UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Chương 02 – Kiến trúc bộ lệnh

Mục tiêu chương:

1. Hiểu cách biểu diễn lệnh trong máy tính, cách các lệnh thực thi
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

Chương 02 – 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
8. Các thủ tục hỗ trợ trong phần cứng máy tính
9. Chuyển đổi và bắt đầu một chương trình

Giới thiệu

❖ Để ra lệnh cho máy tính ta phải nói với máy tính bằng ngôn ngữ của máy tính. Các từ của ngôn ngữ máy tính gọi là các lệnh ***(instructions)*** và tập hợp tất cả các từ gọi là bộ lệnh ***(instruction set)***

❖ Bộ lệnh trong chương này là MIPS, một bộ lệnh của kiến trúc máy tính được thiết kế từ năm 1980. Cùng với 2 bộ lệnh thông dụng nhất ngày nay:

- ARM rất giống MIPS
- The Intel x86,

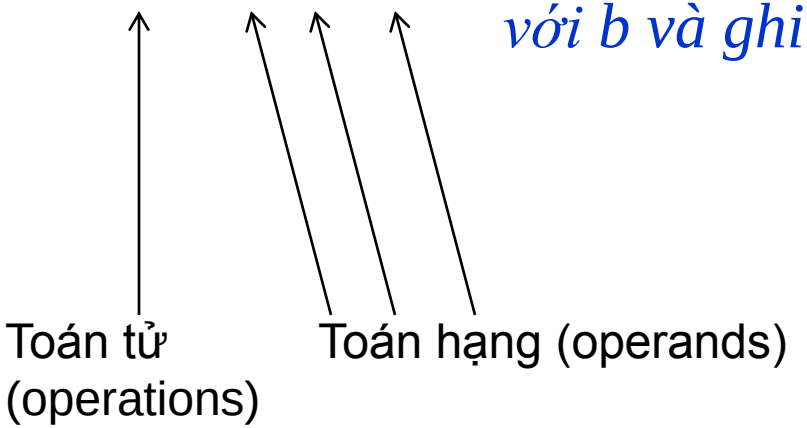
Chương 02 – Kiến trúc bộ lệnh

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
8. Các thủ tục hỗ trợ trong phần cứng máy tính
9. Chuyển đổi và bắt đầu một chương trình

Phép tính (Operations)

Ví dụ:

add *a, b, c* ➔ *Chỉ dẫn cho máy tính thực hiện cộng 2 biến a với b và ghi kết quả vào biến c, c= a + b.*



Ví dụ một số lệnh trên MIPS

Category	Instruction	Example	Meaning
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]
	store half	sh $s1,20($s2)	Memory[$s2 + 20] = $s1
	load byte	lb $s1,20($s2)	$s1 = Memory[$s2 + 20]
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]
	store byte	sb $s1,20($s2)	Memory[$s2 + 20] = $s1
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]
	store condition. word	sc $s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10

Phép tính (Operations)

Ví dụ 1.

C/Java

```
a = b + c;
d = a – e;
```

MIPS

```
add a, b, c
sub d, a, e
```

Ví dụ 2.

C/Java

```
f = (g + h) – (i + j);
```

MIPS

```
add t0, g, h
add t1, i, j
sub f, t0, t1
```

Chương 02 – 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
8. Các thủ tục hỗ trợ trong phần cứng máy tính
9. Chuyển đổi và bắt đầu một chương trình

Toán hạng

Có 3 loại toán hạng:

1. Toán hạng thanh ghi (Register Operands)
2. Toán hạng bộ nhớ (Memory Operands)
3. Toán hạng hằng (Constant or Immediate Operands)

Toán hạng

Toán hạng thanh ghi:

❖ Không giống như các chương trình trong ngôn ngữ cấp cao, các toán hạng của các lệnh số học bị hạn chế, chúng phải đặt trong các vị trí đặc biệt được xây dựng trực tiếp trong phần cứng được gọi là **thanh ghi (số lượng thanh ghi có giới hạn: MIPS-32, ARM Cortex A8-40)**.

❖ Kích thước của một thanh ghi trong kiến trúc MIPS là 32 bit; nhóm 32 bit xuất hiện thường xuyên nên chúng được đặt tên là "từ" (**word**) trong kiến trúc MIPS.
(lưu ý: một "từ" trong kiến trúc bộ lệnh khác có thể không có 32 bit)

❖ Một sự khác biệt lớn giữa các biến của một ngôn ngữ lập trình và các biến thanh ghi là số giới hạn thanh ghi, thường là 32 trên các máy tính hiện nay.

Toán hạng

Các thanh ghi trong MIPS:

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

Toán hạng

Toán hạng bộ nhớ (1):

❖ Bộ vi xử lý chỉ có thể giữ một lượng nhỏ dữ liệu trong các thanh ghi, trong khi bộ nhớ máy tính chứa hàng triệu dữ liệu.

❖ Với lệnh MIPS, phép tính số học chỉ xảy ra trên thanh ghi, do đó, MIPS phải có các lệnh chuyển dữ liệu giữa bộ nhớ và thanh ghi. Lệnh như vậy được gọi là **lệnh chuyển dữ liệu**.

Lệnh chuyển dữ liệu: *Một lệnh di chuyển dữ liệu giữa bộ nhớ và thanh ghi*

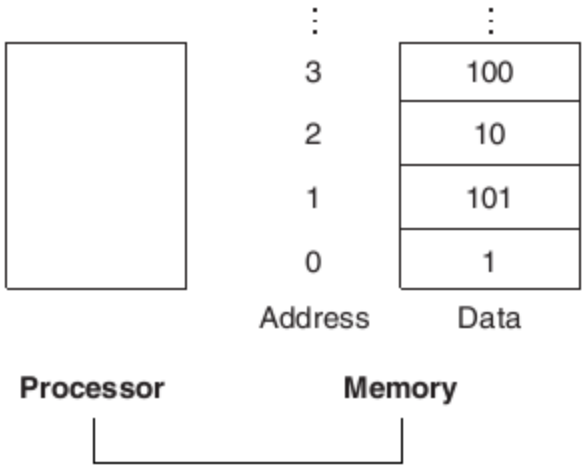
❖ Để truy cập vào một từ trong bộ nhớ, lệnh phải cung cấp **địa chỉ** bộ nhớ.

