII đối với phép trừ.
	AAM	Hiệu chỉnh AL sau phép nhân.
	AAD	Hiệu chỉnh số bị chia mã BCD không nén trong AX trước khi thực hiện chia.

- BCD = Binary Coded Decimal: Số thập phân được mã hoá nhị phân.

+ BCD nén: xử lí từng byte như là 2 nửa byte (nibble), 1 nibble = 4 bit. Mỗi nibble chứa một số thập phân từ 0 - 9, như vậy các tổ hợp bit từ 1010 - 1111 không được dùng. Theo mã BCD, một số nguyên 16 bit có thể chứa một số thập phân từ 0 - 9999.

+ BCD không nén: Mỗi byte chứa một số thập phân từ 0 - 9. Theo mã BCD không nén, 16 bit có thể chứa một số thập phân từ 0 - 99.

1. DAA = Decimal Adjust for Addition: Hiệu chỉnh thập phân đối với phép cộng. Lệnh này điều chỉnh tổng trong AL của 2 toán hạng BCD nén. Phép hiệu chỉnh xẩy ra khi kết quả trong mỗi nibble lớn hơn 9, cách hiệu chỉnh: cộng 6h (0110) cho hàng đơn vị, 60h (0110 0000) cho hàng chục,...VD:

[AL] = 12d = 0001 0010bcd

[BL] = 49d = 0100 1001bcd

ADD AL,BL $\longrightarrow$ [AL] = 0101 1011b

DAA → [AL] = 0101 1011b + 0000 0110b = 0110 0001bcd = 61d.

2. DAS = Decimal Adjust Subtraction: Hiệu chỉnh hiệu trong AL của 2 toán hạng mã BCD nén. Phép hiệu chỉnh xẩy ra khi kết quả trong mỗi nibble lớn hơn 9, cách hiệu chỉnh: nếu nibble thấp của AL lớn hơn 9 thì trừ 6h (0110b) cho hàng đơn vị, nếu nibble cao của AL lớn hơn 9 thì trừ 60h (0110 0000b) cho hàng chục,...VD:

[AL] = 0101 0110bcd = 56d

[BL] = 0011 1001bcd = 39d

SUB AL, BL → [AL] = 0001 1101b (nibble > 9d)

DAS → [AL] = 0001 1101b - 0000 0110b = 0001 0111bcd = 17d.

3. AAA = ASCII Adjust for Addition: Hiệu chỉnh kết quả trong AL sau phép cộng 2 số mã BCD không nén hay 2 chữ số mã ASCII. Lệnh này có toán hạng ẩn trong AX. Phép hiệu chỉnh xẩy ra khi nibble thấp của AL lớn hơn 9 hay cờ A dương (nibble thấp có nhớ). Cách hiệu chỉnh: [AL] + 6d, [AH] + 1d, xoá nibble cao của AL. VD:

[AL] = 00001001bcd = 9d

[BL] = 00001000bcd = 8d

ADD AL, BL → [AL] = 00010001b = 11h (nibble thấp có nhớ)

AAA → [AH] = 00000001bcd

[AL] = 00000111bcd

[AX] = 00000001 00000111bcd = 17d

Khi nhập 2 chữ số từ bàn phím (mã ASCII của mỗi chữ số chứa trong thanh ghi), cộng chúng với nhau và lưu kết quả dạng mã BCD không nén. VD: [AL] = 36h (chữ số 6), [BL] = 37h (chữ số 7), chúng ta cộng 2 số này và sử dụng AAA để hiệu chỉnh:

[AL] = 0011 0110b = 36h (chữ số 6)

[BL] = 0011 0111b = 37h (chữ số 7)

ADD AL, BL → [AL] = 0110 1101b (nibble thấp > 9)

AAA → [AH] = 0000 0001bcd

[AL] = 0000 0011bcd

[AX] = 0000 0001 000 00011bcd = 13d

4. AAS = ASCII Adjust for Subtraction: Hiệu chỉnh kết quả trong AL sau phép trừ 2 số dạng mã BCD không nén. Lệnh này có toán hạng ẩn trong AX. Phép hiệu

chỉnh xẩy ra khi nibble thấp của AL lớn hơn 9 hay cờ A dương (nibble thấp có nhớ). Cách hiệu chỉnh: [AL] - 6d, [AH] - 1d, xoá nibble cao của AL. VD: thực hiện phép trừ 26 cho 7:

[AH] = 0000 0010bcd = 2d

[AL] = 0000 0110bcd = 6d

[BL] = 0000 0111bcd = 7d

SUB AL, BL ⟶ [AL] = 1111 1111b (nibble thấp > 9)

AAS ⟶ [AH] = 0000 0001bcd = 1d

[AL] = 0000 1001bcd = 9d ⟶ [AX] = 19d

5. AAM = Adjust After Multiplication: hiệu chỉnh kết quả trong AL của phép nhân 2 số mã BCD không nén thành số mã BCD không nén trong AX. Cách hiệu chỉnh: Chia [AL] cho 10d, thương số đặt trong AH, số dư đặt trong AL. VD:

[AL] = 00001001bcd = 9d

[BL] = 00000111bcd = 7d

MUL BL⟶ [AL] = 00111111b = 3fh = 63d

AAS ⟶ [AH] = 00000110bcd = 6d

[AL] = 00000011bcd = 3d

6. AAD = Adjust AX before Division: Hiệu chỉnh số bị chia mã BCD không nén trong AX trước khi thực hiện chia. Cách hiệu chỉnh: [AH] nhân với 10d cộng với [AL], sau đó xoá [AH]. VD:

[AX] = 0000 0110 0000 0011bcd = 63d

[BL] = 0000 0111bcd = 7d

AAD ⟶ [AL] = 0011 1111b = 3fh = 63d

DIV BL ⟶ [AL] = 0000 1001b = 9d

Boolean	AND dst, scr	End logic các cặp bit của dst và scr, kết quả 1 khi cả 2 bit là 1, còn lại là 0.
	OR scr, dst	Or logic các cặp bit của dst và scr, kết quả 0 khi cả 2 bit là 0, còn lại là 1.
	XOR scr, dst	Xor logic các cặp bit của dst và scr, kết quả 0 khi 2 bit giống nhau, là 1 khi khác nhau.
	NOT dst	Lấy phủ định các bit của dst.

