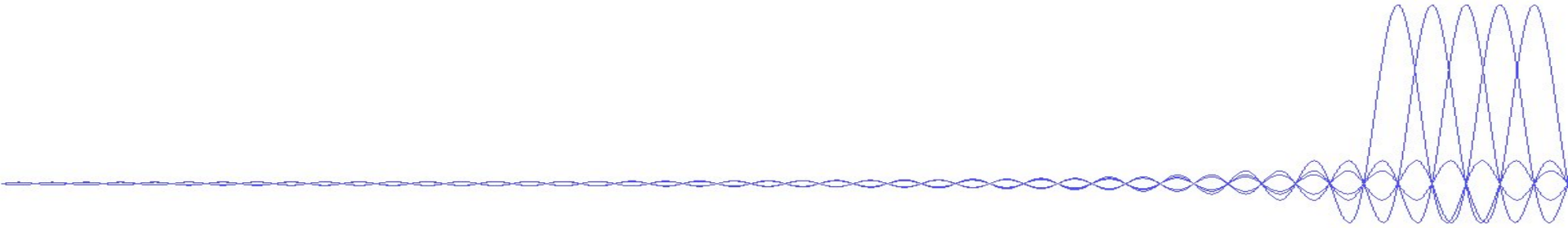
COMPUTER ENGINEERING

KIẾN TRÚC MÁY TÍNH

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Tuần 4

KIẾN TRÚC BỘ LỆNH
(Tiếp theo)



COMPUTER ENGINEERING

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.

COMPUTER ENGINEERING

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?

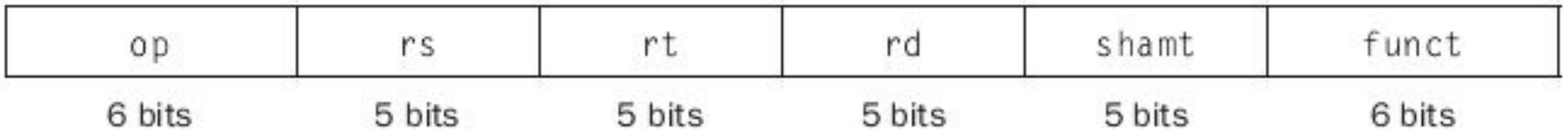
Máy tính chỉ có thể làm việc với các tín hiệu điện tử thấp và cao, do đó một lệnh lưu giữ trong máy tính phải được biểu diễn như là một chuỗi của "0" và "1", được gọi là **mã máy/lệnh máy**.

- **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:

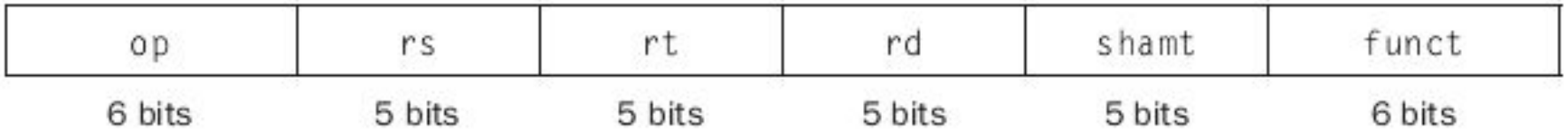


Biểu diễn lệnh

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

add *$t0,$s1,$s2*

Với định dạng lệnh:

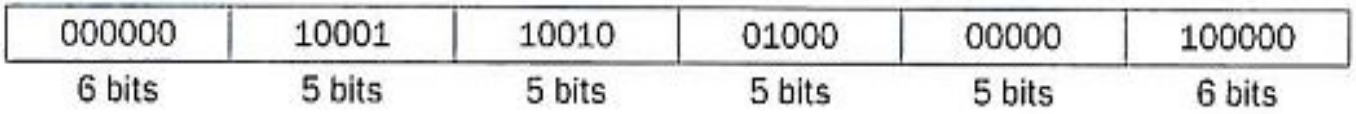


Biểu diễn lệnh

- **Trả lời:** Chuyển đổi một lệnh cộng trong MIPS thành một lệnh máy:
  add *$t0, $s1, $s2*

Định dạng lệnh:

Mã máy:



- Mỗi phân 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.
- 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'?

