

# Chương 2: Tập lệnh máy tính

## Kiến trúc máy tính

ThS. Đinh Xuân Trường  
[truongdx@ptit.edu.vn](mailto:truongdx@ptit.edu.vn)



Posts and Telecommunications  
Institute of Technology  
Faculty of Information Technology 1



CNTT1  
Học viện Công nghệ Bưu chính Viễn thông

January 15, 2023

# Mục tiêu Buổi 4

Tập lệnh máy

Dạng và các thành phần của lệnh

Các dạng địa chỉ và toán hạng

Các chế độ địa chỉ

Chế độ địa chỉ tức thì

Chế độ địa chỉ trực tiếp

Chế độ địa chỉ gián tiếp thanh ghi

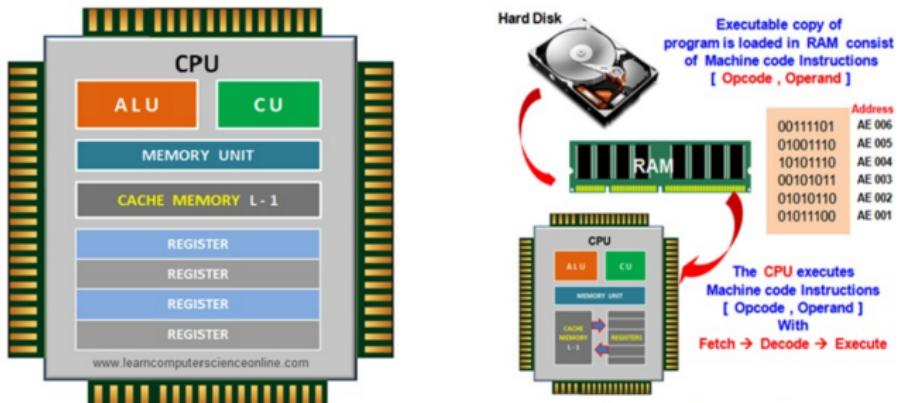
Chế độ địa chỉ gián tiếp bộ nhớ

Chế độ địa chỉ chỉ số (Indexed)

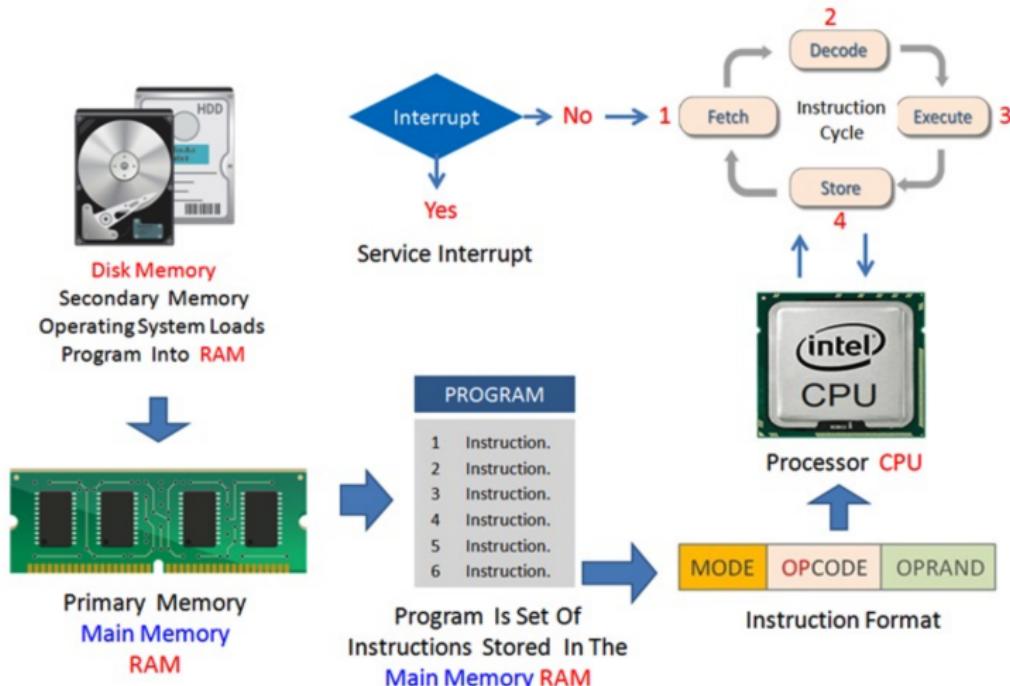
Chế độ địa chỉ tương đối

Một số dạng lệnh thông dụng

1. Tập lệnh máy
2. Khuôn dạng và các thành phần của lệnh
3. Các dạng toán hạng lệnh
4. Các chế độ địa chỉ
5. Một số dạng lệnh thông dụng



# Nhắc lại về Chu kỳ xử lý lệnh của CPU



Chương trình máy tính được tạo nên bởi một loạt các thao tác riêng lẻ được gọi là những câu lệnh / lệnh máy tính.

**Lệnh máy tính (Instruction)** là một từ nhị phân (binary word) được gán một nhiệm vụ cụ thể, hướng dẫn cho máy tính biết phải làm gì.

- ▶ Lệnh chương trình được lưu trong bộ nhớ
- ▶ Lệnh được đọc từ bộ nhớ vào CPU để giải mã và thực hiện
- ▶ Mỗi lệnh có một chức năng riêng

## 8086 INSTRUCTION SET

OPCODE	DESCRIPTION				
AAA	ASCII adjust addition	JNAE	slabel Jump if not above or equal	PUSHF	Push flags onto stack
ASCII adjust division	JNB	slabel Jump if not below	RCL	dt, cnt Rotate left through carry	
AAM	ASCII adjust multiply	JNBE	slabel Jump if below or equal	RCR	dt, cnt Rotate right through carry
AAS	ASCII adjust subtraction	JNC	slabel Jump if no carry	REP	Repeat string operation
ADC	dt, sc Add with carry	JNE	slabel Jump if not equal	REPE	Repeat while equal
ADD	dt, sc Add	JNG	slabel Jump if not greater	REPZ	Repeat while zero
AND	dt, sc Logical AND	JNGE	slabel Jump if not greater or equal	REPNE	Repeat while not equal
CALL	proc Call a procedure	JNL	slabel Jump if not less	REPNZ	Repeat while not zero
CBW	Convert byte to word	JNLE	slabel Jump if not less or equal	RET	[pop] Return from procedure
CLC	Clear carry flag	JNZ	slabel Jump if not zero	ROL	dt, cnt Rotate left
CDL	Clear direction flag	JNO	slabel Jump if not overflow	ROR	dt, cnt Rotate right
CLI	Clear interrupt flag	JNP	slabel Jump if not parity	SAHF	Store AH into flags
CMC	Complement carry flag	JNS	slabel Jump if not sign	SAL	dt, cnt Shift arithmetic left
CMP	dt, sc Compare	JO	slabel Jump if overflow	SHL	dt, cnt Shift logical left
CMPS	[dt, sc] Compare string	JPO	slabel Jump if parity odd	SAR	dt, cnt Shift arithmetic right
CMPSB	" " bytes	JP	slabel Jump if parity	SBB	dt, sc Subtract with borrow
CMPSW	" " words	JPE	slabel Jump if parity even	SCAS	[dt] Scan string
CWD	Convert word to double word	JS	slabel Jump if sign	SCASB	" " byte
		JZ	slabel Jump if zero	SCASW	" " word