Shift/Rotate	SAL/SAR dst, count	Dịch trái/phải số học, thực hiện nhân phép / chia các số. Mỗi lần dịch trái / phải tương tự số được nhân 2 hoặc chia 2.
	SHL/SHR dst, count	Dịch các bit của dst sang trái hay sang phải count bit.
	ROL/ROR dst, count	Dịch các bit sang trái / phải, bit msb (lsb) chuyển vào vị trí lsb (msb).
	RCL/RCR dst, count	Quay trái / phải qua cờ C. Bit msb (lsb) đưa vào C, giá trị của C cũ vào vị trí lsb (msb).

- SAL = Shift Arithmatic Left, SAR = Shift Arithmatic Right, SHL = SHift Left, SHR = SHift Right, ROL = ROtate Left, ROR = ROtate Right, RCL = Rotate Carry Left, RCR = Rotate Carry Right.

Test Compare	TEST scr1, src2	And scr1 và scr2, tác động đến các cờ C, O, P, S,Z.
	CMP src1, src2	S. sánh 2 toán hạng, tác động tới các cờ A, C, O, P, S, Z.

- CMP = ComPare.

Control Transter	JMP addr	Nhẩy vô điều kiện.
	Jxx addr	Nhẩy có điều kiện.
	JCXZ addr	Nhẩy nếu CX = 0.
	CALL addr	Lệnh gọi thủ tục.
	RET	Lệnh quay trở về CT chính, lệnh tiếp theo sau lệnh call
	IRET	Trở về chương trình chính sau khi thực hiện CT con phục vụ ngắt.
	LOOPxx	Lặp có điều kiện.
	INT addr	Gọi chương trình con phục vụ ngắt.
	INTO	Gọi chương trình con phục vụ ngắt khi tràn số (O = 1)

Strings	LODS (B/W/D)	Nạp chuỗi (byte/wod/double word) từ bộ nhớ vào thanh ghi AL/AX/EAX.
	STOS (B/W/D)	Nạp chuỗi (byte/wod/double word) từ thanh ghi AL/AX/EAX vào bộ nhớ.
	MOVS (B/W/D)	Sao chép ND của (byte/wod/double word) từ địa chỉ DS:SI tới địa chỉ ES:DI.
	CMPS (B/W/D)	So sánh các cặp (byte/wod/double word) từ ở địa chỉ DS:SI và địa chỉ ES:DI.
	SCAS (B/W/D)	Tìm (byte/wod/double word - mặc định trong thanh ghi) trong chuỗi ở trong bộ nhớ.

- LODS = LOaD String, STOS = STOre String, MOVS = MOVe String, CMPS = CoMPare String, SCAS = Scan String.

Condition codes	STC	Thiết lập cờ nnhớ (C = 1).
	CLC	Xoá cờ nhớ (C = 0).
	CMC	Lấy bù cờ nhớ.
	STD	Thiết đặt cờ hướng (D = 1).
	CLD	Xoá cờ hướng (D = 0).
	STI	Thiết đặt cờ ngắt (I = 1).
	CLI	Xoá cờ ngắt (I =).
	PUSHF	Cất cờ vào đỉnh stack.
	POPF	Lấy cờ từ đỉnh stack.
	LAHF	8 bit thấp của thanh ghi cờ được copy vào AH.
	SAHF	Lưu trữ ND của AH vào 8 bit thấp của thanh ghi cờ.

- STC = SeT Carry, CLC = CLear Carry, CMC = Complement Carry, STD = SeT Direction, CLD = CLear Direction, STI = SeT Interrupt, CLI = CLear Interrupt, PUSHF = PUSH Flag, POPF = POP Flag, LAHF = Load AH from Flags, SAHF = Store AH in Flags register.

Miscellaneous	CWD	Đổi số 16 bit trong AX thành số 32 bit có dấu trong DX:AX.
	CBW	Đổi số có dấu 8 bit trong AL thành số có dấu 16 bit trong AX.
	XLAT (Translate)	Thay thế ND của AL bằng ND của ô nhớ được xác định bởi BX và AL.
	NOP (NO oPeration)	Không thực hiện thao tác nào.
	HLT (HaLT)	Dừng thực hiện CT để đợi một ngắt ngoài.
	ESC	Cho phép 8087 thực hiện thao tác.
	IN R, port	Nhập vào thanh ghi (byte/word) ND của cổng vào/ra.
	OUT R, port	ND của cổng vào/ra được thay thế bởi ND thanh ghi.
	WAIT	CPU ở trạng thái chờ cho tới khi được kích hoạt.

d. 80286 của Intel

- Về cấu trúc, 80286 khác 8088 như bảng dưới đây:

8088	80286
Chế độ thực	Chế độ thực, chế độ bảo vệ.
Không gian địa chỉ nhớ 1 MB	Không gian địa chỉ nhớ 64 MB
Không hỗ trợ đa chương trình	Hỗ trợ đa chương trình

- Ở cấp máy qui ước, tất cả các lệnh của 8088 đều thực hiện được trong 80286. 80286 có thêm một số lệnh sau đây:

PUSHA	Cất tất cả các thanh ghi AX, BX, CX, DX, SP, BP, SI, DI vào stack theo trật tự trên.
POPA	Khôi phục các thanh ghi đã cất từ ngăn xếp theo trình tự: DI, SI, BP, SP, DX, CX, BX, AX.
ENTER	Thiết lập khung stack cho thủ tục.
LEAVE	Xoá stack khi ra khỏi thủ tục.
BOUND	Thực hiện việc kiểm tra các biên dãy (array).
VERR/VERW	Kiểm tra việc đọc / ghi có quá một segment không.

