

Một số lệnh assembly MIPS cơ bản

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (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 _{hex}
Add Immediate	addi I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 _{hex}
Add Imm. Unsigned	addiu I	$R[rt] = R[rs] + \text{SignExtImm}$	(2) 9 _{hex}
Add Unsigned	addu R	$R[rd] = R[rs] + R[rt]$	0/21 _{hex}
And	and R	$R[rd] = R[rs] \& R[rt]$	0/24 _{hex}
And Immediate	andi I	$R[rt] = R[rs] \& \text{ZeroExtImm}$	(3) c _{hex}
Branch On Equal	beq I	if $R[rs] == R[rt]$ $PC = PC + 4 + \text{BranchAddr}$	(4) 4 _{hex}
Branch On Not Equal	bne I	if $R[rs] \neq R[rt]$ $PC = PC + 4 + \text{BranchAddr}$	(4) 5 _{hex}
Jump	j J	$PC = \text{JumpAddr}$	(5) 2 _{hex}
Jump And Link	jal J	$R[31] = PC + 8; PC = \text{JumpAddr}$	(5) 3 _{hex}
Jump Register	jr R	$PC = R[rs]$	0/08 _{hex}
Load Byte Unsigned	lbu I	$R[rt] = \{24'b0, M[R[rs]] + \text{SignExtImm}(7:0)\}$	(2) 24 _{hex}
Load Halfword Unsigned	lhu I	$R[rt] = \{16'b0, M[R[rs]] + \text{SignExtImm}(15:0)\}$	(2) 25 _{hex}
Load Linked	ll I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2,7) 30 _{hex}
Load Upper Imm.	lui I	$R[rt] = \{\text{imm}, 16'b0\}$	f _{hex}
Load Word	lw I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 23 _{hex}
Nor	nor R	$R[rd] = \sim (R[rs] R[rt])$	0/27 _{hex}
Or	or R	$R[rd] = R[rs] R[rt]$	0/25 _{hex}
Or Immediate	ori I	$R[rt] = R[rs] \text{ZeroExtImm}$	(3) d _{hex}
Set Less Than	slt R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	0/2a _{hex}
Set Less Than Imm.	slti I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	a _{hex}
Set Less Than Imm. Unsigned	sltiu I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2,6) b _{hex}
Set Less Than Unsig.	sltu R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	(6) 0/2b _{hex}
Shift Left Logical	sll R	$R[rd] = R[rt] \ll \text{shamt}$	0/00 _{hex}
Shift Right Logical	srl R	$R[rd] = R[rt] \gg \text{shamt}$	0/02 _{hex}
Store Byte	sb I	$M[R[rs] + \text{SignExtImm}(7:0)] = R[rt](7:0)$	(2) 28 _{hex}
Store Conditional	sc I	$M[R[rs] + \text{SignExtImm}] = R[rt];$ $R[rt] = (\text{atomic}) ? 1 : 0$	(2,7) 38 _{hex}
Store Halfword	sh I	$M[R[rs] + \text{SignExtImm}(15:0)] = R[rt](15:0)$	(2) 29 _{hex}
Store Word	sw I	$M[R[rs] + \text{SignExtImm}] = R[rt]$	(2) 2b _{hex}
Subtract	sub R	$R[rd] = R[rs] - R[rt]$	(1) 0/22 _{hex}
Subtract Unsigned	subu R	$R[rd] = R[rs] - R[rt]$	0/23 _{hex}

- (1) May cause overflow exception
 (2) $\text{SignExtImm} = \{16\{\text{immediate}[15]\}, \text{immediate}\}$
 (3) $\text{ZeroExtImm} = \{16\{1b'0\}, \text{immediate}\}$
 (4) $\text{BranchAddr} = \{14\{\text{immediate}[15]\}, \text{immediate}, 2'b0\}$
 (5) $\text{JumpAddr} = \{PC + 4[31:28], \text{address}, 2'b0\}$
 (6) Operands considered unsigned numbers (vs. 2's comp.)
 (7) Atomic test&set pair; $R[rt] = 1$ if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode		rs	rt	rd	shamt	funct
	31	26 25	21 20	16 15	11 10	6 5	0
I	opcode		rs	rt	immediate		
	31	26 25	21 20	16 15	0		
J	opcode		address				
	31	26 25	0				

ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FUNCT (Hex)
Branch On FP True	bclt FI	if $(FPcond) PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/1/
Branch On FP False	bclt FI	if $(!FPcond) PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/0/
Divide	div R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	0/
Divide Unsigned	divu R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	(6) 0/
FP Add Single	add.s FR	$F[fd] = F[fs] + F[ft]$	11/10/
FP Add Double	add.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} + \{F[ft], F[ft+1]\}$	11/11/
FP Compare Single	c.x.s* FR	$FPcond = (F[fs] op F[ft]) ? 1 : 0$	11/10/
FP Compare Double	c.x.d* FR	$FPcond = (\{F[fs], F[fs+1]\} op \{F[ft], F[ft+1]\}) ? 1 : 0$	11/11/
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)			
FP Divide Single	div.s FR	$F[fd] = F[fs] / F[ft]$	11/10/
FP Divide Double	div.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} / \{F[ft], F[ft+1]\}$	11/11/
FP Multiply Single	mul.s FR	$F[fd] = F[fs] * F[ft]$	11/10/
FP Multiply Double	mul.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} * \{F[ft], F[ft+1]\}$	11/11/
FP Subtract Single	sub.s FR	$F[fd] = F[fs] - F[ft]$	11/10/
FP Subtract Double	sub.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} - \{F[ft], F[ft+1]\}$	11/11/
Load FP Single	lwc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 31/
Load FP Double	ldc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}];$ $F[rt+1] = M[R[rs] + \text{SignExtImm} + 4]$	(2) 35/
Move From Hi	mghi R	$R[rd] = Hi$	0/
Move From Lo	mflr R	$R[rd] = Lo$	0/
Move From Control	mfc0 R	$R[rd] = CR[rs]$	10/0/
Multiply	mult R	$\{Hi, Lo\} = R[rs] * R[rt]$	0/
Multiply Unsigned	multu R	$\{Hi, Lo\} = R[rs] * R[rt]$	(6) 0/
Shift Right Arith.	sra R	$R[rd] = R[rt] \gg \text{shamt}$	0/
Store FP Single	swc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt]$	(2) 39/
Store FP Double	sdc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt];$ $M[R[rs] + \text{SignExtImm} + 4] = F[rt+1]$	(2) 3d/

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31	26 25	21 20	16 15	11 10	6 5
FI	opcode	fmt	ft	immediate		
	31	26 25	21 20	16 15		

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if $R[rs] < R[rt]$ PC = Label
Branch Greater Than	bgt	if $R[rs] > R[rt]$ PC = Label
Branch Less Than or Equal	bte	if $R[rs] \leq R[rt]$ PC = Label
Branch Greater Than or Equal	bge	if $R[rs] \geq R[rt]$ PC = Label
Load Immediate	li	$R[rd] = \text{immediate}$
Move	move	$R[rd] = R[rs]$


REGISTER NAME, NUMBER, USE, CALL CONVENTION

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

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.

Bảng 1. Tóm tắt các lệnh MIPS cơ bản (tham khảo [1])

Các lệnh assembly MIPS trong tài liệu này sẽ được diễn tả theo từng hàng trong bảng 1

①


MIPS Reference Data

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

Tên lệnh (đầy đủ)

Tên lệnh

Lệnh thuộc nhóm gì (R, I hay J)

Lệnh thực hiện chức năng gì

Chức năng của từng lệnh được diễn tả theo kiểu viết của Verilog. Verilog là ngôn ngữ lập trình dùng để mô tả thiết kế phần cứng (sinh viên năm 1, 2, 3 chưa học).

Một số ghi chú cho lệnh tương ứng, được làm rõ ở cuối bảng

opcode và funct cho từng lệnh tương ứng.

Ví dụ: lệnh add có số ở cột này là 0/20_{hex}, tức opcode của add là 0; trường funct trong R-format của add là 20_{hex}

(1) May cause overflow exception

(2) $\text{SignExtImm} = \{ 16\{\text{immediate}[15]\}, \text{immediate} \}$

(3) $\text{ZeroExtImm} = \{ 16\{1b'0\}, \text{immediate} \}$

(4) $\text{BranchAddr} = \{ 14\{\text{immediate}[15]\}, \text{immediate}, 2'b0 \}$

(5) $\text{JumpAddr} = \{ \text{PC}+4[31:28], \text{address}, 2'b0 \}$

(6) Operands considered unsigned numbers (vs. 2's comp.)

(7) Atomic test&set pair; $R[rt] = 1$ if pair atomic, 0 if not atomic

(1) May cause overflow exception

Những lệnh có phần ghi chú (1) sẽ một thông báo lỗi, hay còn gọi là gây ra một ngoại lệ (exception) khi phép toán bị tràn (overflow)

(2) SignExtImm = {16{immediate[15]}, immediate}

Những lệnh có phần ghi chú (2) luôn chứa một số tức thời 16 bits (có dấu dạng bù 2), và số này được mở rộng thành số 32 bits theo kiểu mở rộng có dấu.

Viết theo cấu trúc của Verilog

16{immediate[15]}: là một chuỗi 16 bits; 16 bit này được tạo ra giống y như bit thứ 15 của immediate

{16{immediate[15]}, immediate}: là chuỗi 32 bits, 16 bit thuộc nửa cao được tạo ra giống như bit thứ 15 của immediate, và 16 bit thuộc nửa thấp chính là số tức thời

Ví dụ:

- SignExtImm của 0011 1110 1101 1100 là 0000 0000 0000 0000 0011 1110 1101 1100
- SignExtImm của 1011 1110 1101 1100 là 1111 1111 1111 1111 1011 1110 1101 1100

⇒ Có thể hiểu đơn giản, nếu số tức thời là dương thì 16 bits của nửa cao thêm vào sẽ là 0, còn nếu số tức thời là âm, thì 16bits của nửa cao thêm vào sẽ là 1

(3) ZeroExtImm = {16{1b'0}, immediate}

Những lệnh có phần ghi chú (3) luôn chứa một số tức thời 16 bits (có dấu dạng bù 2), và số này được mở rộng thành số 32 bits theo kiểu mở rộng Zero, tức không cần biết đây là âm hay dương, 16 bits của nửa cao thêm vào đều là 0.

Viết theo cấu trúc của Verilog

16{1b'0}: là một chuỗi 16 bits mà tất cả các bit đều là 0

{16{1b'0}, immediate}: là chuỗi 32 bits, 16 bit thuộc nửa cao là 0 và 16 bit thuộc nửa thấp chính là số tức thời

Ví dụ:

- ZeroExtImm của 0011 1110 1101 1100 là 0000 0000 0000 0000 0011 1110 1101 1100
- ZeroExtImm của 1011 1110 1101 1100 là 0000 0000 0000 0000 1011 1110 1101 1100

(4) BranchAddr = {14{immediate[15]}, immediate, 2'b0}

sẽ được giải thích trong phần lệnh *beq* và *bne*

(5) JumpAddr = { PC + 4[31:28], address, 2'b0}

sẽ được giải thích trong phần lệnh *j* và *jal*

(6) Operations considered unsigned numbers (vs. 2'comp.)

Những lệnh có phần ghi chú (6) luôn làm việc trên số không dấu (unsigned)

(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

Trong bảng 1, chỉ có 2 lệnh *ll* và *sc* là có ghi chú (7)

⇒ 2 lệnh này liên quan đến một số lý thuyết không nằm trong phần giảng dạy lý thuyết, vì vậy bỏ qua 2 lệnh này

Một số ghi chú:**Ký hiệu số:**

- Ký hiệu 0x đầu được dùng để chỉ hệ 16

Ví dụ: $0xffff = ffff_{\text{hex}} = ffff_{(16)}$

- Số ghi bình thường sẽ được hiểu là đang trong hệ 10

Thanh ghi:

- Bộ xử lý chứa 32 thanh ghi để hoạt động, mỗi thanh ghi 32 bits.
- Mỗi thanh ghi sẽ có tên gọi nhớ và số thứ tự tương ứng của nó. Bảng 2 mô tả số thứ tự và tên gọi nhớ của từng thanh ghi
- Như vậy, khi làm việc với thanh ghi có 2 vấn đề cần quan tâm: giá trị và địa chỉ
 - ✓ Giá trị là giá trị đang được chứa trong thanh ghi
 - ✓ Địa chỉ là chỉ số của thanh ghi trong tập 32 thanh ghi.

Ví dụ: Nếu nói thanh ghi \$t3 có giá trị là 5, hoặc thanh ghi \$t3 bằng 5, tức giá trị đang chứa trong \$t3 là 5 và chỉ số/địa chỉ của \$t3 là 11

REGISTER NAME, NUMBER, USE, CALL CONVENTION

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

Bảng 2. Mô tả các thanh ghi (trích từ bảng 1)

Tên gọi nhớ của các thanh ghi

Chỉ số tương ứng của các thanh ghi

Mục đích sử dụng của từng thanh ghi

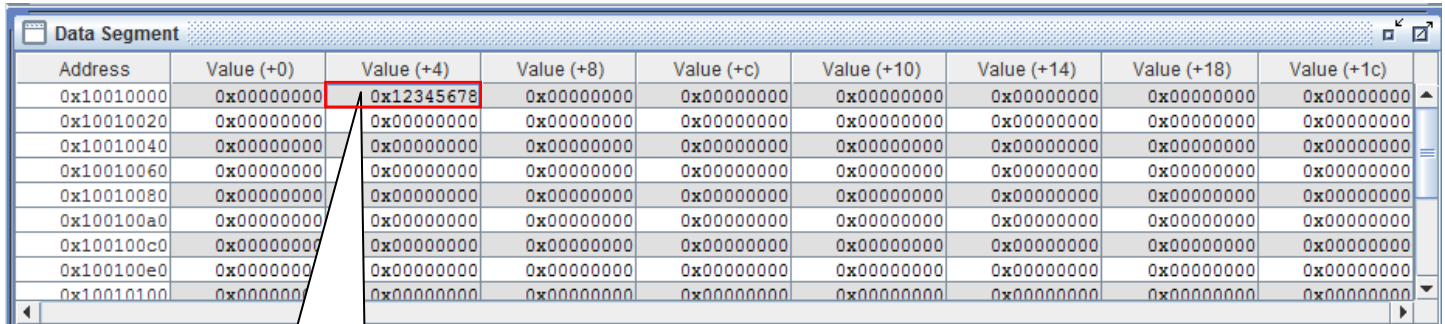
Thanh ghi nào “Yes” là những thanh ghi cần được lưu trữ lại khi thực hiện việc gọi một hàm con

Bộ nhớ:

Tương tự như thanh ghi, khi làm việc với bộ nhớ có 2 vấn đề cần quan tâm: giá trị và địa chỉ

- ✓ Giá trị là giá trị đang được chứa trong một từ nhớ (word), hoặc trong byte
- ✓ Địa chỉ địa chỉ được gán cho word hoặc byte đó.

Ví dụ:



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x12345678	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

Hình 3. Một ví dụ về hình ảnh bộ nhớ từ phần mềm mô phỏng (simulator) MARS 4.4

Đây là word (4 bytes) tại địa chỉ 0x10010004, và có giá trị là 0x12345678

A. Xét các lệnh số học

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data

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 _{hex}
Add Immediate	addi I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 _{hex}
Add Imm. Unsigned	addiu I	$R[rt] = R[rs] + \text{SignExtImm}$	(2) 9 _{hex}
Add Unsigned	addu R	$R[rd] = R[rs] + R[rt]$	0/21 _{hex}
And	and R	$R[rd] = R[rs] \& R[rt]$	0/24 _{hex}
And Immediate	andi I	$R[rt] = R[rs] \& \text{ZeroExtImm}$	(3) c _{hex}
Branch On Equal	beq I	if($R[rs] == R[rt]$) $PC = PC + 4 + \text{BranchAddr}$	(4) 4 _{hex}
Branch On Not Equal	bne I	if($R[rs] != R[rt]$) $PC = PC + 4 + \text{BranchAddr}$	(4) 5 _{hex}
Jump	j J	$PC = \text{JumpAddr}$	(5) 2 _{hex}
Jump And Link	jal J	$R[31] = PC + 8; PC = \text{JumpAddr}$	(5) 3 _{hex}
Jump Register	jrr R	$PC = R[rs]$	0/08 _{hex}
Load Byte Unsigned	lbu I	$R[rt] = \{24'b0, M[R[rs] + \text{SignExtImm}](7:0)\}$	(2) 24 _{hex}
Load Halfword Unsigned	lhu I	$R[rt] = \{16'b0, M[R[rs] + \text{SignExtImm}](15:0)\}$	(2) 25 _{hex}
Load Linked	ll I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2,7) 30 _{hex}
Load Upper Imm.	lui I	$R[rt] = \{\text{imm}, 16'b0\}$	f _{hex}
Load Word	lw I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 23 _{hex}
Nor	nor R	$R[rd] = \sim (R[rs] R[rt])$	0/27 _{hex}
Or	or R	$R[rd] = R[rs] R[rt]$	0/25 _{hex}
Or Immediate	ori I	$R[rt] = R[rs] \text{ZeroExtImm}$	(3) d _{hex}
Set Less Than	slt R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	0/2a _{hex}
Set Less Than Imm.	slti I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2) a _{hex}
Set Less Than Imm. Unsigned	sltiu I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2,6) b _{hex}
Set Less Than Unsig.	sltu R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	(6) 0/2b _{hex}
Shift Left Logical	sll R	$R[rd] = R[rt] \ll \text{shamt}$	0/00 _{hex}
Shift Right Logical	srl R	$R[rd] = R[rt] \gg \text{shamt}$	0/02 _{hex}
Store Byte	sb I	$M[R[rs] + \text{SignExtImm}](7:0) = R[rt](7:0)$	(2) 28 _{hex}
Store Conditional	sc I	$M[R[rs] + \text{SignExtImm}] = R[rt];$ $R[rt] = (\text{atomic}) ? 1 : 0$	(2,7) 38 _{hex}
Store Halfword	sh I	$M[R[rs] + \text{SignExtImm}](15:0) = R[rt](15:0)$	(2) 29 _{hex}
Store Word	sw I	$M[R[rs] + \text{SignExtImm}] = R[rt]$	(2) 2b _{hex}
Subtract	sub R	$R[rd] = R[rs] - R[rt]$	(1) 0/22 _{hex}
Subtract Unsigned	subu R	$R[rd] = R[rs] - R[rt]$	0/23 _{hex}

- (1) May cause overflow exception
 (2) $\text{SignExtImm} = \{16\{\text{immediate}[15]\}, \text{immediate}\}$
 (3) $\text{ZeroExtImm} = \{16\{1b'0\}, \text{immediate}\}$
 (4) $\text{BranchAddr} = \{14\{\text{immediate}[15]\}, \text{immediate}, 2'b0\}$
 (5) $\text{JumpAddr} = \{PC + 4[31:28], \text{address}, 2'b0\}$
 (6) Operands considered unsigned numbers (vs. 2's comp.)
 (7) Atomic test&set pair; $R[rt] = 1$ if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31	26 25	21 20	16 15	11 10	6 5
	0					
I	opcode	rs	rt	immediate		
	31	26 25	21 20	16 15		
	0					
J	opcode	address				
	31	26 25				
	0					

 Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.

ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FUNCT (Hex)
Branch On FP True	bclt FI	if($FPcond$) $PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/1/--
Branch On FP False	bclt FI	if(! $FPcond$) $PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/0/--
Divide	div R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	0/--/--1a
Divide Unsigned	divu R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	(6) 0/--/--1b
FP Add Single	add.s FR	$F[fd] = F[fs] + F[ft]$	11/10/--/0
FP Add Double	add.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} + \{F[ft], F[ft+1]\}$	11/11/--/0
FP Compare Single	c.x.s* FR	$FPcond = (F[fs] op F[ft]) ? 1 : 0$	11/10/--/y
FP Compare Double	c.x.d* FR	$FPcond = (\{F[fs], F[fs+1]\} op \{F[ft], F[ft+1]\}) ? 1 : 0$	11/11/--/y
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)			
FP Divide Single	div.s FR	$F[fd] = F[fs] / F[ft]$	11/10/--/3
FP Divide Double	div.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} / \{F[ft], F[ft+1]\}$	11/11/--/3
FP Multiply Single	mul.s FR	$F[fd] = F[fs] * F[ft]$	11/10/--/2
FP Multiply Double	mul.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} * \{F[ft], F[ft+1]\}$	11/11/--/2
FP Subtract Single	sub.s FR	$F[fd] = F[fs] - F[ft]$	11/10/--/1
FP Subtract Double	sub.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} - \{F[ft], F[ft+1]\}$	11/11/--/1
Load FP Single	lwc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 31/--/--/--
Load FP Double	ldc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}];$ $F[rt+1] = M[R[rs] + \text{SignExtImm} + 4]$	(2) 35/--/--/--
Move From Hi	mghi R	$R[rd] = Hi$	0/--/--/10
Move From Lo	mfl0 R	$R[rd] = Lo$	0/--/--/12
Move From Control	mfc0 R	$R[rd] = CR[rs]$	10/0/--/0
Multiply	mult R	$\{Hi, Lo\} = R[rs] * R[rt]$	0/--/--/18
Multiply Unsigned	multu R	$\{Hi, Lo\} = R[rs] * R[rt]$	(6) 0/--/--/19
Shift Right Arith.	sra R	$R[rd] = R[rt] \gg \text{shamt}$	0/--/--/3
Store FP Single	swc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt]$	(2) 39/--/--/--
Store FP Double	sdc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt];$ $M[R[rs] + \text{SignExtImm} + 4] = F[rt+1]$	(2) 3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31	26 25	21 20	16 15	11 10	6 5
	0					
FI	opcode	fmt	ft	immediate		
	31	26 25	21 20	16 15		
	0					

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if($R[rs] < R[rt]$) $PC = \text{Label}$
Branch Greater Than	bgt	if($R[rs] > R[rt]$) $PC = \text{Label}$
Branch Less Than or Equal	b1e	if($R[rs] \leq R[rt]$) $PC = \text{Label}$
Branch Greater Than or Equal	bge	if($R[rs] \geq R[rt]$) $PC = \text{Label}$
Load Immediate	li	$R[rd] = \text{immediate}$
Move	move	$R[rd] = R[rs]$

REGISTER NAME, NUMBER, USE, CALL CONVENTION

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

Các lệnh số học:

- ❖ *add, addi, addiu, addu*
- ❖ *sub, subu*

R viết tắt của Register
Ví dụ: R[rs] hiểu là giá trị của thanh ghi rs

1. Lệnh add

Add add R $R[rd] = R[rs] + R[rt]$ (1) 0 / 20_{hex}

⇒ Lệnh này thuộc dạng R-format, có opcode là 0 và trường funct giá trị là 20_{hex}

Syntax (cú pháp): (tham khảo Appendix B của sách tham khảo [1])

add rd, rs, rt	0	rs	rt	rd	0	0x20
	6	5	5	5	5	6

Ý nghĩa: $R[rd] = R[rs] + R[rt]$

Thực hiện cộng giá trị thanh ghi rs với giá trị thanh ghi rt, tổng đưa vào thanh ghi rd

Ví dụ:

add \$t0, \$t1, \$t2

Giả sử giá trị đang chứa trong thanh ghi \$t1 là 3, giá trị đang chứa trong thanh ghi \$t2 là 4

Kết quả: Sau khi lệnh add trên thực hiện, giá trị trong thanh ghi \$t0 là 7 ($4 + 3 = 7$).

2. Lệnh addi

Add Immediate addi I $R[rt] = R[rs] + \text{SignExtImm}$ (1,2) 8_{hex}

⇒ Lệnh này thuộc dạng I-format, có opcode 8_{hex}

Syntax (cú pháp):

addi rt, rs, imm	8	rs	rt	imm
	6	5	5	16

Ý nghĩa: $R[rt] = R[rs] + \text{SignExtImm}$

Thực hiện cộng giá trị thanh ghi rs với số tức thời, kết quả đưa vào thanh ghi rt.

Lưu ý: Phạm vi cho số tức thời trong lệnh này là 16 bits. Số tức thời trước khi cộng với thanh ghi rs phải được **mở rộng có dấu thành** (SignExtImm) thành số 32 bits.

Ví dụ:

- a) `addi $t0, $t1, 3`
- b) `addi $t0, $t1, -3`
- c) `addi $t0, $t1, 32768`

Giả sử giá trị đang chứa trong thanh ghi \$t1 cho cả 3 câu đều là 4

Kết quả:

- a) Sau khi `addi` thực hiện xong, giá trị của \$t0 là 7

Quy trình lệnh thực hiện:

số tức thời là $3_{(10)} = 0000\ 0000\ 0000\ 0011_{(2)}$ (số 16 bit có dấu)

`SignExtImm` của $3_{(10)} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011_{(2)}$

Giá trị thanh ghi \$t1 = $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100_{(2)}$

Giá trị trong \$t1 + `SignExtImm` của $3_{(10)} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{(2)}$

- b) Sau khi `addi` thực hiện xong, giá trị của \$t0 là 1

Quy trình lệnh thực hiện:

số tức thời là $-3_{(10)} = 1111\ 1111\ 1111\ 1101_{(2)}$ (số 16 bit có dấu, biểu diễn theo bù 2)

`SignExtImm` của $3_{(10)} = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{(2)}$

Giá trị thanh ghi \$t1 = $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100_{(2)}$

Giá trị trong \$t1 + `SignExtImm` của $3_{(10)} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{(2)}$

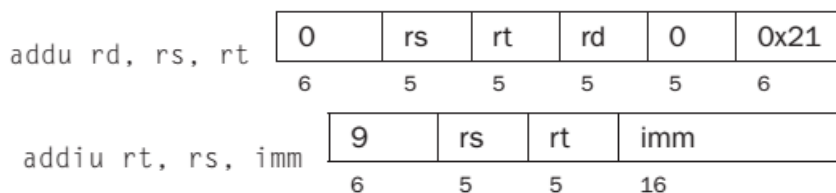
- c) Lệnh bị báo lỗi, do 32768 ra khỏi giới hạn của số 16 bits có dấu

3. Lệnh *addiu* và *addu*

- *Addiu* có cú pháp và thực hiện chức năng giống *addi*
- *Addu* có cú pháp và thực hiện chức năng giống *add*

Tuy nhiên, *addiu* và *addu* không xét kết quả có bị overflow hay không, trong khi đó *addi* và *add* sẽ báo khi overflow xuất hiện

Syntax (cú pháp):



Ví dụ:

a) `addi $t0, $t1, 0x1`

Giả sử thanh ghi `$t1 = 0x7ffffff`

Kết quả:

$$0x1 + 0x7ffffff = 0x80000000$$

Cộng một số dương với một số dương, kết quả ra một số âm => overflow

Khi lệnh `addi` trên thực hiện, một thông báo overflow sẽ xuất hiện

b) `addiu $t0, $t1, 1`

Giả sử thanh ghi `$t1 = 0x7ffffff`

Kết quả: `$t0 = 0x80000000`

Khi lệnh `addi` trên thực hiện, thanh ghi `$t0` vẫn nhận giá trị `0x80000000` và không có bất kì thông báo overflow nào xuất hiện.

4. Lệnh `sub` và `subu`

Lệnh `sub` có cú pháp tương tự như lệnh `add`, nhưng

- **`add`** thực hiện phép toán **cộng** 2 thanh ghi, kết quả lưu vào thanh ghi thứ 3
- trong khi đó, **`sub`** thực hiện phép toán **trừ** 2 thanh ghi, kết quả lưu vào thanh ghi thứ 3

Lệnh `subu` có cú pháp và chức năng giống như `sub`, nhưng

- **`subu`** không xét đến kết quả có bị overflow hay không
- **`sub`** có xét đến kết quả có bị overflow hay không; nếu bị overflow, sẽ có thông báo

Syntax (cú pháp):

<code>sub rd, rs, rt</code>	0	rs	rt	rd	0	0x22
	6	5	5	5	5	6

<code>subu rd, rs, rt</code>	0	rs	rt	rd	0	0x23
	6	5	5	5	5	6

Lưu ý: không có lệnh `subi` (tức trừ một thanh ghi với một số tức thời) vì đã có lệnh `addi` và số tức thời trong `addi` có thể âm hoặc dương, nên `subi` không cần thiết.

Tổng kết:

- ❖ `add, addi, addiu, addu`
- ❖ `sub, subu`

Nhìn lại cột ghi chú của 6 lệnh trên trong bảng 1:

- Chỉ có lệnh `addi` và `addiu` có ghi chú (2) → tức 2 lệnh này làm việc với số tức thời, và số tức thời 16 bits này được mở rộng có dấu thành thành số 32 bits

(có ‘i’ → làm việc với số tức thời)

- Các lệnh không có “u” theo sau: add, addi, sub có thêm ghi chú (1); Các lệnh có “u” theo sau như: addiu, addu và subu không có ghi chú (1) → tức các lệnh không có “u” sẽ báo khi có overflow, còn các lệnh có “u” sẽ không báo khi có overflow

❖ Nhóm lệnh so sánh

slt / sltu

slti / sltiu

Set Less Than	<i>slt</i>	R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	$0 / 2a_{hex}$
Set Less Than Imm.	<i>slti</i>	I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	$(2) a_{hex}$
Set Less Than Imm. Unsigned	<i>sltiu</i>	I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	$(2,6) b_{hex}$
Set Less Than Unsig.	<i>sltu</i>	R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	$(6) 0 / 2b_{hex}$

5. Lệnh slt/sltu

Set Less Than	<i>slt</i>	R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	$0 / 2a_{hex}$
Set Less Than Unsig.	<i>sltu</i>	R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	$(6) 0 / 2b_{hex}$

⇒ Hai lệnh này thuộc nhóm lệnh R-format, có opcode là 0 và funct trong *slt* là $2a_{hex}$, trong *sltu* là $2b_{hex}$

Syntax:

<i>slt rd, rs, rt</i>	0	rs	rt	rd	0	0x2a
	6	5	5	5	5	6
<i>sltu rd, rs, rt</i>	0	rs	rt	rd	0	0x2b
	6	5	5	5	5	6

Ý nghĩa:

slt: $R[rd] = (R[rs] < R[rt]) ? 1 : 0$

Kiểm tra xem giá trị trong thanh ghi rs có nhỏ hơn thanh ghi rt hay không, nếu nhỏ hơn thì thanh ghi rd nhận giá trị là 1; ngược lại thanh ghi rd sẽ nhận giá trị 0

sltu: Ý nghĩa thực hiện giống như *slt*. Nhưng việc kiểm tra giá trị thanh ghi rs có nhỏ hơn thanh ghi rt hay không trong lệnh *slt* thực hiện trên số có dấu, còn trong *sltu* thực hiện trên số không dấu

Ví dụ:

a. `slt $t0, $t1, $t2`

Giả sử $\$t1 = 0xffffffff$, $\$t2 = 0x00000073$

Kết quả: $\$t0 = 1$

Lệnh `slt` so sánh theo kiểu so sánh 2 số có dấu dạng bù 2

$$\$t1 = 0xffffffff = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 0001_{(2)} = -15_{(10)}$$

$$\$t2 = 0x00000073 = 01110011_{(2)} = 115_{(10)}$$

Vậy $\$t1 < \$t2 \rightarrow$ giá trị trong thanh ghi $\$t0 = 1$

b. `sltu $t0, $t1, $t2`

Giả sử $\$t1 = 0xffffffff$, $\$t2 = 0x00000073$

Kết quả: $\$t0 = 0$

Lệnh `sltu` so sánh theo kiểu so sánh 2 số không dấu

$$\$t1 = 0xffffffff = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 0001_{(2)} = 4294967295_{(10)}$$

$$\$t2 = 0x00000073 = 01110011_{(2)} = 115_{(10)}$$

Vậy $\$t1 > \$t2 \rightarrow$ giá trị trong thanh ghi $\$t0 = 0$

6. Lệnh `slti/sltiu`

Set Less Than Imm. `slti` I $R[rt] = (R[rs] < \text{SignExtImm})? 1 : 0 \quad (2) \quad a_{\text{hex}}$

Set Less Than Imm. `sltiu` I $R[rt] = (R[rs] < \text{SignExtImm})? 1 : 0 \quad (2,6) \quad b_{\text{hex}}$
Unsigned

\Rightarrow Hai lệnh này thuộc nhóm lệnh I-format. Opcode của `slti` là a_{hex} , opcode của `sltiu` là b_{hex}

Syntax:

`slti rt, rs, imm`

Oxa	rs	rt	imm
6	5	5	16

`sltiu rt, rs, imm`

Oxb	rs	rt	imm
6	5	5	16

Ý nghĩa:

`slti/sltiu:` $R[rd] = (R[rs] < \text{SignExtImm})? 1 : 0$

Ý nghĩa 2 lệnh này giống nhau là so sánh giá trị một thanh ghi với một số tức thời, nếu giá trị trong thanh ghi rs nhỏ hơn số tức thời thì thanh ghi rd nhận giá trị là 1; ngược lại thanh ghi rd sẽ nhận giá trị 0

Số tức thời cho phép trong lệnh này là số 16 bits. Trước khi so sánh với thanh ghi rs, số tức thời được mở rộng có dấu (SignExtImm) thành số 32 bits

slti khác *sltiu* là *slti* so sánh 2 giá trị theo kiểu có dấu dạng bù 2, trong khi đó *sltiu* so sánh theo kiểu số không dấu

Ví dụ:

c. *slti \$t0, \$t1, 0x73*

Giả sử $\$t1 = 0xffffffff$

Kết quả: $\$t0 = 1$

Lệnh *slt* so sánh theo kiểu so sánh 2 số có dấu dạng bù 2

$$\$t1 = 0xffffffff = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 0001_{(2)} = -15_{(10)}$$

$$\text{Số tức thời} = 0x73 = 01110011_{(2)}$$

$$\text{SignExtImm}(0x73) = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111\ 0011_{(2)} = 115_{(10)}$$

Vậy $\$t1 < \$t2 \rightarrow$ giá trị trong thanh ghi $\$t0 = 1$

d. *sltiu \$t0, \$t1, 0x83*

Giả sử $\$t1 = 0xffffffff$

Kết quả: $\$t0 = 0$

Lệnh *slt* so sánh theo kiểu so sánh 2 số không dấu

$$\$t1 = 0xffffffff = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 0001_{(2)} = 4294967281_{(10)}$$

$$\$t2 = 0x83 = 10000011_{(2)}$$

$$\text{SignExtImm}(0x83) = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1000\ 0011_{(2)} = 4294967171_{(10)}$$