Địa chỉ*: Một giá trị sử dụng để phân định vị trí của một phần tử dữ liệu cụ thể trong một mảng bộ nhớ.*

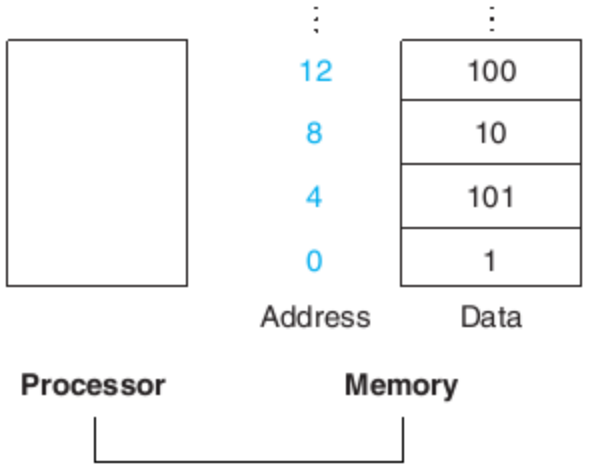
Toán hạng

Toán hạng bộ nhớ (2):

❖ Bộ nhớ chỉ là một mảng đơn chiều lớn, với địa chỉ đóng vai trò là chỉ số trong mảng đó, bắt đầu từ 0. Ví dụ, trong hình 2, địa chỉ của phần tử thứ ba là 2, và giá trị của bộ nhớ [2] là 10.



Hình 2: Địa chỉ bộ nhớ và nội dung của bộ nhớ ở những địa chỉ.



Hình 3: Địa chỉ bộ nhớ MIPS thực tế và nội dung của bộ nhớ cho những từ đó.

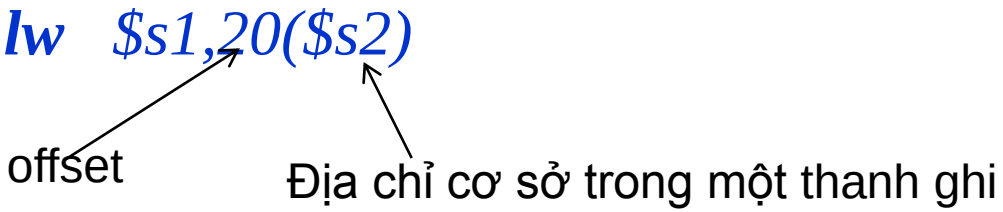
Đây là một sự đơn giản hóa của địa chỉ MIPS; Hình 3 cho thấy địa chỉ MIPS thực tế cho các địa chỉ các từ tuần tự trong bộ nhớ.

Các địa chỉ thay đổi được đánh dấu xanh để tương phản với Hình 2. Từ địa chỉ MIPS trên mỗi byte, địa chỉ từ là bội của bốn: có bốn byte trong một từ.

Toán hạng

Toán hạng bộ nhớ (3):

❖ Lệnh chuyển dữ liệu từ bộ nhớ vào thanh ghi gọi là **load (viết tắt lw-load word)**. Định dạng của các lệnh nạp:



- *$s1:* thanh ghi được nạp dữ liệu vào.
- Một hằng số (20) và thanh ghi ($s2) được sử dụng để truy cập vào bộ nhớ. Tổng số của hằng số và nội dung của thanh ghi thứ hai là địa chỉ bộ nhớ của phần tử cần truy cập đến.

Toán hạng

Toán hạng bộ nhớ (4):

Ví dụ về lệnh nạp:

Giả sử rằng A là một mảng của 100 từ và trình biên dịch đã kết hợp các biến g và h với các thanh ghi $s1 và $s2 như trước. Giả định rằng địa chỉ bắt đầu của mảng A (hay **địa chỉ cơ sở)** chứa trong $s3. Hãy biên dịch đoạn lệnh bằng ngôn ngữ C sau:

g = h + A[8];

➔ Biên dịch:

lw *$t0,* ***8****($s3)* *# $t0 nhận A[8]*

add *$s1,$s2,$t0* *# g = h + A[8]*

Thực tế trong MIPS, 1 từ là 4 bytes
lw $ t0, ***32****($s3)*

❖ Hằng số trong một lệnh truyền dữ liệu (8) được gọi là **offset**, và thanh ghi thêm vào để tạo thành địa chỉ ($s3) được gọi là **thanh ghi cơ sở**.

Toán hạng

Toán hạng bộ nhớ (5):

❖ Qui định sắp xếp:

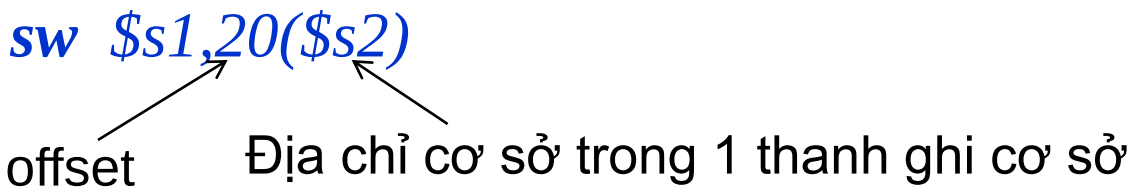
- Trong MIPS, các từ phải bắt đầu từ địa chỉ là bội số của 4. Yêu cầu này được gọi là một **Qui định sắp xếp (alignment restriction)**, và nhiều kiến trúc có nó. (giúp việc truyền dữ liệu nhanh hơn).

