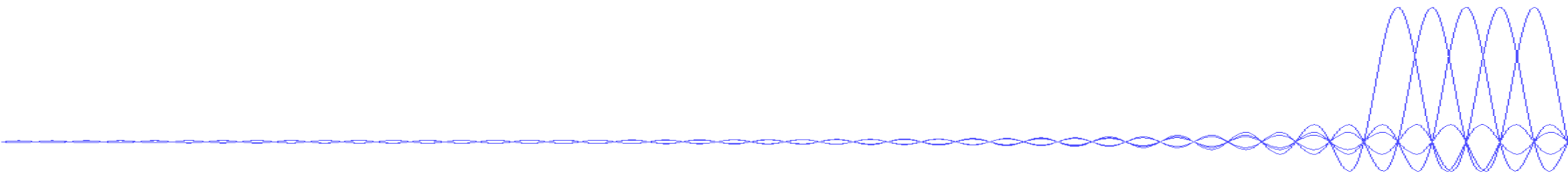
COMPUTER ENGINEERING

KIẾN TRÚC MÁY TÍNH

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

KIẾN TRÚC BỘ LỆNH



COMPUTER ENGINEERING

Nội dung môn học

- Tuần 1: Máy tính − Các khái niệm & Công nghệ.
- Tuần 2: Hiệu suất máy tính.
- Tuần 3: Kiến trúc tập lệnh.
- Tuần 4: Kiến trúc tập lệnh (tiếp theo).
- Tuần 5: Kiến trúc tập lệnh (tiếp theo).
- Tuần 6: Kiến trúc tập lệnh (tiếp theo).
- Tuần 7: Ôn thi giữa kỳ

COMPUTER ENGINEERING

Ôn tập bài cũ

1. **Lịch sử phát triển của máy tính.**
2. **Phân loại máy tính.**
3. **Các lớp thực thi bên trong máy tính.**
4. **Các chức năng và thành phần cơ bản của máy tính.**
5. **Hiệu suất máy tính.**

Tuần 03 – Kiến trúc bộ lệnh

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 3 – 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. **Thủ tục/ hàm**

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)***

To command a computer's hardware, you must speak its language. The words of a computer's language are called *instructions*, and its vocabulary is called an **instruction set**. In this chapter, you will see the instruction set of a real computer, both in the form written by people and in the form read by the computer. We introduce instructions in a top-down fashion. Starting from a notation that looks like a restricted programming language, we refine it step-by-step until you see the real language of a real computer.

(Trích từ sách tham khảo)

Giới thiệu

- Bộ lệnh trong chương này là MIPS, một bộ lệnh kiến trúc máy tính được thiết kế từ năm 1980. Cùng với hai bộ lệnh thông dụng nhất ngày nay:
  - ARM (rất giống MIPS)
  - The Intel x86

1. ARMv7 is similar to MIPS. More than 9 billion chips with ARM processors were manufactured in 2011, making it the most popular instruction set in the world.
2. The second example is the Intel x86, which powers both the PC and the cloud of the PostPC Era.
3. The third example is ARMv8, which extends the address size of the ARMv7 from 32 bits to 64 bits. Ironically, as we shall see, this 2013 instruction set is closer to MIPS than it is to ARMv7.

(Trích từ sách tham khảo)

Tuần 3 – Kiến trúc bộ lệnh

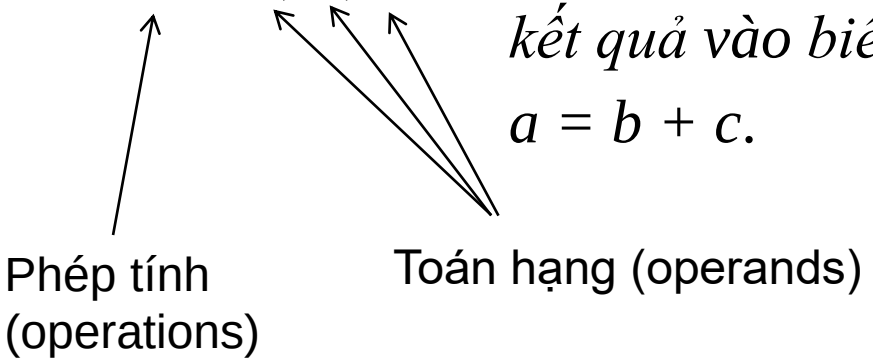
1. **Giới thiệu**
2. **Các phép tính - Các lệnh trong MIPS**
3. **Toán hạng**
4. **Số có dấu và không dấu**
5. **Biểu diễn lệnh**
6. **Các phép tính Logic**
7. **Các lệnh điều kiện và nhảy**