e. 80386 của Intel

- Về cấu trúc, 80386 khác 80286 như bảng dưới đây:

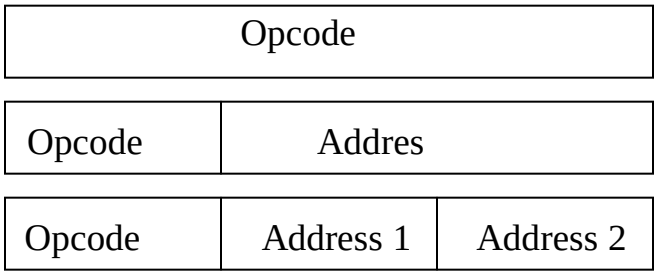
80286	80386
Chế độ thực, chế độ bảo vệ	Chế độ thực, chế độ ảo, chế độ bảo vệ.
Không gian địa chỉ nhớ 64 MB	Không gian địa chỉ nhớ 4 GB
4 thanh ghi đoạn	6 thanh ghi đoạn
Phép toán 8 / 16 bit	Phép toán 8 / 16 / 32 bit
Các thanh ghi 16 bit	Các thanh ghi 32 bit

- Ở cấp máy qui ước, tất cả các lệnh của 8088 và 80286 đều thực hiện được trong 80386. 80386 có thêm một số lệnh sau đây:

Chỉ thị	Mô tả
BSF/BSR	BSF = Bit Scan Forwad, quét từ bit 0 tới bit msb; BSR = Bit Scan Reverse, quét từ bit msb tới bit 0.
BTx	BT = Bit Test, x = C (Complement), R (Reset), S (Set): các lệnh kiểm tra bit.
MOVSX	MOVe with Sign eXtend: chép byte/word của toán hạng nguồn thành word/double word của đích với dấu.
MOVZX	MOVe with Zero eXtend: chép byte/word của toán hạng nguồn thành word/double word của đích với các byte 0 ở phần mở rộng.
SETxx	Thiết lập byte theo điều kiện, byte là 1 nếu điều kiện đúng, 0 nếu điều kiện sai.
SHLD/SHRD	Double precision SHift Left / Double precision SHift Right: dịch trái / phải độ chính xác kép.

2. Các khuôn dạng lệnh

Chương trình bao gồm một tập các chỉ thị, mỗi chỉ thị xác định một thao tác cụ thể nào đó. Một phần của chỉ thị được gọi là opcode (operation code), cho biết thao tác gì được thực hiện. Phần còn lại của chỉ thị chỉ rõ vị trí của dữ liệu được chỉ thị sử dụng. Có nhiều khuôn dạng lệnh khác nhau: lệnh không địa chỉ, lệnh 1 địa chỉ, lệnh 2 địa chỉ,...



2.1. Các tiêu chuẩn thiết kế khuôn dạng lệnh

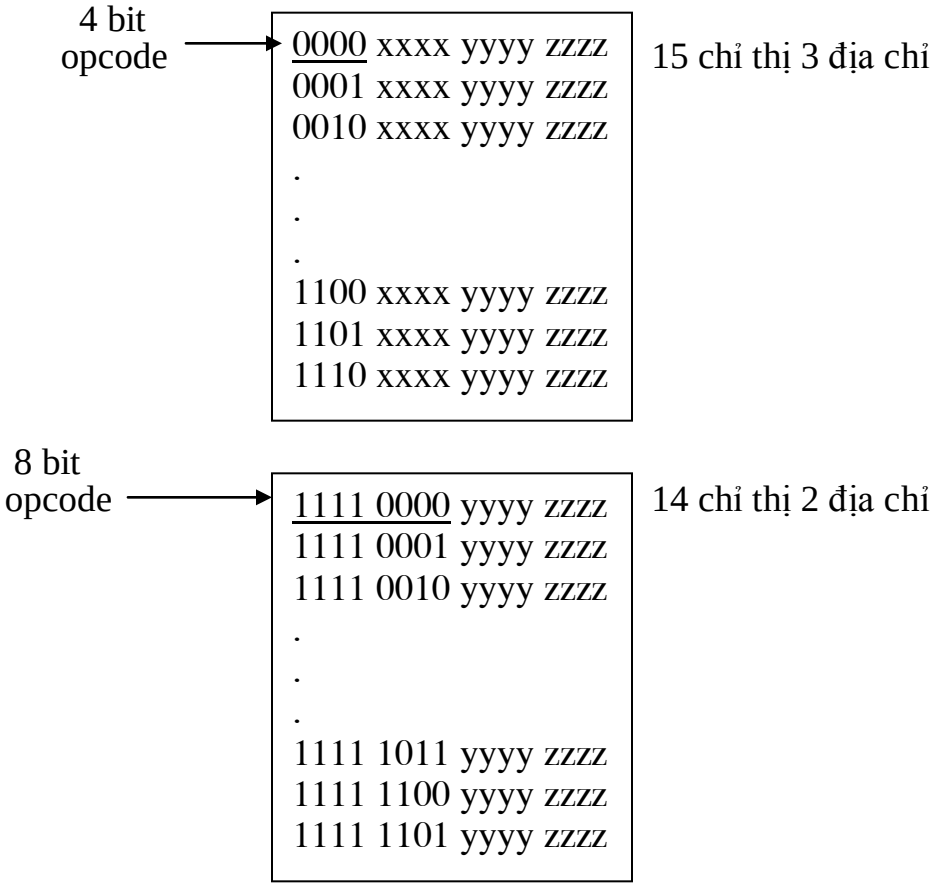
1. Các chỉ thị ngắn (số bit trong một chỉ thị) tốt hơn các chỉ thị dài. Ví dụ, một chương trình n chỉ thị 16 bit chỉ chiếm một nửa không gian nhớ so với n chỉ thị 32 bit. Ngoài ra, việc đọc các chỉ thị ngắn từ bộ nhớ cung cấp cho CPU sẽ nhanh hơn, do đó tốc độ xử lí chương trình với các chỉ thị ngắn sẽ nhanh hơn.

