



COMPUTER ENGINEERING

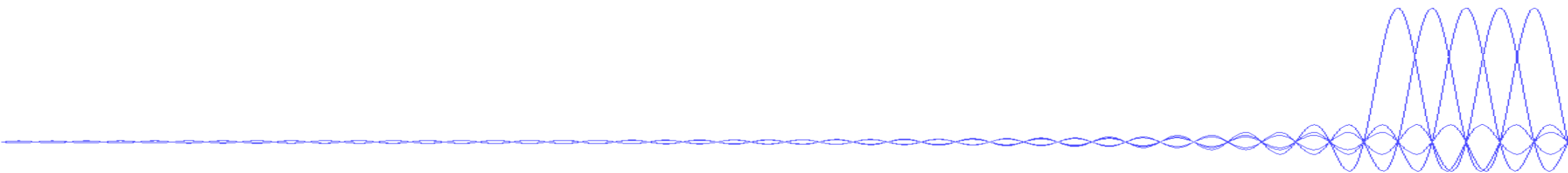
# KIẾN TRÚC MÁY TÍNH



**UIT**  
TRƯỜNG ĐẠI HỌC  
CÔNG NGHỆ THÔNG TIN

## Tuần 4

# KIẾN TRÚC BỘ LỆNH (Tiếp theo)





## Tuần 04 – Kiến trúc bộ lệnh (tiếp theo)

### Mục tiêu:


1. Hiểu cách biểu diễn và cách thực thi các lệnh trong máy tính
2. Chuyển đổi lệnh ngôn ngữ cấp cao sang assembly và mã máy
3. Chuyển đổi lệnh mã máy sang ngôn ngữ cấp cao hơn
4. Biết cách lập trình bằng ngôn ngữ assembly cho MIPS

Slide được dịch và các hình được lấy từ sách tham khảo:

***Computer Organization and Design: The Hardware/Software Interface***, Patterson, D. A., and J. L. Hennessy, Morgan Kaufman, Revised Fourth Edition, 2011.



## Tuần 4 – Kiến trúc bộ lệnh

1. **Giới thiệu** (kiến trúc bộ lệnh là gì, có những kiến trúc nào)
2. **Các phép tính** (các lệnh cơ bản trong kiến trúc tập lệnh MIPS)
3. **Toán hạng** (3 loại toán hạng: thanh ghi, bộ nhớ và hằng số)
4. **Số có dấu và không dấu** (biểu diễn số có dấu và không dấu trên máy tính)
5. **Biểu diễn lệnh** 
6. **Các phép tính Logic**
7. **Các lệnh điều kiện và nhảy**



## Tuần 4 – Kiến trúc bộ lệnh

**1. Giới thiệu**

**2. Các phép tính**

**3. Toán hạng**

**4. Số có dấu và không dấu**

**5. Biểu diễn lệnh**

**6. Các phép tính Logic**

**7. Các lệnh điều kiện và nhảy**



# Biểu diễn lệnh

❖ Làm thế nào một lệnh (~~add \$t0, \$s1, \$s2~~) lưu giữ được trong máy tính?

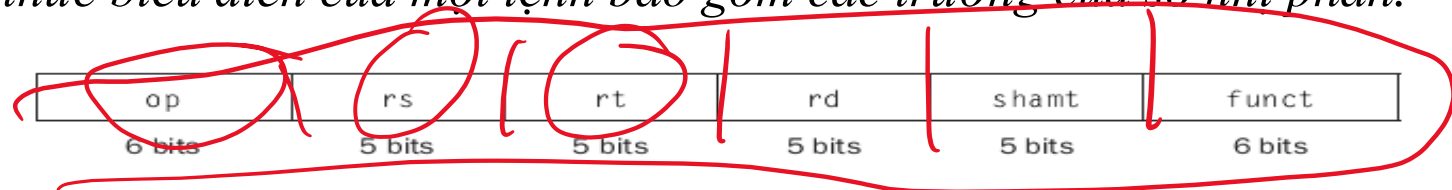
Máy tính chỉ có thể làm việc với các tín hiệu điện tử thấp và cao, do đó một lệnh lưu giữ trong máy tính phải được biểu diễn như là một chuỗi của "0" và "1", được gọi là mã máy/lệnh máy.

❖ **Ngôn ngữ máy (Machine language):** biểu diễn nhị phân được sử dụng để giao tiếp trong một hệ thống máy tính.

❖ Để chuyển đổi từ một lệnh sang mã máy (machine code) sử dụng định dạng lệnh (instruction format).

**Định dạng lệnh:** Một hình thức biểu diễn của một lệnh bao gồm các trường của số nhị phân.

Ví dụ một định dạng lệnh:

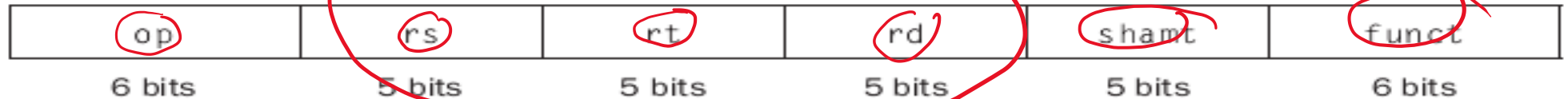




**Ví dụ:** Chuyển đổi một lệnh cộng trong MIPS thành một lệnh máy:

*add \$t0,\$s1,\$s2*

Với định dạng lệnh:



- Mỗi phần đoạn của một định dạng lệnh được gọi là một **trường** (ví dụ trường op, rs, rt, rd, shamt, funct).
- Trong ngôn ngữ assembly MIPS, thanh ghi **\$s0 đến \$s7** có chỉ số tương ứng từ 16 đến 23, và thanh ghi **\$t0 đến \$t7** có chỉ số tương ứng từ 8 đến 15.