- Máy tính phân chia thành đánh số byte trong 1 từ từ trái sang phải (leftmost hay "big en") so với đánh số byte trong 1 từ từ phải sang trái (rightmost hay "litle end"). MIPS thuộc dạng *Big Endian*.

Toán hạng

Toán hạng bộ nhớ (6):

❖ Lệnh **lưu (sw - Store Word)** dữ liệu từ thanh ghi vào bộ nhớ. Định dạng của một lệnh lưu là:



- $s1: thanh ghi chứa dữ liệu cần lưu.
- Một hằng số (20) và thanh ghi ($s2) được sử dụng để truy cập vào bộ nhớ.

Toán hạng

Toán hạng bộ nhớ (7):

Ví dụ lệnh *sw*:

Giả sử biến h được kết nối với thanh ghi $s2 và địa chỉ cơ sở của mảng A là trong $s3. Biên dịch câu lệnh C thực hiện dưới đây sang MIPS?

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

➔Biên dịch:

```
lw  $t0,32($s3)      # $t0 = A[8]
add  $t0,$s2,$t0     # $t0 = h + A[8]
sw  $t0,48($s3)      # A[12] = $t0
```

Toán hạng

Toán hạng hằng:

Nhiều khi một chương trình sẽ sử dụng một hằng số trong một phép toán

Ví dụ:

addi *$s3, $s3, 4* *# $s3 = $s3 + 4*

Toán hạng hằng

Lưu ý:

❖ Mặc dù thanh ghi MIPS xem xét ở đây là 32 bit, có một phiên bản 64-bit của lệnh MIPS thiết lập với thanh ghi 64-bit. Để giữ cả phiên bản cũ, chúng đang chính thức được gọi là MIPS-32 và MIPS-64. (ta quan tâm tập hợp con của MIPS-32)

❖Từ khi MIPS hỗ trợ hằng số âm, không có nhu cầu trừ ngay lập tức trong MIPS.

Chương 02 – Kiến trúc bộ lệnh

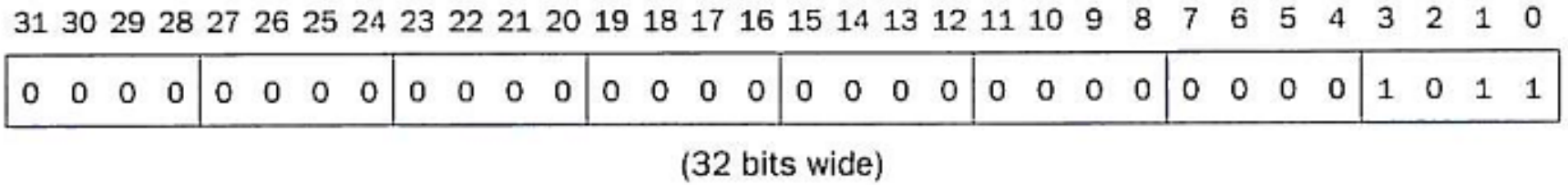
1. Giới thiệu
2. Các phép tính
3. Toán hạng
4. **Số có dấu và không dấu**
5. Biểu diễn lệnh
6. Các phép tính Logic
7. Các lệnh điều kiện và nhảy
8. Các thủ tục hỗ trợ trong phần cứng máy tính
9. Chuyển đổi và bắt đầu một chương trình

Số có dấu và không dấu

- Con người được dạy để suy nghĩ trong hệ cơ số 10, nhưng con số có thể được biểu diễn trong bất kỳ cơ số nào. Ví dụ, 123 cơ số 10 = 1.111.011 cơ số 2.

- Con số này được giữ trong phần cứng máy tính như một loạt các tín hiệu điện thế cao và thấp và do đó chúng được coi là hệ cơ số 2.

Ví dụ: Hình vẽ dưới đây cho thấy số bit trong một từ MIPS và vị trí của các số 1011:



- Từ MIPS có 32 bit độ dài, do đó biểu diễn các số từ 0 đến 2^{32}-1 (4.294.967.295)

- **Bit nhỏ nhất:** Bit ngoài cùng bên phải trong một từ MIPS (bit 0)

- **Bit lớn nhất:** Bit ngoài cùng bên trái trong một từ MIPS (bit 31)

Số có dấu và không dấu

Số dương và âm trong máy tính:

Sử dụng bù 2 biểu diễn

Số đầu tiên là '0 'có nghĩa là dương, số đầu tiên là '1' có nghĩa là âm.

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$
. . . . . .

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 2,147,483,645_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 2,147,483,646_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2,147,483,647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -2,147,483,648_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -2,147,483,647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -2,147,483,646_{ten}$
. . . . . .

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = -3_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -1_{ten}$

Số có dấu và không dấu

- Nửa phần dương của các con số, từ 0 đến $2{,}147{,}483{,}647_{ten}$ $(2^{31} - 1)$, biểu diễn như thường. Phần số âm biểu diễn:

  $1000\ldots0000_{two} = -2{,}147{,}483{,}648_{ten}$

  $1000\ldots0001_{two} = -2{,}147{,}483{,}647_{ten}$

  $1111\ldots1111_{two} = -1_{ten}$

- Bit thứ 32 được gọi là **bit dấu**. Chúng có thể biểu diễn các số dương và âm 32-bit trong điều kiện bit giá trị là một lũy thừa của 2.

- Bù hai có một số âm $-2{,}147{,}483{,}648_{ten}$, mà không có số dương tương ứng.

- Mỗi máy tính ngày nay sử dụng bù hai để biểu diễn nhị phân cho **số có dấu**.