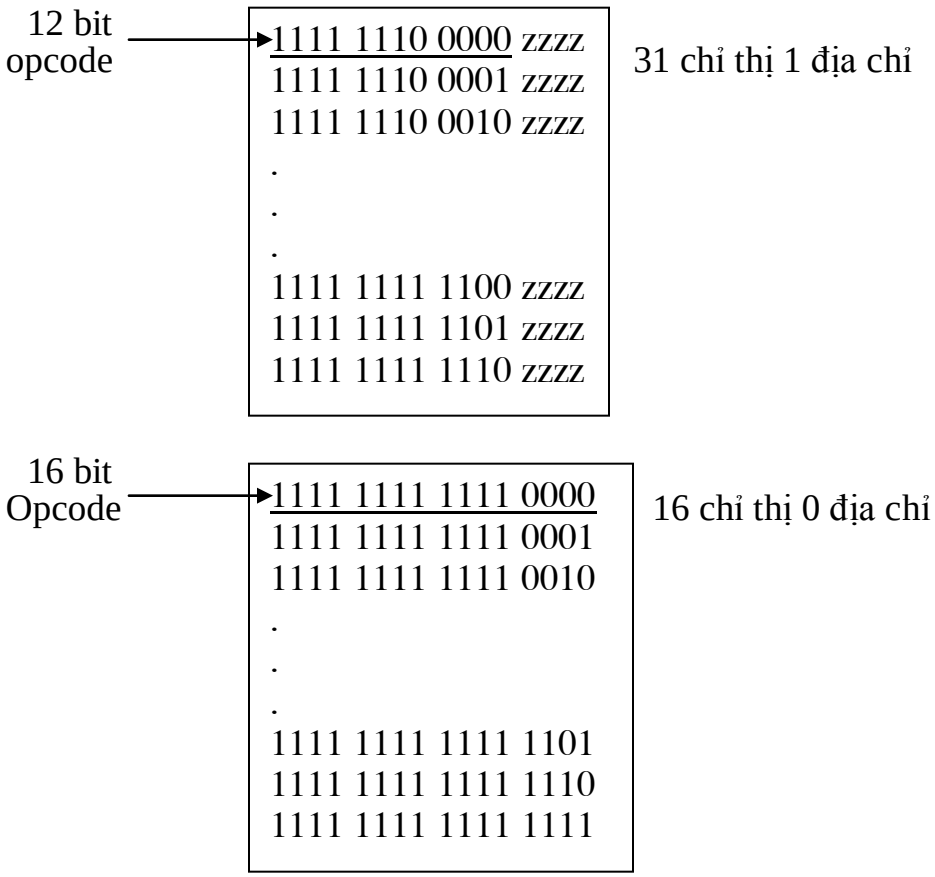
2. Đủ không gian để diễn tả tất cả các thao tác mong muốn của CPU. Một CPU với 2^n thao tác thì kích thước của opcode phải lớn hơn hay bằng n.

3. Chiều dài một chỉ thị nên là bội số nguyên lần chiều dài mã hoá ký tự, để tránh gây lãng phí bộ nhớ khi cất các ký tự.

4. Liên quan đến số bit trong trường địa chỉ. Giả sử một máy mà các ký tự được mã hoá bởi 8 bit và giả sử bộ nhớ chính có dung lượng 2^{16} byte. Nhà thiết kế có thể chọn từ nhớ là 8, 16, 32 bit. Nếu chọn 8 bit, thì chiều dài địa chỉ cần 16 bit. Còn nếu chọn 32 bit thì chiều dài trường địa chỉ cần 14 bit.

2.2. Mở rộng opcode





Hình 4.3. Mở rộng opcode

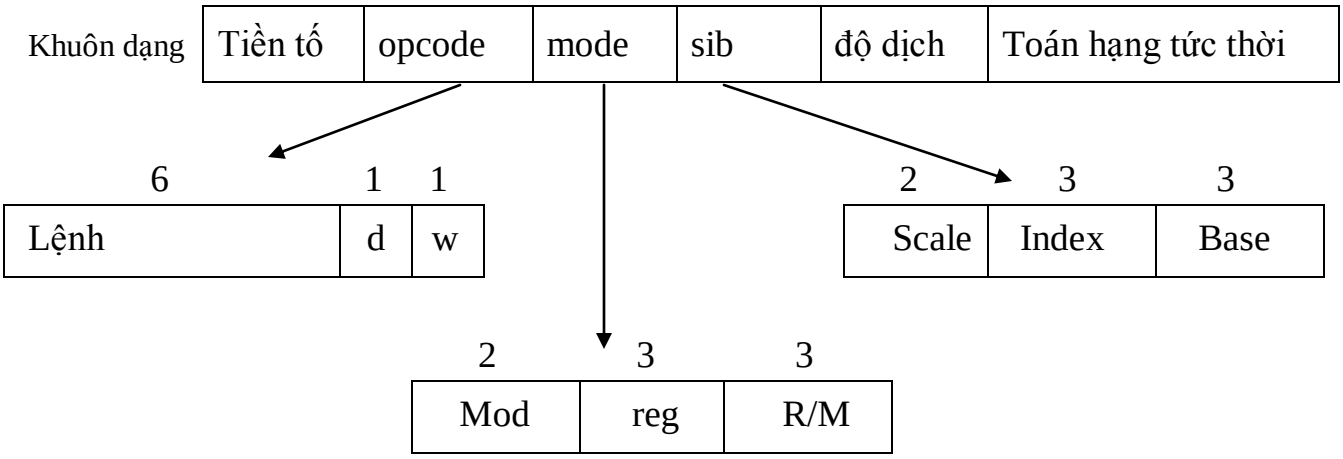
Xét một chỉ thị có độ dài (n + k) bit, k bit opcode và n bit địa chỉ. Chỉ thị này cho phép có 2^k thao tác khác nhau và 2^n địa chỉ nhớ khác nhau. Nếu tăng số bit địa chỉ lên 1 thì địa chỉ nhớ tăng gấp đôi nhưng số các thao tác lại giảm đi một nửa.

