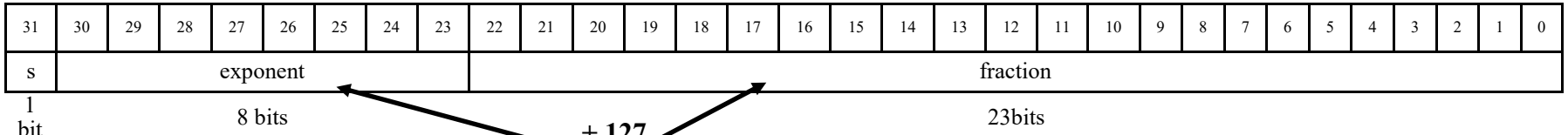


(1: - ; 0: +)

$$1.\underline{xxxxxxxxx}_{two} \times 2^{\underline{yyyy}}$$

$$(-1)^S \times (1 + 0.F) \times 2^{E-127}$$

0	0111 1111	1	1000 0000	2-3
-1	0111 1110	2	1000 0001	4-7
-2	0111 1101	3	1000 0010	8-15
-3	0111 1100	4	1000 0011	16-31

Với độ chính sát kép: 64 bits

1 bit S, 11 bits E, 52 bits F y = E - 1023

*** Một số công thức cần chú ý:**

Performance$_x$ = $\frac{1}{Execution\ time_x}$

Nếu: **Performance$_x$ > Performance$_y$**

Execution time$_y$ > Execution time$_x$

Gọi: **ET: Execution time:** thời gian thực thi chương trình

NC: Number of clock cycles Số vòng clock chương trình cần

T: Clock cycle: Chu kì

f: Clock rate/ Clock frequency: Tần số xung

CPI: Clock cycle per instruction: Số chu kì xung cần để thực thi một lệnh

NI: Number of instruction: Tổng số lệnh cho một chương trình

ET = NC*T = $\frac{NC}{f}$ (với T = 1/f) (1)

NC = NI * CPI

ET = NI/MIPS x 10^6

(1) → ET = NI * CPI * T = $\frac{NI * CPI}{f}$

Chuyển đổi đơn vị:

1GHz = 10^9 Hz

1 ns = 10^{-9} s

1M = 10^6

1ps = 10^{-12}

1µs = 10^{-6}

1ms = 10^{-3}

Gọi: **MIPS (Million instructions per):** Số tập triệu lệnh trên giây mà máy thực hiện được

MIPS = $\frac{NI}{ET\ X\ 10^6} = \frac{f}{CPI\ X\ 10^6}$

CPI$_{tb}$ = $\frac{NC_{total}}{NI_{total}}$

1.7.4 Cộng số nguyên không dấu

- Nếu cộng n-bit với n-bit
  - + Ra n-bit thì không bị tràn $C_{out} = 0$
  - + Ra > n-bit thì bị tràn nhớ $C_{out} = 1$
- Xảy ra khi tổng > $2^n - 1$

1.7.5 Cộng số nguyên có dấu

- **Nếu cộng hai số khác dấu:** kết quả luôn luôn đúng
- **Cộng hai số cùng dấu:**
  - + Nếu kết quả cùng dấu với các số hạng thì kết quả là đúng
  - + Nếu kết quả có dấu ngược lại, khi đó có tràn xảy ra
- **Phạm vi:** (-2^{n-1}) - $(2^{n-1} - 1)$

-4	0111 1011
-5	0111 1010
-6	0111 1001
-7	0111 1000
-8	0111 0111
-9	0111 0110
-10	0111 0101
-11	0111 0100
-12	0111 0011
-13	0111 0010
-14	0111 0001
-15	0111 0000
-16	0110 1111
-17	0110 1110
-18	0110 1101
-19	0110 1100
-20	0110 1011

$2^0 = 1$	$2^{11} = 2048$	$2^{-1} = 0.5$
$2^1 = 2$	$2^{12} = 4096$	$2^{-2} = 0.25$
$2^2 = 4$	$2^{13} = 8192$	$2^{-3} = 0.125$
$2^3 = 8$	$2^{14} = 16384$	$2^{-4} = 0.0625$
$2^4 = 16$	$2^{15} = 32768$	$2^{-5} = 0.03125$
$2^5 = 32$	$2^{16} = 65536$	$2^{-6} = 0.015625$
$2^6 = 64$	$2^{17} = 131072$	$2^{-7} = 0.0078125$
$2^7 = 128$	$2^{18} = 262144$	$2^{-8} = 0.00390625$
$2^8 = 256$	$2^{19} = 524288$	$2^{-9} = 0.001953125$
$2^9 = 512$	$2^{20} = 1048576$	
$2^{10} = 1024$		