- **Một cách tính giá trị của số không cần đổi sang bù 2 của số âm:**

$$(x31 \times -2^{31}) + (x30 \times 2^{30}) + (x29 \times 2^{29}) + \ldots + (x1 \times 2^{1}) + (x0 \times 2^{0})$$

Lưu ý: Bit dấu được nhân với -2^{31}, và phần còn lại của các bit sau đó được nhân với các số dương của các giá trị cơ số nào tương ứng của chúng.

Số có dấu và không dấu

Ví dụ: đổi từ hệ 2 sang hệ 10

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$$

Trả lời:

$$\begin{aligned}
&(1 \times -2^{31}) + (1 \times 2^{30}) + (1 \times 2^{29}) + \ldots + (1 \times 2^{2}) + (0 \times 2^{1}) + (0 \times 2^{0}) \\
&= -2^{31} + 2^{30} + 2^{29} + \ldots + 2^{2} + 0 + 0 \\
&= -2{,}147{,}483{,}648_{ten} + 2{,}147{,}483{,}644_{ten} \\
&= -4_{ten}
\end{aligned}$$

Số có dấu và không dấu

Mở rộng số có dấu:

Làm thế nào để chuyển đổi một số nhị phân được biểu diễn trong n bit thành một số biểu diễn với nhiều hơn n bit?

Ví dụ:

Chuyển đổi số nhị phân 16-bit của số 2_{ten} và -2_{ten} thành số nhị phân 32-bit.

➔ 2_{ten}: $0000\ 0000\ 0000\ 0010_{two}$ ⇒ $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two}$

➔ -2_{ten}: $1111\ 1111\ 1111\ 1110_{two}$ ⇒ $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two}$

Chương 02 – 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
8. Các thủ tục hỗ trợ trong phần cứng máy tính
9. Chuyển đổi và bắt đầu một chương trình

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:** 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, sử dụng **định dạng lệnh.**

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 MIPS cộng thành một lệnh máy:

add *$t0,$s1,$s2*

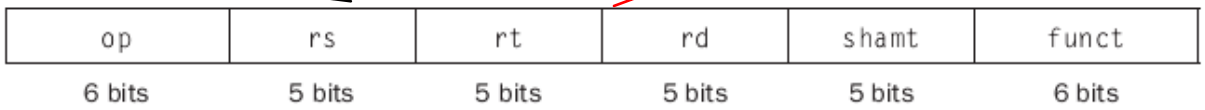
Với định dạng lệnh:

Biểu diễn lệnh

❖ **Trả lời:** Chuyển đổi một lệnh MIPS cộng thành một lệnh máy:

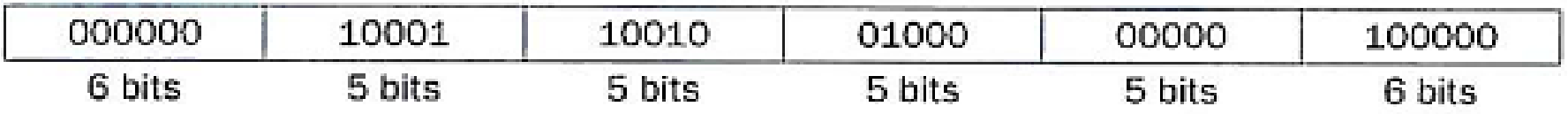
add *$t0,$s1,$s2*

Với định dạng lệnh:



Tra trong bảng MIPS reference data để có các giá trị cần thiết

- Trong ngôn ngữ assembly MIPS, thanh ghi **$s0 đến $s7** tương ứng vào thanh ghi 16 đến 23, và thanh ghi **$t0 đến $t7** tương ứng vào thanh ghi 8 đến 15
- Mỗi phân đoạn của một định dạng lệnh được gọi là một **trường**.
- Các **trường đầu tiên và cuối cùng** (add có phần opcode và Function tương ứng với 0/20hex) kết hợp báo cho máy tính rằng lệnh MIPS này thực hiện **phép cộng**.
- **Trường thứ hai** cho biết số thanh ghi đó là toán hạng nguồn đầu tiên của phép toán cộng ($s1 là thanh ghi số 17)
- **Trường thứ ba** cung cấp cho các toán hạng nguồn khác cho phép cộng ($s2 là thanh ghi số 18).
- **Trường thứ tư** là thanh ghi đích để nhận được tổng ($t0 là thanh ghi số 8).

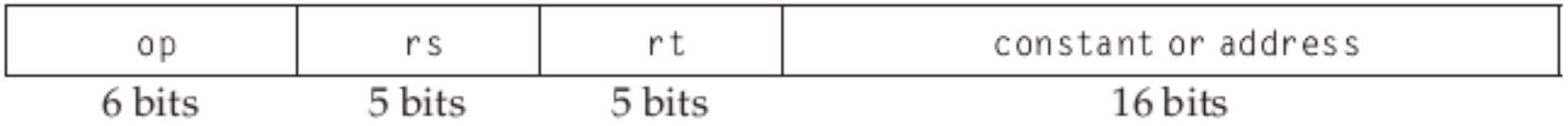


Biểu diễn lệnh

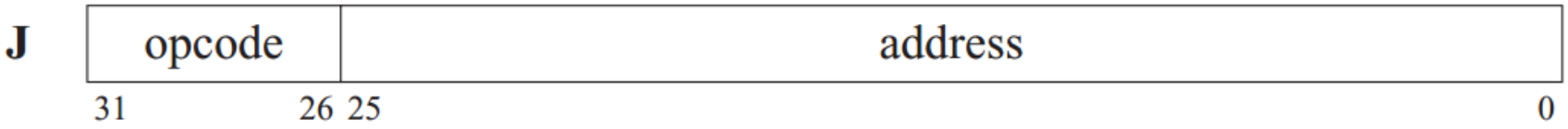
Các dạng khác nhau của định dạng lệnh MIPS :

- **R-type(cho thanh ghi) or R-format**

- **I-type (cho tức thời) hoặc I-format và sử dụng bởi lệnh truyền dữ liệu trực tiếp (tức thời)**