Khái niệm mở rộng opcode liên quan đến việc thay đổi số các bit opcode. Ví dụ, xét một máy có chỉ thị dài 16 bit, các nhà thiết kế có thể thiết kế 15 chỉ thị 3 địa chỉ (mỗi địa chỉ dài 4 bit), 14 chỉ thị 2 địa chỉ, 31 chỉ thị 1 địa chỉ và 16 chỉ thị không địa chỉ như trong hình 4.3.

2.3. Khuôn dạng lệnh của Intel

Các khuôn dạng chỉ thị của 8088, 80286 và 80386 được trình bày trong hình 4.4. Mỗi chỉ thị có thể có 6 trường, mỗi trường có độ dài như trong hình vẽ.

8088	0-3	1	0-1	0	0-2	0-2
80286	0-3	1	0-1	0	0-2	0-2
80386	0-3	1-2	0-1	0-1	0-4	0-4



Hình 4.4. Khuôn dạng lệnh của Intel

- Trường w và reg qui định cho các thanh ghi như trong hình 4.5.

W = 1	W = 0	REG
AX	AL	000
BX	BL	011
CX	CL	001
DX	DL	010
SP	AH	100
DI	BH	111
BP	DH	101
SI	CH	110
CS		01
DS		11
ES		00
SS		10

Hình 4.5. Trường w và trường reg

- Bit d chỉ hướng đi của dữ liệu, d = 1: dữ liệu đi đến thanh ghi, d = 0: dữ liệu đến từ thanh ghi. W = 1: địa chỉ nhớ là từ, w = 0: địa chỉ nhớ là byte.
- Trường mode và trường R/M điều khiển các kiểu định địa chỉ được trình bày trong hình 4.6.

Mod / R/M	00	01	10	11	
	Chế độ địa chỉ bộ nhớ			Chế độ đc tg	
				W = 0	W = 1
000	[BX]+[SI]	[BX]+[SI]+disp.8	[BX]+[SI]+disp.16	AL	AX
001	[BX]+[DI]	[BX]+[DI]+disp.8	[BX]+[DI]+disp.16	CL	CX
010	[BP]+[SI]	[BP]+[SI]+disp.8	[BP]+[SI]+disp.16	DL	DX
011	[BP]+[DI]	[BP]+[DI]+disp.8	[BP]+[DI]+disp.16	BL	BX
100	[SI]	[SI]+disp.8	[SI]+disp.16	AH	SP
101	[DI]	[DI]+disp.8	[DI]+disp.16	CH	BP
110	disp.16	[BP]+disp.8	[BP]+disp.16	DH	SI
111	[BX]	[BX]+disp.8	[BX]+disp.16	BH	DI

Hình 4.6. Định địa chỉ toán hạng được điều khiển bởi trường mod và trường R/M

- Từ 80386 trở đi, Intel bổ sung thêm trường SIB. Địa chỉ toán hạng được tính bằng cách : nhân thanh ghi chỉ số (index) với hệ số (scale) sau đó cộng với thanh ghi cơ sở (base).

Ví dụ 1 : lệnh nạp số 89h vào thanh ghi AL :

1011 0000 1000 1001b = b089h

Viết theo từ gợi nhớ cho dễ hiểu: MOV AL, 89h.

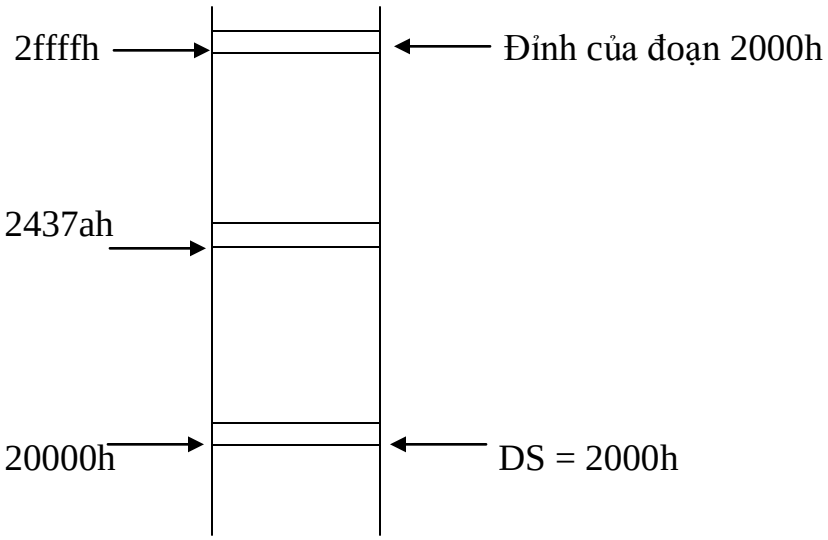
3. Các chế độ địa chỉ của họ vi xử lí Intel

Cách thức cho biết các toán hạng trong lệnh ở đâu được gọi là *phương pháp định địa chỉ* hay *chế độ địa chỉ*. Một lệnh có thể có 1, 2 hoặc 3 địa chỉ.

Để truy cập dữ liệu trong bộ nhớ, bộ xử lí phải tính được địa chỉ vật lí 20 bit. Để tính được địa chỉ 20 bit, 8086/8088 làm như sau:

+ Nhân địa chỉ đoạn với 10h.

+ Cộng địa chỉ offset hay effective address (EA) hay độ lệch (displacement) vào địa chỉ đoạn sau khi đã nhân.



Hình 4.7. Cách tính địa chỉ vật lí của 8086/8088

3.1. Chế độ địa chỉ tức thời (immediate addressing)

Trong chế độ này, phần địa chỉ toán hạng nguồn của lệnh chính là toán hạng. Ví dụ:

MOV CX, 437bh; nạp số 437bh vào thanh ghi CX.