# Biểu diễn lệnh

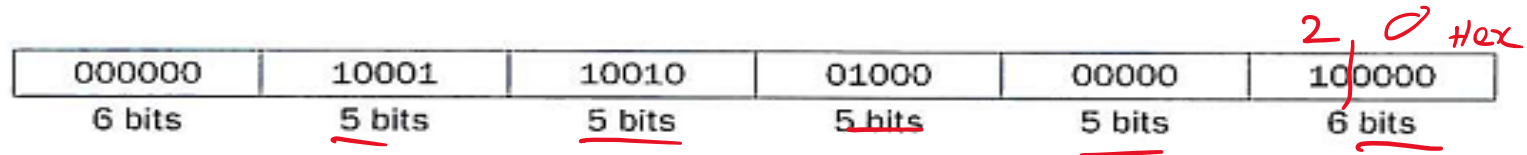
**Trả lời:** Chuyển đổi một lệnh công trong MIPS thành một lệnh máy:

*add* \$t0, \$s1, \$s2

Định dạng lệnh:



Mã máy:



- Các trường rs, rt, rd chứa chỉ số của các thanh ghi tương ứng; trường op và funct có giá trị bao nhiêu cho từng loại lệnh do MIPS quy định.
- Trường 'shamt'? — *sl / srl → số bit cần dịch*

**Tra trong bảng “MIPS reference data” (trang 2 sách tham khảo chính) để có các giá trị cần thiết**



# Biểu diễn lệnh

COMPUTER ENGINEERING

(columns 3 and 4) together

## MIPS Reference Data

①



### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR- MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R $R[rd] = R[rs] + R[rt]$	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I $R[rt] = R[rs] + \text{SignExtImm}$	(1, 2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I $R[rt] = R[rs] + \text{SignExtImm}$	(2) 9 <sub>hex</sub>
Add Unsigned	addu	R $R[rd] = R[rs] + R[rt]$	0 / 21 <sub>hex</sub>
And	and	R $R[rd] = R[rs] \& R[rt]$	0 / 24 <sub>hex</sub>





# Biểu diễn lệnh

Từ một mã máy đang có, như thế nào máy tính hiểu?

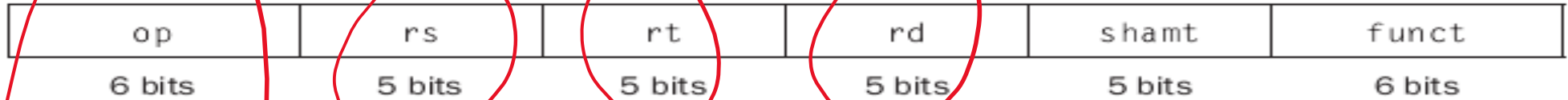
op	rs	rt	rd	shamt	funct
000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- Trường đầu tiên (**op**, tức **opcode** có giá trị 0) và trường cuối cùng (**funct**, tức **function** có giá trị 20<sub>hex</sub>) kết hợp báo cho máy tính biết rằng đây là **lệnh cộng (add)**.
- Trường thứ hai (**rs**) cho biết toán hạng thứ nhất của phép toán cộng (rs hiện có giá trị 17, tức toán hạng thứ nhất của phép cộng là thanh ghi \$s1).
- Trường thứ ba (**rt**) cho biết toán hạng thứ hai của phép toán cộng (rt hiện có giá trị 18, tức toán hạng thứ hai của phép cộng là thanh ghi \$s2).
- Trường thứ tư (**rd**) là thanh ghi đích chứa tổng của phép cộng (rd hiện có giá trị 8, tức thanh ghi đích chứa tổng là \$t0).
- Trường thứ năm (**shamt**) không sử dụng trong lệnh add này.



## Các dạng khác nhau của định dạng lệnh MIPS :

- **R-type hoặc R-format** (cho các lệnh chỉ làm việc với thanh ghi)



- **I-type hoặc I-format** (cho các lệnh có liên quan đến số tức thời và truyền dữ liệu)



- **J-type hoặc J-format** (lệnh nhảy, lệnh ra quyết định)





# Biểu diễn lệnh

## Các dạng khác nhau của định dạng lệnh MIPS :

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

op	address
6 bits	26 bits

**op** (Hay còn gọi là opcode, mã tác vụ): Trong cả ba định dạng của lệnh, trường op luôn chiếm 6 bits.

Khi máy tính nhận được mã máy, phân tích op sẽ cho máy tính biết được đây là lệnh gì (\*), từ đó cũng biết được mã máy thuộc loại định dạng nào, sau đó các trường tiếp theo sẽ được phân tích.

(\*) **Lưu ý:** MIPS quy định nhóm các lệnh làm việc với 3 thanh ghi (R-format) đều có op là 0. Vì vậy, với R-format, cần dùng thêm trường 'funct' để biết chính xác lệnh cần thực hiện là lệnh nào.