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



Biểu diễn lệnh

Trường MIPS của R-format:

Các trường MIPS được đặt tên để làm cho chúng dễ nhớ hơn:

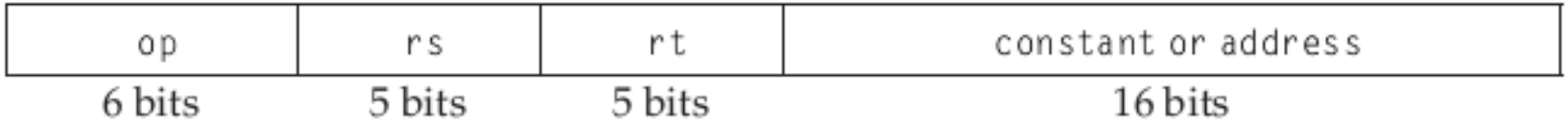
- **op:** theo truyền thống được gọi là **mã tác vụ**.

  Opcode: Trường biểu thị phép toán và định dạng của một lệnh.

- **rs:** Thanh ghi đầu tiên toán hạng nguồn.
- **rt:** Thanh ghi thứ hai toán hạng nguồn.
- **rd:** Thanh ghi toán hạng đích. Nó nhận kết quả của các phép toán.
- **shamt:** số lượng bit dịch chuyển được dùng trong các câu lệnh dịch bit (shift). (không được sử dụng sẽ chứa 0.)
- **funct:** Chức năng. Lĩnh vực này lựa chọn phiên bản cụ thể của các hoạt động trong lĩnh vực op và đôi khi được gọi là mã chức năng.

Biểu diễn lệnh

Trường MIPS của I-format:



Địa chỉ 16-bit có nghĩa là một lệnh có thể truy cập đến một địa chỉ bằng giá trị trong thanh ghi rs cộng với số 16 bit này.

Biểu diễn lệnh

❖ Hình sau: cho thấy mỗi trường cho 1 vài lệnh MIPS

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

Fig.6 MIPS instruction encoding.

- "reg" nghĩa là 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 "cộng" và "trừ" có cùng giá trị trong trường "op"; phần cứng sử dụng trường "funct" để quyết định các biến thể của các phép toán: "cộng" (32) hoặc "trừ" (34) .

Biểu diễn lệnh

Ví dụ: Chuyển ngôn ngữ cấp cao→asembly→mã máy:

- Nếu $t1 chứa địa chỉ cơ sở của mảng A và $s2 tương ứng với h, câu lệnh gán:

A[300] = h + A[300];

- Được chuyển thành:

```
lw $t0,1200($t1)     # Tạm thời reg $t0 nhận A[300]
add $t0,$s2,$t0      # Tạm thời reg $t0 nhận h + A[300]
sw $t0,1200($t1)     # Lưu h + A[300] trở lại vào A[300]
```

- Mã máy ngôn ngữ MIPS 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 thực hiện một chương trình, bạn chỉ cần nạp chương trình và dữ liệu vào bộ nhớ và sau đó báo với máy tính để bắt đầu thực hiện tại một vị trí nhất định trong bộ nhớ.

Chương 02 – 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
8. Chuyển đổi và bắt đầu một chương trình

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**.

Chương 02 – 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**
8. Chuyển đổi và bắt đầu một chương trình

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 bao gồm hai lệnh 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 trong thanh ghi 1 bằng trong thanh ghi 2. Từ beq là viết tắt của "branch if equal" (rẽ nhánh nếu bằng)

➔ Các lệnh như vậy được gọi là **lệnh rẽ nhánh có điều kiện**.

Các lệnh điều kiện và nhảy

Các lệnh rẽ nhánh có điều kiện của MIPS:

Conditional branch	branch on equal	`beq $s1,$s2,25`	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1,$s2,25`	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for `beq`, `bne`
	set on less than unsigned	`sltu $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	`slti $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	`sltiu $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to $ra	For switch, procedure return
	jump and link	`jal 2500`	$ra = PC + 4; go to 10000	For procedure call

Các lệnh điều kiện và nhảy

❖ Biên dịch ***if-then-else*** thành nhánh có điều kiện:

Trong đoạn mã sau đây 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 biên dịch cho câu lệnh *if* này là gì?

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

❖ **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*** trong C

Đây là 1 vòng lặp truyền thống trong C:

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

Giả định rằng i và k tương ứng với thanh ghi $s3 và $s5 và địa chỉ cơ sở của mảng ***save*** lưu trong $s6. Mã assembly MIPS tương ứng với đoạn mã C này 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:
```

Chương 02 – 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
8. **Các thủ tục hỗ trợ trong phần cứng máy tính**
9. Chuyển đổi và bắt đầu một chương trình

Các Thủ Tục Hỗ Trợ Trong Phần Cứng Máy Tính

❖ Một thủ tục hay một hàm là một công cụ mà lập trình viên sử dụng để xây dựng cấu trúc của những chương trình, với mục đích vừa làm cho các chương trình đó dễ hiểu hơn vừa làm cho mã nguồn của các chương trình này có thể được tái sử dụng.

❖ Các thủ tục này cho phép lập trình viên tại một thời điểm chỉ cần tập trung vào một phần của công việc (task) .

❖ Để thực thi một thủ tục, chương trình phải tuân theo sáu bước sau:

1. *Đặt các tham số ở một nơi mà thủ tục có thể truy xuất được.*
2. *Chuyển quyền điều khiển cho thủ tục.*
3. *Yêu cầu tài nguyên lưu trữ cần thiết cho thủ tục đó.*
4. *Thực hiện công việc (task).*
5. *Lưu kết quả ở một nơi mà chương trình có thể truy xuất được.*

6 Trả điều khiển về vị trị mà thủ tục được gọi. Vì một thủ tục có thể được gọi từ nhiều vị trí trong một chương trình.

Các Thủ Tục Hỗ Trợ Trong Phần Cứng Máy Tính

❖ Thanh ghi (Registers) là loại bộ nhớ có tốc độ truy xuất nhanh nhất được dùng để lưu trữ dữ liệu trong một máy tính, cho nên chúng ta muốn tận dụng chúng một cách tối đa

❖ Các phần mềm theo kiến trúc MIPS tuân theo các quy ước về việc gọi thủ tục trong việc cấp phát các thanh ghi 32 bit của nó như sau:

- $a0-@a3 : là 4 thanh ghi lưu tham số được dùng để truyền tham số.
- $v0-$v1: là 2 thanh ghi giá trị được dùng để lưu giá trị trả về.
- $ra: là 1 thanh ghi chứa giá trị địa chỉ để trở về vị trí gọi hàm.

Các Thủ Tục Hỗ Trợ Trong Phần Cứng Máy Tính

❖ *Hợp ngữ trong kiến trúc MIPS bao gồm một lệnh dành riêng cho các thủ tục: nó nhảy tới một địa chỉ và đồng thời lưu lại địa chỉ của lệnh sau vào thanh ghi* ***$ra****. Lệnh nhảy-và-liên kết (jump-and-link)(****jal****) được viết một cách đơn giản như sau:*

jal *ProcedureAddress*