3.2. Chế độ địa chỉ trực tiếp (direct addressing)

Trong chế độ này, địa chỉ của từ nhớ chứa toán hạng nguồn nằm trong phần địa chỉ của lệnh. Ví dụ:

MOV BL, [437ah]; nạp ND của ô nhớ có địa chỉ offset 437ah vào thanh ghi BL.

Khi thực hiện lệnh, CPU tính địa chỉ 20 bit của toán hạng như đã trình bày ở trên. Địa chỉ đoạn chính là địa chỉ của đoạn chứa dữ liệu của chương trình.

3.3. Chế độ địa chỉ thanh ghi (register addressing)

Trong chế độ này, trường địa chỉ toán hạng nguồn của lệnh chứa số hiệu của thanh ghi chứa toán hạng.Ví dụ:

MOV CX, AX; copy ND thanh ghi AX vào CX.

3.4. Chế độ địa chỉ gián tiếp (indirect addressing)

Địa chỉ offset của toán hạng nguồn chứa trong thanh ghi BX, BP, SI hoặc DI. Ví dụ:

MOV AX, [SI]; nạp ND của từ nhớ có địa chỉ offset chứa trong thanh ghi SI vào thanh ghi AX.

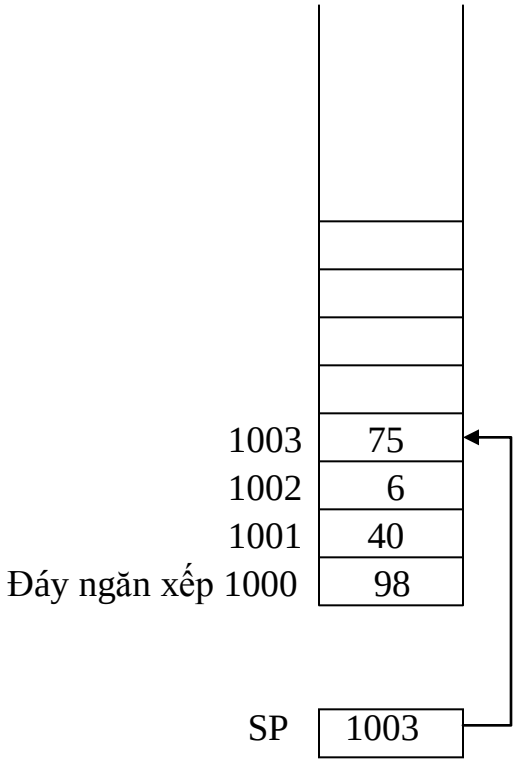
3.5. Chế độ địa chỉ chỉ số (indexed addressing)

Trong chế độ này, địa chỉ offset là tổng của độ dịch chuyển và ND của các thanh ghi. Ví dụ:

MOV AL, [SI] + displacement; chuyển ND ô nhớ có địa chỉ offset là tổng của ND SI với độ dịch chuyển vào thanh ghi AL.

3.6. Địa chỉ ngăn xếp (stack addressing)

Ngăn xếp lưu trữ dữ liệu theo nguyên tắc: phần tử đầu tiên được cất vào đáy ngăn xếp, phần tử mới nhất được cất vào đỉnh ngăn xếp. Kết hợp với ngăn xếp là một thanh ghi (SP) chứa địa chỉ của đỉnh ngăn xếp.



Hình 4.8. Địa chỉ ngăn xếp

Ví dụ: PUSH AX; cất ND của thanh ghi AX vào đỉnh ngăn xếp.

4. Các loại chỉ thị

4.1. Các chỉ thị di chuyển dữ liệu

Di chuyển (sao chép) dữ liệu từ nơi này sang nơi khác là một trong những loại thao tác cơ bản nhất của các chỉ thị. Khi ta nói nội dung của địa chỉ nhớ 2000 được chuyển tới một thanh ghi nào đó, điều đó có nghĩa một bản sao nội dung của địa chỉ

nhớ được chứa trong thanh ghi và bản thân từ nhớ không thay đổi. Vì vậy thuật ngữ " movement " được thay bằng " duplication " có lẽ phù hợp hơn.

Dữ liệu thường được cất ở 3 nơi trong máy tính: từ nhớ, thanh ghi, ngăn xếp. Việc di chuyển dữ liệu giữa các thanh ghi, từ nhớ cần phải xác định địa chỉ rõ ràng, trong khi di chuyển dữ liệu vào hoặc ra khỏi ngăn xếp thì không cần.

4.2. Các thao tác hai ngôi

Các thao tác 2 ngôi là các thao tác kết hợp 2 toán hạng để sinh ra một kết quả. Ví dụ: các lệnh thực hiện các phép toán số học: cộng, trừ, nhân, chia,... các phép toán lôgic: and, or, xor,...

4.3. Các thao tác một ngôi

Các thao tác một ngôi chỉ có một toán hạng và tạo ra một kết quả. Vì vậy, các lệnh thực hiện các thao tác một ngôi chỉ có 1 địa chỉ và ngắn hơn các lệnh thực hiện các thao tác 2 ngôi. Ví dụ: các thao tác dịch bit, quay bit,... là các thao tác một ngôi.

4.4. Các lệnh so sánh và nhẩy có điều kiện

Khi thực hiện tính căn bậc hai ($\sqrt{x}$), phải tiến hành kiểm tra nếu x < 0 thì đưa ra thông báo lỗi còn nếu x $\geq$ 0 thì tính căn bậc 2. Tức là phải kiểm tra x và sau đó thực hiện nhẩy tuỳ thuộc vào kết quả kiểm tra.

Khi cần kiểm tra 2 từ hoặc 2 ký tự xem cúng có bằng nhau không, từ nào lớn hơn,...Các lệnh này cần 3 địa chỉ: 2 cho dữ liệu so sánh và 1 cho địa chỉ nhẩy tới khi điều kiện là đúng.