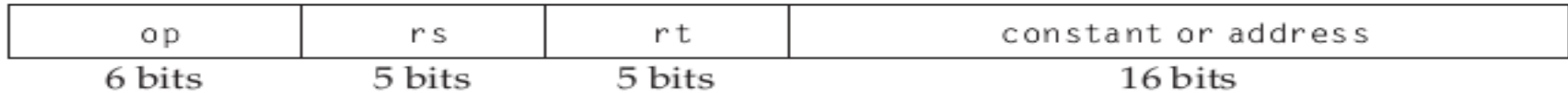
## Các trường của R-format:



- **rs:** Thanh ghi chứa toán hạng nguồn thứ nhất
- **rt:** Thanh ghi chứa toán hạng nguồn thứ hai
- **rd:** Thanh ghi toán hạng đích, nhận kết quả của các phép toán.
- **shamt:** Chỉ dùng trong các câu lệnh dịch bit (shift) - chứa số lượng bit cần dịch (không được sử dụng sẽ chứa 0)
- **funct:** Kết hợp với op (khi op bằng 0) để cho biết mã máy là lệnh gì

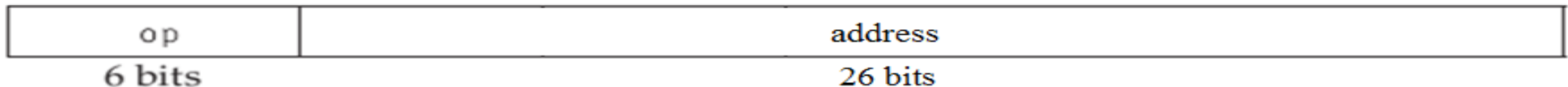


## Các trường của I-format và J-format:



Vùng “constant or address” (thỉnh thoảng gọi là vùng immediate) là vùng chứa số 16 bit.

- ✓ Với lệnh liên quan đến memory (như lw, sw): giá trị trong thanh ghi rs cộng với số 16 bits này sẽ là địa chỉ của vùng nhớ mà lệnh này truy cập đến.
- ✓ Với lệnh khác (như addi): 16 bits này chứa số tức thời



Vùng “address” là vùng chứa số 26 bit (dùng cho lệnh ‘j’)



# Biểu diễn lệnh

Ví dụ một số lệnh MIPS và các trường tương ứng

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	$32_{ten}$	n.a.
sub (subtract)	R	0	reg	reg	reg	0	$34_{ten}$	n.a.
add immediate	I	$8_{ten}$	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	$35_{ten}$	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	$43_{ten}$	reg	reg	n.a.	n.a.	n.a.	address

- “reg” nghĩa là chỉ số thanh ghi (giữa 0 và 31)
- “address” nghĩa là 1 địa chỉ 16 bit.
- “n.a.” (không áp dụng) nghĩa là trường này không xuất hiện trong định dạng này.
- Lưu ý rằng lệnh ‘add’ và ‘sub’ có cùng giá trị trong trường “op”; do đó phần cứng sẽ sử dụng thêm trường “funct” để quyết định đây là lệnh gì
  - $Funct = 32_{ten} = 20_{hex}$  (lệnh ‘add’)
  - $Funct = 34_{ten} = 22_{hex}$  (lệnh ‘sub’)



# Biểu diễn lệnh

## Ví dụ 1: Chuyển ngôn ngữ Assembly (ASM) MIPS sang mã máy

Chuyển câu lệnh assembly MIPS sau sang mã máy:

and \$t3, \$s0, \$s2

**Đáp án:** Thực hiện việc chuyển đổi theo các bước sau:

- Bước 1: Tra bảng "MIPS reference data" xem lệnh and thuộc định dạng nào ☐ R type

And

and

**R**

$R[rd] = R[rs] \& R[rt]$

0 / 24<sub>hex</sub>

6

5

5

5

5

6

op	rs	rt	rd	shamt	funct



## Ví dụ 1: Chuyển ngôn ngữ Assembly (ASM) MIPS sang mã máy

Chuyển câu lệnh assembly MIPS sau sang mã máy:

*and \$t3, \$s0, \$s2*

**Đáp án:** Thực hiện việc chuyển đổi theo các bước sau:

- Bước 1: Tra bảng “MIPS reference data” xem lệnh *and* thuộc định dạng nào ☐ R type
- Bước 2: Tra các trường opcode và function

And                      and                      R    R[rd] = R[rs] & R[rt]

6

~~0100100~~  
0/24<sub>hex</sub>

op	rs	rt	rd	shamt	funct
000000					100100





# Biểu diễn lệnh

COMPUTER ENGINEERING

## Ví dụ 1: Chuyển ngôn ngữ Assembly (ASM) MIPS sang mã máy

Chuyển câu lệnh assembly MIPS sau sang mã máy:

*and \$t3, \$s0, \$s2*

**Đáp án:** Thực hiện việc chuyển đổi theo các bước sau:

- Bước 1: Tra bảng “MIPS reference data” xem lệnh *and* thuộc định dạng nào
- Bước 2: Tra các trường opcode và function
- Bước 3: Tra vị trí và chỉ số các thanh ghi

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

$t_1 = 9$

$t_2 = 10$

$t_3 = 11$

01011

↓

10000

↓

$s_1 = 17$

$s_2 = 18 - 70010$

And

and

R R[rd] = R[rs] & R[rt]

0 / 24<sub>hex</sub>

op	rs	rt	rd	shamt	funct
000000	10000	10010	01011		100100



## Ví dụ 1: Chuyển ngôn ngữ Assembly (ASM) MIPS sang mã máy

Chuyển câu lệnh assembly MIPS sau sang mã máy:

and \$t3, \$s0, \$s2

**Đáp án:** Thực hiện việc chuyển đổi theo các bước sau:

- Bước 1: Tra bảng “MIPS reference data” xem lệnh **and** thuộc định dạng nào □ R type
- Bước 2: Tra các trường opcode và function
- Bước 3: Tra vị trí và chỉ số các thanh ghi
- Bước 4: Điền trường shamt và hoàn thành mã máy của lệnh

op	rs	rt	rd	shamt	funct
000000	10000	10010	01011	00000	100100



# Biểu diễn lệnh

## Ví dụ 2: Chuyển ngôn ngữ Assembly (ASM) MIPS sang mã máy

Chuyển câu lệnh assembly MIPS sau sang mã máy:

*sll \$t1, \$t5, 7*

**Đáp án:** Thực hiện việc chuyển đổi theo các bước sau:

- Bước 1: Tra bảng “MIPS reference data” xem lệnh *sll* thuộc định dạng nào ☐ R type