Chuyển đổi số thập phân sang nhị phân:

- 0.6875 x 2 = 1.375 phần nguyên = 1
- 0.375 x 2 = 0.75 phần nguyên = 0
- 0.75 x 2 = 1.5 phần nguyên = 1
- 0.5 x 2 = 1.0 phần nguyên = 1

Kết quả: $0.6875_{(10)} = 0.1011_{(2)}$

Lưu ý khi gọi hàm trong MIPS:

$a0-$a3: tham số truyền vào hàm (ori, move)

jal tên_hàm #nhảy đến hàm

$v0-$v1: giá trị mà hàm trả về

jr $ra #nhảy về

```
do{
(1)
}while(mệnh đề)

do:
        (1)
        if(!mệnh đề), exit
        j do
exit:
```

```
i=0;
while(điều kiện chạy)
{ (1)  i++; }
ori     $s3, $zero , 0
for:    if(!mệnh đề chạy), exit
        (1)
        addi $s3, $s3, 1
        j for
exit:
```

```
for(int i = 0; điều kiện chạy; i++){
        (1)
}
ori     $s3, $zero , 0
for:    if(!mệnh đề chạy), exit
        (1)
        addi $s3, $s3, 1
        j for
exit:
```

Với i = $s3, j = $s4

```
if (i == j)
(1)
else (2)

bne $s3, $s4, else
(1)
j exit
else: (2)
exit:
```

```
if (i != j)
(1)
else (2)

beq $s3, $s4, else
(1)
j exit
else: (2)
exit:
```

```
if (i < j)
(1)
else (2)

slt $t0, $s3, $s4
beq $t0, $zero, else
(1)
j exit
else: (2)
exit:
```

```
if (i <= j)
(1)
else (2)

slt $t0, $s4, $s3 # i > j
bne $t0, $zero, else
(1)
j exit
else: (2)
exit:
```

```
if (i > j)
(1)
else (2)

slt $t0, $s4, $s3 # i > j
beq $t0, $zero, else
(1)
j exit
else: (2)
exit:
```

```
if (i >= j)
(1)
else (2)

slt $t0, $s3, $s4 #i < j
bne $t0, $zero, else
(1)
j exit
else: (2)
exit:
```

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

R-Format

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

I-Format và J-Format

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

op	address
6 bits	26 bits

Hệ thập lục phân (16)

4 bit	Chữ số Hex	4 bit	Chữ số Hex
0000	0x0	1000	0x8
0001	0x1	1001	0x9
0010	0x2	1010	0xA
0011	0x3	1011	0xB
0100	0x4	1100	0xC
0101	0x5	1101	0xD
0110	0x6	1110	0xE
0111	0x7	1111	0xF

Hệ bát phân (8)

3 bit	Bát phân
000	0_8
001	1_8
010	2_8
011	3_8
100	4_8
101	5_8
110	6_8
111	7_8

Thanh ghi $a0 - $a3 (argument registers):

Thanh ghi	Số hiệu thanh ghi	Mã nhị phân
$a0	4	00100
$a1	5	00101
$a2	6	00110
$a3	7	00111

Thanh ghi $t0 - $t7 (temporary registers):

Thanh ghi	Số hiệu thanh ghi	Mã nhị phân
$t0	8	01000
$t1	9	01001
$t2	10	01010
$t3	11	01011
$t4	12	01100
$t5	13	01101
$t6	14	01110
$t7	15	01111

Thanh ghi $s0 - $s7 (saved registers):

Thanh ghi	Số hiệu thanh ghi	Mã nhị phân
$s0	16	10000
$s1	17	10001
$s2	18	10010
$s3	19	10011
$s4	20	10100
$s5	21	10101
$s6	22	10110
$s7	23	10111

Nếu A = $s1, i = $s2, **truy cập vào A[i]**

```
sll   $t0, $s2, 2
add   $t0, $t0, $s1
```

Donate:
TRAN MINH PHU

BẢNG TỔNG HỢP NHỮNG MÃ MIPS VÀ NHỊ PHÂN

Chú ý: Label = PC + 4 + 4 x offset (Imm lưu offset)