Vậy $\$t1 > \$t2 \rightarrow$ giá trị trong thanh ghi $\$t0 = 0$

B. Các lệnh logic

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (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 _{hex}
Add Immediate	addi I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 _{hex}
Add Imm. Unsigned	addiu I	$R[rt] = R[rs] + \text{SignExtImm}$	(2) 9 _{hex}
Add Unsigned	addu R	$R[rd] = R[rs] + R[rt]$	0/21 _{hex}
And	and R	$R[rd] = R[rs] \& R[rt]$	0/24 _{hex}
And Immediate	andi I	$R[rt] = R[rs] \& \text{ZeroExtImm}$	(3) c _{hex}
Branch On Equal	beq I	if($R[rs] == R[rt]$) $PC = PC + 4 + \text{BranchAddr}$	(4) 4 _{hex}
Branch On Not Equal	bne I	if($R[rs] != R[rt]$) $PC = PC + 4 + \text{BranchAddr}$	(4) 5 _{hex}
Jump	j J	$PC = \text{JumpAddr}$	(5) 2 _{hex}
Jump And Link	jal J	$R[31] = PC + 8; PC = \text{JumpAddr}$	(5) 3 _{hex}
Jump Register	jrr R	$PC = R[rs]$	0/08 _{hex}
Load Byte Unsigned	lbu I	$R[rt] = \{24'b0, M[R[rs]] + \text{SignExtImm}(7:0)\}$	(2) 24 _{hex}
Load Halfword Unsigned	lhu I	$R[rt] = \{16'b0, M[R[rs]] + \text{SignExtImm}(15:0)\}$	(2) 25 _{hex}
Load Linked	ll I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2,7) 30 _{hex}
Load Upper Imm.	lui I	$R[rt] = \{\text{imm}, 16'b0\}$	f _{hex}
Load Word	lw I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 23 _{hex}
Nor	nor R	$R[rd] = \sim(R[rs] R[rt])$	0/27 _{hex}
Or	or R	$R[rd] = R[rs] R[rt]$	0/25 _{hex}
Or Immediate	ori I	$R[rt] = R[rs] \text{ZeroExtImm}$	(3) d _{hex}
Set Less Than	slt R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	0/2a _{hex}
Set Less Than Imm.	slti I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2) a _{hex}
Set Less Than Imm. Unsigned	sltiu I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2,6) b _{hex}
Set Less Than Unsig.	sltu R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	(6) 0/2b _{hex}
Shift Left Logical	sll R	$R[rd] = R[rt] \ll \text{shamt}$	0/00 _{hex}
Shift Right Logical	srl R	$R[rd] = R[rt] \gg \text{shamt}$	0/02 _{hex}
Store Byte	sb I	$M[R[rs] + \text{SignExtImm}(7:0)] = R[rt](7:0)$	(2) 28 _{hex}
Store Conditional	sc I	$M[R[rs] + \text{SignExtImm}] = R[rt];$ $R[rt] = \{\text{atomic}\} ? 1 : 0$	(2,7) 38 _{hex}
Store Halfword	sh I	$M[R[rs] + \text{SignExtImm}(15:0)] = R[rt](15:0)$	(2) 29 _{hex}
Store Word	sw I	$M[R[rs] + \text{SignExtImm}] = R[rt]$	(2) 2b _{hex}
Subtract	sub R	$R[rd] = R[rs] - R[rt]$	(1) 0/22 _{hex}
Subtract Unsigned	subu R	$R[rd] = R[rs] - R[rt]$	0/23 _{hex}

- (1) May cause overflow exception
 (2) $\text{SignExtImm} = \{16\{\text{immediate}[15]\}, \text{immediate}\}$
 (3) $\text{ZeroExtImm} = \{16\{1b'0\}, \text{immediate}\}$
 (4) $\text{BranchAddr} = \{14\{\text{immediate}[15]\}, \text{immediate}, 2'b0\}$
 (5) $\text{JumpAddr} = \{PC + 4[31:28], \text{address}, 2'b0\}$
 (6) Operands considered unsigned numbers (vs. 2's comp.)
 (7) Atomic test&set pair; $R[rt] = 1$ if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31	26 25	21 20	16 15	11 10	6 5
	0					
I	opcode	rs	rt	immediate		
	31	26 25	21 20	16 15		
	0					
J	opcode	address				
	31	26 25				
	0					

ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FUNCT (Hex)
Branch On FP True	bclt FI	if($FPcond$) $PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/1/--
Branch On FP False	bclt FI	if(! $FPcond$) $PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/0/--
Divide	div R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	0/--/--/1a
Divide Unsigned	divu R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	(6) 0/--/--/1b
FP Add Single	add.s FR	$F[fd] = F[fs] + F[ft]$	11/10/--/0
FP Add	add.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} + \{F[ft], F[ft+1]\}$	11/11/--/0
FP Compare Single	c.x.s* FR	$FPcond = (F[fs] \text{ op } F[ft]) ? 1 : 0$	11/10/--/y
FP Compare	c.x.d* FR	$FPcond = (\{F[fs], F[fs+1]\} \text{ op } \{F[ft], F[ft+1]\}) ? 1 : 0$	11/11/--/y
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)			
FP Divide Single	div.s FR	$F[fd] = F[fs] / F[ft]$	11/10/--/3
FP Divide	div.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} / \{F[ft], F[ft+1]\}$	11/11/--/3
FP Multiply Single	mul.s FR	$F[fd] = F[fs] * F[ft]$	11/10/--/2
FP Multiply	mul.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} * \{F[ft], F[ft+1]\}$	11/11/--/2
FP Subtract Single	sub.s FR	$F[fd] = F[fs] - F[ft]$	11/10/--/1
FP Subtract	sub.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} - \{F[ft], F[ft+1]\}$	11/11/--/1
Load FP Single	lwc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 31/--/--/0
Load FP	ldc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}];$ $F[rt+1] = M[R[rs] + \text{SignExtImm} + 4]$	(2) 35/--/--/0
Move From Hi	mfmhi R	$R[rd] = Hi$	0/--/--/10
Move From Lo	mfmlo R	$R[rd] = Lo$	0/--/--/12
Move From Control	mfc0 R	$R[rd] = CR[rs]$	10/0/--/0
Multiply	mult R	$\{Hi, Lo\} = R[rs] * R[rt]$	0/--/--/18
Multiply Unsigned	multu R	$\{Hi, Lo\} = R[rs] * R[rt]$	(6) 0/--/--/19
Shift Right Arith.	sra R	$R[rd] = R[rt] \gg \text{shamt}$	0/--/--/3
Store FP Single	swc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt]$	(2) 39/--/--/0
Store FP	sdc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt];$ $M[R[rs] + \text{SignExtImm} + 4] = F[rt+1]$	(2) 3d/--/--/0

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31	26 25	21 20	16 15	11 10	6 5
	0					
FI	opcode	fmt	ft	immediate		
	31	26 25	21 20	16 15		
	0					

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if($R[rs] < R[rt]$) $PC = \text{Label}$
Branch Greater Than	bgt	if($R[rs] > R[rt]$) $PC = \text{Label}$
Branch Less Than or Equal	btle	if($R[rs] \leq R[rt]$) $PC = \text{Label}$
Branch Greater Than or Equal	bge	if($R[rs] \geq R[rt]$) $PC = \text{Label}$
Load Immediate	li	$R[rd] = \text{immediate}$
Move	move	$R[rd] = R[rs]$

REGISTER NAME, NUMBER, USE, CALL CONVENTION

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

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.

Nhóm lệnh logic:

and, andi

nor

or, ori

sll, srl

7. Lệnh *and*

And *and* R $R[rd] = R[rs] \& R[rt]$ 0 / 24_{hex}

⇒ Lệnh này thuộc dạng R-format, có opcode là 0 và trường funct là 24_{hex}

Syntax (cú pháp):

<i>and rd, rs, rt</i>	0	rs	rt	rd	0	0x24
	6	5	5	5	5	6

Ý nghĩa: $R[rd] = R[rs] \& R[rt]$

Thực hiện *and* từng bit giá trị của thanh ghi rs và rt với nhau, kết quả lưu vào thanh ghi rd

Ví dụ:

and \$t0, \$t1, \$t2

Giả sử giá trị đang chứa trong thanh ghi \$t1 là 0x12345678, giá trị đang chứa trong thanh ghi \$t2 là 0x0000000f thì

Kết quả: sau lệnh add trên, giá trị trong thanh ghi \$t0 là 0x00000008.

8. Lệnh *andi*

And Immediate *andi* I $R[rt] = R[rs] \& \text{ZeroExtImm}$ (3) c_{hex}

⇒ Lệnh này thuộc dạng I-format, có opcode là 0xc

Syntax (cú pháp):

<i>andi rt, rs, imm</i>	0xc	rs	rt	imm
	6	5	5	16

Ý nghĩa: $R[rt] = R[rs] \& \text{ZeroExtImm}$

Lệnh này thực hiện *and* từng bit giá trị thanh ghi *rs* và một số tức thời. Số tức thời đang là số 16 bits, mở rộng thành số 32 bits theo kiểu ZeroExtImm, tức 16 bits nữa cao còn thiếu sẽ điền 0 vào. Sau đó thực hiện *and* từng bit giá trị của thanh ghi *rs* và số tức thời đã được mở rộng thành 32 bits với nhau, kết quả lưu vào thanh ghi *rd*

Ví dụ:

a) *andi \$t0, \$t1, 0xffff*

Giả sử giá trị đang chứa trong thanh ghi \$t1 là 0x12345678.

Kết quả: sau lệnh trên, giá trị thanh ghi \$t0 = 0x00005678

Quy trình lệnh thực hiện:

Số tức thời: 0xffff = 1111 1111 1111 1111₍₂₎

ZeroExtImm(0xffff) = 0000 0000 0000 0000 1111 1111 1111 1111₍₂₎

\$t0 = \$t1 & ZeroExtImm(0xffff) = 0x00005678

b) *andi \$t0, \$t1, -3*

Vấn đề đặt ra là imm ở đây có thể là số âm không?

- Một số simulator chấp nhận imm có thể là âm, ví dụ số -3 trên sẽ chuyển sang thành bù 2 của số 16 bits, sau đó mở rộng theo kiểu ZeroExtImm
- Một số simulator không chấp nhận imm có thể là âm, ví dụ số -3 trên đưa vào sẽ báo lỗi

⇒ Trong phạm vi môn học, chọn trường hợp thứ 2, không chấp nhận imm là âm

9. Các lệnh or, ori, nor

or và *nor* cách viết tương tự như *and*, nhưng thay vì thực hiện phép toán *and*, 2 lệnh này sẽ thực hiện phép toán *or* hoặc *nor* cho từng bit trong 2 thanh ghi, kết quả lưu vào thanh ghi thứ 3

ori tương tự như *andi*, thực hiện *or* một thanh ghi và một số tức thời 16 bits được mở rộng ZeroExtImm thành 32 bits

10. Lệnh sll/srl



sll

Shift Left Logical *sll* R $R[rd] = R[rt] \ll \text{shamt}$ 0 / 00_{hex}

⇒ lệnh dịch trái số học, thuộc nhóm lệnh R, có opcode là 0 và funct 00_{hex}

Shift Right Logical *srl* R $R[rd] = R[rt] \gg \text{shamt}$ 0 / 02_{hex}

⇒ lệnh dịch phải số học, thuộc nhóm lệnh R, opcode là 0 và funct là 02_{hex}

Syntax (cú pháp):

sll rd, rt, shamt	0	rs	rt	rd	shamt	0
	6	5	5	5	5	6

srl rd, rt, shamt	0	rs	rt	rd	shamt	2
	6	5	5	5	5	6

Ý nghĩa:

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

Thanh ghi rt dịch trái shamt bit và kết quả lưu vào thanh ghi rd (‘<< ‘ là ký hiệu của phép toán dịch trái)

srl: $R[rd] = R[rt] \gg \text{shamt}$

Thanh ghi rt dịch phải shamt bit và kết quả lưu vào thanh ghi rd (‘>> ‘ là ký hiệu của phép toán dịch phải)

Ví dụ:

a. *sll \$t0, \$t1, 5*

Giả sử thanh ghi \$t1 đang chứa giá trị 0x12345678

Kết quả: sau lệnh trên, thanh ghi \$t0 = 0x468ACF00

Quy trình lệnh thực hiện: lệnh trên dịch trái 5 bit thanh ghi \$t1

$$\$t1 = 0x12345678 = 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111\ 1000_{(2)}$$

$$\text{Dịch trái 5 bit } \$t1 = 0100\ 0110\ 1000\ 1010\ 1100\ 1111\ 0000\ 0000_{(2)} = 0x468ACF00$$

$$\text{Vậy kết quả thanh ghi } \$t0 = 0x468ACF00$$

b. *srl \$t0, \$t1, 5*

Giả sử thanh ghi \$t1 đang chứa giá trị 0x12345678

Kết quả: sau lệnh trên, thanh ghi \$t0 = 0x91A2B3

Quy trình lệnh thực hiện: lệnh trên dịch phải 5 bit thanh ghi \$t1

$$\$t1 = 0x12345678 = 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111\ 1000_{(2)}$$

$$\text{Dịch phải 5 bit } \$t1 = 0000\ 0000\ 1001\ 0001\ 1010\ 0010\ 1011\ 0011_{(2)} = 0x91A2B3$$

$$\text{Vậy kết quả thanh ghi } \$t0 = 0x91A2B3$$

Tổng kết:

Các lệnh trong nhóm:

and, andi

nor

or, ori

sll, srl

Trong cột ghi chú ở bảng 1, chú ý chỉ có 2 lệnh *andi* và *ori* có ghi chú (3) – ghi chú ‘zeroExtImm’, tức các lệnh làm việc với số tức thời trong nhóm này khi mở rộng từ số tức thời 16 bits sang số 32 bits thì dùng zeroExtImm, không phải SignExtImm như nhóm lệnh số học.

C. Nhóm lệnh Nhánh/Nhảy (Branch/Jump)

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data

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 _{hex}
Add Immediate	addi I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 _{hex}
Add Imm. Unsigned	addiu I	$R[rt] = R[rs] + \text{SignExtImm}$	(2) 9 _{hex}
Add Unsigned	addu R	$R[rd] = R[rs] + R[rt]$	0 / 21 _{hex}
And	and R	$R[rd] = R[rs] \& R[rt]$	0 / 24 _{hex}
And Immediate	andi I	$R[rt] = R[rs] \& \text{ZeroExtImm}$	(3) c _{hex}
Branch On Equal	beq I	if $R[rs] == R[rt]$ $PC = PC + 4 + \text{BranchAddr}$	(4) 4 _{hex}
Branch On Not Equal	bne I	if $R[rs] != R[rt]$ $PC = PC + 4 + \text{BranchAddr}$	(4) 5 _{hex}
Jump	j J	$PC = \text{JumpAddr}$	(5) 2 _{hex}
Jump And Link	jal J	$R[31] = PC + 8; PC = \text{JumpAddr}$	(5) 3 _{hex}
Jump Register	jr R	$PC = R[rs]$	0 / 08 _{hex}
Load Byte Unsigned	lbu I	$R[rt] = \{24'b0, M[R[rs]] + \text{SignExtImm}(7:0)\}$	(2) 24 _{hex}
Load Halfword Unsigned	lhu I	$R[rt] = \{16'b0, M[R[rs]] + \text{SignExtImm}(15:0)\}$	(2) 25 _{hex}
Load Linked	ll I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2,7) 30 _{hex}
Load Upper Imm.	lui I	$R[rt] = \{\text{imm}, 16'b0\}$	f _{hex}
Load Word	lw I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 23 _{hex}
Nor	nor R	$R[rd] = \sim (R[rs] R[rt])$	0 / 27 _{hex}
Or	or R	$R[rd] = R[rs] R[rt]$	0 / 25 _{hex}
Or Immediate	ori I	$R[rt] = R[rs] \text{ZeroExtImm}$	(3) d _{hex}
Set Less Than	slt R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	0 / 24 _{hex}
Set Less Than Imm.	slti I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2) a _{hex}
Set Less Than Imm. Unsigned	sltiu I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2,6) b _{hex}
Set Less Than Unsig.	sltu R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	(6) 0 / 2b _{hex}
Shift Left Logical	sll R	$R[rd] = R[rt] << \text{shamt}$	0 / 00 _{hex}
Shift Right Logical	srl R	$R[rd] = R[rt] >>> \text{shamt}$	0 / 02 _{hex}
Store Byte	sb I	$M[R[rs] + \text{SignExtImm}(7:0)] = R[rt](7:0)$	(2) 28 _{hex}
Store Conditional	sc I	$M[R[rs] + \text{SignExtImm}] = R[rt];$ $R[rt] = (\text{atomic}) ? 1 : 0$	(2,7) 38 _{hex}
Store Halfword	sh I	$M[R[rs] + \text{SignExtImm}(15:0)] = R[rt](15:0)$	(2) 29 _{hex}
Store Word	sw I	$M[R[rs] + \text{SignExtImm}] = R[rt]$	(2) 2b _{hex}
Subtract	sub R	$R[rd] = R[rs] - R[rt]$	(1) 0 / 22 _{hex}
Subtract Unsigned	subu R	$R[rd] = R[rs] - R[rt]$	0 / 23 _{hex}

- (1) May cause overflow exception
 (2) $\text{SignExtImm} = \{16\{\text{immediate}[15]\}, \text{immediate}\}$
 (3) $\text{ZeroExtImm} = \{16\{1b'0\}, \text{immediate}\}$
 (4) $\text{BranchAddr} = \{14\{\text{immediate}[15]\}, \text{immediate}, 2'b0\}$
 (5) $\text{JumpAddr} = \{PC + 4[31:28], \text{address}, 2'b0\}$
 (6) Operands considered unsigned numbers (vs. 2's comp.)
 (7) Atomic test&set pair; $R[rt] = 1$ if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31	26 25	21 20	16 15	11 10	6 5
	0					
I	opcode	rs	rt	immediate		
	31	26 25	21 20	16 15		
	0					
J	opcode	address				
	31	26 25				
	0					

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.

ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FUNCT (Hex)
Branch On FP True	bclt FI	if $(FPcond) PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/1/
Branch On FP False	bclt FI	if $(!FPcond) PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/0/
Divide	div R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	0/
Divide Unsigned	divu R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	(6) 0/
FP Add Single	add.s FR	$F[fd] = F[fs] + F[ft]$	11/10/
FP Add Double	add.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} + \{F[ft], F[ft+1]\}$	11/11/
FP Compare Single	c.x.s* FR	$FPcond = (F[fs] op F[ft]) ? 1 : 0$	11/10/
FP Compare Double	c.x.d* FR	$FPcond = (\{F[fs], F[fs+1]\} op \{F[ft], F[ft+1]\}) ? 1 : 0$	11/11/
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)			
FP Divide Single	div.s FR	$F[fd] = F[fs] / F[ft]$	11/10/
FP Divide Double	div.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} / \{F[ft], F[ft+1]\}$	11/11/
FP Multiply Single	mul.s FR	$F[fd] = F[fs] * F[ft]$	11/10/
FP Multiply Double	mul.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} * \{F[ft], F[ft+1]\}$	11/11/
FP Subtract Single	sub.s FR	$F[fd] = F[fs] - F[ft]$	11/10/
FP Subtract Double	sub.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} - \{F[ft], F[ft+1]\}$	11/11/
Load FP Single	lwc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 31/
Load FP Double	ldc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}];$ $F[rt+1] = M[R[rs] + \text{SignExtImm} + 4]$	(2) 35/
Move From Hi	mghi R	$R[rd] = Hi$	0 /
Move From Lo	mfl0 R	$R[rd] = Lo$	0 /
Move From Control	mfc0 R	$R[rd] = CR[rs]$	10 /
Multiply	mult R	$\{Hi, Lo\} = R[rs] * R[rt]$	0/
Multiply Unsigned	multu R	$\{Hi, Lo\} = R[rs] * R[rt]$	(6) 0/
Shift Right Arith.	sra R	$R[rd] = R[rt] >> \text{shamt}$	0/
Store FP Single	swc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt]$	(2) 39/
Store FP Double	sdc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt];$ $M[R[rs] + \text{SignExtImm} + 4] = F[rt+1]$	(2) 3d/

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31	26 25	21 20	16 15	11 10	6 5
	0					
FI	opcode	fmt	ft	immediate		
	31	26 25	21 20	16 15		
	0					

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if $R[rs] < R[rt]$ PC = Label
Branch Greater Than	bgt	if $R[rs] > R[rt]$ PC = Label
Branch Less Than or Equal	b1e	if $R[rs] \leq R[rt]$ PC = Label
Branch Greater Than or Equal	bge	if $R[rs] \geq R[rt]$ PC = Label
Load Immediate	li	$R[rd] = \text{immediate}$
Move	move	$R[rd] = R[rs]$

REGISTER NAME, NUMBER, USE, CALL CONVENTION

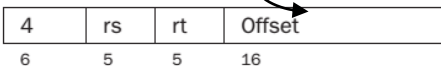
NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$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

<p>được thực hiện là lệnh “<i>or \$t1, \$t2, \$t3</i>”. Sau khi “<i>or</i>” thực hiện xong thì luồng lệnh theo sau đó sẽ được thực hiện (ví dụ lệnh <i>sub</i> tiếp theo sau sẽ được thực hiện)</p> <ul style="list-style-type: none"> 2 thanh ghi này không bằng nhau, thì lệnh tiếp theo được thực hiện là lệnh “<i>add \$s0, \$t3, \$t4</i>”. Sau khi “<i>add</i>” thực hiện xong thì luồng lệnh theo sau đó sẽ được thực hiện (ví dụ chuỗi các lệnh <i>addi, or, sub</i> tiếp theo sau sẽ được thực hiện) 	<ul style="list-style-type: none"> 2 thanh ghi này bằng nhau, thì lệnh tiếp theo được thực hiện là lệnh cách beq 2 lệnh, tức là lệnh “<i>or \$t1, \$t2, \$t3</i>”. Sau khi “<i>or</i>” thực hiện xong thì luồng lệnh theo sau đó sẽ được thực hiện (ví dụ lệnh <i>sub</i> tiếp theo sau sẽ được thực hiện) 2 thanh ghi này không bằng nhau, thì lệnh tiếp theo được thực hiện là lệnh “<i>add \$s0, \$t3, \$t4</i>”. Sau khi “<i>add</i>” thực hiện xong thì luồng lệnh theo sau đó sẽ được thực hiện (ví dụ chuỗi các lệnh <i>addi, or, sub</i> tiếp theo sau sẽ được thực hiện)
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

⇒ Khi lập trình, ta có thể sử dụng một trong 2 cách như 2 ví dụ trên. Nhưng thực tế lệnh mà bộ xử lý hiểu là lệnh như ví dụ 2. Khi ta lập trình theo như ví dụ 1 thì lệnh cũng sẽ được chuyển về như ví dụ 2 để gởi cho bộ xử lý.

Như vậy beq chuẩn theo dạng:

beq rs, rt, label/imm



Số tức thời label/imm này chính là số lệnh mà lệnh beq hiện tại cách lệnh sẽ nhảy tới bao nhiêu, được lưu vào 16 bits của offset

Ý nghĩa:

if(R[rs] == R[rt]) PC = PC + 4 + BranchAddr

⇒ Nếu giá trị thanh ghi *rs* bằng *rt* thì chương trình nhảy tới lệnh mà cách lệnh beq đang xét là **imm** lệnh, tức địa chỉ con trỏ/thanh ghi PC sẽ chuyển thành **PC + 4 + imm*4** (đối với trường hợp mỗi lệnh lưu trong một word 4 bytes) = PC + 4 + BranchAddr

BranchAddr = imm * 4 (đối với trường hợp mỗi lệnh lưu trong một word 4 bytes)

12. Lệnh bne:

Cách viết tương tự như beq, nhưng ý nghĩa trái ngược:

- beq: kiểm tra nếu 2 thanh ghi **bằng nhau** thì nhảy đến lệnh mong muốn
- bne: kiểm tra nếu 2 thanh ghi **không bằng nhau** thì nhảy đến lệnh mong muốn

13. Lệnh bge/bgt/ble/blt

Ngoài ra, còn một số lệnh so sánh và nhảy khác (trong bảng pseudoInstruction Set)

bge \$t1, \$t2, label → Nhảy tới label thực hiện lệnh nếu thanh ghi $\$t1 \geq \$t2$

bgt \$t1, \$t2, label → Nhảy tới label thực hiện lệnh nếu thanh ghi $\$t1 > \$t2$

ble \$t1, \$t2, label → Nhảy tới label thực hiện lệnh nếu thanh ghi $\$t1 \leq \$t2$

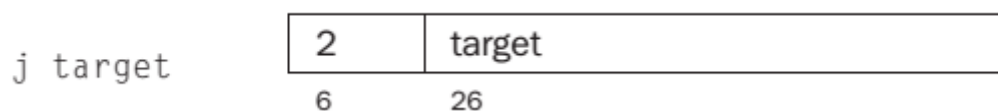
blt \$t1, \$t2, label → Nhảy tới label thực hiện lệnh nếu thanh ghi $\$t1 < \$t2$

14. Lệnh *j* – lệnh nhảy không điều kiện

Jump *j* J PC=JumpAddr (5) 2_{hex}

⇒ Lệnh thuộc nhóm lệnh J-format, có opcode 2_{hex}

Syntax (cú pháp):



Ví dụ:

Chạy đoạn lệnh sau trên MARS 4.4:

```

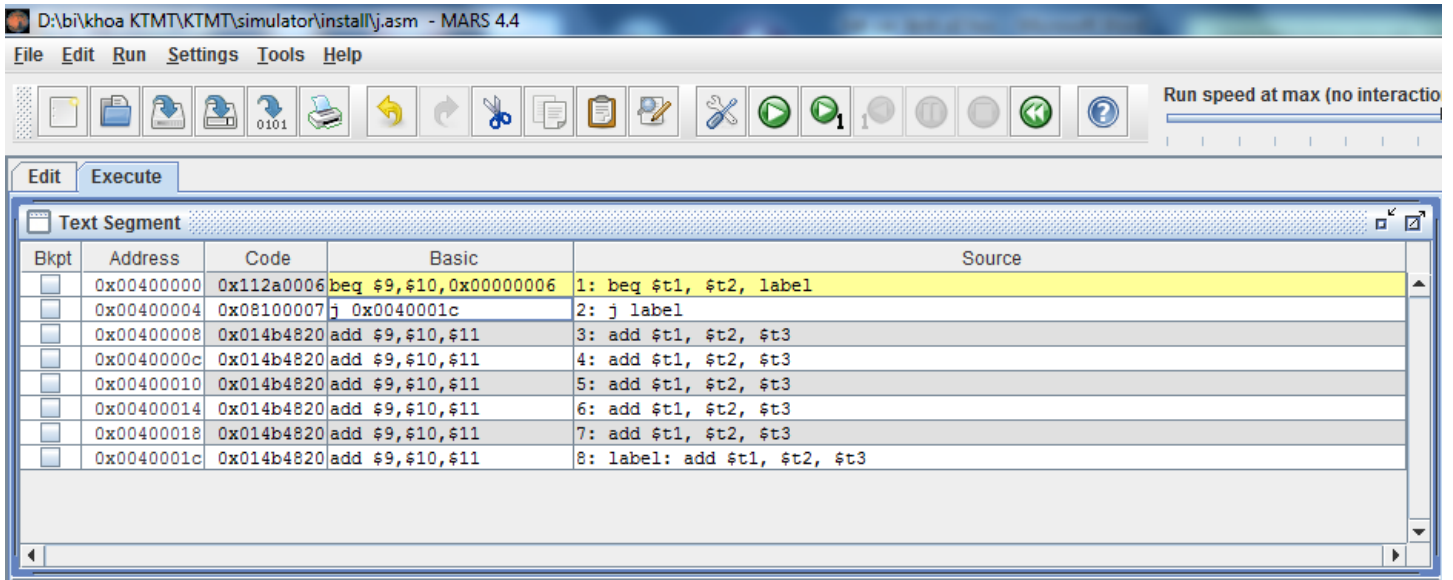
beq $t1, $t2, label
j label
add $t1, $t2, $t3
add $t1, $t2, $t3
add $t1, $t2, $t3
add $t1, $t2, $t3
add $t1, $t2, $t3
add $t1, $t2, $t3
label: add $t1, $t2, $t3

```

Kết quả: sau khi tới lệnh “*j label*”, chương trình sẽ nhảy đến lệnh *add* cuối cùng sẽ để thực tiếp

Quá trình thực hiện lệnh:

Khi biên dịch đoạn lệnh trên trong MARS 4.4, ta được hình như sau:



Cột Source là cột chứa các lệnh từ chương trình mà ta viết, cột này có thể chứa một số lệnh giả (pseudo-code); Cột basic là cột chứa các lệnh mà thực sự processor sẽ chuẩn bị chạy.

Lệnh “j label” khi thật sự chạy sẽ chuyển thành “j 0x0040001c”, số này được tạo ra như thế nào?

- ⇒ Lệnh “j label” cách lệnh chứa nhãn ‘label’ 5 lệnh, vì vậy lệnh này cách lệnh cần nhảy tới $5 \times 4 = 20$ byte
- Địa chỉ của lệnh cần nhảy tới = PC + 4 + số byte cách lệnh sẽ nhảy tới
- PC của lệnh “j label” = 0x00400004
- số byte cách lệnh sẽ nhảy tới = 20
- Vậy Địa chỉ của lệnh cần nhảy tới = $0x00400004 + 4_{10} + 20_{10}$
- = 0x0040001C

Chú ý: Lệnh này chỉ cho nhảy trong phạm vi 256 MB = 2^{28} byte

Lưu ý lệnh này khác với *beq/bne* là target trong syntax phải là nhãn của lệnh cần nhảy tới, không thể gán địa chỉ trực tiếp của lệnh cần nhảy tới, tức không thể gán một số vào đây được. Địa chỉ của lệnh cần nhảy tới sẽ do compiler tính toán và gởi cho processor trước khi thực hiện.

15. Lệnh jal

Jump And Link jal J R[31]=PC+8;PC=JumpAddr (5) 3_{hex}

⇒ Lệnh này thuộc nhóm J-format, có opcode là 3_{hex}

Syntax (cú pháp):

jal target

3	target
6	26

Sửa lại thành:

R[31] = PC + 4

Ý nghĩa:

$$R[31] = PC + 4; PC = \text{JumpAddr}$$

⇒ Lệnh này thực hiện việc nhảy giống y như lệnh *j*; nhưng địa chỉ của lệnh ngay sau lệnh *jal* được lưu vào thanh ghi 31 (thanh ghi *ra*) trước khi nhảy

Lệnh này nhằm phục vụ cho việc gọi một hàm con. Theo quy tắc, sau khi hàm con được gọi và thực hiện xong sẽ quay trở về chương trình chính. Do đó việc lưu lại địa chỉ của lệnh ngay sau *jal* vào *ra* nhằm lưu lại địa chỉ quay về này

Ví dụ:**Chạy đoạn lệnh sau trên MARS 4.4:**

```
jal label
add $t1, $t2, $t3
add $t1, $t2, $t3
add $t1, $t2, $t3
add $t1, $t2, $t3
add $t1, $t2, $t3
label: add $t1, $t2, $t3
```

Khi biên dịch:

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x0c100006	jal 0x00400018	1: jal label
<input type="checkbox"/>	0x00400004	0x014b4820	add \$9,\$10,\$11	2: add \$t1, \$t2, \$t3
<input type="checkbox"/>	0x00400008	0x014b4820	add \$9,\$10,\$11	3: add \$t1, \$t2, \$t3
<input type="checkbox"/>	0x0040000c	0x014b4820	add \$9,\$10,\$11	4: add \$t1, \$t2, \$t3
<input type="checkbox"/>	0x00400010	0x014b4820	add \$9,\$10,\$11	5: add \$t1, \$t2, \$t3
<input type="checkbox"/>	0x00400014	0x014b4820	add \$9,\$10,\$11	6: add \$t1, \$t2, \$t3
<input type="checkbox"/>	0x00400018	0x014b4820	add \$9,\$10,\$11	7: label: add \$t1, \$t2, \$t3

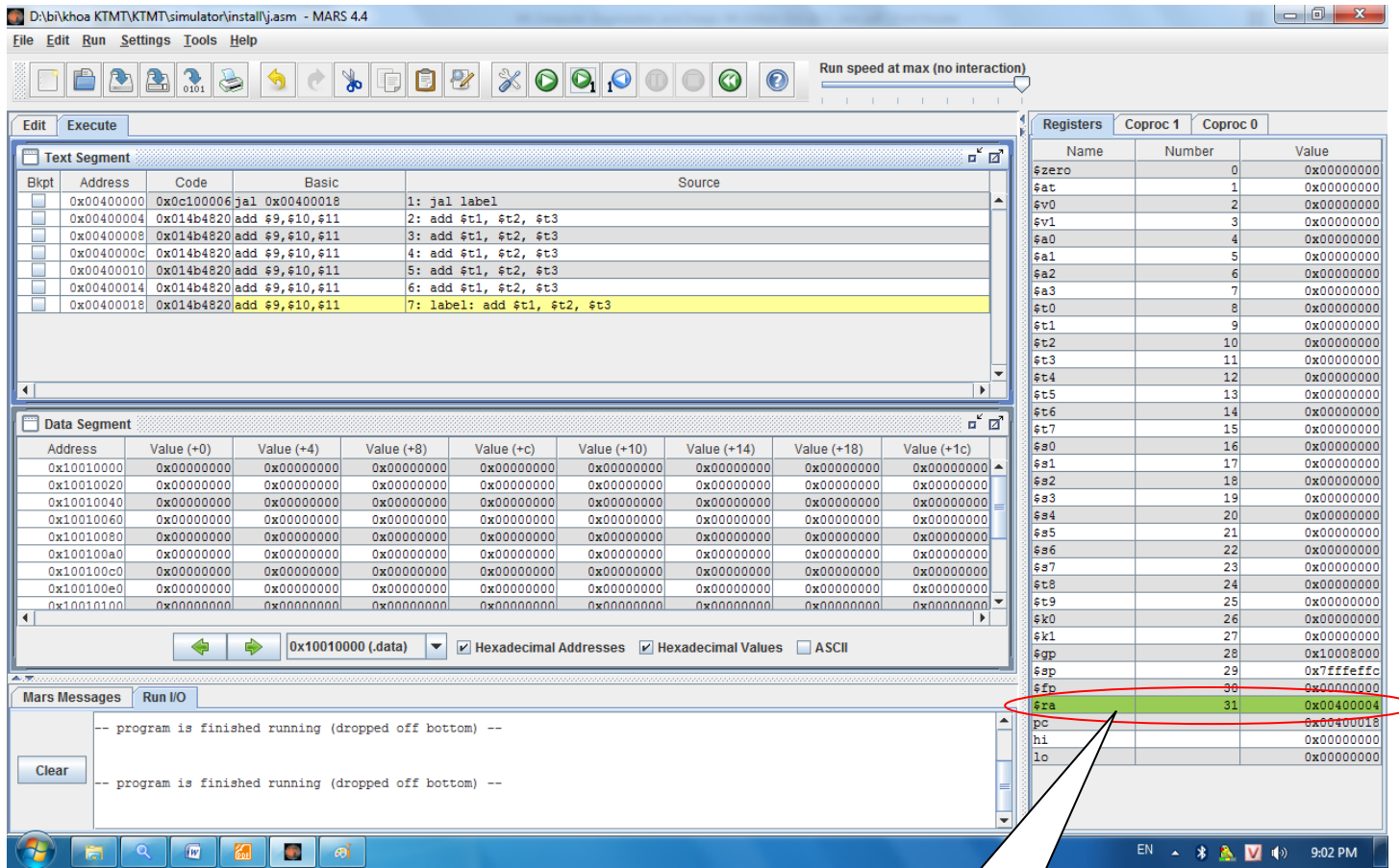
Ta thấy:

“jal label” được chuyển thành “jal 0x00400018” (giá trị ‘target’ trong format lệnh lúc này sẽ bằng 0x00400018) trước khi gửi cho processor

Địa chỉ của lệnh này hiện tại là 0x00400000

Khi chương trình chạy:

- $PC = 0x00400000$
- Đầu tiên, lệnh theo sau *jal* được lưu lại vào thanh ghi *ra* $\rightarrow ra = PC + 4 = 0x00400004$
- Sau đó lệnh sẽ nhảy đến lệnh thứ 7, tức PC đang bằng $0x00400000$ sẽ chuyển thành $PC = \text{target/JumpAddr} = 0x00400018$



\$ra = 0x00400004

16. Lệnh jr

Jump Register jr R PC=R[rs] 0 / 08_{hex}

⇒ Lệnh thuộc nhóm lệnh R (nhưng khá đặc biệt - chỉ quan tâm vùng thanh ghi rs), có opcode 0 và funct 08_{hex}

Syntax (cú pháp):

jr rs	0	rs	0	8
	6	5	15	6

Ý nghĩa: PC = R[rs]

⇒ Nhảy tới lệnh có địa chỉ đang chứa trong thanh ghi rs

Ví dụ:

Chạy đoạn chương trình sau trong MARS

```
jr $s1
add $t1, $t2, $t3
addi $t1, $t2, 3
or $t1, $t2, $t3
xor $t1, $t2, $t3
addi $t1, $t2, 1
sub $t1, $t2, $t3
```

Khi chương trình được biên dịch:

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x02200008	jr \$17	1: jr \$s1
<input type="checkbox"/>	0x00400004	0x014b4820	add \$9,\$10,\$11	2: add \$t1, \$t2, \$t3
<input type="checkbox"/>	0x00400008	0x21490003	addi \$9,\$10,0x00000003	3: addi \$t1, \$t2, 3
<input type="checkbox"/>	0x0040000c	0x014b4825	or \$9,\$10,\$11	4: or \$t1, \$t2, \$t3
<input type="checkbox"/>	0x00400010	0x014b4826	xor \$9,\$10,\$11	5: xor \$t1, \$t2, \$t3
<input type="checkbox"/>	0x00400014	0x21490001	addi \$9,\$10,0x00000001	6: addi \$t1, \$t2, 1
<input type="checkbox"/>	0x00400018	0x014b4822	sub \$9,\$10,\$11	7: sub \$t1, \$t2, \$t3

Giả sử lúc này giá trị trong thanh ghi \$s1 = 0x0040000c, lệnh *or* sẽ được thực hiện ngay sau *jr* khi chương trình chạy.

Giả sử lúc này giá trị trong thanh ghi \$s1 = 0x00400018, lệnh *sub* sẽ được thực hiện ngay sau *jr* khi chương trình chạy.

Tổng kết

Nhóm lệnh:

beq, bne

j, jal, jr

Xét nhóm 4 lệnh: *beq, bne, j, jal* có cấu trúc như ví dụ sau:

beq/bne \$t1, \$t2, số_16_bits/label

j/jal label

Đứng trên phương diện người lập trình, ta chỉ cần quan tâm:

Lệnh mà *beq/bne* nhảy tới có thể được đưa vào bằng cách gán “*label*” hoặc dùng “*số_16_bits*”, “*số_16_bits*” trong *beq/bne* là số lệnh cách lệnh sẽ nhảy tới bao nhiêu.

Lệnh mà *j/jal* nhảy tới chỉ có thể được đưa vào bằng cách gán “*label*”

Đứng trên phương diện thiết kế processor, như thế nào processor tính toán ra địa chỉ của lệnh tiếp theo cần nhảy tới dựa vào các label hoặc các số 16_bits này?

Nhìn lại bảng 1 ta thấy *beq/bne* có ghi chú (4), còn nhóm *j/jal* có ghi chú (5), trong khi *jr* không có ghi chú gì cả:

(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }

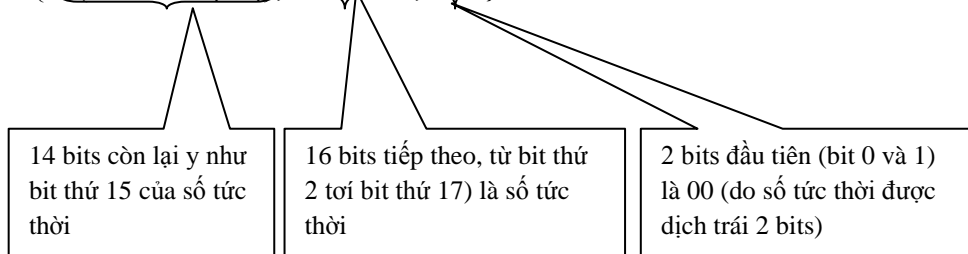
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }

- ✓ Trong cột ý nghĩa của lệnh *beq/bne* ta thấy: $PC = PC + 4 + \text{BranchAddr}$
- ✓ Trong cột ý nghĩa của lệnh *j/jal* ta thấy: $PC = \text{JumpAddr}$

Như đã trình bày trong phần trước, lệnh cần nhảy tới trong *beq/bne* có thể đưa vào là “label” hoặc số tức thời 16 bits - chỉ lệnh sẽ nhảy đến cách lệnh hiện tại bao nhiêu lệnh. Nếu người lập trình đưa vào bằng “label”, thì compiler sẽ tự tính toán ra số lệnh sẽ nhảy tới cách lệnh hiện tại bao nhiêu. Tóm lại:

số tức thời trong lệnh *beq/bne* là số lệnh cách lệnh cần nhảy tới bao nhiêu, nên BranchAddr được tính bằng cách lấy số tức thời 16 bits này nhân 4, tức dịch trái 2 bits rồi mở rộng theo kiểu có dấu thành số 32 bits, sau đó được cộng với $PC + 4$. Cách viết trong ghi chú (4) tương tự ý nghĩa này

{ 14{immediate[15]}, immediate, 2'b0 } : là số 32 bits

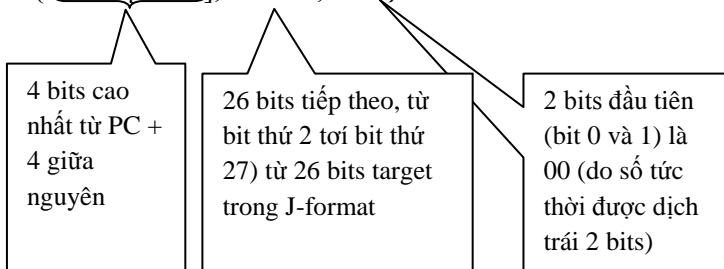


Trong khi đó đối với lệnh *j/jal*, người lập trình đưa vào nhãn của địa chỉ cần nhảy tới, compiler sẽ tự tính toán ra lệnh cần nhảy tới cách lệnh hiện tại bao nhiêu lệnh. Địa chỉ của lệnh cần nhảy tới (JumpAddr) sẽ được tính bằng cách lấy số lệnh này nhân 4 (dịch trái 2 bits) và cộng với $PC + 4$. Tuy nhiên, vì lệnh này chỉ cho phép nhảy trong phạm vi $256\text{MB} = 2^{28}$ bytes, tức 4 bit cao nhất trong $PC + 4$ không đổi.

$\text{JumpAdd} = PC + 4 + \text{số lệnh cách lệnh sẽ nhảy tới} * 4$

Cách viết trong ghi chú (4) tương tự ý nghĩa này

{ PC + 4[31:28], address, 2'b0 } : là số 32 bits



D. Nhóm lệnh memory-instruction

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data

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 _{hex}
Add Immediate	addi I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 _{hex}
Add Imm. Unsigned	addiu I	$R[rt] = R[rs] + \text{SignExtImm}$	(2) 9 _{hex}
Add Unsigned	addu R	$R[rd] = R[rs] + R[rt]$	0 / 21 _{hex}
And	and R	$R[rd] = R[rs] \& R[rt]$	0 / 24 _{hex}
And Immediate	andi I	$R[rt] = R[rs] \& \text{ZeroExtImm}$	(3) c _{hex}
Branch On Equal	beq I	if $R[rs] == R[rt]$ $PC = PC + 4 + \text{BranchAddr}$	(4) 4 _{hex}
Branch On Not Equal	bne I	if $R[rs] != R[rt]$ $PC = PC + 4 + \text{BranchAddr}$	(4) 5 _{hex}
Jump	j J	$PC = \text{JumpAddr}$	(5) 2 _{hex}
Jump And Link	jal J	$R[31] = PC + 8; PC = \text{JumpAddr}$	(5) 3 _{hex}
Jump Register	jrr R	$PC = R[rs]$	0 / 08 _{hex}
Load Byte Unsigned	lbu I	$R[rt] = \{24'b0, M[R[rs]] + \text{SignExtImm}(7:0)\}$	(2) 24 _{hex}
Load Halfword Unsigned	lhu I	$R[rt] = \{16'b0, M[R[rs]] + \text{SignExtImm}(15:0)\}$	(2) 25 _{hex}
Load Linked	ll I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2,7) 30 _{hex}
Load Upper Imm.	lui I	$R[rt] = \{\text{imm}, 16'b0\}$	f _{hex}
Load Word	lw I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 23 _{hex}
Nor	nor R	$R[rd] = \sim (R[rs] R[rt])$	0 / 27 _{hex}
Or	or R	$R[rd] = R[rs] R[rt]$	0 / 25 _{hex}
Or Immediate	ori I	$R[rt] = R[rs] \text{ZeroExtImm}$	(3) d _{hex}
Set Less Than	slt R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	0 / 24 _{hex}
Set Less Than Imm.	slti I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2) a _{hex}
Set Less Than Imm. Unsigned	sltiu I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2,6) b _{hex}
Set Less Than Unsig.	sltu R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	(6) 0 / 2b _{hex}
Shift Left Logical	sll R	$R[rd] = R[rt] \ll \text{shamt}$	0 / 00 _{hex}
Shift Right Logical	srl R	$R[rd] = R[rt] \gg \text{shamt}$	0 / 02 _{hex}
Store Byte	sb I	$M[R[rs] + \text{SignExtImm}(7:0)] = R[rt](7:0)$	(2) 28 _{hex}
Store Conditional	sc I	$M[R[rs] + \text{SignExtImm}] = R[rt];$ $R[rt] = (\text{atomic}) ? 1 : 0$	(2,7) 38 _{hex}
Store Halfword	sh I	$M[R[rs] + \text{SignExtImm}(15:0)] = R[rt](15:0)$	(2) 29 _{hex}
Store Word	sw I	$M[R[rs] + \text{SignExtImm}] = R[rt]$	(2) 2b _{hex}
Subtract	sub R	$R[rd] = R[rs] - R[rt]$	(1) 0 / 22 _{hex}
Subtract Unsigned	subu R	$R[rd] = R[rs] - R[rt]$	0 / 23 _{hex}

(1) May cause overflow exception
 (2) SignExtImm = { 16{immediate[15]}, immediate }
 (3) ZeroExtImm = { 16{1b'0}, immediate }
 (4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
 (5) JumpAddr = { PC+4[31:28], address, 2'b0 }
 (6) Operands considered unsigned numbers (vs. 2's comp.)
 (7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode		rs		rt		rd		shamt		funct	
	31	26 25	21 20		16 15		11 10		6 5		0	
I	opcode		rs		rt		immediate					
	31	26 25	21 20		16 15		0					
J	opcode		address									
	31	26 25	0									

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.

①



ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FUNCT (Hex)
Branch On FP True	bclt FI	if(FPcond) $PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/1/
Branch On FP False	bclt FI	if(!FPcond) $PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/0/
Divide	div R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	0/
Divide Unsigned	divu R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	(6) 0/
FP Add Single	add.s FR	$F[fd] = F[fs] + F[ft]$	11/10/
FP Add Double	add.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} + \{F[ft], F[ft+1]\}$	11/11/
FP Compare Single	c.x.s* FR	$FPcond = (F[fs] \text{ op } F[ft]) ? 1 : 0$	11/10/
FP Compare Double	c.x.d* FR	$FPcond = (\{F[fs], F[fs+1]\} \text{ op } \{F[ft], F[ft+1]\}) ? 1 : 0$	11/11/
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)			
FP Divide Single	div.s FR	$F[fd] = F[fs] / F[ft]$	11/10/
FP Divide Double	div.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} / \{F[ft], F[ft+1]\}$	11/11/
FP Multiply Single	mul.s FR	$F[fd] = F[fs] * F[ft]$	11/10/
FP Multiply Double	mul.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} * \{F[ft], F[ft+1]\}$	11/11/
FP Subtract Single	sub.s FR	$F[fd] = F[fs] - F[ft]$	11/10/
FP Subtract Double	sub.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} - \{F[ft], F[ft+1]\}$	11/11/
Load FP Single	lwc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 31/
Load FP Double	ldc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}];$ $F[rt+1] = M[R[rs] + \text{SignExtImm} + 4]$	(2) 35/
Move From Hi	mfmhi R	$R[rd] = Hi$	0/
Move From Lo	mfmlo R	$R[rd] = Lo$	0/
Move From Control	mfc0 R	$R[rd] = CR[rs]$	10/
Multiply	mult R	$\{Hi, Lo\} = R[rs] * R[rt]$	0/
Multiply Unsigned	multu R	$\{Hi, Lo\} = R[rs] * R[rt]$	(6) 0/
Shift Right Arith.	sra R	$R[rd] = R[rt] \gg \text{shamt}$	0/
Store FP Single	swc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt];$ $F[rt+1] = M[R[rs] + \text{SignExtImm} + 4] = F[rt+1]$	(2) 39/
Store FP Double	sdc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt];$ $M[R[rs] + \text{SignExtImm} + 4] = F[rt+1]$	(2) 3d/

②

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmat	ft	fs	fd	funct
	31	26 25	21 20	16 15	11 10	6 5
	0					
FI	opcode	fmat	ft	immediate		
	31	26 25	21 20	16 15		
	0					

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if $R[rs] < R[rt]$ $PC = \text{Label}$
Branch Greater Than	bgt	if $R[rs] > R[rt]$ $PC = \text{Label}$
Branch Less Than or Equal	ble	if $R[rs] \leq R[rt]$ $PC = \text{Label}$
Branch Greater Than or Equal	bge	if $R[rs] \geq R[rt]$ $PC = \text{Label}$
Load Immediate	li	$R[rd] = \text{immediate}$
Move	move	$R[rd] = R[rs]$

REGISTER NAME, NUMBER, USE, CALL CONVENTION

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

Các lệnh xét:

lbu, lhu, lui, lw

sb, sh, sw

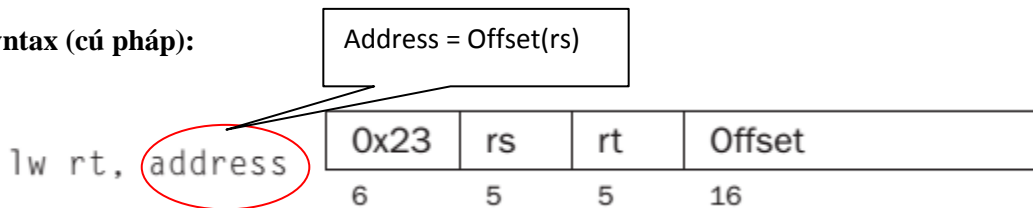
Chú ý: lệnh ll và sc bỏ qua

17. Lệnh lw

Load Word lw I $R[rt] = M[R[rs] + \text{SignExtImm}]$ (2) 23_{hex}

⇒ Lệnh thuộc dạng I-format, có opcode = 23_{hex}

Syntax (cú pháp):



Ý nghĩa: $R[rt] = M[R[rs] + \text{SignExtImm}]$

⇒ Lấy giá trị trong thanh ghi rs cộng với số tức thời đang lưu trong offset (số tức thời này này được mở rộng có dấu thành 32 bits) ta được địa chỉ của từ nhớ cần lấy dữ liệu. Dữ liệu của từ nhớ này sẽ được lấy để lưu vào thanh ghi rt

Lưu ý:

$M[X]$: là lấy giá trị của từ nhớ tại địa chỉ X

Ví dụ:

lw \$t1, 4(\$t0)

giả sử \$t0 = 0x10010000

và từ nhớ tại địa chỉ 0x10010004 có giá trị 0x12345678

⇒ Lệnh lw thực hiện việc load một từ nhớ (word) tại địa chỉ \$t0 + 4 = 0x10010004 vào thanh ghi \$t1

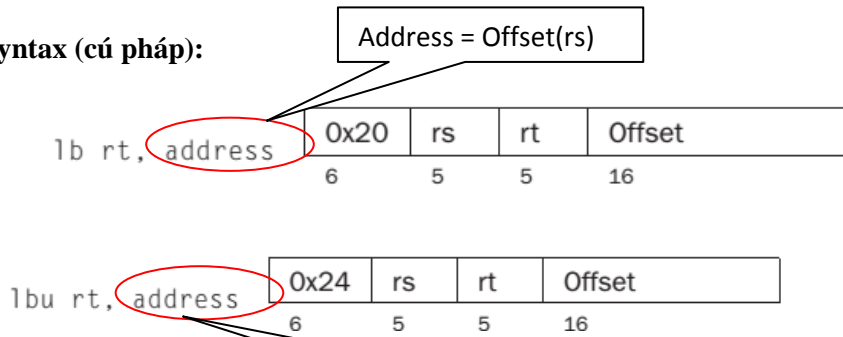
Kết quả: \$t1 = 0x12345678

18. Lệnh lbu/lb

lbu

Load Byte Unsigned lbu I $R[rt] = \{24'b0, M[R[rs] + \text{SignExtImm}](7:0)\}$ (2) 24_{hex}