4.5. Các lệnh gọi thủ tục

Thủ tục (subroutine: thủ tục con hay procedure: thủ tục trong kinh doanh, pháp lí) là một nhớ các lệnh hay chỉ thị thực hiện một công việc nào đó và có thể được gọi từ nhiều nơi trong chương trình. Khi thủ tục hoàn tất công việc, thủ tục phải trở về câu lệnh tiếp theo ngay sau câu lệnh gọi thủ tục trong chương trình.

4.6. Các lệnh điều khiển vòng lặp

Trong thực tế có một số lệnh cần phải một số lần liên tiếp nhất định. Do vậy, có một số chỉ thị nhằm thực hiện điều này. Các chỉ thị loại này bao gồm một bộ đếm. Bộ đếm được khởi động ở ngoài vòng lặp, chỉ thị cuối cùng của vòng lặp sẽ cập nhật bộ đếm và nếu điều kiện kết thúc vòng lặp chưa thoả chương trình sẽ quay trở lại chỉ thị đầu tiên của vòng lặp, ngược lại kết thúc và thực thi chỉ thị đầu tiên ngoài vòng lặp.

4.7. Các lệnh vào / ra

Các chỉ thị vào / ra rất đa dạng, hiện có 4 phương pháp vào / ra khác nhau:

1. Vào / ra được lập được lập trình (Programmed I/O).

2. Vào / ra được điều khiển bởi ngắt (interrapt driven I/O).

3. Vào / ra truy cập bộ nhớ trực tiếp (Direct Memory Access – DMA I/O).

4. Vào / ra sử dụng các kênh dữ liệu (I/O using data channels).

Nguyên lí của các phương pháp vào / ra trên đã được trình bày trong môn học kiến trúc máy tính.

5. Luồng điều khiển

Luồng điều khiển có liên quan đến trình tự thực thi các chỉ thị. Thường các chỉ thị được thực thi lần lượt từ các vị trí nhớ liên tiếp. Các chỉ thị gọi thủ tục là nguyên nhân làm thay đổi luồng điều khiển vì nó dừng thủ tục đang thực hiện để bắt đầu thủ tục được gọi. Ngoài ra, các bẫy và các ngắt cũng làm cho luồng điều khiển thay đổi khi có các điều kiện đặc biệt xẩy ra.

5.1. Luồng điều khiển tuần tự và các chỉ thị nhảy

Như đã nói ở trên, đa số các lệnh không làm thay đổi luồng điều khiển. Sau khi một lệnh được thực hiện, lệnh tiếp theo trong bộ nhớ được tìm - nạp và thực hiện. Sau mỗi lệnh, nội dung thanh ghi đếm chương trình tự động tăng lên theo số byte chiều dài của lệnh. Nếu chương trình chứa các lệnh nhảy thì trật tự các chỉ thị trong bộ nhớ và trật tự thực thi các lệnh sẽ khác nhau, khi đó người lập trình khó có thể theo dõi được chuỗi các chỉ thị mà CPU sẽ thực thi từ chương trình. Vì vậy, Dijkstra đã viết một bài báo có tựa đề " GO TO Statement Considered Harmful " trong đó ông đề nghị tránh dùng các câu lệnh GO TO. Bài báo này đã khởi đầu cho cuộc cách mạng khai sinh ra phương pháp lập trình có cấu trúc. Một trong những nguyên lí cơ bản của phương pháp này là thay thế các câu lệnh go to bằng những dạng có cấu trúc hơn của luồng điều khiển như vòng lặp: While, repeat ... until,...