Phép tính (Operations)

Ví dụ:

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

$a = b + c.$



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
```

COMPUTER ENGINEERING

Ví dụ một số lệnh trên MIPS

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three register operands
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	sh $s1,20($s2)	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	lb $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	sb $s1,20($s2)	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	sc $s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
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

COMPUTER ENGINEERING

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}

COMPUTER ENGINEERING

Ví dụ một số lệnh trên MIPS

Category	Instruction	Example	Meaning
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

Ví dụ một số lệnh trên MIPS

Category	Instruction	Example	Meaning
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

s1 = 00110 = 6

sll s2, s1, 1

s2 = 01100 = 12

sll s3, s1, 2

s3 = 11000 = 24

srl s2, s1, 1

s2 = 00011 = 3

COMPUTER ENGINEERING

Ví dụ một số lệnh trên MIPS

Category	Instruction	Example	Meaning
Conditional branch	branch on equal	`beq $s1,$s2,25`	if ($s1 == $s2) go to PC + 4 + 100
	branch on not equal	`bne $s1,$s2,25`	if ($s1!= $s2) go to PC + 4 + 100
	set on less than	`slt $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0
	set on less than unsigned	`sltu $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0
	set less than immediate	`slti $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0
	set less than immediate unsigned	`sltiu $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0
Unconditional jump	jump	`j 2500`	go to 10000
	jump register	`jr $ra`	go to $ra
	jump and link	`jal 2500`	$ra = PC + 4; go to 10000

Tuần 3 – Kiến trúc bộ lệnh

1. **Giới thiệu**
2. **Các phép tính**
3. **Toán hạng (*add* *a, b, c*)**
4. **Số có dấu và không dấu**
5. **Biểu diễn lệnh**
6. **Các phép tính Logic**
7. **Các lệnh điều kiện và nhảy**

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 là 32 bit)

❖ Một sự khác biệt lớn giữa các biến của một ngôn ngữ lập trình và các biến thanh ghi là số thanh ghi bị giới hạn (thường là 32 thanh ghi 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):