Category	Format	Instruction	Example	Meaning	Op (6 bits)	Rs (5 bits)	Rt (5 bits)	Imm (Address or constant) (16 bits)		
								Rd (5 bits)	Shamt (5 bits)	Funct (6 bits)
Arithmetic	R	add	add $s1, $s2, $s3	$s1 = $s2 + $s3	0	$s2	$s3	$s1	0	32 (100000)
	R	sub	sub $s1, $s2, $s3	$s1 = $s2 - $s3	0	$s2	$s3	$s1	0	34 (100010)
	I	Add immediate	addi $s1, $s2, 20	$s1 = $s2 + 20	8 (001000)	$s2	$s1	constant		
Data transfer	I	Load word	lw $s1, 20($s2)	$s1 = Mem[$s2 + 20]	35 (100011)	$s2	$s1	address		
	I	Store word	sw $s1, 20($s2)	Mem[$s2 + 20] = $s1	43 (101011)	$s2	$s1	address		
Logical	R(I)	and(i)	and $s1, $s2, $s3	$s1 = $s2 & $s3	0(12-001100)	$s2	$s3	$s1	0	36 (100100)
	R(I)	or(i)	or $s1, $s2, $s3	$s1 = $s2 | $s3	0(13-001101)	$s2	$s3	$s1	0	37 (100101)
	R(I)	xor(i)	xor $s1, $s2, $s3	$s1 = $s2 ^ $s3	0(14-001110)	$s2	$s3	$s1	0	38 (100110)
	R	nor	nor $s1, $s2, $s3	$s1 = ~($s2 | $s3)	0	$s2	$s3	$s1	0	39 (100111)
	R	Shift left	sll $s1, $s2, 20	$s1 = $s2 << 20	0	0	$s2	$s1	20	0 (000000)
	R	Shift right	srl $s1, $s2, 20	$s1 = $s2 >> 20	0	0	$s2	$s1	20	2 (000010)
Conditional branch	I	Branch on equal	beq $s1, $s2, offset	if($s1 == $s2) go to PC + 4 + offset * 4	4 (000100)	$s1	$s2	offset		
	I	Branch on not equal	bne $s1, $s2, offset	if($s1 != $s2) go to PC + 4 + offset * 4	5 (000101)	$s1	$s2	offset		
	R	Set on less than Set on less than unsigned	slt $s1, $s2, $s3 slu $s1, $s2, $s3	if($s2 < $s3) $s1 = 1; else $s1 = 0	0	$s2	$s3	$s1	0	(42) 101010 (43) 101011
	I	Set on less than immediate Set on less than immediate unsigned	slti $s1, $s2, 20 sltiu $s1, $s2, 20	if($s2 < 20) $s1 = 1; else $s1 = 0	10 (001010) 11 (001011)	$s2	$s1	20		
Unconditional jump	J	jump	j label	Nhảy đến mục tiêu cố định	2 (000010)	(26 bits) address (Label >> 2, bỏ 4 bit đầu)				
	R	Jump register	jr $ra	Nhảy đến thanh ghi có đ/c	0	$ra	0	0	0	8 (001000)
	J	Jump and link	jal label	Nhảy địa chỉ và lưu địa chỉ vào thanh ghi $ra	3 (000011)	(26 bits) address (Label >> 2, bỏ 4 bit đầu)				