Shift Left Logical

*sll*



$R[rd] = R[rt] \ll \text{shamt}$

0 / 00<sub>hex</sub>

op	rs	rt	rd	shamt	funct



## Ví dụ 2: Chuyển ngôn ngữ Assembly (ASM) MIPS sang mã máy

Chuyển câu lệnh assembly MIPS sau sang mã máy:

*sll \$t1, \$t5, 7*

**Đáp án:** Thực hiện việc chuyển đổi theo các bước sau:

- Bước 1: Tra bảng “MIPS reference data” xem lệnh *sll* thuộc định dạng nào □ R type
- Bước 2: Tra các trường opcode và function

Shift Left Logical    *sll*    R     $R[\underline{rd}] = R[\underline{rt}] \ll \text{shamt}$

0 / 00<sub>hex</sub>

op	rs	rt	rd	shamt	funct
000000					000000



## Ví dụ 2: Chuyển ngôn ngữ Assembly (ASM) MIPS sang mã máy

Chuyển câu lệnh assembly MIPS sau sang mã máy:

`sll $t1, $t5, 7`

**Đáp án:** Thực hiện việc chuyển đổi theo các bước sau:

- Bước 1: Tra bảng “MIPS reference data” xem lệnh `sll` thuộc định dạng nào ☐ R type
- Bước 2: Tra các trường opcode và function
- Bước 3: Tra vị trí và chỉ số các thanh ghi

Shift Left Logical `sll` R

`R[rd] = R[rt] << shamt`

op	rs	rt	rd	shamt	funct
000000	00000	01101	01001		000000

$t_0 = 8$   
 $t_7 = 9 \rightarrow 01001$   
 $t_2 = 10$   
 $t_3 = 11$   
 $t_4 = 12$   
 $t_5 = 13$   
 $01101$   
 $0/00_{hex}$



## Ví dụ 2: Chuyển ngôn ngữ Assembly (ASM) MIPS sang mã máy

Chuyển câu lệnh assembly MIPS sau sang mã máy:

*sll \$t1, \$t5, 7*

**Đáp án:** Thực hiện việc chuyển đổi theo các bước sau:

- Bước 1: Tra bảng “MIPS reference data” xem lệnh *sll* thuộc định dạng nào □ R type
- Bước 2: Tra các trường opcode và function
- Bước 3: Tra vị trí và chỉ số các thanh ghi
- Bước 4: Điền trường shamt và hoàn thành mã máy của lệnh

op	rs	rt	rd	shamt	funct
000000	00000	01101	01001	00111	000000



## Ví dụ 2: Chuyển ngôn ngữ Assembly (ASM) MIPS sang mã máy

Chuyển câu lệnh assembly MIPS sau sang mã máy:

*sll \$t1, \$t5, 7*

**Đáp án:** Thực hiện việc chuyển đổi theo các bước sau:

- Bước 1: Tra bảng “MIPS reference data” xem lệnh *sll* thuộc định dạng nào □ R type
- Bước 2: Tra các trường opcode và function
- Bước 3: Tra vị trí và chỉ số các thanh ghi
- Bước 4: Điền trường shamt và hoàn thành mã máy của lệnh

op	rs	rt	rd	shamt	funct
000000	00000	01101	01001	00111	000000



# Biểu diễn lệnh

**Ví dụ: Chuyển ngôn ngữ cấp cao □ Assembly MIPS □ mã máy**

Chuyển câu lệnh sau sang assembly MIPS và sau đó chuyển thành mã máy:

$$A[300] = h + A[300]$$

Biết  $A$  là một mảng nguyên, mỗi phần tử của  $A$  cần một từ nhớ để lưu trữ;  $\$t1$  chứa địa chỉ nền/cơ sở của mảng  $A$  và  $\$s2$  tương ứng với biến nguyên  $h$ .

**Đáp án: Assembly MIPS:**

*lw \$t0,1200(\$t1) # Dùng thanh ghi tạm \$t0 nhận A[300]*

*add \$t0,\$s2,\$t0 # Dùng thanh ghi tạm \$t0 nhận  $h + A[300]$*

*sw \$t0,1200(\$t1) # Lưu  $h + A[300]$  trở lại vào A[300]*

Load Word	lw	I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2)	$23_{\text{hex}}$
Add	add	R	$R[rd] = R[rs] + R[rt]$	(1)	$0 / 20_{\text{hex}}$
Store Word	sw	I	$M[R[rs] + \text{SignExtImm}] = R[rt]$	(2)	$2b_{\text{hex}}$

Mã máy cho ba lệnh trên:

100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		





## Kết luận:

1. Các lệnh được biểu diễn như là các con số.
  2. Chương trình được lưu trữ trong bộ nhớ được đọc hay viết giống như các con số.
- Xem lệnh như là dữ liệu là cách tốt nhất để đơn giản hóa cả bộ nhớ và phần mềm của máy tính.
  - Để chạy/thực thi một chương trình, đơn giản chỉ cần nạp chương trình và dữ liệu vào bộ nhớ; sau đó báo với máy tính để bắt đầu thực thi chương trình tại vị trí mà nó đã được cấp phát.