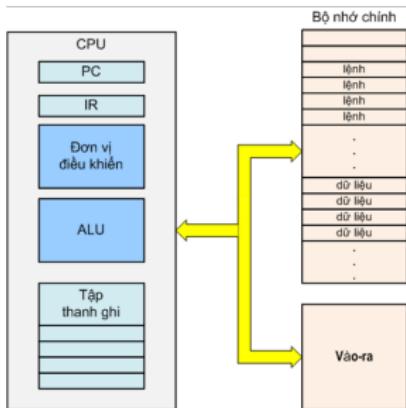
Mỗi bộ xử lý có một tập lệnh xác định:

- ▶ Tập lệnh thường có hàng chục đến hàng trăm lệnh
- ▶ Mỗi lệnh là một chuỗi số nhị phân mà bộ xử lý hiểu được để thực hiện một thao tác xác định.
- ▶ Các lệnh được mô tả bằng các ký hiệu gợi nhớ dạng text: chính là các lệnh của hợp ngữ (assembly language). Ví dụ: ADD A, 1

Tập lệnh gồm nhiều lệnh, được chia thành các nhóm theo chức năng:

- ▶ Chuyển dữ liệu (data movement)
- ▶ Tính toán (computational)
- ▶ Điều kiện và rẽ nhánh (conditioning & branching)
- ▶ Các lệnh khác

Quá trình thực hiện lệnh được chia thành các pha hay giai đoạn (stage).  
Mỗi lệnh có thể được thực hiện theo 4 giai đoạn:



- ▶ Đọc lệnh IF (Instruction Fetch): lệnh được đọc từ bộ nhớ vào CPU
- ▶ Giải mã lệnh ID (Instruction Decode): CPU giải mã lệnh
- ▶ Chạy lệnh IE (Instruction Execution): CPU thực hiện lệnh
- ▶ Ghi WB (Write Back): kết quả (nếu có) ghi vào thanh ghi/bộ nhớ

**Chu kỳ lệnh** (Instruction Cycle) là khoảng thời gian để CPU thực hiện xong một lệnh kể từ khi CPU cấp phát tín hiệu địa chỉ ô nhớ chứa lệnh đến khi nó hoàn tất việc thực hiện lệnh đó.

- ▶ Một chu kỳ thực hiện lệnh gồm một số giai đoạn thực hiện lệnh
- ▶ Một giai đoạn thực hiện lệnh có thể gồm một số chu kỳ máy
- ▶ Một chu kỳ máy có thể gồm một số chu kỳ đồng hồ

Một chu kỳ thực hiện lệnh có thể gồm các thành phần sau:

- ▶ Chu kỳ đọc lệnh
- ▶ Chu kỳ đọc/ghi bộ nhớ (memory read / write)
- ▶ Chu kỳ đọc/ghi thiết bị ngoại vi (I/O read/write)
- ▶ Chu kỳ bus rỗi (bus idle)

# Dạng và các thành phần của lệnh

Dạng tổng quát của lệnh máy tính gồm có 2 phần chính:

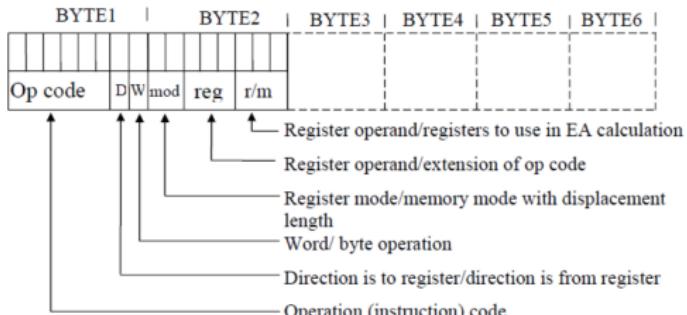
- ▶ (1) Mã lệnh (Opcode – Operation code)
- ▶ (2) Địa chỉ của các toán hạng (Addresses of Operands).

Mã lệnh	Địa chỉ của các toán hạng	
Opcode	Addresses of Operands	
Opcode	Des addr.	Source addr.

- ▶ **Mã lệnh** (operation code - opcode): mã hóa cho thao tác mà bộ xử lý phải thực hiện
- ▶ **Địa chỉ toán hạng**: chỉ ra nơi chứa các toán hạng mà thao tác sẽ tác động:
  - Toán hạng nguồn (*source operand*): dữ liệu vào của thao tác
  - Toán hạng đích (*destination operand*): dữ liệu ra của thao tác

# Dạng và các thành phần của lệnh (cont.)

Một số ví dụ về dạng lệnh thực tế:



## Vi xử lý 8086:

	31	26 25	21 20	16 15	11 10	6 5	0
R-Type	opcode	rs	rt	rd	sa	function	

	31	26 25	21 20	16 15	0
I-Type	opcode	rs	rt	immediate	

	31	26 25	0
Mã MIPS: J-Type	opcode	Instr_index	

# Các dạng địa chỉ và toán hạng

Dựa vào số lượng các toán hạng, các lệnh đưa chia thành 5 loại:

- ▶ Toán hạng 3 địa chỉ;
- ▶ Toán hạng 2 địa chỉ
- ▶ Toán hạng 1.5 địa chỉ;
- ▶ Toán hạng 1 địa chỉ
- ▶ Toán hạng 0 địa chỉ.