Instruction	Form	op	funct	VD	Meaning
Add Imm Ued(addiu)	I	9	N/A	addiu [rt], [rs], Imm	Thực hiện phép công với một số mà không kiểm tra tràn.
Add unsigned (addu)	R	0	33	addu [rd], [rs], [rt]	Thực hiện phép cộng không kiểm tra tràn.
Load byte (lb)	I	32	N/A	lb [rt], 0([rs])	Lấy 1 byte dữ liệu từ bộ nhớ
Load byte ued (lbu)	I	36	N/A	lbu [rt], 0([rs])	Nạp 1 byte từ bộ nhớ vào thanh ghi đích (rt). Byte được nạp từ địa chỉ được tính bằng cách cộng giá trị immediate (16 bit) với nội dung của thanh ghi nguồn (rs). Byte được nạp sẽ được lưu vào byte thấp nhất của thanh ghi đích, các byte còn lại được gán bằng 0.
Load halfword (lh)	I	33	N/A	lh [rt], 0([rs])	Lấy 2 byte dữ liệu (halfword) từ bộ nhớ. Lưu ý: Giá trị halfword được load sẽ được coi là số có dấu.
Load halfword uned (lhu)	I	37	N/A	lhu [rt], 0([rs])	Nạp 2 byte (halfword) từ bộ nhớ vào thanh ghi đích (rt). Halfword được nạp từ địa chỉ được tính bằng cách cộng giá trị imm (16 bit) với nội dung của thanh ghi nguồn (rs). Halfword được nạp sẽ được lưu vào 2 byte thấp nhất của thanh ghi đích, các byte còn lại được gán bằng 0.
Load Linked (ll)	I	48	N/A	ll [rt], 0([rs])	Nạp 1 word (4 byte) từ bộ nhớ vào thanh ghi đích (rt). Lệnh này thường được sử dụng trong môi trường đa xử lý để đảm bảo dữ liệu được nạp một cách nguyên vẹn.
Load upper imm (lui)	I	15	N/A	lui [rt], Imm	Nạp giá trị immediate (16 bit) vào 16 bit cao của thanh ghi đích (rt), 16 bit thấp của thanh ghi đích được gán bằng 0.
Store byte (sb)	I	40	N/A	sb [rt], 0([rs])	Lưu 1 byte thấp nhất của thanh ghi nguồn (rt) vào bộ nhớ. Địa chỉ bộ nhớ được tính bằng cách cộng giá trị immediate (16 bit) với nội dung của thanh ghi nguồn (rs).
Store conditional (sc)	I	56	N/A	sc [rt], 0([rs])	Lưu 1 word (4 byte) từ thanh ghi nguồn (rt) vào bộ nhớ. Lệnh này thường được sử dụng trong môi trường đa xử lý để đảm bảo dữ liệu được lưu một cách nguyên vẹn.
Store halfword (sh)	I	41	N/A	sh [rt], 0([rs])	Lưu 2 byte thấp nhất của thanh ghi nguồn (rt) vào bộ nhớ. Địa chỉ bộ nhớ được tính bằng cách cộng giá trị immediate (16 bit) với nội dung của thanh ghi nguồn (rs).
Subtract ued (subu)	R	0	35	subu [rd], [rs], [rt]	Thực hiện phép trừ không kiểm tra tràn.
Sr arithmetic (sra)	R	0	3	sra [rd], [rt], Shamt	Dịch phải bit mà giữ nguyên dấu (Chú ý: [rs] = 0)
Move from hi (mfhi)	R	0	16	mfhi [rd]	Di chuyển nội dung của thanh ghi HI vào thanh ghi đích (rd). Thanh ghi HI thường được sử dụng để lưu trữ phần cao của kết quả phép chia (phần dư).
Move from lo (mflo)	R	0	18	mflo [rd]	Di chuyển nội dung của thanh ghi LO vào thanh ghi đích (rd). Thanh ghi LO thường được sử dụng để lưu trữ phần thấp của kết quả phép nhân (tích) hoặc phép chia (thương).
Multiply (mult)	R	0	24	mult [rs], [rt]	Nhân nội dung của hai thanh ghi nguồn (rs và rt). Kết quả 64 bit được lưu vào hai thanh ghi đặc biệt HI và LO.
Multiply ued (multu)	R	0	25	multu [rs], [rt]	Nhân không dấu của hai register nguồn (rs và rt). Kết quả 64 bit được lưu vào hai thanh ghi đặc biệt HI và LO.
Divide (div)	R	0	26	div [rs], [rt]	Chia nội dung của thanh ghi nguồn (rs) cho nội dung của thanh ghi nguồn (rt). Thương số được lưu vào thanh ghi LO, và phần dư được lưu vào thanh ghi HI.
Divide unsigned (divu)	R	0	27	divu [rs], [rt]	Chia nội dung của thanh ghi nguồn (rs) cho nội dung của thanh ghi nguồn (rt), coi cả hai toán hạng là số không dấu. Thương số được lưu vào thanh ghi LO, và phần dư được lưu vào thanh ghi HI.