❖ *Ngày nay, những máy tính như MIPS sử dụng lệnh thanh ghi nhảy (jump regiser instruction)* ***(jr)****, có nghĩa là một lệnh nhảy không điều kiện tới địa chỉ được mô tả trong một thanh ghi:*

jr *$ra*

Các Thủ Tục Hỗ Trợ Trong Phần Cứng Máy Tính

Các khái niệm và định nghĩa:

❖Địa chỉ trả về (return address): là một liên kết tới vùng đang gọi cho phép một thủ tục trả về đúng địa chỉ; trong MIPS, nó được lưu trữ ở thanh ghi $ra.

❖**caller:** Là chương trình gọi một thủ tục và cung cấp những giá trị tham số cần thiết.

❖ **callee**: Là một thủ tục thực thi một chuỗi những lệnh được lưu trữ dựa trên những tham số được cung cấp bởi caller và sau đó trả điều khiều về cho caller.

❖ **program counter (PC):** Là thanh ghi chứa địa chỉ của lệnh đang được thực thi trong chương trình.

❖ **stack(ngăn xếp):** Là một cấu trúc dữ liệu cho việc nạp những thanh ghi được tổ chức theo hàng đợi dạng vào-sau ra-trước (last-in first-out) (trong trường hợp trình biên dịch cần nhiều thanh ghi cho một thủ hơn là chỉ có bốn thanh ghi biến và hai thanh ghi giá trị trả về).

❖ **stack pointer (SP):** Là một giá trị biểu thị địa chỉ được cấp gần đây nhất trong ngăn xếp và cho biết vị trí các thanh ghi nên được nạp dữ liệu hoặc vị trí mà các giá trị thanh ghi cũ có thể được tìm thấy. Trong MIPS, nó là thanh ghi $sp.

❖ **push:** là một lệnh, lệnh này sẽ thêm 1 phần tử vào ngăn xếp..

❖ **pop:** là một lệnh làm nhiệm vụ lấy và xóa một phần tử ra khỏi ngăn xếp..

Các Thủ Tục Hỗ Trợ Trong Phần Cứng Máy Tính

Nested procedure:

- Các thủ tục mà không gọi các thủ tục khác là các thủ tục lá (leaf procedures). Ngược lại là nested procedures.

- Chúng ta cần cẩn thận khi sử dụng các thanh ghi trong các thủ tục, càng cần phải cẩn thận hơn khi gọi một nested procedure.

Các Thủ Tục Hỗ Trợ Trong Phần Cứng Máy Tính

Cấp phát không gian cho dữ liệu mới trên ngăn xếp (stack)

❖ **procedure frame (**activation record): Phần của ngăn xếp mà chứa những biến cục bộ và các thanh ghi được lưu trữ của một thủ tục.

❖ **frame pointer:** Giả sử ta có một thủ tục, frame pointer là một giá trị biểu thị vị trí của những biến cục bộ và thanh ghi được lưu trữ cho thủ tục đó.

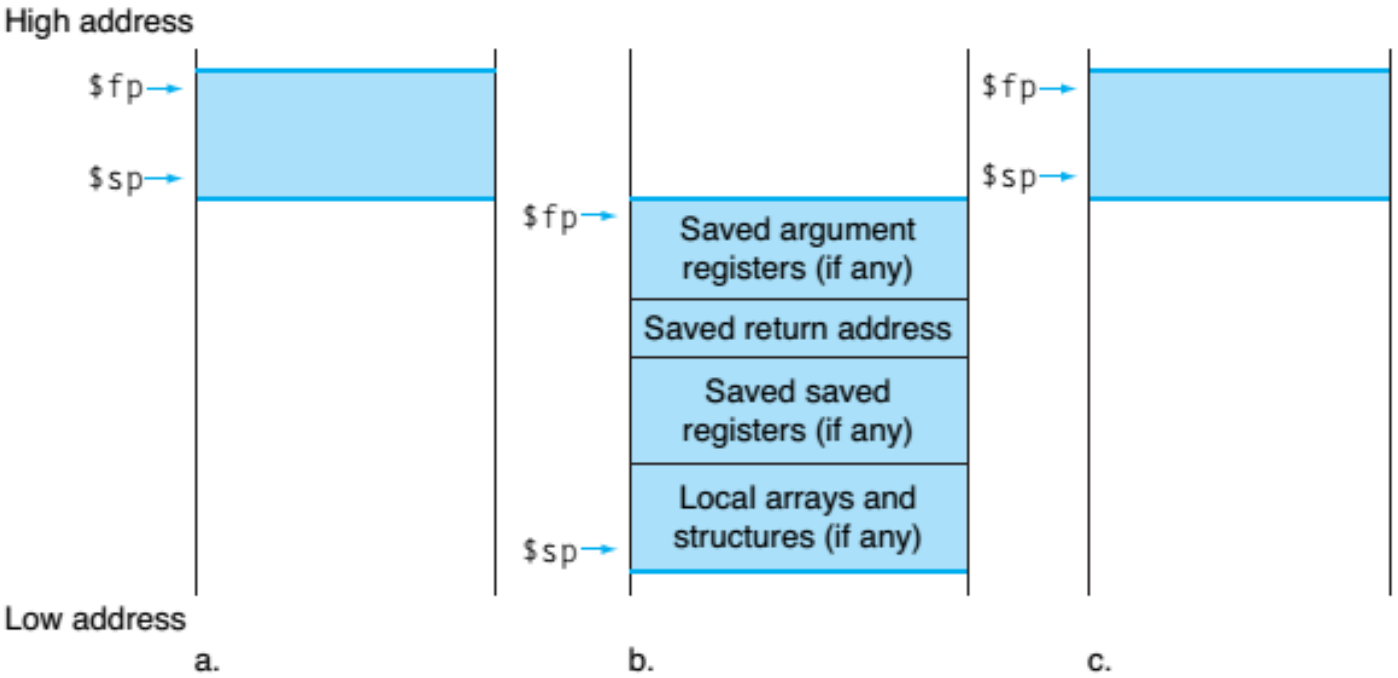


Fig.8 Mô tả của việc cấp phát trên stack (a) trước khi gọi thủ tục, (b) đang gọi thủ tục, (c) sau khi gọi thủ tục.

Các Thủ Tục Hỗ Trợ Trong Phần Cứng Máy Tính