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

COMPUTER ENGINEERING

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

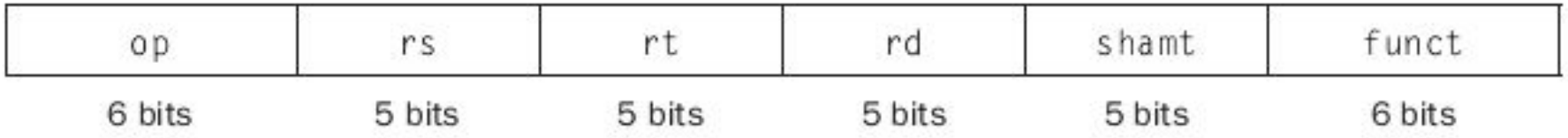
- **Trường đầu tiên (op, tức opcode có giá trị 0) và trường cuối cùng (funct, tức function có giá trị 20_{hex})** kết hợp báo cho máy tính biết rằng đây là **lệnh cộng (add)**.
- **Trường thứ hai (rs)** cho biết toán hạng thứ nhất của phép toán cộng (rs hiện có giá trị 17, tức toán hạng thứ nhất của phép công 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 công 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

COMPUTER ENGINEERING

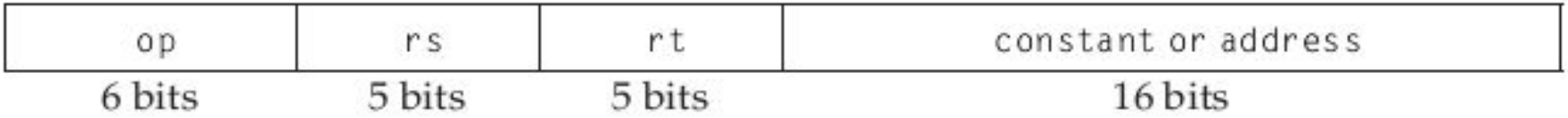
Biểu diễn lệnh

Các dạng khác nhau của định dạng lệnh MIPS :

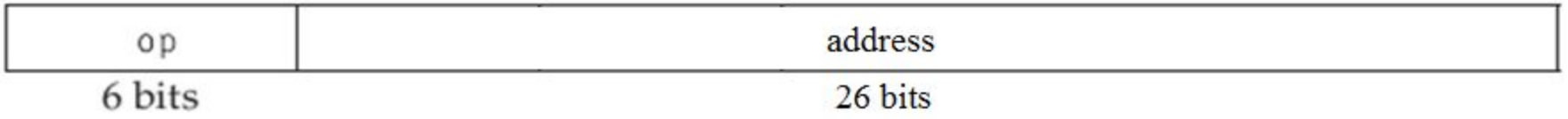
- **R-type hoặc R-format (cho các lệnh chỉ làm việc với thanh ghi)**



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

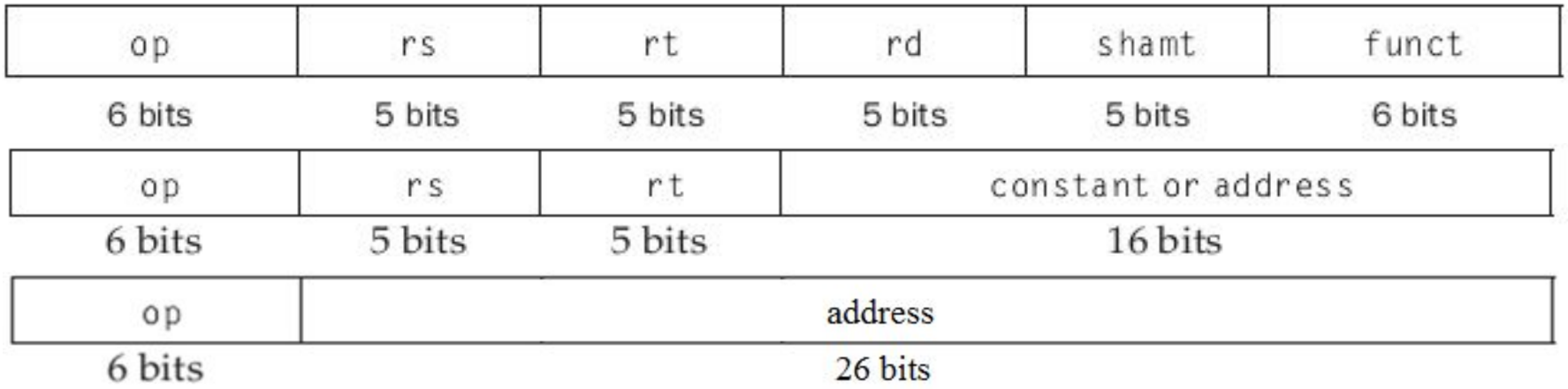


- **J-type hoặc J-format (lệnh nhảy, lệnh ra quyết định)**



Biểu diễn lệnh

Các dạng khác nhau của định dạng lệnh MIPS :