MIPS Reference Data

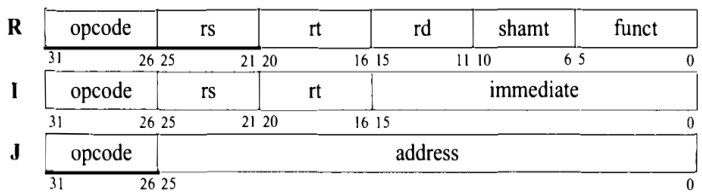
①

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] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + 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] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=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]+SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (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] | 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] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < 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] << shamt		$0 / 00_{hex}$
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		$0 / 02_{hex}$
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+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



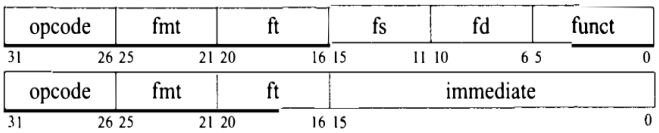
ARITHMETIC CORE INSTRUCTION SET

②

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	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
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*
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]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
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] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

(*x* is `eq`, `lt`, or `le`) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)

FLOATING-POINT INSTRUCTION FORMATS



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	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = 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

MỘT SỐ LỆNH GIẢ

Lệnh giả	Ý nghĩa	Chuyển đổi sang lệnh thực
move $d, $s	Sao chép giá trị từ $s sang $d.	add $d, $s, $zero
li $d, imm	Nạp một giá trị số nguyên imm vào thanh ghi $d.	Nếu imm 16-bit: addi $d, $zero, imm
		Nếu imm 32-bit: lui $at, upper(imm) và ori $d, $at, lower(imm)
la $d, address	Nạp địa chỉ address vào thanh ghi $d.	Nếu địa chỉ là 16-bit: addi $d, $zero, address
		Nếu 32-bit: lui $at, upper (address) và ori $d, $at, lower(address)
nop	Không làm gì cả, dùng để chờ (no operation).	sll $zero, $zero, 0
clear $d	Đặt giá trị thanh ghi $d về 0.	add $d, $zero, $zero
bgt $s, $t, label	Nhảy đến label nếu $s > $t.	slt $at, $t, $s
		bne $at, $zero, label
blt $s, $t, label	Nhảy đến label nếu $s < $t.	slt $at, $s, $t
		bne $at, $zero, label
bge $s, $t, label	Nhảy đến label nếu $s >= $t.	slt $at, $s, $t
		beq $at, $zero, label
ble $s, $t, label	Nhảy đến label nếu $s <= $t.	slt $at, $t, $s
		beq $at, $zero, label
abs $d, $s	Lấy giá trị tuyệt đối của $s và lưu vào $d.	slt $at, $s, $zero
		sub $d, $zero, $s
		movn $d, $s, $at
neg $d, $s	Lấy giá trị âm của $s và lưu vào $d.	sub $d, $zero, $s
not $d, $s	Lấy giá trị bit đảo của $s và lưu vào $d.	nor $d, $s, $zero

Lệnh giả	Ý nghĩa	Chuyển đổi sang lệnh thực
mul $d, $s, $t	Nhân $s và $t, lưu vào $d.	mult $s, $t
		mflo $d
div $d, $s, $t	Chia $s cho $t, lưu thương vào $d.	div $s, $t
		mflo $d
rem $d, $s, $t	Chia $s cho $t, lưu phần dư vào $d.	div $s, $t
		mfhi $d
push $s	Đẩy giá trị của $s vào stack.	addi $sp, $sp, -4
		sw $s, 0($sp)
pop $d	Lấy giá trị từ stack và lưu vào $d.	lw $d, 0($sp)
		addi $sp, $sp, 4
subi $d, $s, imm	Trừ đi giá trị imm từ $s và lưu vào $d.	addi $d, $s, -imm
beqz $s, label	Nhảy đến label nếu $s == 0.	beq $s, $zero, label
bnez $s, label	Nhảy đến label nếu $s != 0.	bne $s, $zero, label
seq $d, $s, $t	Đặt $d = 1 nếu $s == $t, ngược lại $d = 0.	sub $at, $s, $t
		sltiu $d, $at, 1
sne $d, $s, $t	Đặt $d = 1 nếu $s != $t, ngược lại $d = 0.	xor $d, $s, $t
		sltu $d, $zero, $d

NHÂN 2 SỐ TRÊN MÁY TÍNH

0100 x 0101 (4 x 5)