## Ví dụ 3: Chuyển mã máy sang ngôn ngữ Assembly (ASM) MIPS

Chuyển mã máy sau sang câu lệnh assembly MIPS:

***0x01304024***

**Đáp án:** Thực hiện việc chuyển đổi theo các bước sau:

- Bước 1: Chuyển mã máy sang hệ nhị phân

0x	0	1	3	0	4	0	2	4
	0000	0001	0011	0000	0100	0000	0010	0100



## Ví dụ 3: Chuyển mã máy sang ngôn ngữ Assembly (ASM) MIPS

Chuyển mã máy sau sang câu lệnh assembly MIPS:

**0x01304024**

**Đáp án:** Thực hiện việc chuyển đổi theo các bước sau:

- Bước 1: Chuyển mã máy sang hệ nhị phân
- Bước 2: Chọn 6 bit đầu và tra opcode, nếu opcode bằng 000000 thì tra tiếp function 6 bit cuối trong bảng MIPS

### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add R	$R[rd] = R[rs] + R[rt]$	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu I	$R[rt] = R[rs] + \text{SignExtImm}$	(2) 9 <sub>hex</sub>
Add Unsigned	addu R	$R[rd] = R[rs] + R[rt]$	0 / 21 <sub>hex</sub>
And	and <b>R</b>	$R[rd] = R[rs] \& R[rt]$	<b>0 / 24<sub>hex</sub></b>

0x	0	1	3	0	4	0	2	4
	0000	0001	0011	0000	0100	0000	0010	0100

op	func
000000	01 0011 0000 0100 0000 00 100100

Từ đó suy ra lệnh “and” và thuộc format “R”



# Biểu diễn lệnh

## Ví dụ 3: Chuyển mã máy sang ngôn ngữ Assembly (ASM) MIPS

Chuyển mã máy sau sang câu lệnh assembly MIPS:

**0x01304024**

**Đáp án:** Thực hiện việc chuyển đổi theo các bước sau:

- Bước 2: Chọn 6 bit đầu và tra opcode, nếu opcode bằng 000000 thì tra tiếp function 6 bit cuối trong bảng MIPS
- Bước 3: Tìm các thanh ghi tương ứng với các trường trong format R

### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add R	$R[rd] = R[rs] + R[rt]$	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu I	$R[rt] = R[rs] + \text{SignExtImm}$	(2) 9 <sub>hex</sub>
Add Unsigned	addu R	$R[rd] = R[rs] + R[rt]$	0 / 21 <sub>hex</sub>
And	and R	$R[rd] = R[rs] \& R[rt]$	0 / 24 <sub>hex</sub>

op					funct
000000	01 0011 0000 0100 0000 00				100100

op	rs	rt	rd	shamt	funct
000000	01001	10000	01000	00000	100100
	9 $\Rightarrow$ t1	16 $\Rightarrow$ s0	8 $\Rightarrow$ t0		



## Ví dụ 3: Chuyển mã máy sang ngôn ngữ Assembly (ASM) MIPS

Chuyển mã máy sau sang câu lệnh assembly MIPS:

**0x01304024**

**Đáp án:** Thực hiện việc chuyển đổi theo các bước sau:

- Bước 2: Chọn 6 bit đầu và tra opcode, nếu opcode bằng 000000 thì tra tiếp function 6 bit cuối trong bảng MIPS
- Bước 3: Tìm các thanh ghi tương ứng với các trường trong format R
- Bước 4: Hoàn thành lệnh: *and \$t0, \$t1, \$s0*

### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add R	$R[rd] = R[rs] + R[rt]$	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu I	$R[rt] = R[rs] + \text{SignExtImm}$	(2) 9 <sub>hex</sub>
Add Unsigned	addu R	$R[rd] = R[rs] + R[rt]$	0 / 21 <sub>hex</sub>
And	and R	$R[rd] = R[rs] \& R[rt]$	0 / 24 <sub>hex</sub>

op	rs	rt	rd	shamt	funct
000000	01001	10000	01000	00000	100100
	9 $\Rightarrow$ t1	16 $\Rightarrow$ s0	8 $\Rightarrow$ t0		



**1. Giới thiệu**

**2. Các phép tính**

**3. Toán hạng**

**4. Số có dấu và không dấu**

**5. Biểu diễn lệnh**

**6. Các phép tính Logic**

**7. Các lệnh điều kiện và nhảy**



# Các phép tính Logic

Logical operations	C operators	Java operators	MIPS instructions
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR			or, ori

Hình 7: C và Java các phép tính logic và lệnh MIPS tương ứng.

- **Shift:** Lệnh dịch chuyển bit.
- **AND:** là phép toán logic “*VÀ*”.
- **OR:** là một phép toán logic “*HOẶC*”
- **NOR:** NOT OR.
- Hằng số rất hữu ích trong các phép toán logic AND và OR cũng như trong phép tính số học, vì vậy MIPS cung cấp các lệnh trực tiếp **andi** và **ori**.



## Ví dụ: Lệnh Logic (1)

### Ví dụ về lệnh shift:

Giả sử thanh ghi s1 chứa giá trị nhị phân 8 bit 00000110 (6)

Sau khi thực hiện câu lệnh sll \$t0, \$s1, 1 thì giá trị thanh ghi t0 = 00001100 (12)

Sau khi thực hiện câu lệnh sll \$t1, \$s1, 2 thì giá trị thanh ghi t1 = 00011000 (24)

Sau khi thực hiện câu lệnh srl \$t2, \$s1, 1 thì giá trị thanh ghi t2 = 00000011 (3)





## Ví dụ: Lệnh Logic (2)

### Ví dụ về lệnh or:

Cho giá trị các thanh ghi như sau:

$\$t0 = 0x55555555$

$\$t1 = 0x12345678$

Hãy cho biết giá trị của thanh ghi  $\$t2$  sau khi chạy các lệnh sau:

*sll \$t2, \$t0, 4*

*or \$t2, \$t2, \$t1*



## Ví dụ: Lệnh Logic (2)

### Ví dụ về lệnh or:

Cho giá trị các thanh ghi như sau:

$\$t0 = 0x55555555 = 01010101010101010101010101010101$

$\$t1 = 0x12345678 = 00010010001101000101011001111000$

Hãy cho biết giá trị của thanh ghi  $\$t2$  sau khi chạy các lệnh sau:

$sll \$t2, \$t0, 4 \Rightarrow \$t2 = 010101010101010101010101010101010000$

$or \$t2, \$t2, \$t1$  với  $\$t1 = 00010010001101000101011001111000$   
 $\Rightarrow \$t2 = 01010111011101010101011101111000$



## Ví dụ: Lệnh Logic (3)

### Ví dụ về lệnh andi:

Cho giá trị các thanh ghi như sau:

$\$t0 = 0x55555555$

Hãy cho biết giá trị của thanh ghi  $\$t2$  sau khi chạy các lệnh sau:

*srl \$t2, \$t0, 3*

*andi \$t2, \$t2, 0xFFEF*



## Ví dụ: Lệnh Logic (3)

### Ví dụ về lệnh andi:

Cho giá trị các thanh ghi như sau:

$\$t0 = 0x55555555 = 01010101010101010101010101010101$

Hãy cho biết giá trị của thanh ghi  $\$t2$  sau khi chạy các lệnh sau:

$srl \$t2, \$t0, 3 \Rightarrow \$t2 = 00001010101010101010101010101010$

$andi \$t2, \$t2, 0xFFEF = 11111111111111111111111111110111$

$\Rightarrow \$t2 = 00001010101010101010101010101010$



**1. Giới thiệu**

**2. Các phép tính**

**3. Toán hạng**

**4. Số có dấu và không dấu**

**5. Biểu diễn lệnh**

**6. Các phép tính Logic**

**7. Các lệnh điều kiện và nhảy**



# Các lệnh điều kiện và nhảy

- ❖ Một máy tính (PC) khác với các máy tính tay (calculator) chính là dựa trên khả năng đưa ra quyết định.
- ❖ Trong ngôn ngữ lập trình, đưa ra quyết định thường được biểu diễn bằng cách sử dụng câu lệnh “if”, đôi khi kết hợp với câu lệnh “go to”.
- ❖ Ngôn ngữ Assembly MIPS cũng chứa các lệnh hỗ trợ ra quyết định, tương tự với câu lệnh “if” và “go to”.

Ví dụ: ***beq register1, register2, L1***

Lệnh này có nghĩa là đi đến câu lệnh có nhãn *L1* nếu giá trị của thanh ghi *register1* bằng giá trị thanh ghi *register2*.

Từ ‘*beq*’ là viết tắt của “branch if equal” ( rẽ nhánh nếu bằng)

➔ Các lệnh như ‘*beq*’ được gọi là **lệnh rẽ nhánh có điều kiện**.



# Các lệnh điều kiện và nhảy

## Các lệnh rẽ nhánh có điều kiện (conditional branch) của MIPS:

Conditional branch	branch on equal	<code>beq \$s1,\$s2,25</code>	if ( $\$s1 == \$s2$ ) go to $PC + 4 + 100$
	branch on not equal	<code>bne \$s1,\$s2,25</code>	if ( $\$s1 \neq \$s2$ ) go to $PC + 4 + 100$
	set on less than	<code>slt \$s1,\$s2,\$s3</code>	if ( $\$s2 < \$s3$ ) $\$s1 = 1$ ; else $\$s1 = 0$
	set on less than unsigned	<code>sltu \$s1,\$s2,\$s3</code>	if ( $\$s2 < \$s3$ ) $\$s1 = 1$ ; else $\$s1 = 0$
	set less than immediate	<code>slti \$s1,\$s2,20</code>	if ( $\$s2 < 20$ ) $\$s1 = 1$ ; else $\$s1 = 0$
	set less than immediate unsigned	<code>sltiu \$s1,\$s2,20</code>	if ( $\$s2 < 20$ ) $\$s1 = 1$ ; else $\$s1 = 0$



## Các lệnh điều kiện và nhảy - Điều kiện $\geq$

Nếu ta muốn so sánh *if* ( $s0 \geq s1$ ) *goto* *ABC* thì ta dùng cặp *slt* và *beq*

*slt* \$t0, \$s0, \$s1

*#* \$t0 = 1 *if* \$s0 < \$s1

*beq* \$t0, \$zero, ABC

*# if* \$s0  $\geq$  \$s1, *goto* ABC

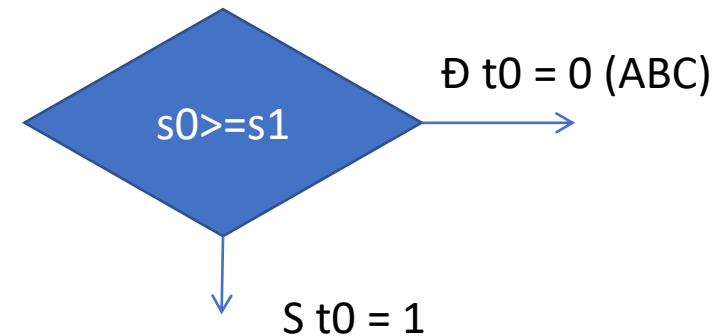
<stuff>

*# do if* \$s0 < \$s1

*j* Exit

ABC: ....

Exit:







## Các lệnh điều kiện và nhảy - Điều kiện <

Nếu ta muốn so sánh *if* ( $s0 < s1$ ) *goto* *ABC* thì ta dùng cặp *slt* và *bne*

```
slt $t0, $s0, $s1
```

*# \$t0 = 1 if \$s0 < \$s1*

```
bne $t0, $zero, ABC
```

*# if \$s0 < \$s1, goto ABC*

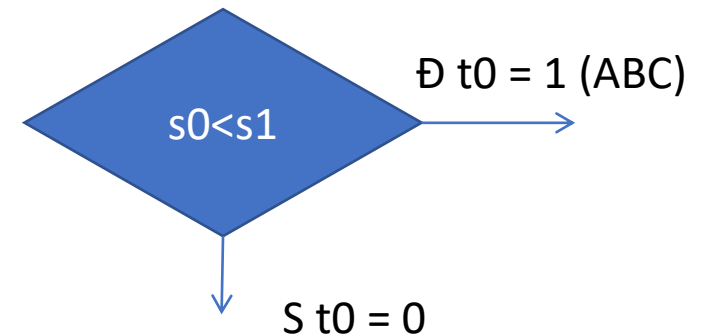
```
<stuff>
```

*# do if \$s0 >= \$s1*

```
j Exit
```

ABC: ....