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

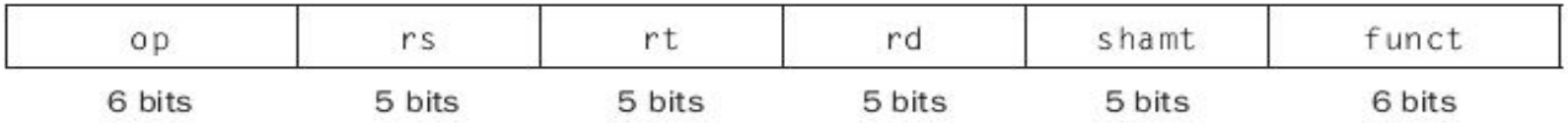
Khi máy tính nhận được mã máy, phân tích op sẽ cho máy tính biết được đây là lệnh gì (*), từ đó cũng biết được mã máy thuộc loại định dạng nào, sau đó các trường tiếp theo sẽ được phân tích.

(*)***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.*

COMPUTER ENGINEERING

Biểu diễn lệnh

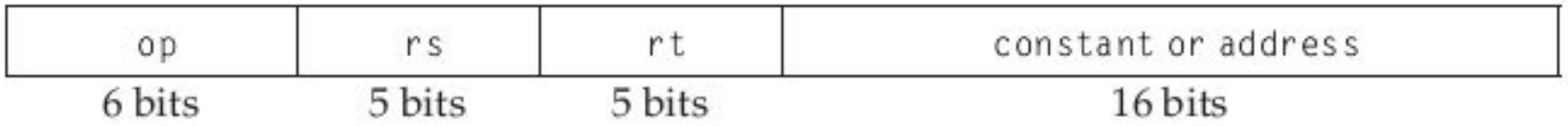
Các trường của R-format:



- **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

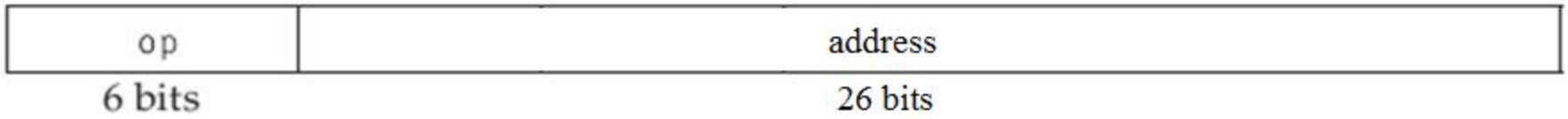
Biểu diễn lệnh

Các trường của I-format và J-format:



Vùng "constant or address" (thỉnh thoảng gọi là vùng immediate) là vùng chứa số16 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')

COMPUTER ENGINEERING

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: 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]*

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		

Biểu diễn lệnh

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.

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

COMPUTER ENGINEERING

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
Bit-by-bit NOT	~	~	nor

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"*
- **NOT:** kết quả là 1 nếu bit đó là 0 và ngược lại.
- **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**.

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**

COMPUTER ENGINEERING

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	beq $s1, $s2 25	if ($s1 == $s2) goto PC + 4 + 100
	branch on not equal	bne $s1, $s2, 25	if ($s1 != $s2) goto PC + 4 + 100
	set on less than	slt $s1, $s2, $s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0
	set on less than unsigned	slt $s1, $s2, $s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0
	set on less than immediate	slt $s1, $s2, 20	if ($s2 < 20) $s1 = 1; else $s1 = 0
	set on less than immediate unsigned	slt $s1, $s2, 20	if ($s2 < 20) $s1 = 1; else $s1 = 0

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

Cặp (slt □ beq) tương đương với if(… ≥ …) goto…

Cặp (slt □ bne) tương đương với if(… < …) goto…

```
slt $t0,$s0,$s1          # $t0 = 1 if $s0 <$s1
beq $t0,$zero,skip           # if $s0 >= $s1, goto skip
<stuff>                  # do if $s0 < $s1
```

```
slt $t0,$s0,$s1          # $t0 = 1 if $s0 <$s1
bne $t0,$zero,skip           # if $s0 < $s1, goto skip
<stuff>                  # do if $s0 >= $s1
```