P/M là thanh ghi chép P và M

M_0 là bit thấp nhất của Mcand

Phổ thông

Lần lặp	Bước	Multiplier	Multiplicand	Product
0	Initial values	0101	0000 0100	0000 0000
1	1a: 1 => Prod = Prod + Mcand	0101	0000 0100	0000 0100
	2: Shift left Multiplicand	0101	0000 1000	0000 0100
	3: Shift right Multiplier	0010	0000 1000	0000 0100
2	1a: 0 => No operation	0010	0000 1000	0000 0100
	2: Shift left Multiplicand	0010	0001 0000	0000 0100
	3: Shift right Multiplier	0001	0001 0000	0000 0100
3	1a: 1 => Prod = Prod + Mcand	0001	0001 0000	0001 0100
	2: Shift left Multiplicand	0001	0010 0000	0001 0100
	3: Shift right Multiplier	0000	0010 0000	0001 0100
4	1a: 0 => No operation	0000	0010 0000	0001 0100
	2: Shift left Multiplicand	0000	0100 0000	0001 0100
	3: Shift right Multiplier	0000	0100 0000	0001 0100

Cải tiến

Lần lặp	Bước	Multiplicand	Product/Multiplier
0	Initial values	0100	0000 0101
1	1: $M_0 = [P/M]_0 = 1$	0100	0000 0101
	1a): Prod = Prod + Mcand	0100	0100 0101
	2: Shift right [P/M]	0100	0010 0010
2	1: $M_0 = [P/M]_0 = 0$	0100	0010 0010
	=> No operation	0100	0010 0010
	2: Shift right [P/M]	0100	0001 0001
3	1: $M_0 = [P/M]_0 = 1$	0100	0001 0001
	1a): Prod = Prod + Mcand	0100	0101 0001
	2: Shift right [P/M]	0100	0010 1000
4	1: $M_0 = [P/M]_0 = 0$	0100	0010 1000
	=> No operation	0100	0010 1000
	2: Shift right [P/M]	0100	0001 0100

- Bit multiplier:

+ Bit cuối là 0 => không công

+ Bit cuối là 1 => 1.a) P= P + Mcand

- Số vòng lặp = số bit biểu diễn

Chú ý:

Nếu nhân 2 số khác dấu thì chuyển 2 số thành 2 số tuyệt đối rồi nhân bình thường, kết quả sau đó chuyển về âm

Nếu nhân 2 số đều âm thì kết quả là 2 số tuyệt đối nhân với nhau

Có thể dùng phép XOR để xét dấu

Chia 2 số trên máy tính

Rem = Rem - Div

2b: Rem < 0 => +Div , sll Q, $Q_0 = 0$

2a: Rem ≥ 0 => sll Q, $Q_0 = 1$

1011 / 0010 (11 / 2)

Sll [R/Q] Rem = Rem - Div

2b: Rem < 0 => +Div

2a: Rem ≥ 0 => $Q_0 = 1$

Phổ thông

Lần lặp	Bước	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 1011
1	1: Rem = Rem - Div	0000	0010 0000	1110 1011
	2b: Rem < 0 => +Div , sll Q, $Q_0 = 0$	0000	0010 0000	0000 1011
	3: Shift right Div	0000	0001 0000	0000 1011
2	1: Rem = Rem - Div	0000	0001 0000	1111 1011
	2b: Rem < 0 => +Div , sll Q, $Q_0 = 0$	0000	0001 0000	0000 1011
	3: Shift right Div	0000	0000 1000	0000 1011
3	1: Rem = Rem - Div	0000	0000 1000	0000 0011
	2a: Rem ≥ 0 => sll Q, $Q_0 = 1$	0001	0000 1000	0000 0011
	3: Shift right Div	0001	0000 0100	0000 0011
4	1: Rem = Rem - Div	0001	0000 0100	1111 1111
	2b: Rem < 0 => +Div , sll Q, $Q_0 = 0$	0010	0000 0100	0000 0011
	3: Shift right Div	0010	0000 0010	0000 0011
5	1: Rem = Rem - Div	0010	0000 0010	0000 0001
	2a: Rem ≥ 0 => sll Q, $Q_0 = 1$	0101	0000 0010	0000 0001
	3: Shift right Div	0101	0000 0001	0000 0001