⇒ Lệnh thuộc nhóm lệnh I-format, có opcode = 24_{hex}

Syntax (cú pháp):**Ý nghĩa:**

$$\text{Address} = \text{Offset}(\text{rs})$$

$$R[\text{rt}] = \{24'b0, M[R[\text{rs}] + \text{SignExtImm}](7:0)\}$$

⇒ Không giống `lw` là load hết toàn bộ 1 từ nhớ vào thanh ghi `rt`, lệnh này chỉ load về 1 byte đầu tiên trong từ nhớ vào `rt`

Vì `rt` là thanh ghi 32 bits nên 24 bit còn lại của `rt` có 2 kiểu mở rộng dấu:

- `lb`: sign-ext
- `lbu`: zero-ext

Ví dụ:

a. `lbu $t1, 8($t0)`

giả sử `$t0 = 0x10010000`

và từ nhớ tại địa chỉ `0x10010008` có giá trị `0x12345678`

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	78

⇒ Lệnh `lbu` thực hiện việc load một byte vào thanh ghi `$t1`, và byte được load là byte 0. Nhưng do thanh ghi `$t1` là 32 bits, nên 24 bits còn lại là 0

Kết quả: `$t1 = 0x00000078`

b. `lb $t1, 8($t0)`

giả sử `$t0 = 0x10010000`

và từ nhớ tại địa chỉ `0x10010008` có giá trị `0x123456f8`

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	f8

⇒ Lệnh `lb` thực hiện việc load một byte vào thanh ghi `$t1`, và byte được load là byte 0. Nhưng do thanh ghi `$t1` là 32 bits, nên 24 bits còn lại trong lệnh này **được mở rộng có dấu theo bit lớn nhất của byte được load về**

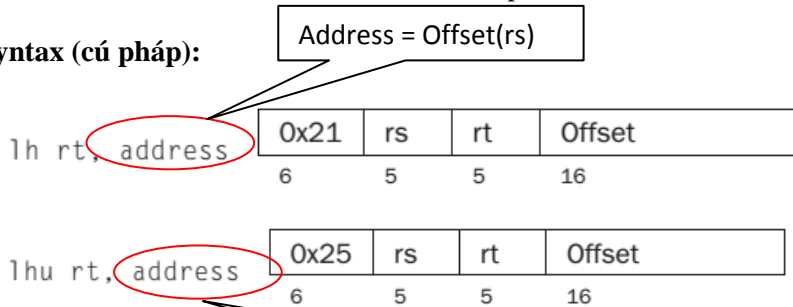
Kết quả: `$t1 = 0xfffffff8`

19. Lệnh lhu/lh

Load Halfword Unsigned lh I $R[rt] = \{16'b0, M[R[rs] + \text{SignExtImm}](15:0)\}$ (2) 25_{hex}

⇒ Lệnh thuộc nhóm lệnh I-format, có opcode 25_{hex}

Syntax (cú pháp):



Ý nghĩa:

$R[rt] = \{16'b0, M[R[rs] + \text{SignExtImm}](15:0)\}$

⇒ Lệnh này chỉ load về 2 byte đầu tiên (nửa word thấp) trong từ nhớ vào rt . Vì rt là thanh ghi 32 bits nên 16 bit còn lại của rt có 2 kiểu mở rộng dấu:

- lh : sign-ext
- lhu : zero-ext

Ví dụ:

- a. $lhu\ \$t1, 8(\$t0)$
giả sử $\$t0 = 0x10010000$
và từ nhớ tại địa chỉ $0x10010008$ có giá trị $0x12345678$

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	78

⇒ Lệnh lhu thực hiện việc load 2 byte vào thanh ghi $\$t1$, và byte được load là byte 0 và byte 1. Nhưng do thanh ghi $\$t1$ là 32 bits, nên 16 bits còn lại là 0.
Kết quả: $\$t1 = 0x00005678$

- b. $lh\ \$t1, 8(\$t0)$
giả sử $\$t0 = 0x10010000$
và từ nhớ tại địa chỉ $0x10010008$ có giá trị $0x123456f8$

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	f8

⇒ Lệnh lh thực hiện việc load 2 byte vào thanh ghi \$t1, và byte được load là byte 0 và byte 1. Nhưng do thanh ghi \$t1 là 32 bits, nên 16 bits còn lại trong lệnh này **được mở rộng có dấu theo bit lớn nhất của byte được load về**
 Kết quả: \$t1 = 0x000056f8

c. lh \$t1, 8(\$t0)

giả sử \$t0 = 0x10010000

và từ nhớ tại địa chỉ 0x10010008 có giá trị 0x12348cde

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	8c	de

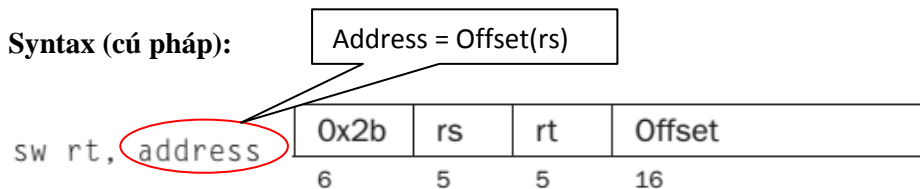
⇒ Lệnh lh thực hiện việc load 2 byte vào thanh ghi \$t1, và byte được load là byte 0 và byte 1. Nhưng do thanh ghi \$t1 là 32 bits, nên 16 bits còn lại trong lệnh này **được mở rộng có dấu theo bit lớn nhất của byte được load về**
 Kết quả: \$t1 = 0xffff8cde

20. Lệnh sw

Store Word *sw* I $M[R[rs] + \text{SignExtImm}] = R[rt]$ (2) $2b_{\text{hex}}$

⇒ Lệnh thuộc nhóm lệnh I-format, có opcode = $2b_{\text{hex}}$

Syntax (cú pháp):



Ý nghĩa: $M[R[rs] + \text{SignExtImm}] = R[rt]$

⇒ Lưu giá trị thanh ghi rt vào từ nhớ có địa chỉ được tính bằng giá trị thanh ghi rs cộng với offset (offset được mở rộng có dấu thành số 32 bits trước khi cộng)

Ví dụ:

sw \$t1, 8(\$t0)

giả sử \$t0 = 0x10010000

\$t1 = 0x87654321

và từ nhớ tại địa chỉ 0x10010008 có giá trị 0x12345678

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	78

⇒ Lệnh sw thực hiện việc lưu giá trị của thanh ghi \$t1 vào từ nhớ có địa chỉ = \$t0 + 8 = 0x10010008

Giá trị của từ nhớ tại địa chỉ 0x10010008 sau khi lệnh trên thực hiện là:

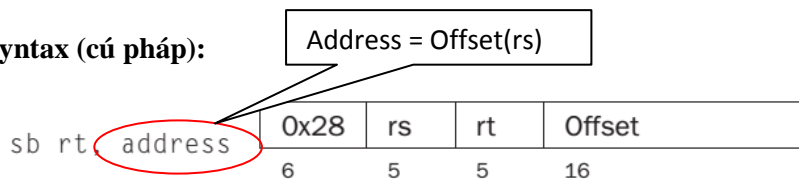
Byte 4	Byte 2	Byte 1	Byte 0
87	65	43	21

21. Lệnh sb

Store Byte sb I $M[R[rs] + \text{SignExtImm}](7:0) = R[rt](7:0)$ (2) 28_{hex}

⇒ lệnh thuộc nhóm I-format, có opcode = 28_{hex}

Syntax (cú pháp):



Ý nghĩa: $M[R[rs] + \text{SignExtImm}](7:0) = R[rt](7:0)$

⇒ Lấy byte thấp nhất của trong thanh ghi rt lưu vào byte thấp nhất của từ nhớ có địa chỉ được tính bằng giá trị thanh ghi rs cộng với offset (offset được mở rộng có dấu thành số 32 bits trước khi cộng)

Ví dụ:

sb \$t1, 8(\$t0)

giả sử \$t0 = 0x10010000

\$t1 = 0x87654321

và từ nhớ tại địa chỉ 0x10010008 có giá trị 0x12345678

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	78

⇒ Lệnh sb thực hiện việc lưu byte 0 của thanh ghi t1 (0x21) vào byte 0 của từ nhớ tại địa chỉ 0x10010008. Nên sau lệnh trên, hình ảnh từ nhớ:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	21

22. Lệnh sh

$$\text{Store Halfword} \quad \text{sh} \quad \text{I} \quad M[R[rs] + \text{SignExtImm}](15:0) = R[rt](15:0) \quad (2) \quad 29_{\text{hex}}$$

⇒ lệnh thuộc nhóm I-format, có opcode = 29_{hex}

Syntax (cú pháp):

sh rt, address	0x29	rs	rt	Offset
	6	5	5	16

Ý nghĩa: $M[R[rs] + \text{SignExtImm}](15:0) = R[rt](15:0)$

⇒ Lấy 2 byte thấp nhất trong thanh ghi rt (nửa thấp) lưu vào 2 byte thấp nhất của từ nhớ có địa chỉ được tính bằng giá trị thanh ghi rs cộng với offset (offset được mở rộng có dấu thành số 32 bits trước khi cộng)

Ví dụ:

sh \$t1, 8(\$t0)

giả sử $\$t0 = 0x10010000$

$\$t1 = 0x87654321$

và từ nhớ tại địa chỉ $0x10010008$ có giá trị $0x12345678$

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	78

⇒ Lệnh sh thực hiện việc lưu byte 0 và byte 1 của thanh ghi t1 ($0x4321$) vào byte 0 và byte 1 của từ nhớ tại địa chỉ $0x10010008$. Nên sau lệnh trên, hình ảnh từ nhớ:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	43	21

23. Lệnh lui

$$\text{Load Upper Imm.} \quad \text{lui} \quad \text{I} \quad R[rt] = \{\text{imm}, 16'b0\} \quad f_{\text{hex}}$$

⇒ Lệnh thuộc I-format, có opcode là f_{hex}

Syntax (cú pháp):

lui rt, imm	0xf	0	rt	imm
	6	5	5	16

Ý nghĩa: $R[rt] = \{imm, 16'b0\}$

⇒ Gán số tức thời 16 bits vào nửa cao của thanh ghi rt, nửa thấp đưa 0 vào

Ví dụ:

a. lui \$t1, 0x1234

Kết quả: \$t1 = 0x12340000

b. lui \$t1, 0x12345

⇒ báo lỗi do số tức thời tràn quá số 16 bits

❖ Ngoài ra còn 2 lệnh thuộc nhóm PseudoInstruction set: *li* và *move*

Load Immediate

li

$R[rd] = \text{immediate}$

Move

move

$R[rd] = R[rs]$

24. Lệnh li

Ý nghĩa: đưa một số tức thời (32 bits) vào một thanh ghi

Ví dụ:

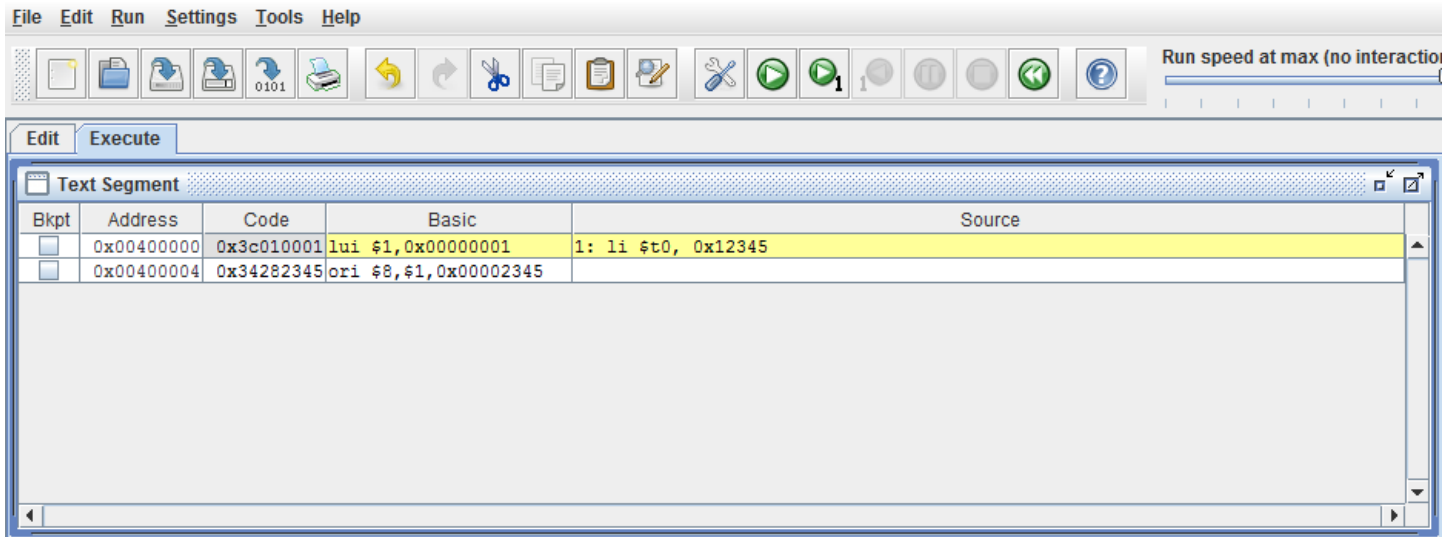
li \$t0, 0x12345

Kết quả: \$t0 = 0x00012345

Lưu ý:

Số đưa vào lui chỉ được phép là số 16 bits, còn số đưa và li có thể lên đến 32 bits

Lệnh *li* này thực chất là lệnh giả, lệnh này được chuyển thành 2 lệnh (lui và or) khi processor chạy thật sự:



25. Lệnh *move*

Ý nghĩa: sao chép/di chuyển giá trị từ thanh ghi này sang thanh ghi kia

Ví dụ:

move \$t1, \$t2

Giả sử $\$t1 = 0x12345678$

$\$t2 = 0x87654321$

Khi lệnh trên thực thi, giá trị thanh ghi $\$t2$ được đưa vào thanh ghi $\$t1$

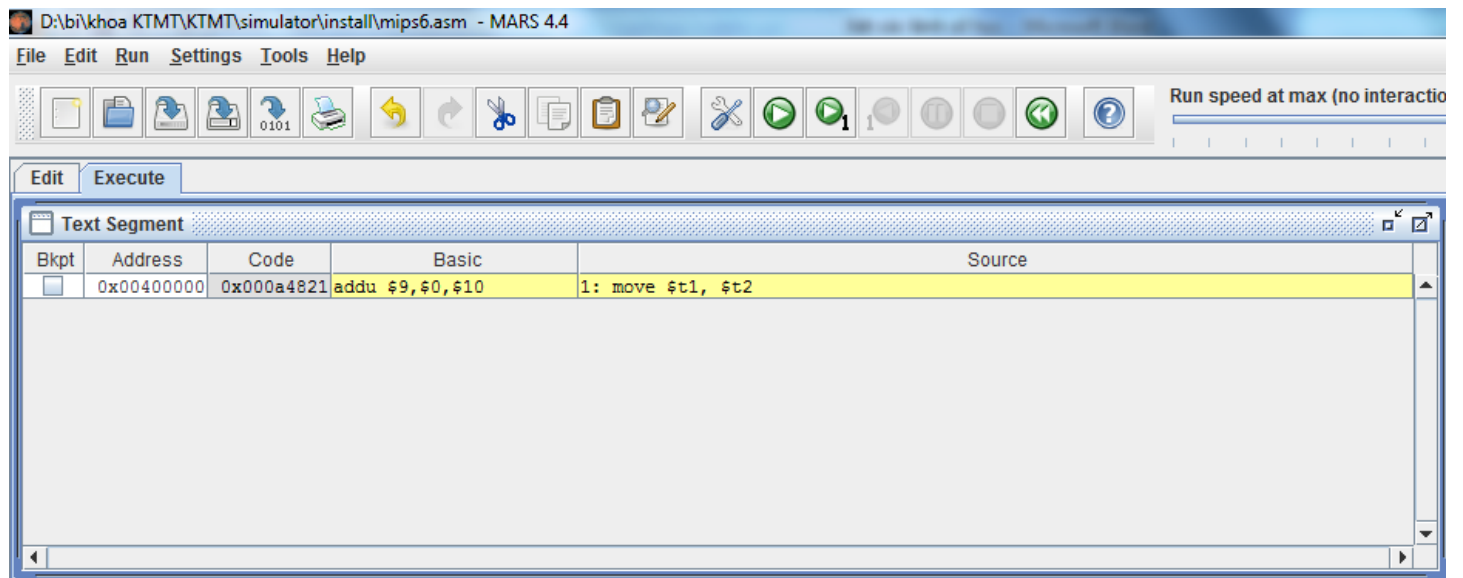
Kết quả sau khi chạy:

$\$t1 = 0x87654321$

$\$t2 = 0x87654321$

Chú ý:

Lệnh *li* này thực chất là lệnh giả, lệnh này được chuyển thành lệnh '*addu*' khi processor chạy thật sự (thực hiện việc cộng thanh ghi $\$t2$ với thanh ghi zero, kết quả nạp vào thanh ghi $\$t1$):



CÁC LỆNH ASSEMBLY TRONG KHỐI “ARITHMETIC CORE INSTRUCTION SET”

Các lệnh assembly nằm ở bảng con bên phải của bảng số 1 chứa các lệnh số học phức tạp hơn so với các lệnh bên bảng con trái.

ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR- MAT	OPERATION	② OPCODE / FMT / FT / FUNCT (Hex)
Branch On FP True bclt	FI	if(FPcond)PC=PC+4+BranchAddr (4)	11/8/1/--
Branch On FP False bclf	FI	if(!FPcond)PC=PC+4+BranchAddr(4)	11/8/0/--
Divide div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	0/--/--/1a
Divide Unsigned divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt] (6)	0/--/--/1b
FP Add Single add.s	FR	F[fd]=F[fs]+F[ft]	11/10/--/0
FP Add Double add.d	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}+ {F[ft],F[ft+1]}	11/11/--/0

Do trong bảng này chứa một số lệnh làm việc với số dấu chấm động (floating-point), nên lệnh nào có format có “F” phía trước tức là lệnh làm việc với số dấu chấm động

Do các lệnh làm việc với số floating-point có format lệnh hơi khác so với R-format hoặc I-format chuẩn (Có thêm trường ‘fmt’: để phân biệt làm việc với số floating-point độ chính xác đơn hay độ chính kép)

Ví dụ: lệnh bclt có giá trị cột này là “11/8/1/...” tức opcode của lệnh = 11_{hex}, fmt = 8_{hex}, ft = 1_{hex} và funct không quan tâm

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31	26 25	21 20	16 15	11 10	6 5 0
FI	opcode	fmt	ft	immediate		
	31	26 25	21 20	16 15		0

A. Các lệnh nhân và chia

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (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 _{hex}
Add Immediate	addi I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 _{hex}
Add Imm. Unsigned	addiu I	$R[rt] = R[rs] + \text{SignExtImm}$	(2) 9 _{hex}
Add Unsigned	addu R	$R[rd] = R[rs] + R[rt]$	0/21 _{hex}
And	and R	$R[rd] = R[rs] \& R[rt]$	0/24 _{hex}
And Immediate	andi I	$R[rt] = R[rs] \& \text{ZeroExtImm}$	(3) c _{hex}
Branch On Equal	beq I	if($R[rs] == R[rt]$) $PC = PC + 4 + \text{BranchAddr}$	(4) 4 _{hex}
Branch On Not Equal	bne I	if($R[rs] != R[rt]$) $PC = PC + 4 + \text{BranchAddr}$	(4) 5 _{hex}
Jump	j J	$PC = \text{JumpAddr}$	(5) 2 _{hex}
Jump And Link	jal J	$R[31] = PC + 8; PC = \text{JumpAddr}$	(5) 3 _{hex}
Jump Register	jrr R	$PC = R[rs]$	0/08 _{hex}
Load Byte Unsigned	lbu I	$R[rt] = \{24'b0, M[R[rs]] + \text{SignExtImm}(7:0)\}$	(2) 24 _{hex}
Load Halfword Unsigned	lhu I	$R[rt] = \{16'b0, M[R[rs]] + \text{SignExtImm}(15:0)\}$	(2) 25 _{hex}
Load Linked	ll I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2,7) 30 _{hex}
Load Upper Imm.	lui I	$R[rt] = \{\text{imm}, 16'b0\}$	f _{hex}
Load Word	lw I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 23 _{hex}
Nor	nor R	$R[rd] = \sim (R[rs] R[rt])$	0/27 _{hex}
Or	or R	$R[rd] = R[rs] R[rt]$	0/25 _{hex}
Or Immediate	ori I	$R[rt] = R[rs] \text{ZeroExtImm}$	(3) d _{hex}
Set Less Than	slt R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	0/2a _{hex}
Set Less Than Imm.	slti I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	a _{hex}
Set Less Than Imm. Unsigned	sltiu I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2,6) b _{hex}
Set Less Than Unsig.	sltu R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	(6) 0/2b _{hex}
Shift Left Logical	sll R	$R[rd] = R[rt] << \text{shamt}$	0/00 _{hex}
Shift Right Logical	srl R	$R[rd] = R[rt] >>> \text{shamt}$	0/02 _{hex}
Store Byte	sb I	$M[R[rs] + \text{SignExtImm}(7:0)] = R[rt](7:0)$	(2) 28 _{hex}
Store Conditional	sc I	$M[R[rs] + \text{SignExtImm}] = R[rt];$ $R[rt] = (\text{atomic}) ? 1 : 0$	(2,7) 38 _{hex}
Store Halfword	sh I	$M[R[rs] + \text{SignExtImm}(15:0)] = R[rt](15:0)$	(2) 29 _{hex}
Store Word	sw I	$M[R[rs] + \text{SignExtImm}] = R[rt]$	(2) 2b _{hex}
Subtract	sub R	$R[rd] = R[rs] - R[rt]$	(1) 0/22 _{hex}
Subtract Unsigned	subu R	$R[rd] = R[rs] - R[rt]$	0/23 _{hex}

- (1) May cause overflow exception
 (2) $\text{SignExtImm} = \{16\{\text{immediate}[15]\}, \text{immediate}\}$
 (3) $\text{ZeroExtImm} = \{16\{1b'0\}, \text{immediate}\}$
 (4) $\text{BranchAddr} = \{14\{\text{immediate}[15]\}, \text{immediate}, 2'b0\}$
 (5) $\text{JumpAddr} = \{PC + 4[31:28], \text{address}, 2'b0\}$
 (6) Operands considered unsigned numbers (vs. 2's comp.)
 (7) Atomic test&set pair; $R[rt] = 1$ if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode		rs		rt		rd		shamt		funct	
	31	26 25	21 20		16 15		11 10		6 5		0	
I	opcode		rs		rt		immediate					
	31	26 25	21 20		16 15							
J	opcode		address									
	31	26 25										

ARITHMETIC CORE INSTRUCTION SET

②

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FUNCT (Hex)
Branch On FP True	bclt FI	if($FPcond$) $PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/1/--
Branch On FP False	bclt FI	if(! $FPcond$) $PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/0/--
Divide	div R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	0/--/--/1a
Divide Unsigned	divu R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	(6) 0/--/--/1b
FP Add Single	add.s FR	$F[fd] = F[fs] + F[ft]$	11/10/--/0
FP Add	add.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} + \{F[ft], F[ft+1]\}$	11/11/--/0
FP Compare Single	c.x.s* FR	$FPcond = (F[fs] op F[ft]) ? 1 : 0$	11/10/--/y
FP Compare	c.x.d* FR	$FPcond = (\{F[fs], F[fs+1]\} op \{F[ft], F[ft+1]\}) ? 1 : 0$	11/11/--/y
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)			
FP Divide Single	div.s FR	$F[fd] = F[fs] / F[ft]$	11/10/--/3
FP Divide	div.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} / \{F[ft], F[ft+1]\}$	11/11/--/3
FP Multiply Single	mul.s FR	$F[fd] = F[fs] * F[ft]$	11/10/--/2
FP Multiply	mul.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} * \{F[ft], F[ft+1]\}$	11/11/--/2
FP Subtract Single	sub.s FR	$F[fd] = F[fs] - F[ft]$	11/10/--/1
FP Subtract	sub.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} - \{F[ft], F[ft+1]\}$	11/11/--/1
Load FP Single	lwc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 31/--/--/0
Load FP	ldc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}];$ $F[rt+1] = M[R[rs] + \text{SignExtImm} + 4]$	(2) 35/--/--/0
Move From Hi	mfmhi R	$R[rd] = Hi$	0/--/--/10
Move From Lo	mfmlo R	$R[rd] = Lo$	0/--/--/12
Move From Control	mfc0 R	$R[rd] = CR[rs]$	10/0/--/0
Multiply	mult R	$\{Hi, Lo\} = R[rs] * R[rt]$	0/--/--/18
Multiply Unsigned	multu R	$\{Hi, Lo\} = R[rs] * R[rt]$	(6) 0/--/--/19
Shift Right Arith.	sra R	$R[rd] = R[rt] >>> \text{shamt}$	0/--/--/3
Store FP Single	swc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt]$	(2) 39/--/--/0
Store FP	sdc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt];$ $M[R[rs] + \text{SignExtImm} + 4] = F[rt+1]$	(2) 3d/--/--/0

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31	26 25	21 20	16 15	11 10	6 5 0
FI	opcode	fmt	ft	immediate		
	31	26 25	21 20	16 15	0	

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if($R[rs] < R[rt]$) $PC = \text{Label}$
Branch Greater Than	bgt	if($R[rs] > R[rt]$) $PC = \text{Label}$
Branch Less Than or Equal	btle	if($R[rs] <= R[rt]$) $PC = \text{Label}$
Branch Greater Than or Equal	bge	if($R[rs] >= R[rt]$) $PC = \text{Label}$
Load Immediate	li	$R[rd] = \text{immediate}$
Move	move	$R[rd] = R[rs]$

REGISTER NAME, NUMBER, USE, CALL CONVENTION

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

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.

Các lệnh assembly cần khi thực hiện phép nhân và chia:

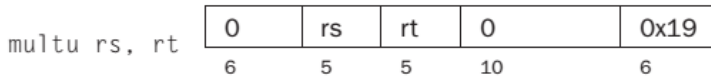
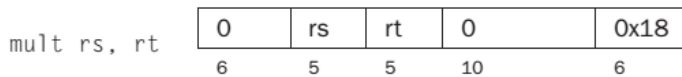
- Nhân: *mult/multu*
- Chia: *div/divu*
- Hai lệnh hỗ trợ: *mfhi/mflo*

Lệnh *mult/multu*

Multiply *mult* R {Hi,Lo} = R[rs] * R[rt] 0/--/--/18
 Multiply Unsigned *multu* R {Hi,Lo} = R[rs] * R[rt] (6) 0/--/--/19

⇒ Hai lệnh này thuộc nhóm lệnh R-format, có opcode là 0. Lệnh *mult* có funct là 18, lệnh *multu* có funct là 19

Syntax:



Ý nghĩa:

{Hi, Lo} = R[rs] * R[rt]

⇒ Giá trị trong thanh ghi rs (số 32 bits) nhân với giá trị trong thanh ghi rt (số 32 bits), kết quả là số 64 bits. 32 bits thuộc nửa thấp của kết quả được lưu trong thanh ghi **Lo**, và 32 bits thuộc nửa cao của kết quả được lưu trong thanh ghi **Hi**

Chú ý: *Hi* và *Lo* là 2 thanh ghi phụ thêm cho processor khi thực hiện phép toán nhân chia. Vì các thanh ghi đều là 32 bits, mà kết quả phép toán nhân là 64 bits, nên phải dùng 2 thanh ghi tạm này ghép lại

- *mult*: nhân 2 số có dấu
- *multu*: nhân 2 số không dấu

Ví dụ:

a. *mult \$t1, \$t2*

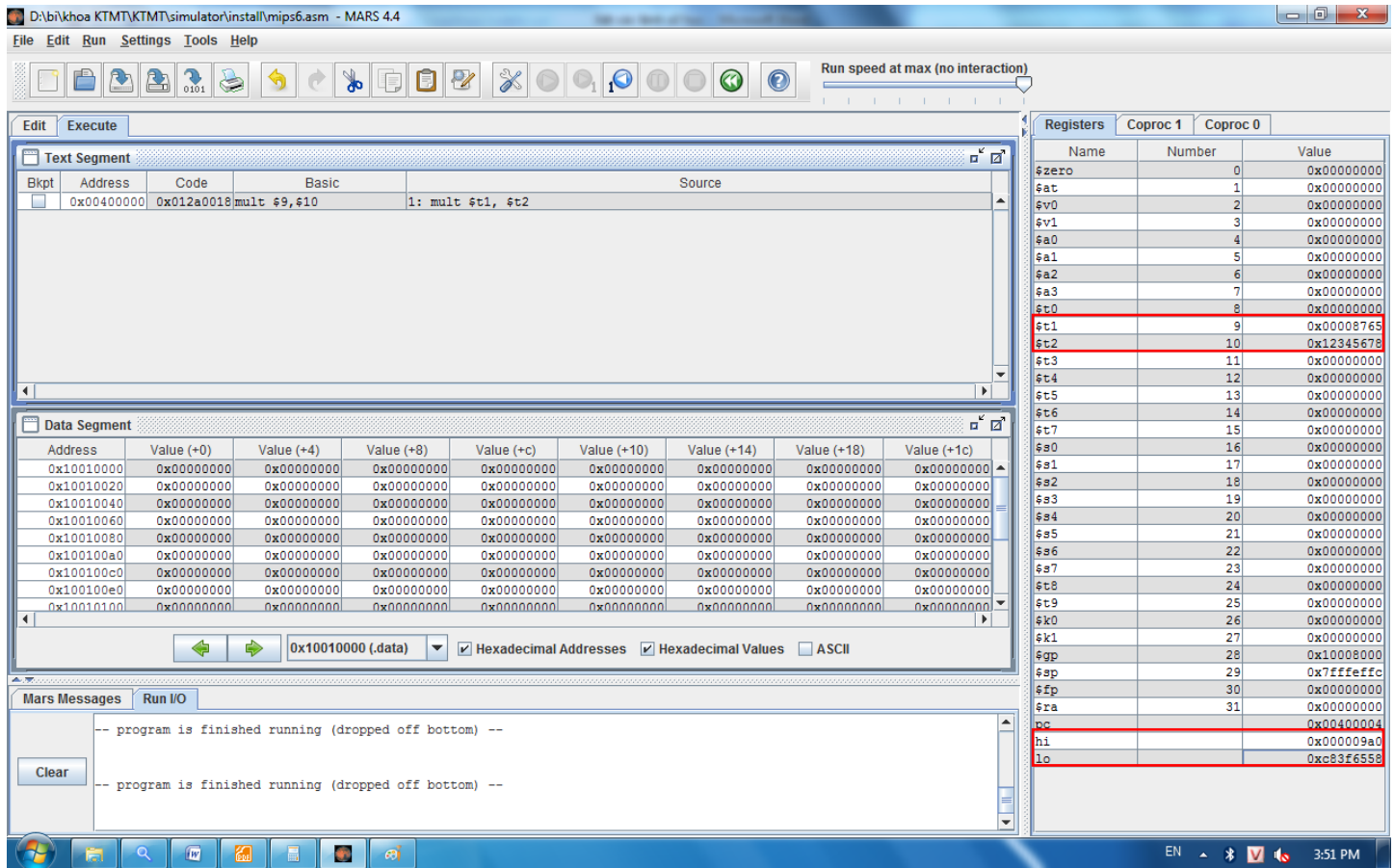
Giả sử \$t1 = 0x00008765

\$t2 = 0x12345678

Kết quả:

\$hi = 0x000009a0

\$lo = 0xc83f6558



b. *mult \$t1, \$t2*

Giả sử $\$t1 = 0x80008765$

$\$t2 = 0x12345678$

Kết quả:

$\$hi = 0xf6e5de64$

$\$lo = 0xc83f6558$

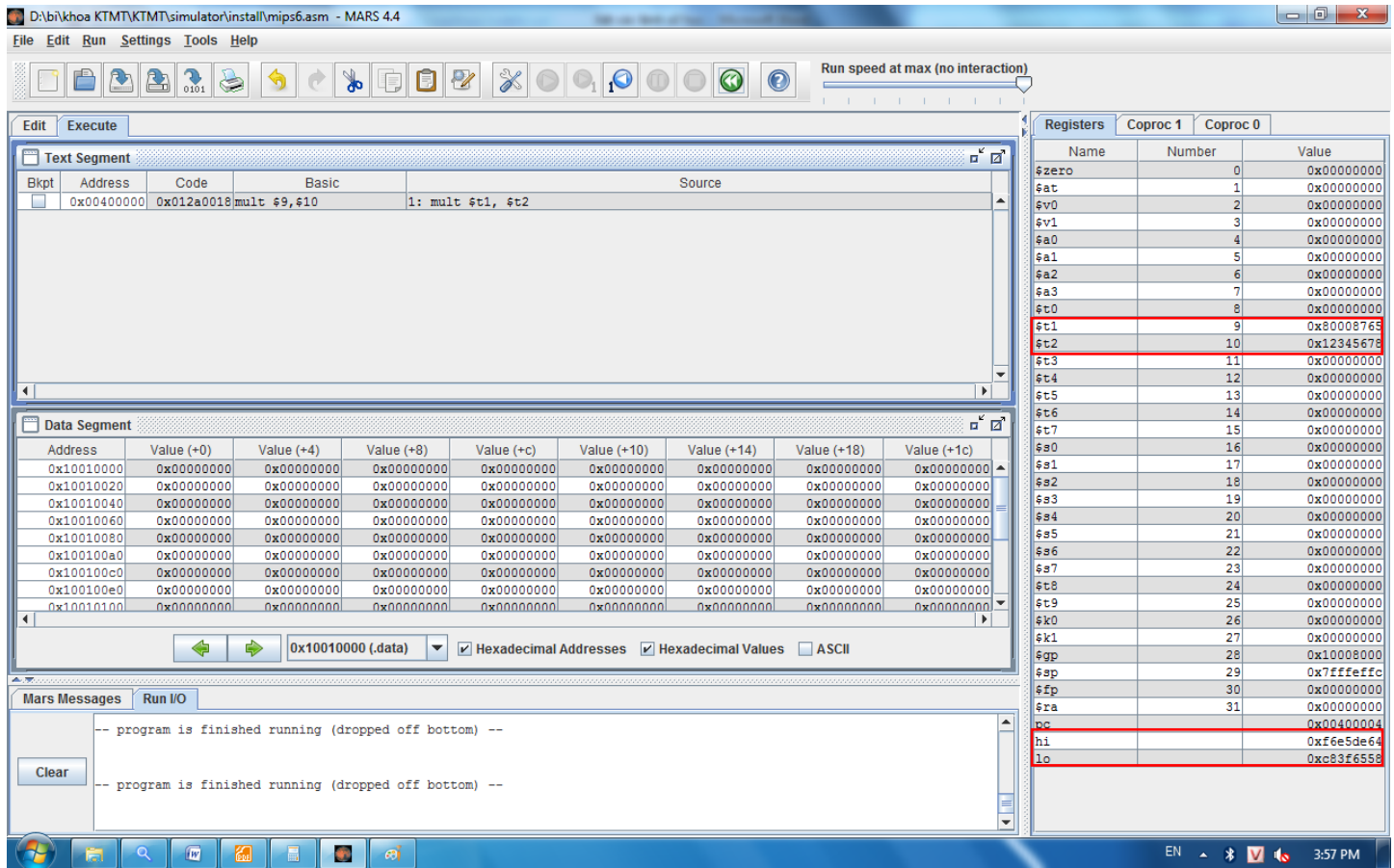
Tức $\$t1 * \$t2 = -0x91A219B37C09AA8$ (xét trên số có dấu dạng bù 2)