Một số quy ước về lệnh dạng CPU:

- ▶  $R_i$ : là các thanh ghi của CPU
- ▶  $R_{acc}$ : là thanh ghi Accumulation - tích luỹ / Tổng
- ▶  $(R_i)$ : Nội dung ô nhớ có địa chỉ được lưu trong thanh ghi  $R_i$
- ▶  $(100)$ : Nội dung ô nhớ có địa chỉ được lưu trong ô nhớ 100
- ▶ A, B, C, 1000 là địa chỉ các ô nhớ
- ▶ M[100]: tham chiếu đến nội dung ô nhớ thứ 100
- ▶ #100: Hằng số 100

### Toán hạng 3 địa chỉ

Khuôn dạng: **OPCODE Addr1, Addr2, Addr3**

Mỗi địa chỉ  $Addr1, Addr2, Addr3$  tham chiếu đến một ô nhớ hoặc một thanh ghi.

Ví dụ:

► ADD  $R_1, R_2, R_3;$

$$R_3 + R_2 \rightarrow R_1$$

$R_2$  cộng  $R_3$  sau đó kết quả đưa vào  $R_1$

$R_i$  là các thanh ghi CPU

► ADD A, B, C;

$$M[B] + M[C] \rightarrow M[A]$$

A, B, C là các vị trí trong bộ nhớ

### Toán hạng 2 địa chỉ

Khuôn dạng: **OPCODE Addr1, Addr2**

Mỗi địa chỉ  $Addr1, Addr2$  tham chiếu đến một ô nhớ hoặc một thanh ghi.

Ví dụ:

- ▶ ADD  $R_1, R_2;$

$$R_1 + R_2 \rightarrow R_1$$

$R_2$  cộng  $R_1$  sau đó kết quả đưa vào  $R_1$

$R_i$  là các thanh ghi CPU

- ▶ ADD A, B;

$$M[B] + M[A] \rightarrow M[A]$$

A, B là các vị trí trong bộ nhớ

### Toán hạng 1 địa chỉ

Khuôn dạng: **OPCODE Addr**

Mỗi địa chỉ *Addr* tham chiếu đến một ô nhớ hoặc một thanh ghi.

Khuôn dạng này sử dụng  $R_{acc}$  (thanh ghi tích lũy) mặc định cho địa chỉ thứ 2

Ví dụ:

- ▶ ADD  $R_1$ ;

$$R_1 + R_{acc} \rightarrow R_{acc}$$

$R_{acc}$  cộng  $R_1$  sau đó kết quả đưa vào  $R_{acc}$

$R_i$  là các thanh ghi CPU

- ▶ ADD A;

$$M[A] + R_{acc} \rightarrow R_{acc}$$

A là vị trí trong bộ nhớ

### Toán hạng 1.5 địa chỉ

Khuôn dạng: **OPCODE Addr1, Addr2**

Một địa chỉ tham chiếu tới 1 ô nhớ và địa chỉ còn lại tham chiếu tới 1 thanh ghi

Là dạng hỗn hợp giữa các toán hạng thanh ghi và vị trí bộ nhớ.  
Ví dụ:

- ▶ ADD  $R_1$ , B;

$$M[B] + R_1 \rightarrow R_1$$

$R_1$  cộng  $M[B]$  sau đó kết quả đưa vào  $R_1$

$R_i$  là các thanh ghi CPU

B là vị trí trong bộ nhớ

## Toán hạng 0 địa chỉ

Khuôn dạng: **OPCODE**

Được thực hiện trong các lệnh mà thực hiện các thao tác ngăn xếp:

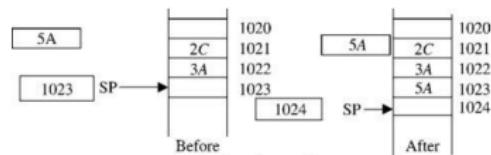
**PUSH & POP**

Ví dụ:

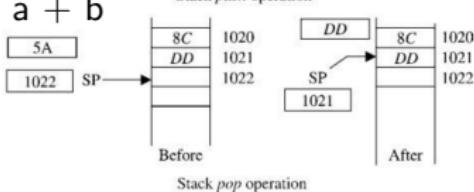
PUSH a

PUSH b

ADD POP c



► Có nghĩa  $c = a + b$



# Các chế độ địa chỉ

Mã lệnh	Địa chỉ của các toán hạng	
Opcode	Addresses of Operands	
Opcode	Des addr.	Source addr.

Toán hạng của lệnh có thể là:

- ▶ Một giá trị cụ thể nằm ngay trong lệnh
- ▶ Nội dung của thanh ghi
- ▶ Nội dung của ngăn nhớ hoặc cổng vào-ra

**Chế độ địa chỉ** (*Addressing modes*) là phương thức hoặc cách thức CPU tổ chức các toán hạng của lệnh hay cách thức địa chỉ hóa trong trường địa chỉ của lệnh để xác định nơi chứa toán hạng

- ▶ Chế độ địa chỉ cho phép CPU kiểm tra dạng lệnh và tìm các toán hạng của lệnh.
- ▶ Số lượng các chế độ địa chỉ phụ thuộc vào thiết kế của CPU.

# Các chế độ địa chỉ (cont.)

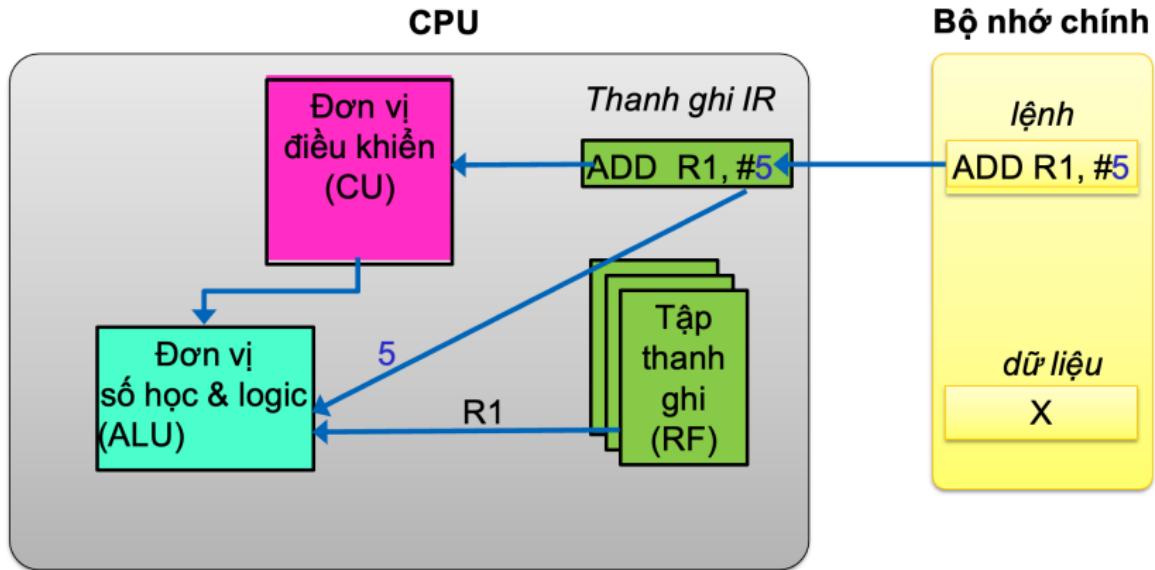
## Các chế độ địa chỉ:

Chế độ địa chỉ	Ý nghĩa	Ví dụ	Thực hiện
Tức thì	Giá trị của toán hạng được chứa trong lệnh	LOAD Ri, #1000	Ri $\leftarrow$ 1000
Trực tiếp	Địa chỉ của toán hạng được chứa trong lệnh	LOAD Ri, 1000	Ri $\leftarrow$ M[1000]
Gián tiếp thanh ghi	Giá trị của thanh ghi trong lệnh là địa chỉ bộ nhớ chứa toán hạng	LOAD Ri, (Rj)	Ri $\leftarrow$ M[Rj]
Gián tiếp bộ nhớ	Địa chỉ bộ nhớ trong lệnh chứa địa chỉ bộ nhớ của toán hạng	LOAD Ri, (1000)	Ri $\leftarrow$ M[M[1000]]
Chỉ số	Địa chỉ của toán hạng là tổng của hằng số (trong lệnh) và giá trị của một thanh ghi chỉ số	LOAD Ri, X(Rind)	Ri $\leftarrow$ M[X+ Rind]
Tương đối	Địa chỉ của toán hạng là tổng của hằng số và giá trị của thanh ghi con đếm chương trình	LOAD Ri, X(PC)	Ri $\leftarrow$ M[X+ PC]

# Các chế độ địa chỉ

## Chế độ địa chỉ tức thì

Lệnh: ADD R1, #5



# Các chế độ địa chỉ (cont.)

## Chế độ địa chỉ tức thì



Dạng lệnh trong chế độ địa chỉ tức thì:

Mã lệnh		# Hằng số
---------	--	-----------

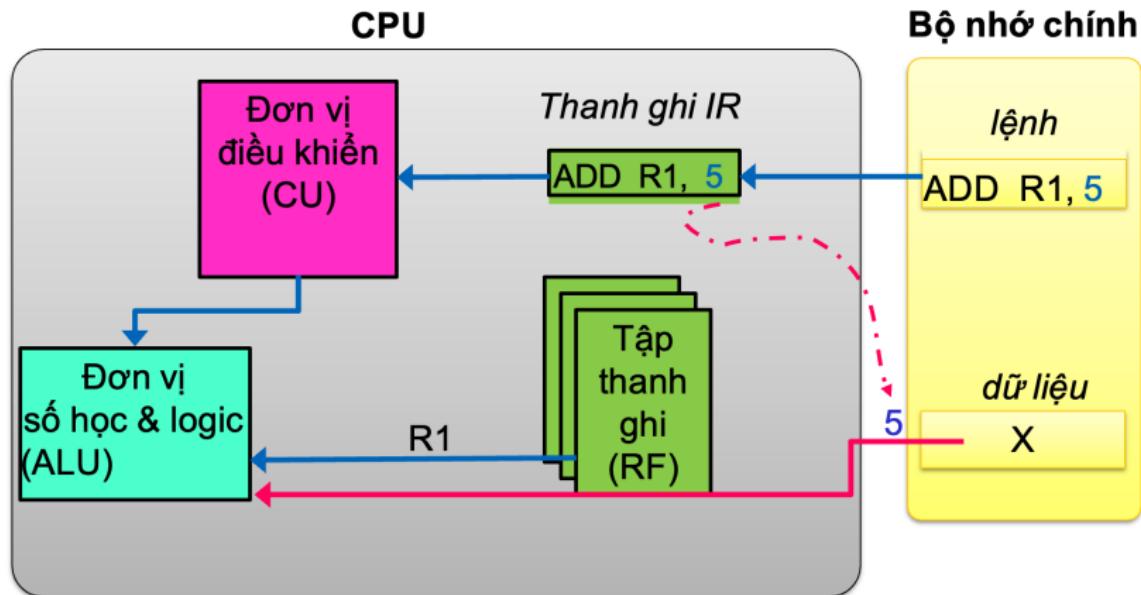
- ▶ Giá trị của toán hạng nguồn có sẵn trong lệnh (hằng số)
- ▶ Toán hạng đích có thể là thanh ghi hoặc một vị trí bộ nhớ
- ▶ Ví dụ:

- LOAD R1, #1000;                            1000 → R1  
*Giá trị 1000 được tải vào thanh ghi R1*
- ADD 100, #5;                                 $M[100] = M[100] + 5$

- ▶ **Nhận xét** về chế độ địa chỉ tức thì:

- Không tham chiếu bộ nhớ
- Truy nhập toán hạng rất nhanh
- Dải giá trị của toán hạng bị hạn chế