Số vòng lặp = số bit biểu diễn + 1

Cải tiến

Lần lặp	Bước	Divisor	R/Q
0	Initial values	0010	0000 1011
1	1: Sll [R/Q]	0010	0001 0110
	Rem = Rem - Div	0010	1111 0110
	2b: Rem < 0 => +Div	0010	0001 0110
2	1: Sll [R/Q]	0010	0010 1100
	Rem = Rem - Div	0010	0000 1100
	2a: Rem ≥ 0 => $Q_0 = 1$	0010	0000 1101
3	1: Sll [R/Q]	0010	0001 1010
	Rem = Rem - Div	0010	1111 1010
	2b: Rem < 0 => +Div	0010	0001 1010
4	1: Sll [R/Q]	0010	0011 0100
	Rem = Rem - Div	0010	0001 0100
	2a: Rem ≥ 0 => $Q_0 = 1$	0010	0001 0101

Số vòng lặp = số bit biểu diễn

Kết luận: Q = 0101, R = 0001

VD: 7/-3 xét 7/3 (lập bảng)

R cùng dấu với số bị chia => R dương (0001)

7 và –3 trái dấu => Q là số âm Q = $CP_2(|Q|)$ = 1110

a | **b**

r | **c**

a: Dividend (Số bị chia)

b: Divisor (Số chia)

c: Quotient (Thương số)

r: Remainder (Số dư)

Xét dấu khi chia 2 số có dấu:

- Dấu âm (-) khi dấu của số chia và số bị chia trái ngược nhau

- Dấu của số dư (cùng dấu với **số bị chia**)

a = b x c + r => r = a - b x c

Donate:
TRAN MINH PHU

Phép cộng trên dấu chấm động (significant 4 bits)

Bước 1: So sánh mũ (tăng mũ nhỏ bằng mũ lớn)

Bước 2: Cộng phần trị

Bước 3: Chuẩn hóa, xét tràn số mũ

Bước 4: Làm tròn tổng, kiểm tra lại

Ví dụ: 0,5 + -(0.4375)

$0.5 = 0.1_2 = 1.000 \times 2^{-1}$ $-0.4375 = -0.0111_2 = -1.110 \times 2^{-2}$

1) So sánh mũ (nhỏ ➞ lớn)

$-1.110 \times 2^{-2} = -0.111 \times 2^{-1}$

2) Cộng phần trị

$1.000 \times 2^{-1} + (-0.111 \times 2^{-1}) = (1.000 - 0.111) \times 2^{-1} = 0.001 \times 2^{-1}$

*** Lưu ý:**

+ Nếu số lớn trừ số nhỏ -> giữ nguyên

+ Nếu số nhỏ trừ số lớn -> - (số lớn – số nhỏ) ➞ để tránh bù hai.

VD: 3 – 5 -> - (5-3)

3) Chuẩn hóa tổng, xét tràn số mũ

$0.001 \times 2^{-1} = 1.000 \times 2^{-4}$

$y = -4 \Rightarrow E = y + 127 = -4 + 127 = 123 \in [1; 254]$ (Không tràn số mũ)

4) Làm tròn tổng

1.000×2^{-4} (không đổi)

Kiểm tra lại

$1.000 \times 2^{-4} = 0.0001 = 0.0625_{10}$ (0.5 + (-0.4375)

Phép nhân trên dấu chấm động (significant 4 bits)

Bước 1: Cộng số mũ

Bước 2: Nhân phần trị

Bước 3: Chuẩn hóa, xét tràn số mũ

Bước 4: Làm tròn

Bước 5: Xét dấu

Ví dụ: 0.5 x (-0.4375)

$0.5 = 0.1_2 = 1.000 \times 2^{-1}$ $-0.4375 = -0.0111_2 = -1.110 \times 2^{-2}$

1) Cộng phần mũ

$-1 + (-2) = -3$

$E = y + 127 = -3 + 127 = 124$

2) Nhân phần trị

$1.000 \times 1.110 = 1.110000$

$\Rightarrow 1.110000 \times 2^{-3}$ $\Rightarrow 1.110 \times 2^{-3}$

3) Chuẩn hóa tích, xét tràn số mũ

1.110×2^{-3} (không đổi)

$y = -3$ ➞ $E = y + 127 = -3 + 127 = 124 \in [1; 254]$

(Không tràn số mũ)

4) Làm tròn tích: 1.110×2^{-3} (không đổi)