COMPUTER ENGINEERING

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

COMPUTER ENGINEERING

Các lệnh điều kiện và nhảy

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

Cho đoạn mã sau:

if (i == j) f = g + h; 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:

bne *$s3,$s4,Else* *# go to Else if i != j*

add *$s0, $s1, $s2* *# f = g + h (skipped if i != j)*

j exit *# go to Exit*

Else: ***sub*** *$s0, $s1, $s2* *# f = g – h (skipped if i = j)*

exit:

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:
```

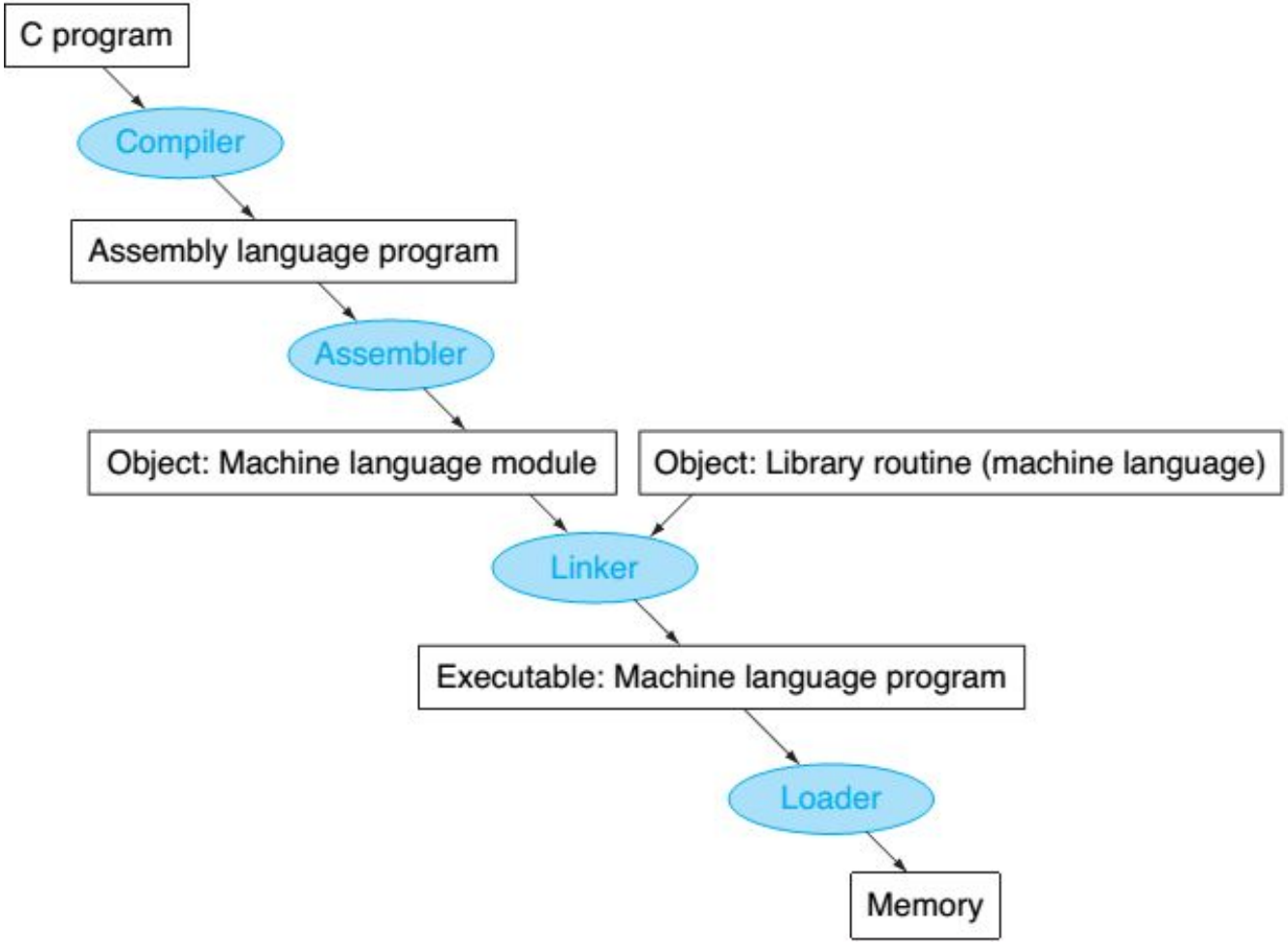
Tuần 4 – Kiến trúc bộ lệnh

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**

COMPUTER ENGINEERING

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