Lệnh: ADD R1, 5



# Các chế độ địa chỉ (cont.)

## Chế độ địa chỉ trực tiếp

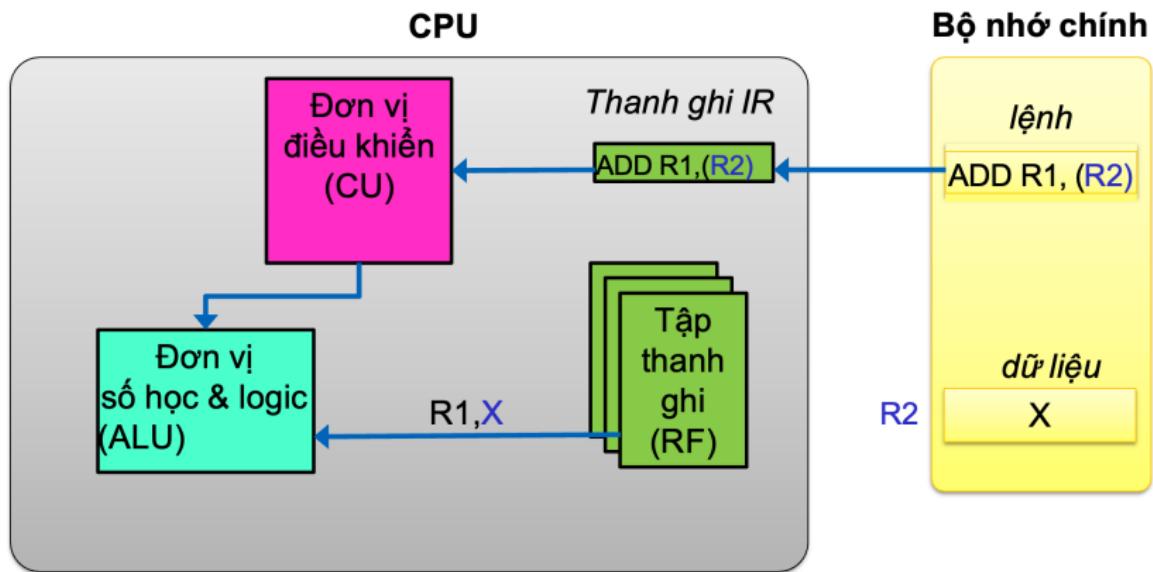


Dạng lệnh trong chế độ địa chỉ trực tiếp:

Mã lệnh		Ô nhớ
---------	--	-------

- ▶ Một toán hạng là ngăn nhớ có địa chỉ trực tiếp trong lệnh
- ▶ Toán hạng kia là thanh ghi hoặc 1 địa chỉ ô nhớ
- ▶ Ví dụ:
  - LOAD R1, 1000;  $M[1000] \rightarrow R1$   
*Giá trị lưu trong vị trí ô nhớ 1000 được tải vào thanh ghi R1*
  - ADD R1, 5;  $R1 = R1 + M[5]$   
*Tìm toán hạng trong bộ nhớ ở địa chỉ 5 và cộng nội dung thanh ghi R1 với nội dung của ngăn nhớ có địa chỉ là 5*
- ▶ **Nhận xét về chế độ địa chỉ trực tiếp:**
  - CPU tham chiếu bộ nhớ một lần để truy nhập dữ liệu

Lệnh: ADD R1, (R2)



# Các chế độ địa chỉ (cont.)

## Chế độ địa chỉ gián tiếp thanh ghi



Dạng lệnh trong chế độ địa chỉ gián tiếp thanh ghi:

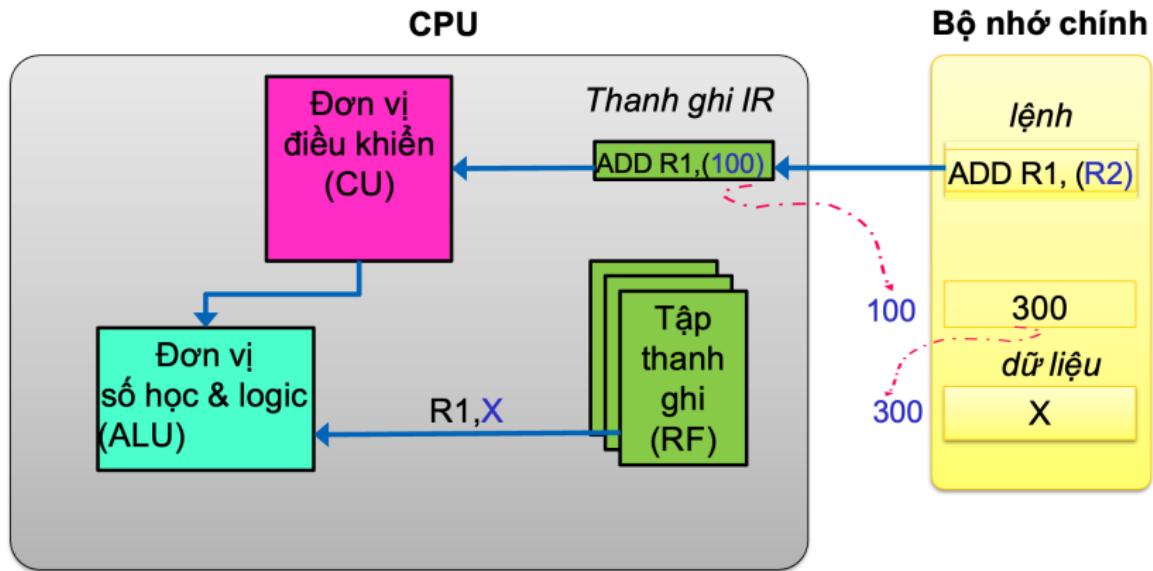
Mã lệnh	(Thanh ghi)
---------	-------------

- ▶ Một thanh ghi được sử dụng để lưu địa chỉ của toán hạng
- ▶ Ví dụ:

- LOAD Rj , (Ri);                           $M[Ri] \rightarrow Rj;$   
*Tải giá trị tại vị trí bộ nhớ có địa chỉ được lưu trong Ri vào thanh ghi Rj*

- ▶ **Nhận xét** về chế độ địa chỉ gián tiếp thanh ghi:
  - CPU cần tới một lần tham chiếu bộ nhớ cho một truy nhập.

Lệnh: ADD R1, (100)



# Các chế độ địa chỉ (cont.)

## Chế độ địa chỉ gián tiếp bộ nhớ

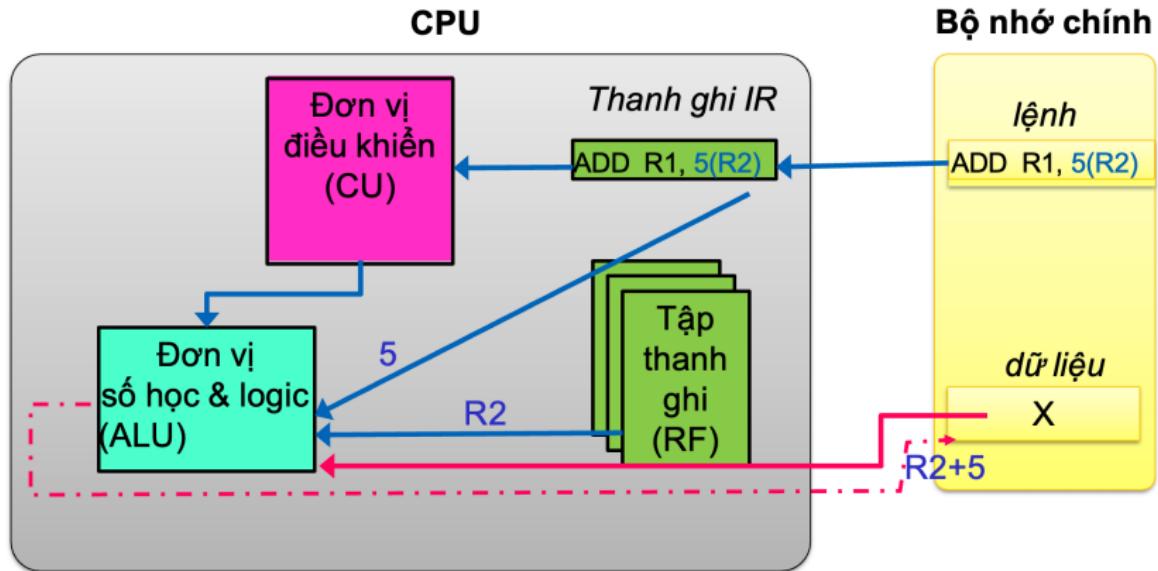


Dạng lệnh trong chế độ địa chỉ gián tiếp bộ nhớ:

Mã lệnh	(Ô nhớ)
---------	---------

- ▶ Một vị trí trong bộ nhớ được sử dụng để lưu địa chỉ của toán hạng
- ▶ Ví dụ:
  - LOAD Rj , (100);  $M[M[100]] \rightarrow Rj;$   
*Tải giá trị tại vị trí bộ nhớ có địa chỉ được lưu trong ô nhớ 100 vào thanh ghi Rj*
- ▶ **Nhận xét** về chế độ địa chỉ gián tiếp bộ nhớ:
  - CPU tham chiếu bộ nhớ hai lần trong mỗi lần truy nhập.

Lệnh: ADD R1, 5(R2)



# Các chế độ địa chỉ (cont.)

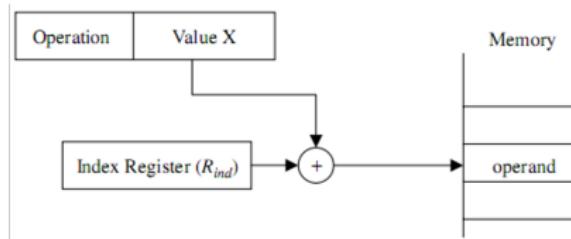
## Chế độ địa chỉ chỉ số (Indexed)

Dạng lệnh trong chế độ địa chỉ chỉ số:

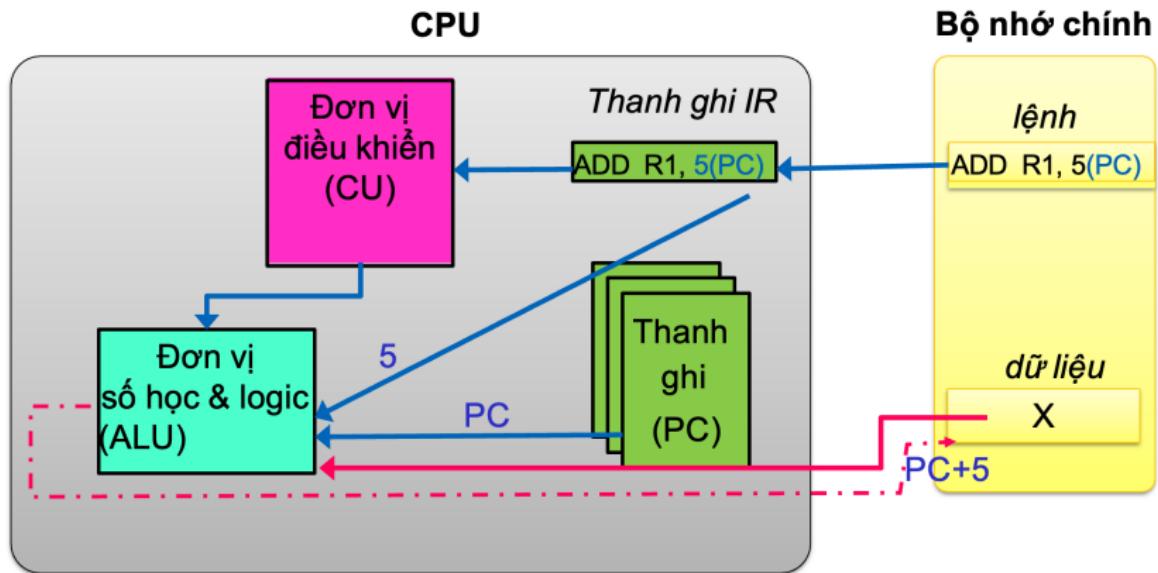
Mã lệnh	Hằng số (Thanh ghi)
---------	---------------------

- ▶ Để xác định toán hạng trường địa chỉ chứa hai phần: Tên thanh ghi và Hằng số (offset)
- ▶ Địa chỉ của toán hạng = nội dung thanh ghi + hằng số
- ▶ Ví dụ:

- LOAD Ri, X(Rind);  $M[X+Rind] \rightarrow R_i;$   
*Tải giá trị tại vị trí bộ nhớ có địa chỉ được lưu trong thanh ghi Rind cộng với hằng số X vào thanh ghi R*



Lệnh: ADD R1, 5(PC)



# Các chế độ địa chỉ (cont.)

## Chế độ địa chỉ tương đối

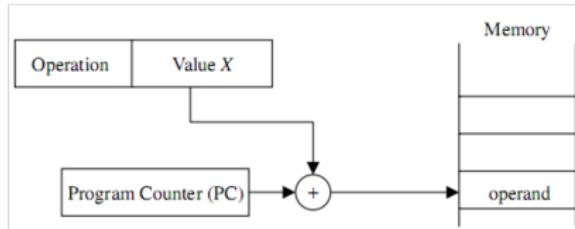
Dạng lệnh trong chế độ địa chỉ tương đối:



- Để xác định toán hạng trường địa chỉ chứa hai phần: Thanh ghi PC và Hằng số (offset)
- Địa chỉ của toán hạng = nội dung thanh ghi PC + hằng số
- Ví dụ:

- LOAD Ri, X(PC); ;  $M[PC+X] \rightarrow Ri;$

Tải giá trị tại vị trí bộ nhớ có địa chỉ được lưu trong thanh ghi PC cộng với hằng số X vào thanh ghi R



# Một số dạng lệnh thông dụng

Phụ thuộc thiết kế CPU, tập lệnh của CPU có thể có số lượng các lệnh rất khác nhau bao gồm :

- ▶ Các lệnh vận chuyển dữ liệu
- ▶ Các lệnh số học và logic
- ▶ Các lệnh điều khiển chương trình
- ▶ Các lệnh vào & ra