Exit:





## Các lệnh điều kiện và nhảy

Ngoài ra còn có các lệnh rẽ nhánh có điều kiện khác, nhưng là nhóm **lệnh giả (pseudo instructions)**

Conditional branch (pseudo instruction)	branch on less than	blt
	branch greater than	bgt
	branch less than or equal	ble
	branch greater than or equal	bge

*(Tham khảo trang số 2, sách tham khảo chính)*



# Các lệnh điều kiện và nhảy

Các lệnh rẽ nhánh không điều kiện (unconditional branch) của MIPS:

Unconditional jump	jump	j label
	jump register	jr \$ra
	jump and link	jal label



# Các lệnh điều kiện và nhảy

Biên dịch *if-then-else* từ ngôn ngữ cấp cao sang assembly MIPS:

Cho đoạn mã sau:

$$\text{if } (i == j) f = g + h; \text{ else } f = g - h;$$

Biết  $f$ ,  $g$ ,  $h$ ,  $i$  và  $j$  là các biến. Nếu năm biến  $f$  đến  $j$  tương ứng với 5 thanh ghi \$s0 đến \$s4, mã MIPS cho câu lệnh *if* này là gì?

Trả lời:

<i>bne</i> \$s3,\$s4, <i>Else</i>	# go to Else if $i \neq j$
<i>add</i> \$s0, \$s1, \$s2	# $f = g + h$ (skipped if $i \neq j$ )
<i>j exit</i>	# go to Exit
<i>Else: sub</i> \$s0, \$s1, \$s2	# $f = g - h$ (skipped if $i = j$ )
<i>exit:</i>	



# Các lệnh điều kiện và nhảy

Biên dịch 1 vòng lặp **while** từ ngôn ngữ cấp cao sang assembly MIPS. Cho đoạn mã sau:

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

Giả định rằng  $i$  và  $k$  tương ứng với thanh ghi  $\$s3$  và  $\$s5$ ; và địa chỉ nền/cơ sở của mảng **save** lưu trong  $\$s6$ . Mã assembly MIPS tương ứng với đoạn mã C trên là gì?

## Trả lời:

```
Loop: sll $t1,$s3,2           # Temp reg $t1 = 4 * i
add $t1,$t1,$s6               # $t1 = address of save[i]
lw $t0,0($t1)                 # Temp reg $t0 = save[i]
bne $t0,$s5, Exit             # go to Exit if save[i] != k
addi $s3,$s3,1                # i = i + 1
j Loop                        # go to Loop
Exit:
```



## Các lệnh điều kiện và nhảy - Ví dụ

Biên dịch đoạn lệnh *if* từ ngôn ngữ cấp cao sang assembly MIPS. Cho đoạn mã sau:

*if* ( $i > j$ )

$\{ A[i] = A[3] + 1; \}$

*else*

$\{ A[i+1] = 10; \}$

$i++;$

Biết  $i$  và  $j$  tương ứng với các thanh ghi \$s0 và \$s1. Mảng A là mảng mà các phần tử là số nguyên, mỗi phần tử chiếm 1 từ nhớ (4 bytes) và địa chỉ nền của mảng A lưu trong thanh ghi \$s3



# Các lệnh điều kiện và nhảy - Ví dụ

Biên dịch đoạn lệnh *if* từ ngôn ngữ cấp cao sang assembly MIPS. Cho đoạn mã sau:

<i>if</i> ( $i > j$ ) $\Rightarrow$ <i>if</i> ( $j < i$ )	$\longrightarrow$	<i>slt</i> \$t0, \$s1, \$s0
{ $A[i] = A[3] + 1$ ;		<i>bne</i> \$t0, \$zero, <i>if</i> hoặc <i>beq</i> \$t0, \$zero, <i>else</i>
<i>else</i>		<i>if</i> : <i>lw</i> \$t1, 12 (\$s3)
{ $A[i+1] = 10$ ;	<i>else</i> :	<i>addi</i> \$t1, \$t1, 1
$i++$ ;		<i>sll</i> \$t2, \$s0, 2
$\downarrow$		<i>add</i> \$t2, \$t2, \$s3
<i>Exit</i> : <i>addi</i> \$s0 \$s0, 1		<i>sw</i> \$t1, 0 (\$t2)
		<i>j</i> <i>Exit</i>
		<i>j</i> <i>Exit</i>

Biết  $i$  và  $j$  tương ứng với các thanh ghi \$s0 và \$s1. Mảng A là mảng mà các phần tử là số nguyên, mỗi phần tử chiếm 1 từ nhớ (4 bytes) và địa chỉ nền của mảng A lưu trong thanh ghi \$s3



## Các lệnh điều kiện và nhảy - Ví dụ

Chương trình sau khi ghép lại với nhau:

*slt \$t0, \$s1, \$s0*

*bne \$t0, \$zero, if*

*else: addi \$t3, \$zero, 10*

*addi \$t4, \$s0, 1*

*sll \$t4, \$t4, 2*

*add \$t4, \$t4, \$s3*

*sw \$t3, 0 (\$t4)*

*j Exit*

*if: lw \$t1, 12 (\$s3)*

*addi \$t1, \$t1, 1*

*sll \$t2, \$s0, 2*

*add \$t2, \$t2, \$s3*

*sw \$t1, 0 (\$t2)*

*j Exit*

*Exit: addi \$s0 \$s0, 1*





# Các lệnh điều kiện và nhảy

Biên dịch 1 vòng lặp **for** từ ngôn ngữ cấp cao sang assembly MIPS. Cho đoạn mã sau:

```
j = value;  
for(i = 1; i < j; i++)  
    A[i] = B[i];  
j = 0;
```