❖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*

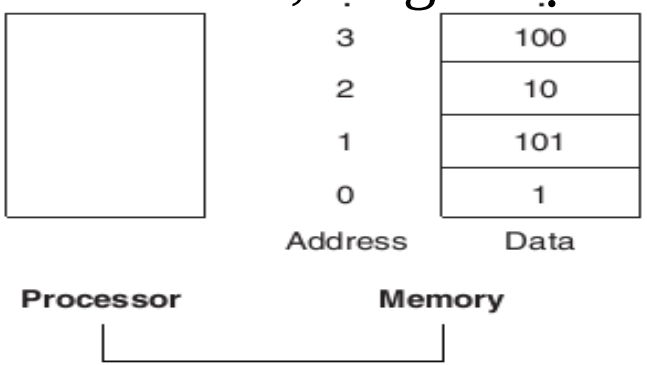
❖ Để truy cập vào một từ trong bộ nhớ, lệnh phải cung cấp **địa chỉ** bộ nhớ.

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.*

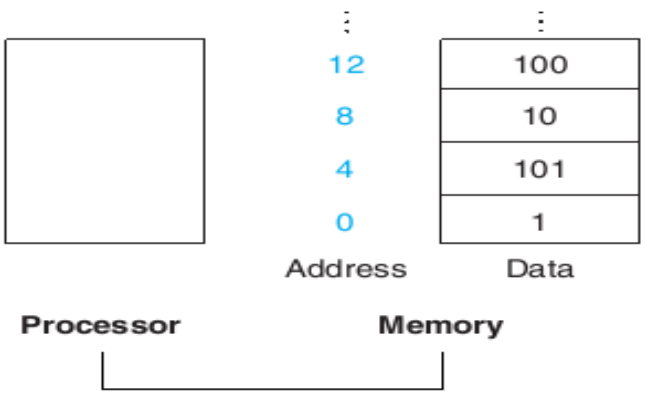
Toán hạng

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 1, địa chỉ của phần tử thứ ba là 2, và giá trị của bộ nhớ [2] là 10.



Hình 1: Địa chỉ và nội dung của bộ nhớ giả lập như mảng.



Hình 2: Địa chỉ và nội dung bộ nhớ MIPS thực tế.

Mỗi từ nhớ (word) của MIPS là 4 bytes. MIPS định địa chỉ theo byte, địa chỉ của mỗi word là địa chỉ của byte đầu tiên trong word đó. **Do đó, địa chỉ mỗi word trong MIPS phải là bội của 4.**

Toán hạng

Toán hạng bộ nhớ (3):

RAM Hexadecimal View

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	FE	C0	0F	48	45	4C	4C	4F	20	57	4F	52	4C	44	21	00
10	D0	02	C0	D0	01	03	D3	00	01	DB	00	00	C1	0B	D4	02
20	00	A4	02	A4	01	C0	F1	D0	01	0C	D0	00	AA	F1	01	AD
30	00	A5	01	C2	FA	C0	04	C0	D9	D0	01	0C	D0	00	FF	F1
40	02	D0	00	FE	F1	02	D0	00	01	F1	02	D0	00	00	F1	02
50	A5	01	C2	EA	F0	03	DB	00	01	C1	09	D0	00	80	F1	03
60	C0	07	D0	00	00	F1	03	D0	01	20	A5	01	C2	FE	D0	00
70	FF	F1	04	D0	01	0A	D0	00	4F	F1	04	A5	01	C2	FC	D0
80	01	20	D0	00	11	F1	05	9A	00	A5	01	C2	FA	D0	02	C0
90	D0	01	0C	D0	00	20	D4	02	00	A4	02	A5	01	C2	F9	C0
A0	98	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
D0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
E0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
F0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20

X Hexadecimal Y ASCII Z Source

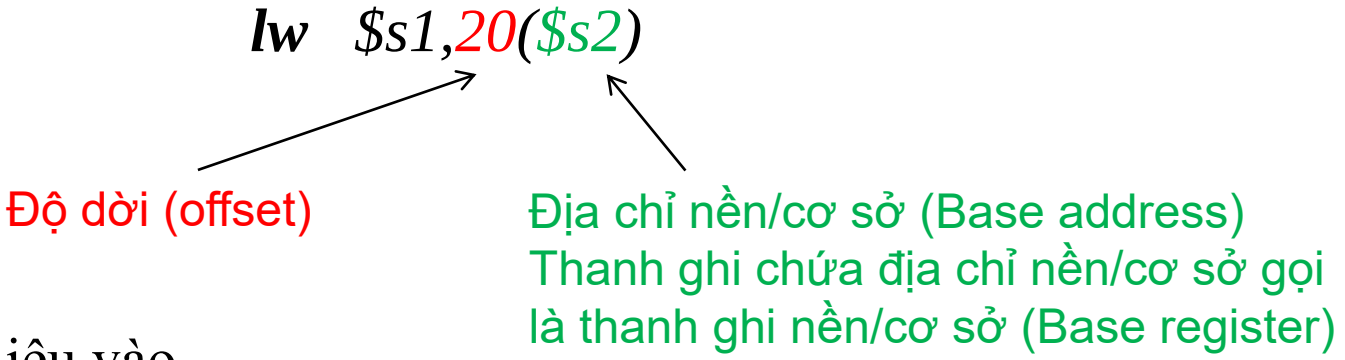
Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Toán hạng

Toán hạng bộ nhớ (3):

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



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

Toán hạng bộ nhớ (4):

Ví dụ về lệnh lw:

Giả sử rằng *A* là một mảng của 100 phần tử (mỗi phần tử cần 1 word lưu trữ) và trình biên dịch đã kết hợp các biến g và h với các thanh ghi *$s1* và *$s2*. Giả định rằng địa chỉ bắt đầu của mảng A (hay **địa chỉ cơ sở/nền)** chứa trong *$s3*. Hãy biên dịch đoạn lệnh bằng ngôn ngữ C sau sang MIPS:

g = h + A[8];

Thực tế trong MIPS một word là 4 bytes, do đó lệnh đúng phải là:

lw $ t0, 32($s3)

➔ Biên dịch:

```
lw  $t0, 8($s3)        # $t0 nhận A[8]
add  $s1,$s2,$t0       # g = h + A[8]
```

❖ Hằng số trong một lệnh truyền dữ liệu (*8*) gọi là **offset**, và thanh ghi chứa địa chỉ bắt đầu của mảng (*$s3*) gọi là **thanh ghi cơ sở**.

Toán hạng

Bài tập về lệnh lw (1):

Giả sử rằng f, g, h được lưu lần lượt ở các thanh ghi $s0, $s1, $s2. Địa chỉ cơ sở/nền (base address) của mảng B được lưu trong các thanh ghi $s7. Hãy biên dịch đoạn lệnh bằng ngôn ngữ C sau sang MIPS:

$$f = g + h + B[4];$$

➔ Biên dịch:

```
lw  $t0, 16($s7)          # $t0 nhận B[4] $t0 = M [$s7+16]
add  $s0,$s1,$s2          # f = g + h
add $s0, $s0, $t0         # f = g + h + B[4]
```

Toán hạng

Bài tập về lệnh lw (1):

Giả sử rằng f, g, h, i, j được lưu lần lượt ở các thanh ghi $s0, $s1, $s2, $s3, $s4. Địa chỉ cơ sở/nền (base address) của mảng A và B được lưu trong các thanh ghi $s6 và $s7. Hãy biên dịch đoạn lệnh bằng ngôn ngữ C sau sang MIPS:

f = A[i];

➔ Biên dịch:

lw *$s0,* ***$s3*4****($s6)* *# $s0 nhận A[i] $t0 = M [$s6+$s3*4]*

sll *$t1, $s3, 2* *# $t1 = $s3*4*

lw *$s0,* ***$t1****($s6)* *# $s0 nhận A[i] $s0 = M [$s6+$s3*4]*

Bài tập về lệnh lw (1):

Giả sử rằng f, i được lưu lần lượt ở các thanh ghi $s0, $s3. Địa chỉ cơ sở/nền (base address) của mảng A được lưu trong các thanh ghi $s6. Hãy biên dịch đoạn lệnh bằng ngôn ngữ C sau sang MIPS:

f = A[i];

➔ Biên dịch:

```
lw  $s0, $s3*4($s6)     # $s0 nhận A[i] $t0 = M [$s6+$s3*4]
sll $t1, $s3, 2         # $t1 = $s3*4
add  $t2,$s6,$t1        # $t2 = $s6+$s3*4
lw  $s0, 0($t2)         # $s0 nhận A[i] $s0 = M [0 + t2] = M [0 + $s6+$s3*4]
```

Toán hạng

Bài tập về lệnh lw (2):

Giả sử rằng f, g được lưu lần lượt ở các thanh ghi $s0, $s1. Địa chỉ cơ sở/nền (base address) của mảng A và B được lưu trong các thanh ghi $s6 và $s7, mỗi phần tử của mảng lưu 1 từ. Hãy biên dịch đoạn lệnh bằng ngôn ngữ C sau sang MIPS:

$$f = g - A[B[4]];$$

➔ Biên dịch:

```
lw  $t0, 16($s7)      # $t0 nhận B[4] = M[4*4 + s7]
sll $t1, $t0, 2       # $t1 = B[4]*4
add $t2, $t1, $s6     # $t2 = (B[4]*4) + s6
lw  $t3, 0($t2)       #  $t3 = A[B[4]] = M [0 + (B[4]*4) + s6]
sub $s0, $s1, $t3     #  f = g – A[B[4]]
```

Toán hạng

Bài tập về lệnh lw (3):

Giả sử rằng f, g được lưu lần lượt ở các thanh ghi $s0, $s1. Địa chỉ cơ sở/nền (base address) của mảng A và B được lưu trong các thanh ghi $s6 và $s7, mỗi phần tử của mảng lưu 1 từ. Hãy biên dịch đoạn lệnh bằng ngôn ngữ C sau sang MIPS:

f = A[B[g] + 1];

➔ Biên dịch:

```
sll $t0, $s1, 2        # $t0 = g*4
add $t1, $t0, $s7      # $t1 = (g*4) + s7
lw  $t2, 0($t1)        # $t2 nhận B[g] = M[g*4 + s7]
addi $t3, $t2, 1       # $t3 = B[g] + 1
sll $t4, $t3, 2        # $t4 = (B[g] + 1)*4
add $t5, $t4, $s6      # $t5 = (B[g] + 1)*4 + s6
lw  $s0, 0($t5)        #  f = A[B[g] + 1] = M[(B[g] + 1)*4 + s6]
```

1. **Giới thiệu**
2. **Các phép tính**
3. **Toán hạng (*add* *a, b, c*)**
4. **Số có dấu và không dấu**
5. **Biểu diễn lệnh**
6. **Các phép tính Logic**
7. **Các lệnh điều kiện và nhảy**

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 bộ nhớ (5):

❖ Lệnh chuyển dữ liệu từ thanh ghi ra bộ nhớ, gọi là lệnh lưu (store) **(viết tắt sw – store word)**. Định dạng của các lệnh lưu:

sw *$s1,20($s2)*

offset — Base address in base register

•$s1: thanh ghi chứa dữ liệu cần lưu.

•Một hằng số (20) và thanh ghi ($s2) được sử dụng để truy cập vào bộ nhớ. Tổng số của hằng số và nội dung của thanh ghi này là địa chỉ bộ nhớ, nơi mà nội dung đang chứa trong thanh ghi *$s1* sẽ được lưu vào đây.

Toán hạng bộ nhớ (6):

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

Bài tập về lệnh sw:

Giả sử rằng h, i được lưu lần lượt ở các thanh ghi $s2, $s3. Địa chỉ cơ sở/nền (base address) của mảng A được lưu trong các thanh ghi $s6. Hãy biên dịch đoạn lệnh bằng ngôn ngữ C sau sang MIPS:

A[i] = h;

➔ Biên dịch:

sw *$s2,* ***$s3*4****($s6)* *# ô nhớ A[i] chứa $s2 -> M [$s6+$s3*4] = $s2*

sll *$t1, $s3, 2* *# $t1 = $s3*4*

sw *$s2,* ***$t1****($s6)* *# ô nhớ A[i] chứa $s2 -> M [$s6+$s3*4] = $s2*

Bài tập về lệnh sw:

Giả sử rằng h, i được lưu lần lượt ở các thanh ghi $s2, $s3. Địa chỉ cơ sở/nền (base address) của mảng A được lưu trong các thanh ghi $s6. Hãy biên dịch đoạn lệnh bằng ngôn ngữ C sau sang MIPS:

A[i] = h;

➔ Biên dịch:

sw *$s2,* ***$s3*4****($s6)* *# ô nhớ A[i] chứa $s2 -> M [$s6+$s3*4] = $s2*

sll *$t1, $s3, 2* *# $t1 = $s3*4*
add *$t2,$s6,$t1* *# $t2 = $s6+$s3*4*
sw *$s2,* ***0****($t2)* *# ô nhớ A[i] chứa $s2 -> M [$s6+$s3*4] = $s2*

Toán hạng bộ nhớ (7):

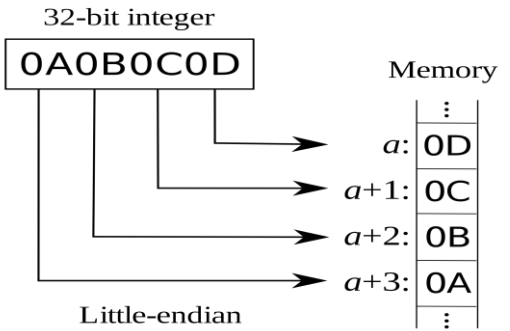
❖**Alignment Restriction**: Trong MIPS, các từ phải bắt đầu từ địa chỉ là bội số của 4. Yêu cầu này được gọi là một **"alignment restriction"** và nhiều kiến trúc hiện nay buộc tuân theo quy định này nhằm giúp việc truyền dữ liệu nhanh hơn. Tuy nhiên một số kiến trúc vẫn không bắt buộc quy định này.

(Chú ý: Tại sao tuân theo điều này giúp truyền dữ liệu nhanh hơn → đọc chương 5 sách tham khảo chính)

❖Leftmost - **"Big End", "Big Endian"**

Rightmost - **"Little End", "Little Endian"**

➔ MIPS thuộc dạng *nào?*



Toán hạng

Toán hạng hằng:

Một hằng số/số tức thời (constant/immediate number) có thể được sử dụng trong trong một phép toán

Ví dụ:

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

Toán hạng hằng

Toán hạng

Tóm lại, chỉ có 3 loại toán hạng trong một lệnh của MIPS

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

Lưu ý:

❖ Các hằng số trong MIPS có thể âm nên không cần phép trừ một thanh ghi và một số tức thời trong MIPS.

❖ Trong thực tế, có một phiên bản khác của MIPS làm việc với các thanh ghi 64 bits, gọi là MIPS-64. MIPS xem xét trong môn học này là MIPS làm việc với các thanh ghi chỉ 32 bit, gọi là MIPS-32.

⇒**Trong phạm vi môn học này, MIPS dùng chung sẽ hiểu là MIPS-32**

Kiến trúc bộ lệnh

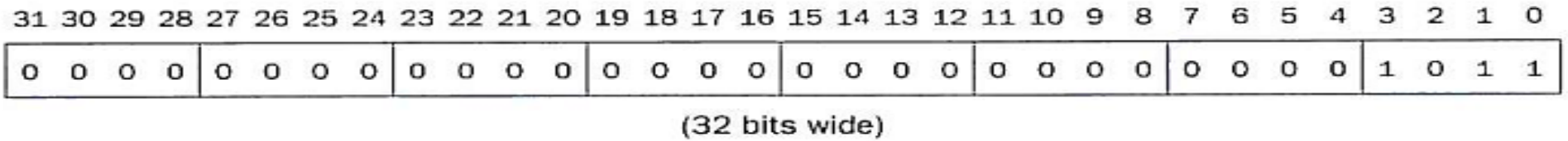
1. **Giới thiệu**
2. **Các phép tính**
3. **Toán hạng**
4. **Số có dấu và không dấu**
5. **Biểu diễn lệnh**
6. **Các phép tính Logic**