Cấp phát không gian cho dữ liệu mới trên ngăn xếp

❖ **Heap:** Là khu vực bộ nhớ cấp phát động, có thể được cấp phát thêm khi đạt đến giới hạn.

❖ **Text segment:** Đoạn mã chương trình.

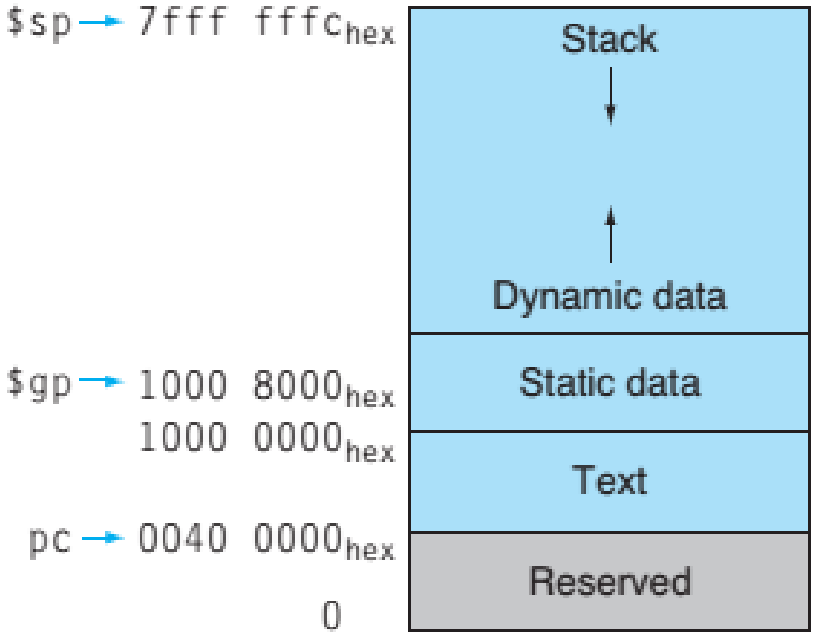


Fig.9 Cấp phát bộ nhớ cho chương trình và dữ liệu kiến trúc MIPS

Các Thủ Tục Hỗ Trợ Trong Phần Cứng Máy Tính

Name	Register number	Usage	Preserved on call?
$zero	0	The constant value 0	n.a.
$v0–$v1	2–3	Values for results and expression evaluation	no
$a0–$a3	4–7	Arguments	no
$t0–$t7	8–15	Temporaries	no
$s0–$s7	16–23	Saved	yes
$t8–$t9	24–25	More temporaries	no
$gp	28	Global pointer	yes
$sp	29	Stack pointer	yes
$fp	30	Frame pointer	yes
$ra	31	Return address	yes

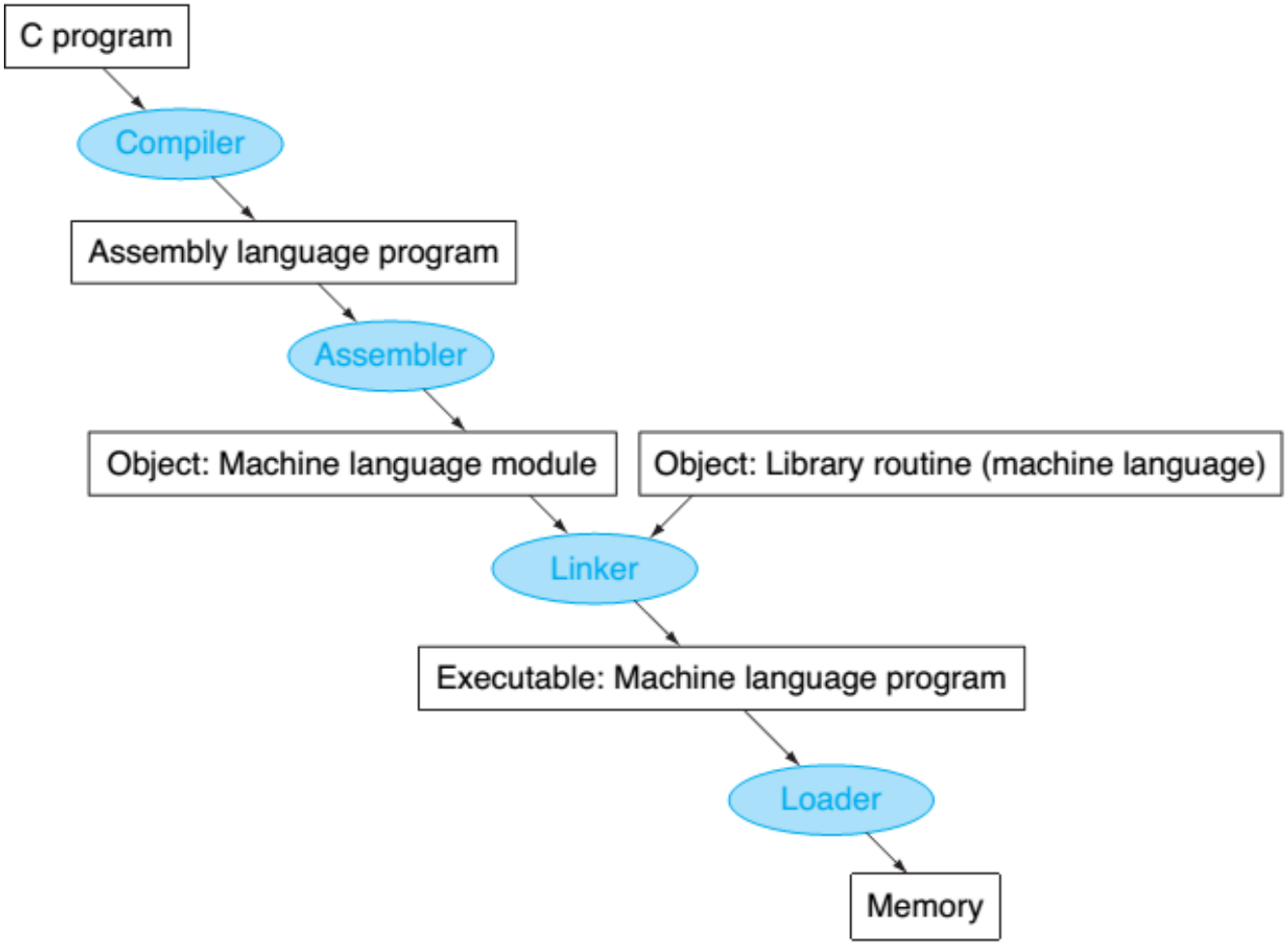
Fig.10 Các quy ước của thanh ghi theo kiến trúc MIPS. Thanh ghi 1, tên gọi là $at, thì được để dành cho assembler, và các thanh ghi 26-27, tên gọi là $k0-$k1, được để dành cho hệ điều hành.

Chương 02 – 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
8. Các thủ tục hỗ trợ trong phần cứng máy tính
9. Chuyển đổi và bắt đầu một chương trình

Chuyển đổi và bắt đầu một chương trình

Phần này mô tả 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.



Hình 12: Một hệ thống phân cấp chuyển đổi cho ngôn ngữ C

Câu hỏi và bài tập chương 2

- Chuyển mã assembly MIPS sang dạng mã máy MIPS
- Chuyển dạng mã máy MIPS sang dạng mã assembly MIPS
- Chuyển dạng mã ngôn ngữ cấp cao sang dạng mã assembly MIPS và ngược lại

Chuyển mã assembly MIPS sang dạng mã máy MIPS

- **add $t0, $s1, $s2**
- sub $s0, $s2, $s3
- sll $s2, $s4, 12
- addi $t0, $t0, -1
- lw $s1, 32($s0)
- sw $a0, 16($t0)
- beq $t0, $0, 50

Chuyển dạng mã máy MIPS sang dạng mã assembly MIPS

- 00001025_{hex}
- $0005402A_{hex}$
- 11000003_{hex}
- 00441020_{hex}
- $20A5FFFF_{hex}$
- 08100001_{hex}

Bài tập

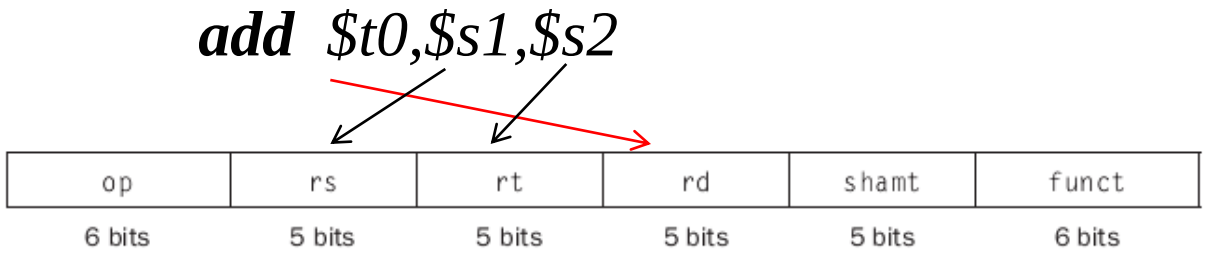
- Chuyển dạng mã ngôn ngữ cấp cao sang dạng mã assembly MIPS, với a, b, c chứa trong các thanh ghi $t0, $t1, $t2

  a = – b – c + 120.

  f = g + h + B[4]; //f,g,h → $s0, $s1, $s2

  // địa chỉ nền của mảng B trong $s6

Chuyển mã assembly MIPS sang dạng mã máy MIPS



- Xác định loại lệnh
  - Thanh ghi: add, sub, or, …
  - Giá trị tức thời: lw, sw, addi, subi, …
- addi $t2, $t2, 2