Biết i và j tương ứng với các thanh ghi \$s0 và \$s1. Mảng A là mảng mà các phần tử là số nguyên, mỗi phần tử chiếm 1 từ nhớ (4 bytes) và địa chỉ nền của mảng A lưu trong thanh ghi \$s3. Với địa chỉ nền mảng B đang lưu trong thanh ghi \$s4 và biến value tương ứng thanh ghi \$s5

**Trả lời:**



**1. Giới thiệu**

**2. Các phép tính**

**3. Toán hạng**

**4. Số có dấu và không dấu**

**5. Biểu diễn lệnh**

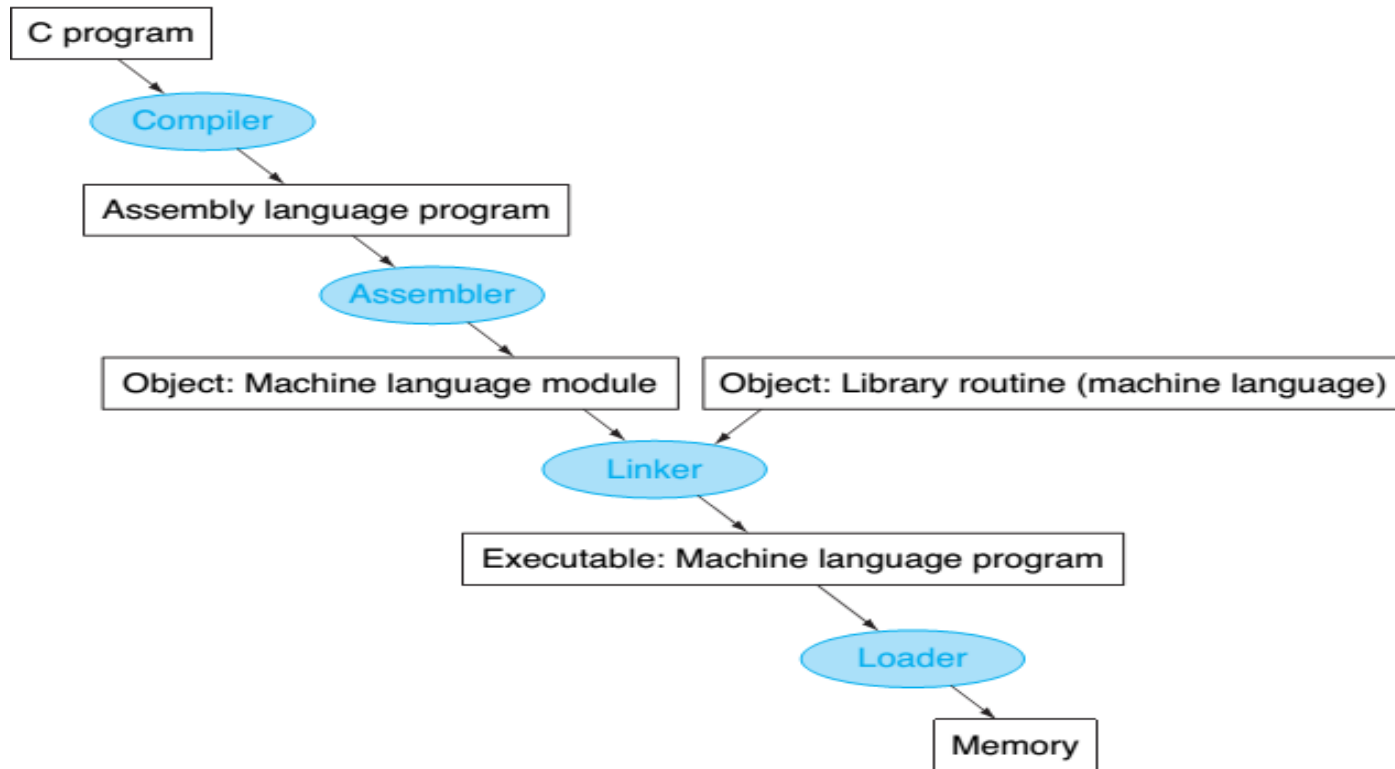
**6. Các phép tính Logic**

**7. Các lệnh điều kiện và nhảy**



# Chuyển đổi và bắt đầu một chương trình

Bốn bước trong việc chuyển đổi một chương trình C trong một tập tin trên đĩa vào một chương trình đang chạy trên máy tính.





### Tổng kết:

- MIPS có ba định dạng lệnh: R-format, I-format, J-format. Từ đó, hiểu cách một lệnh từ ngôn ngữ cấp cao chuyển thành assembly của MIPS, và từ assembly của MIPS chuyển thành mã máy dựa theo ba định dạng trên
- Biết quy tắc hoạt động của nhóm lệnh logic của MIPS
- Biết quy tắc hoạt động của nhóm lệnh nhảy (nhảy có điều kiện và không điều kiện) của MIPS



## Tuần 4 – Kiến trúc bộ lệnh

### ❖ Lý thuyết: Đọc sách tham khảo

- Mục: 2.5, 2.6, 2.7
- Sách: *Computer Organization and Design: The Hardware/Software Interface*, Patterson, D. A., and J. L. Hennessy, Morgan Kaufman, Revised Fourth Edition, 2011.

### ❖ Bài tập: file đính kèm