### Chuyển dữ liệu giữa các phần nhớ của máy tính:

- ▶ Giữa các thanh ghi trong CPU
  - MOVE Ri, Rj ;               $Rj \rightarrow Ri$
- ▶ Giữa thanh ghi CPU và một vị trí trong bộ nhớ
  - MOVE Ri, 1000 ;               $M[1000] \rightarrow R$
- ▶ Giữa các vị trí trong bộ nhớ
  - MOVE 1000, (Rj) ;               $M[Rj] \rightarrow M[1000]$

### Một số lệnh vận chuyển dữ liệu thông dụng:

- ▶ MOVE: chuyển dữ liệu giữa các thanh ghi và ô nhớ
- ▶ LOAD: nạp nội dung 1 ô nhớ vào 1 thanh ghi
- ▶ STORE: lưu nội dung 1 thanh ghi ra 1 ô nhớ
- ▶ PUSH/POP: đẩy/lấy dữ liệu vào/ra ngăn xếp

### Thực hiện các thao tác số học và logic giữa các thanh ghi và nội dung ô nhớ

► Ví dụ:

- ADD R1, R2, R3;                   $R2 + R3 \rightarrow R1$  ;
- SUBTRACT R1, R2, R3 ;                   $R2 - R3 \rightarrow R1$

Một số lệnh toán học thông :

- ADD: cộng 2 toán hạng
- SUBTRACT: trừ 2 toán hạng
- MULTIPLY: nhân 2 toán hạng
- DIVIDE: chia số học
- INCREMENT: tăng 1
- DECREMENT: giảm 1

# Một số dạng lệnh thông dụng (cont.)

## Lệnh số học và logic



Các lệnh logic thông dụng:

- ▶ NOT: phủ định
- ▶ AND: và
- ▶ OR: hoặc
- ▶ XOR: hoặc loại trừ
- ▶ COMPARE: so sánh
- ▶ SHIFT: dịch
- ▶ ROTATE: quay

- ▶ Được dùng để thay đổi trình tự các lệnh được thực hiện:
  - Các lệnh rẽ nhánh (nhảy) có điều kiện (conditional branching/ jump)
  - Các lệnh rẽ nhánh (nhảy) không điều kiện (unconditional branching/ jump)
  - CALL và RETURN: gọi thực hiện và trở về từ chương trình con
- ▶ Đặc tính chung của các lệnh này là quá trình thực hiện lệnh của chúng làm thay đổi giá trị PC
- ▶ Sử dụng các cờ ALU để xác định các điều kiện

# Một số dạng lệnh thông dụng (cont.)

Lệnh điều khiển/ tuần tự



Một số lệnh vận chuyển dữ liệu thông dụng:

- ▶ BRANCH – IF – CONDITION: chuyển đến thực hiện lệnh ở địa chỉ mới nếu điều kiện là đúng
- ▶ JUMP: chuyển đến thực hiện lệnh ở địa chỉ mới
- ▶ BRANCH-IF-EQUAL: chuyển đến thực hiện lệnh ở địa chỉ mới nếu kết quả của phép toán  $= 0$
- ▶ BRANCH-IF-GREATER-THAN: chuyển đến thực hiện lệnh ở địa chỉ mới nếu kết quả của phép toán  $> 0$
- ▶ CALL: chuyển đến thực hiện chương trình con
- ▶ RETURN: trở về (từ chương trình con) thực hiện tiếp chương trình

# Một số dạng lệnh thông dụng (cont.)

Lệnh điều khiển/ tuần tự



Ví dụ: Cộng nội dung 100 ô nhớ cạnh nhau, bắt đầu từ địa chỉ 1000. Kết quả lưu vào R0.

LOAD R1, #100;	R1 <- 100
LOAD R2, #1000;	R2 <- 1000
LOAD R0, #0;	R0 <- 0
LOOP: ADD R0, (R2);	R0 <- R0 + M[R2]
INCREMENT R2;	R2 <- R2 + 1
DECREMENT R1;	R1 <- R1 - 1
BRANCH-IF-GREATER-THAN Loop;	
; Quay lại thực hiện lệnh sau nhãn Loop nếu $R1 > 0$ .	

# Một số dạng lệnh thông dụng

## Các lệnh vào/ ra



- ▶ Được dùng để truyền dữ liệu giữa máy tính và các thiết bị ngoại vi
- ▶ Các thiết bị ngoại vi giao tiếp với máy tính thông qua các cổng. Mỗi cổng có một địa chỉ dành riêng
- ▶ Hai lệnh I/O cơ bản được sử dụng là các lệnh INPUT và OUTPUT
  - Lệnh INPUT được dùng để chuyển dữ liệu từ thiết bị ngoại vi vào tới bộ vi xử lý
  - Lệnh OUTPUT dùng để chuyển dữ liệu từ VXL ra thiết bị đầu ra

# Các ví dụ

CLEAR R0;            R0 <- 0  
CLEAR R2;            R2 <- 0  
MOVE R1, #100;       R1 <- 100

LAP:

ADD R0, 1000(R2);        R0 <- R0 + M[R2+1000]  
INCREMENT R2;            R2 <- R2 + 1  
DECREMENT R1;            R1 <- R1 - 1  
BRANCH\_IF > 0 LAP;      go to LAP if R1 > 0  
STORE 2000, R0;           M[2000] <- R0

## Chương 2

- ▶ Tập lệnh máy
- ▶ Khuôn dạng và các thành phần của lệnh
- ▶ Các dạng toán hạng lệnh
- ▶ Các chế độ địa chỉ
- ▶ Một số dạng lệnh thông dụng

## Tiếp theo Chương 2 - CPU Pipeline

- ▶ Giới thiệu về CPU pipeline
- ▶ Các vấn đề của pipeline
- ▶ Xử lý xung đột dữ liệu và tài nguyên
- ▶ Xử lý rẽ nhánh (branch)
- ▶ Super pipeline

# Câu hỏi và bài tập

► **Bài 1:** Cho đoạn lệnh sau:

```
MOVE R0, #100;  
CLEAR R1;  
CLEAR R2;
```

LAP:

```
ADD R1, 2000(R2);  
DECREMENT R2;  
DECREMENT R0;  
BRANCH_IF>0 LAP;  
STORE 3000, R1;
```

1. Hãy giải thích ý nghĩa của từng lệnh
2. Chỉ ra chế độ địa chỉ của từng lệnh (đối với các lệnh có 2 toán hạng)
3. Đoạn lệnh trên thực hiện công việc gì?