7. **Các lệnh điều kiện và nhảy**

COMPUTER ENGINEERING

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 = 1111011 cơ số 2.
- Số lưu trữ trong máy tính như một chuỗi các tín hiệu điện thế cao và thấp, do đó chúng được xem như hệ cơ số 2.

Ví dụ: Hình vẽ dưới đây cho thấy như thế nào một word của MIPS lưu trữ số 1011:



- Một word của MIPS có 32 bit, do đó có thể biểu diễn các số từ 0 đến $2^{32}-1$ (4.294.967.295)
- **Bit trọng số nhỏ nhất *(The least significant bit – LSB)*:** Bit ngoài cùng bên phải trong một từ nhớ (bit 0)
- **Bit trọng số lớn nhất *(The most significant bit – MSB)*:** Bit ngoài cùng bên trái trong một từ nhớ (bit 31)

Số có dấu và không dấu

❖ Số dương và âm trong máy tính:

Các máy tính hiện tại sử dụng **bù hai** để biểu diễn nhị phân cho **số có dấu**.

- Nếu MSB = 0: số dương
- Nếu MSB = 1: số âm.

➔Bit thứ 32 (MSB) còn được gọi là **bit dấu**.

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

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

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

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:

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

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

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

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

$$
\begin{aligned}
&0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}\\
&0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}\\
&0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}\\
&\ldots \qquad\qquad\qquad\qquad\qquad\qquad \ldots\\
&0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 2{,}147{,}483{,}645_{ten}\\
&0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 2{,}147{,}483{,}646_{ten}\\
&0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2{,}147{,}483{,}647_{ten}\\
&1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -2{,}147{,}483{,}648_{ten}\\
&1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -2{,}147{,}483{,}647_{ten}\\
&1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -2{,}147{,}483{,}646_{ten}\\
&\ldots \qquad\qquad\qquad\qquad\qquad\qquad \ldots\\
&1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = -3_{ten}\\
&1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2_{ten}\\
&1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -1_{ten}
\end{aligned}
$$

Số có dấu và không dấu

Công thức chuyển từ một số bù hai sang số hệ 10:

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

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.

Ví dụ: đổi từ hệ 2 sang hệ 10

0…0100

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

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}: $1111\ 1111\ 1111\ 1110_{two}$ ⟹ $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two}$

Số có dấu và không dấu

Khi làm việc với các lệnh của MIPS, lưu ý:

- Mở rộng có dấu (Sign-extend)
- Mở rộng không dấu (Zero-extend)

Tổng kết:

- Giới thiệu lệnh máy tính, tập lệnh là gì

(*Tập lệnh được sử dụng cụ thể trong môn học này là MIPS 32 bits)*

- Tập lệnh bao gồm các nhóm lệnh cơ bản: Nhóm lệnh logic, nhóm lệnh số học, nhóm lệnh trao đổi dữ liệu và nhóm lệnh nhảy

- Với MIPS, toán hạng cho các lệnh được chia thành ba nhóm: nhóm toán hạng thanh ghi, nhóm toán hạng bộ nhớ và nhóm toán hạng là số tức thời

- Nhắc lại số có dấu và số không dấu

Tuần 3 – Kiến trúc bộ lệnh

- **Lý thuyết: Đọc sách tham khảo**
- Mục: 2.1, 2.2, 2.3, 2.4
- 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