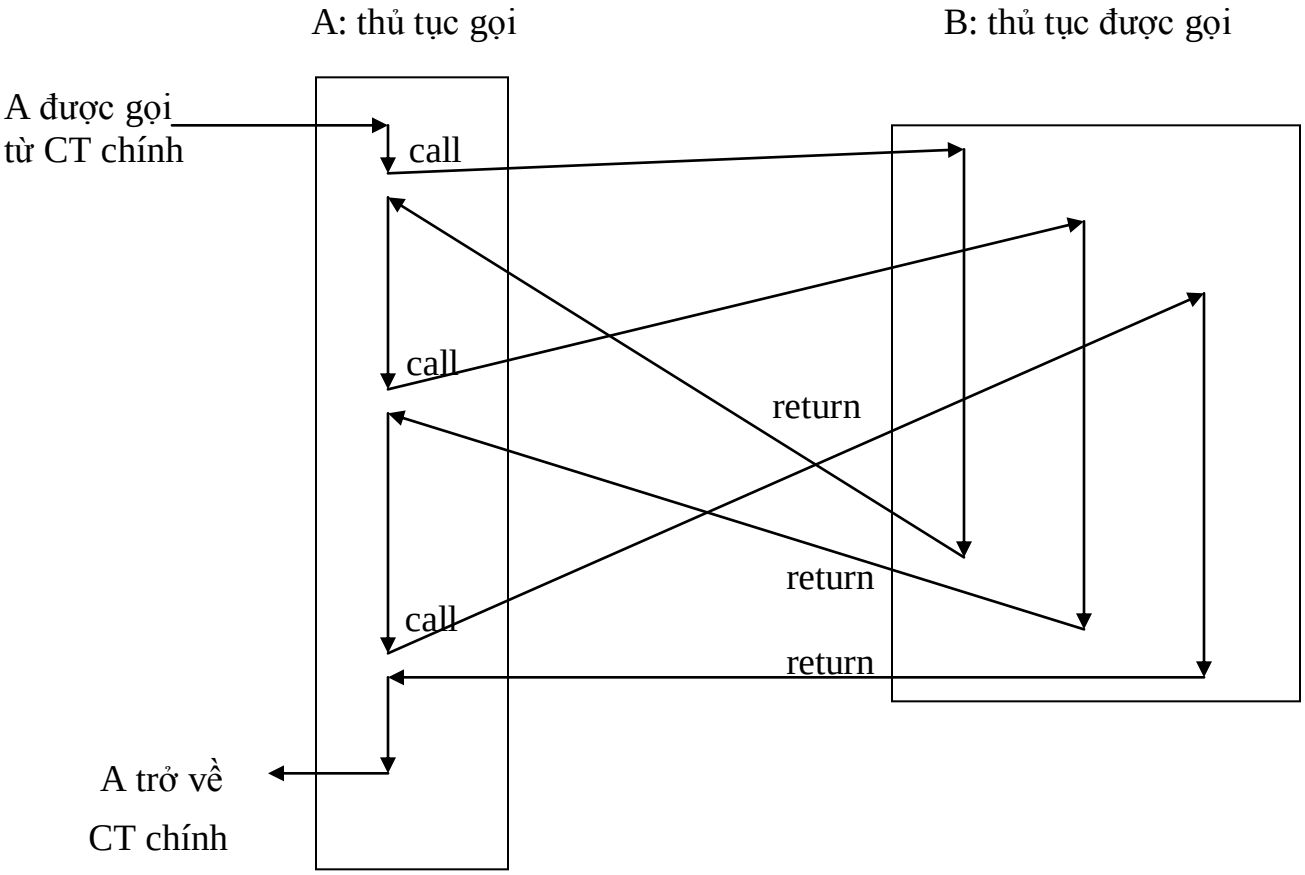
5.2. Thủ tục

Chỉ thị gọi thủ tục cũng làm thay đổi luồng điều khiển nhưng không giống trong các lệnh nhẩy. Khi thực hiện xong công việc, thủ tục bị gọi trả điều khiển về cho chỉ thị ngay sau chỉ thị gọi thủ tục.

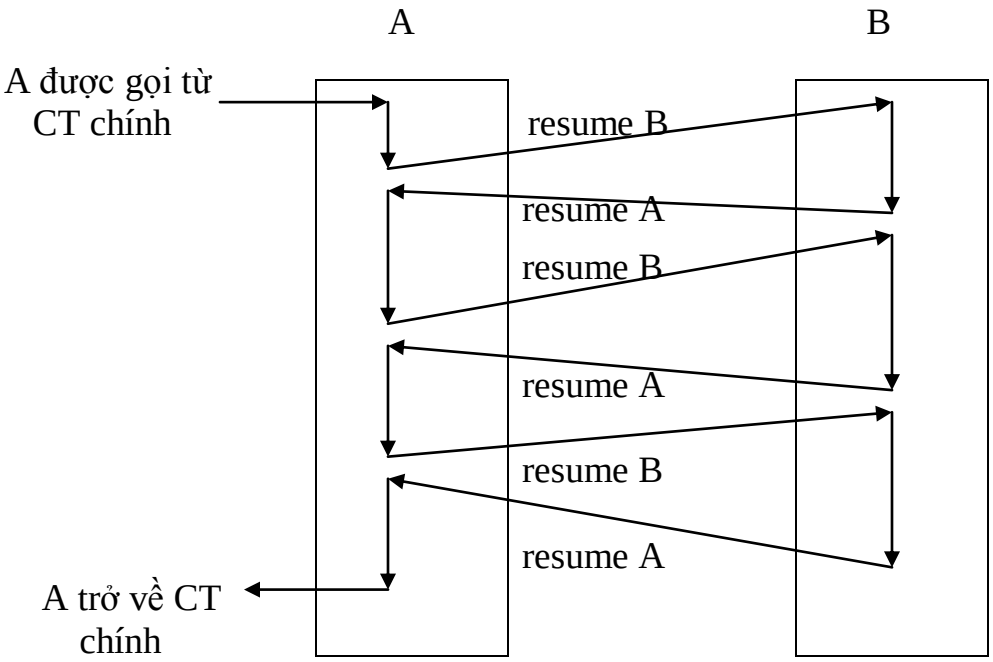
5.3. Đồng thủ tục

Thông thường khi một thủ tục A gọi một thủ tục B. Thủ tục B được thực hiện từ đầu thủ tục. Khi trở về A, việc thực thi A bắt đầu ở chỉ thị ngay sau chỉ thị gọi B (hình 4.9).

Đôi khi, xẩy ra trường hợp A và B có thể gọi lẫn nhau. Khi A chuyển điều khiển cho B, trừ lần đầu tiên được bắt đầu ở chỉ thị đầu tiên của B, còn các lần khác được bắt đầu từ chỉ thị tiếp theo ngay sau chỉ thị trở về A lần gần nhất. Trong trường hợp này, A và B được gọi là đồng thủ tục (co-routine) (hình 4.10).



Hình 4.9. Thủ tục gọi và thủ tục được gọi



Hình 4.10. Đồng thủ tục

5.4. Bẫy

Bẫy là một cách gọi thủ tục tự động khi có một điều kiện đặc biệt nào đó do chương trình tạo ra. Ví dụ: khi tràn số, tức là kết quả của một phép toán số học vượt quá phạm vi biểu diễn. Khi đó một bẫy xuất hiện, luồng điều khiển được chuyển đến một vị trí nhớ cố định nào đó thay vì tiếp tục chương trình theo trình tự. Tại vị trí đó, một thủ tục được gọi, thủ tục này thực hiện một thao tác thích hợp nào đó như là đưa ra một thông báo lỗi chẳng hạn.

5.5. Ngắt

Ngắt là sự kiện làm thay đổi luồng điều khiển của chương trình đang được thực thi không phải do chính chương trình đó gây ra mà thường liên quan đến các thiết bị vào / ra. Ngắt cũng làm ngừng chương trình đang được thực thi và chuyển điều khiển tới bộ điều khiển ngắt để thực hiện một thao tác thích hợp. Khi hoàn tất, bộ điều khiển ngắt trả điều khiển cho chương trình bị ngắt. Chương trình này khôi phục lại trạng thái như là trước khi xẩy ra ngắt.