# Câu hỏi và bài tập (cont.)

- **Bài 2:** Biết  $R0 = 1500$ ,  $R1 = 4500$ ,  $R2 = 1000$ ,  $M[1500] = 3000$ ,  $M[4500] = 500$

```
ADD R2, (R0);  
SUBTRACT R2, (R1);  
MOVE 500(R0), R2;  
LOAD R2, #5000;  
STORE 100(R2), R0;
```

1. Chỉ rõ chế độ địa chỉ của từng lệnh
  2. Hãy chỉ ra giá trị của thanh ghi và tại vị trí trong bộ nhớ qua mỗi lệnh thực hiện
- **Bài 3:** Cho một mảng gồm 10 số, được lưu trữ liên tiếp nhau trong bộ nhớ, bắt đầu từ vị trí ô nhớ 1000.  
Viết đoạn chương trình tính tổng các số **lớn hơn 1** trong mảng đó và lưu kết quả vào ô nhớ 2000.  
Biết rằng mỗi ô nhớ lưu trữ một phần tử trong mảng!

# Câu hỏi và bài tập (cont.)

► **Bài 4:** Cho một mảng gồm 10 số, được lưu trữ cách nhau 1 ô nhớ trong bộ nhớ, bắt đầu từ vị trí ô nhớ 1000.

Viết đoạn chương trình tính tổng các số **nhỏ hơn -1** trong mảng đó và lưu kết quả vào ô nhớ 2500.

► **D20 – Đề 1:** Cho đoạn chương trình sau (R1, R2 là các thanh ghi và lệnh quy ước theo dạng LỆNH <ĐÍCH> <GỐC>):

- (1) MOVE R0, #400
- (2) LOAD R1, #2000
- (3) STORE (R1), R0
- (4) SUBSTRACT R0, #20
- (5) ADD 2000, #10
- (6) ADD R0, (R1)

1. Nêu ý nghĩa của từng lệnh và xác định giá trị R0 sau khi thực hiện xong lệnh số (6)

# Câu hỏi và bài tập (cont.)

- **D20 – Đề 2:** Cho đoạn chương trình sau (R1, R2 là các thanh ghi và lệnh quy ước theo dạng LỆNH <ĐÍCH> <GỐC>):

- (1) STORE -100(R2), R1
- (2) LOAD R1, (00FF)
- (3) COMPARE R3, R4
- (4) JUMP-IF-EQUAL Label
- (5) ADD R3, R4
- (6) ADD R2, 2
- (7) Label:

1. Xác định chế độ địa chỉ và ý nghĩa của từng lệnh;
2. Nêu hướng giải quyết xung đột dữ liệu trong pipeline khi thực hiện đoạn chương trình trên biết mỗi lệnh được chia thành 5 giai đoạn.
3. Giả thiết  $R3 \neq R4$  và mỗi giai đoạn thực hiện lệnh đều thực hiện trong thời gian là 0.1ns, so sánh thời gian CPU chạy hết 6 lệnh đầu tiên trong trường hợp không sử dụng cơ chế pipeline và có sử dụng cơ chế pipeline trong ý 2.

# Câu hỏi và bài tập (cont.)

► **D20 – Đề 5:** Cho đoạn chương trình sau (R1, R2 là các thanh ghi và lệnh quy ước theo dạng LỆNH <ĐÍCH> <GỐC>):

- (1) LOAD R2, #400
- (2) LOAD R1, #1200
- (3) STORE (R1), R2
- (4) SUBSTRACT R2, #20
- (5) ADD 1200, #10
- (6) ADD R2, (R1)

1. Nêu ý nghĩa của từng lệnh;
2. Xác định giá trị của thanh ghi R2 sau khi thực hiện xong lệnh số (6)
3. Nêu một hướng giải quyết xung đột dữ liệu trong pipeline khi thực hiện đoạn chương trình trên biết rằng mỗi lệnh được chia thành 5 giai đoạn trong pipeline: Đọc lệnh (IF), giải mã & đọc toán hạng (ID), truy nhập bộ nhớ (MEM), thực hiện (EX) và lưu kết quả (WB).

# Các dạng bài tập tập lệnh CPU

Dịch các kiểu lệnh mã C thành mã máy dạng lệnh CPU. Sau đó, nêu hướng giải quyết xung đột dữ liệu trong pipeline khi thực hiện đoạn chương trình trên.

- ▶ **Bài 1 - Mảng trong C:** A là mảng các phần tử 32-bit

$$g = g - A[8];$$

$$A[12] = h + A[8];$$

Biết g được lưu ở R1, h ở R2 và R3 chứa địa chỉ cơ sở của mảng A.

- ▶ **Bài 2.1 - Các lệnh rẽ nhánh và lệnh nhảy:**

$$\text{if } (i == j)$$

$$f = g + h;$$

$$f = f - i;$$

Biết f, g, h, i, j đang được lưu giá trị ở thanh ghi R1, R2, R3, R4, R5.

# Các dạng bài tập tập lệnh CPU (cont.)

## ► Bài 2.2 - Các lệnh rẽ nhánh và lệnh nhảy:

```
if (i==j)
    f = g+h;
else
    f = f-i;
```

Biết f, g, h, i, j đang được lưu giá trị ở thanh ghi R1, R2, R3, R4, R5.

## ► Bài 3 - Lệnh switch/case: với mã C biết amount được lưu tại R1, fee được lưu tại R2:

```
switch (amount) {
    case 20: fee = 2;
    break;
    case 50: fee = 3;
    break;
    case 100: fee = 5;
    break;
    default: fee = 0;
}
```

- **Bài 4 - Dịch câu lệnh vòng lặp While:** Biết i được lưu ở R1, k ở R2, địa chỉ của mảng A ở R3 và A là mảng các phần tử 32-bit.

```
while (A[i] == k)
    i += 1;
```

- **Bài 5.1 - Dịch vòng lặp For:**

```
int sum = 0;
int i;
for (i=0; i!=10; i = i+1) {
    sum = sum + i;
}
```

- **Bài 5.2 - Dịch vòng lặp For:**

```
int sum = 0;
int i;
for (i=1; i < 101; i = i*2) {
    sum = sum + i; }
```

# Các dạng bài tập tập lệnh CPU (cont.)

- **Bài 5.3 - Dịch vòng lặp For:** Vòng lặp truy cập mảng dữ liệu giả thiết array có địa chỉ cơ sở đặt tại R0 và mảng A là mảng các phần tử 32-bit.

```
int array[1000];
int i;
for (i=0; i < 1000; i = i + 1)
    array[i] = array[i] * 8;
```