Kiểm tra lại:

$$\begin{aligned} \$t1 = 0x80008765 &= 1000\ 0000\ 0000\ 0000\ 1000\ 0111\ 0110\ 0101_{(2)} \\ &= -0x7FFF789B \end{aligned}$$

$$\$t2 = 0x12345678$$

$$\begin{aligned} \Rightarrow \$t1 * \$t2 &= -(0x7FFF789B \times 0x12345678) = -0x91A219B37C09AA8 \\ &= 0xF6E5DE64C83F6558 \text{ (bù 2)} \end{aligned}$$



c. *multu \$t1, \$t2*

Giả sử $\$t1 = 0x80008765$

$\$t2 = 0x12345678$

Kết quả:

$\$hi = 0x091a34dc$

$\$lo = 0xc83f6558$

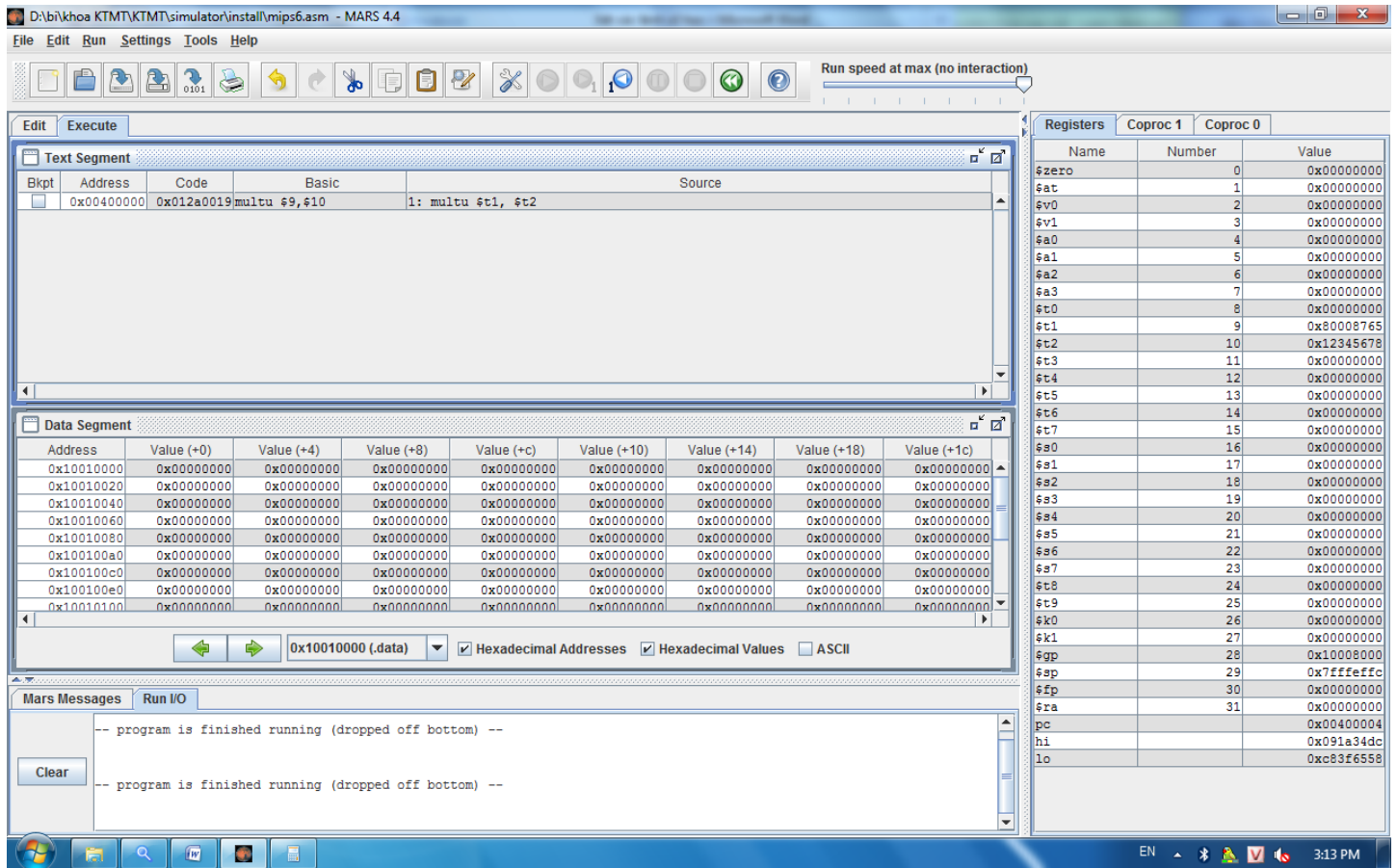
Kiểm chứng kết quả:

multu thực hiện phép nhân 2 số dạng không dấu

$\$t1 = 0x80008765 = 20000103545_{(10)}$

$\$t2 = 0x12345678 = 2215053170_{(10)}$

$\$t1 \times \$t2 = 20000103545_{(10)} \times 2215053170_{(10)} = 44321515631017662530_{(10)} = 0x91A34DCC83F6558$



Chia: *div/divu*

Divide `div R` $Lo=R[rs]/R[rt]; Hi=R[rs]\%R[rt]$ 0/--/--/1a
 Divide Unsigned `divu R` $Lo=R[rs]/R[rt]; Hi=R[rs]\%R[rt]$ (6) 0/--/--/1b

⇒ Hai lệnh này thuộc nhóm lệnh R-format, có opcode là 0. Lệnh *div* có funct là 1a_{hex}, lệnh *divu* có funct là 1b_{hex}

Syntax:

Divide (with overflow)

`div rdest, rsrc1, src2` *pseudoinstruction*

Divide (without overflow)

`divu rdest, rsrc1, src2` *pseudoinstruction*

Ý nghĩa:

rdest, rsrcl và src2 là 3 thanh ghi.

Hai lệnh trên lấy giá trị trong thanh ghi rsrcl chia cho src2, thương số đặt vào thanh ghi rdest. Đồng thời thanh ghi Lo cũng chứa thương số giống thanh ghi rdest và thanh ghi Hi chứa phần dư.

- *div*: chia 2 số có xét tràn
- *divu*: chia 2 số không xét tràn

Ví dụ:

a. `div $t1, $t2, $t3`

giả sử $\$t2 = 0x6$, $\$t3 = 0x5$

Kết quả: $\$t1 = 0x1$

$\$Lo = 0x1$

$\$Hi = 0x1$

✚ Đưa thêm ví dụ kiểm chứng trường hợp tràn và không tràn (div/divu)

CÁC LỆNH ASSEMBLY CHO SỐ DẤU CHẤM ĐỘNG (floating-point number)

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (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 _{hex}
Add Immediate	addi I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 _{hex}
Add Imm. Unsigned	addiu I	$R[rt] = R[rs] + \text{SignExtImm}$	(2) 9 _{hex}
Add Unsigned	addu R	$R[rd] = R[rs] + R[rt]$	0/21 _{hex}
And	and R	$R[rd] = R[rs] \& R[rt]$	0/24 _{hex}
And Immediate	andi I	$R[rt] = R[rs] \& \text{ZeroExtImm}$	(3) c _{hex}
Branch On Equal	beq I	if($R[rs] == R[rt]$) $PC = PC + 4 + \text{BranchAddr}$	(4) 4 _{hex}
Branch On Not Equal	bne I	if($R[rs] != R[rt]$) $PC = PC + 4 + \text{BranchAddr}$	(4) 5 _{hex}
Jump	j J	$PC = \text{JumpAddr}$	(5) 2 _{hex}
Jump And Link	jal J	$R[31] = PC + 8; PC = \text{JumpAddr}$	(5) 3 _{hex}
Jump Register	jrr R	$PC = R[rs]$	0/08 _{hex}
Load Byte Unsigned	lbu I	$R[rt] = \{24'b0, M[R[rs] + \text{SignExtImm}(7:0)]\}$	(2) 24 _{hex}
Load Halfword Unsigned	lhu I	$R[rt] = \{16'b0, M[R[rs] + \text{SignExtImm}(15:0)]\}$	(2) 25 _{hex}
Load Linked	ll I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2,7) 30 _{hex}
Load Upper Imm.	lui I	$R[rt] = \{\text{imm}, 16'b0\}$	f _{hex}
Load Word	lw I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 23 _{hex}
Nor	nor R	$R[rd] = \sim (R[rs] \mid R[rt])$	0/27 _{hex}
Or	or R	$R[rd] = R[rs] \mid R[rt]$	0/25 _{hex}
Or Immediate	ori I	$R[rt] = R[rs] \mid \text{ZeroExtImm}$	(3) d _{hex}
Set Less Than	slt R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	0/2a _{hex}
Set Less Than Imm.	slti I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	a _{hex}
Set Less Than Imm. Unsigned	sltiu I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2,6) b _{hex}
Set Less Than Unsig.	sltu R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	(6) 0/2b _{hex}
Shift Left Logical	sll R	$R[rd] = R[rt] << \text{shamt}$	0/00 _{hex}
Shift Right Logical	srl R	$R[rd] = R[rt] >>> \text{shamt}$	0/02 _{hex}
Store Byte	sb I	$M[R[rs] + \text{SignExtImm}(7:0)] = R[rt](7:0)$	(2) 28 _{hex}
Store Conditional	sc I	$M[R[rs] + \text{SignExtImm}] = R[rt];$ $R[rt] = (\text{atomic}) ? 1 : 0$	(2,7) 38 _{hex}
Store Halfword	sh I	$M[R[rs] + \text{SignExtImm}(15:0)] = R[rt](15:0)$	(2) 29 _{hex}
Store Word	sw I	$M[R[rs] + \text{SignExtImm}] = R[rt]$	(2) 2b _{hex}
Subtract	sub R	$R[rd] = R[rs] - R[rt]$	(1) 0/22 _{hex}
Subtract Unsigned	subu R	$R[rd] = R[rs] - R[rt]$	0/23 _{hex}

- (1) May cause overflow exception
 (2) $\text{SignExtImm} = \{16\{\text{immediate}[15]\}, \text{immediate}\}$
 (3) $\text{ZeroExtImm} = \{16\{1b'0\}, \text{immediate}\}$
 (4) $\text{BranchAddr} = \{14\{\text{immediate}[15]\}, \text{immediate}, 2'b0\}$
 (5) $\text{JumpAddr} = \{PC + 4[31:28], \text{address}, 2'b0\}$
 (6) Operands considered unsigned numbers (vs. 2's comp.)
 (7) Atomic test&set pair; $R[rt] = 1$ if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode		rs		rt		rd		shamt		funct	
	31	26 25	21 20		16 15		11 10		6 5		0	
I	opcode		rs		rt		immediate					
	31	26 25	21 20		16 15							
J	opcode		address									
	31	26 25										

ARITHMETIC CORE INSTRUCTION SET

② OPCODE

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FUNCT (Hex)
Branch On FP True	bclt FI	if($FPcond$) $PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/1/--
Branch On FP False	bclt FI	if(! $FPcond$) $PC = PC + 4 + \text{BranchAddr}$	(4) 11/8/0/--
Divide	div R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	0/--/--/1a
Divide Unsigned	divu R	$Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$	(6) 0/--/--/1b
FP Add Single	add.s FR	$F[fd] = F[fs] + F[ft]$	11/10/--/0
FP Add Double	add.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} + \{F[ft], F[ft+1]\}$	11/11/--/0
FP Compare Single	c.x.s* FR	$FPcond = (F[fs] op F[ft]) ? 1 : 0$	11/10/--/y
FP Compare Double	c.x.d* FR	$FPcond = (\{F[fs], F[fs+1]\} op \{F[ft], F[ft+1]\}) ? 1 : 0$ * (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)	11/11/--/y
FP Divide Single	div.s FR	$F[fd] = F[fs] / F[ft]$	11/10/--/3
FP Divide Double	div.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} / \{F[ft], F[ft+1]\}$	11/11/--/3
FP Multiply Single	mul.s FR	$F[fd] = F[fs] * F[ft]$	11/10/--/2
FP Multiply Double	mul.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} * \{F[ft], F[ft+1]\}$	11/11/--/2
FP Subtract Single	sub.s FR	$F[fd] = F[fs] - F[ft]$	11/10/--/1
FP Subtract Double	sub.d FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} - \{F[ft], F[ft+1]\}$	11/11/--/1
Load FP Single	lwc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 31/--/--/0
Load FP Double	ldc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}];$ $F[rt+1] = M[R[rs] + \text{SignExtImm} + 4]$	(2) 35/--/--/0
Move From Hi	mfhi R	$R[rd] = Hi$	0/--/--/10
Move From Lo	mflo R	$R[rd] = Lo$	0/--/--/12
Move From Control	mfc0 R	$R[rd] = CR[rs]$	10/0/--/0
Multiply	mult R	$\{Hi, Lo\} = R[rs] * R[rt]$	0/--/--/18
Multiply Unsigned	multu R	$\{Hi, Lo\} = R[rs] * R[rt]$	(6) 0/--/--/19
Shift Right Arith.	sra R	$R[rd] = R[rt] >> \text{shamt}$	0/--/--/3
Store FP Single	swc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt]$	(2) 39/--/--/0
Store FP Double	sdc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt];$ $M[R[rs] + \text{SignExtImm} + 4] = F[rt+1]$	(2) 3d/--/--/0

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31	26 25	21 20	16 15	11 10	6 5 0
FI	opcode	fmt	ft	immediate		
	31	26 25	21 20	16 15	0	

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if($R[rs] < R[rt]$) $PC = \text{Label}$
Branch Greater Than	bgt	if($R[rs] > R[rt]$) $PC = \text{Label}$
Branch Less Than or Equal	btle	if($R[rs] \leq R[rt]$) $PC = \text{Label}$
Branch Greater Than or Equal	bge	if($R[rs] \geq R[rt]$) $PC = \text{Label}$
Load Immediate	li	$R[rd] = \text{immediate}$
Move	move	$R[rd] = R[rs]$

REGISTER NAME, NUMBER, USE, CALL CONVENTION

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

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.

Nhóm lệnh số học:**Cộng***add.s**add.d***Trừ***sub.s**sub.d***Nhân***mul.s**mul.d***Chia***div.s**div.d*

Chú ý: Các lệnh làm việc với số floating-point làm việc trên 32 thanh ghi f

FP Add Single	<i>add.s</i>	FR	$F[fd] = F[fs] + F[ft]$	11/10/--/0
FP Add Double	<i>add.d</i>	FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} + \{F[ft], F[ft+1]\}$	11/11/--/0
FP Subtract Single	<i>sub.s</i>	FR	$F[fd] = F[fs] - F[ft]$	11/10/--/1
FP Subtract Double	<i>sub.d</i>	FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} - \{F[ft], F[ft+1]\}$	11/11/--/1
FP Multiply Single	<i>mul.s</i>	FR	$F[fd] = F[fs] * F[ft]$	11/10/--/2
FP Multiply Double	<i>mul.d</i>	FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} * \{F[ft], F[ft+1]\}$	11/11/--/2
FP Divide Single	<i>div.s</i>	FR	$F[fd] = F[fs] / F[ft]$	11/10/--/3
FP Divide Double	<i>div.d</i>	FR	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} / \{F[ft], F[ft+1]\}$	11/11/--/3

⇒ Các lệnh này thuộc nhóm lệnh FR, có opcode đều bằng 11_{hex}

Lệnh add có funct bằng 0, lệnh sub có funct bằng 1_{hex}, lệnh mul có funct bằng 2_{hex}, lệnh div có funct bằng 3_{hex}

Chú ý: các lệnh .s là làm việc với độ chính xác đơn, có trường fmt luôn bằng 10_{hex}; các lệnh .d là làm việc với độ chính xác kép, có trường fmt luôn bằng 11_{hex}

add.s

Syntax

add.s fd, fs, ft	0x11	0x10	ft	fs	fd	0
	6	5	5	5	5	6

Ý nghĩa:

$$F[fd] = F[fs] + F[ft]$$

⇒ Giá trị trong thanh ghi fs cộng với giá trị trong thanh ghi ft, tổng lưu vào thanh ghi fd

add.d

Syntax

add.d fd, fs, ft	0x11	0x11	ft	fs	fd	0
	6	5	5	5	5	6

Ý nghĩa:

$$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} + \{F[ft], F[ft+1]\}$$

⇒ Số tức thời lúc này do biểu diễn trong độ chính xác kép nên cần 64 bits, tức phải cần 2 thanh ghi f liên tục nhau.

Lệnh add.d sẽ thực hiện việc cộng giá trị của số floating point độ chính xác kép đang lưu trong

⇒ Giá trị trong thanh ghi fs cộng với giá trị trong thanh ghi ft, tổng lưu vào thanh ghi fd