5) Xét dấu: Hai thừa số trái dấu ➞ tích âm

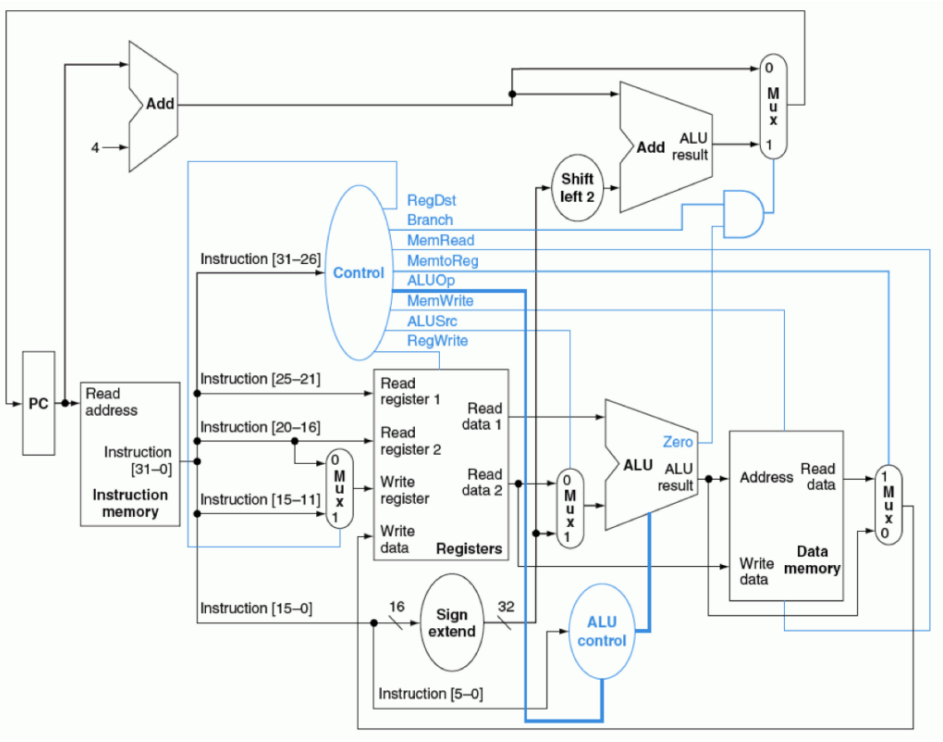
Tích = -1.110×2^{-3}

Kiểm tra lại

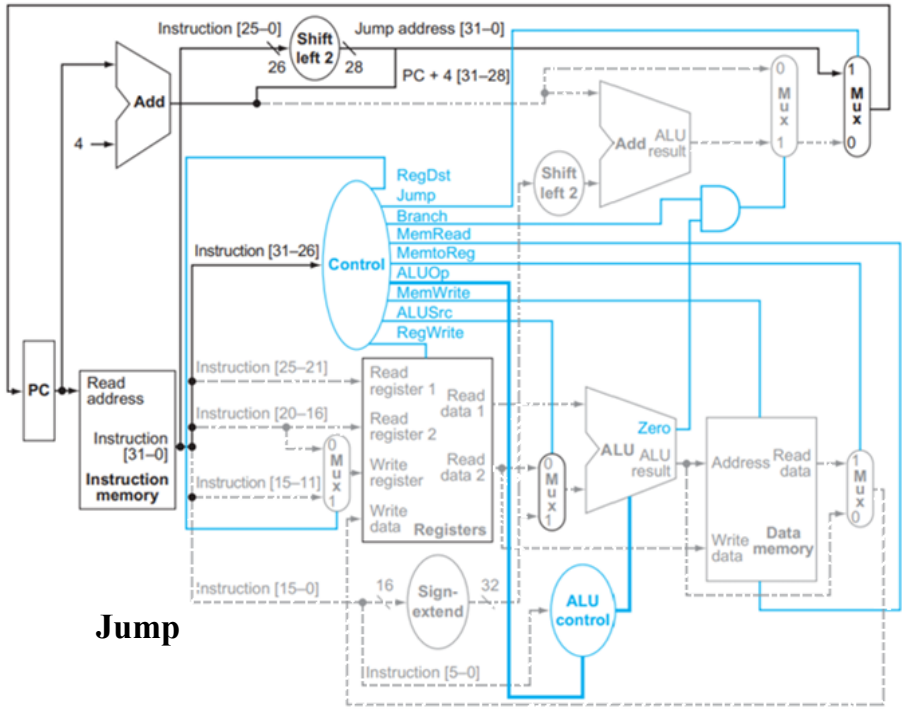
$-1.110 \times 2^{-3} = -0.001110 = -0.21875_{10}$

Bảng control: lấy 6 bit từ opcode để đưa ra các tín hiểu điều khiển tương ứng

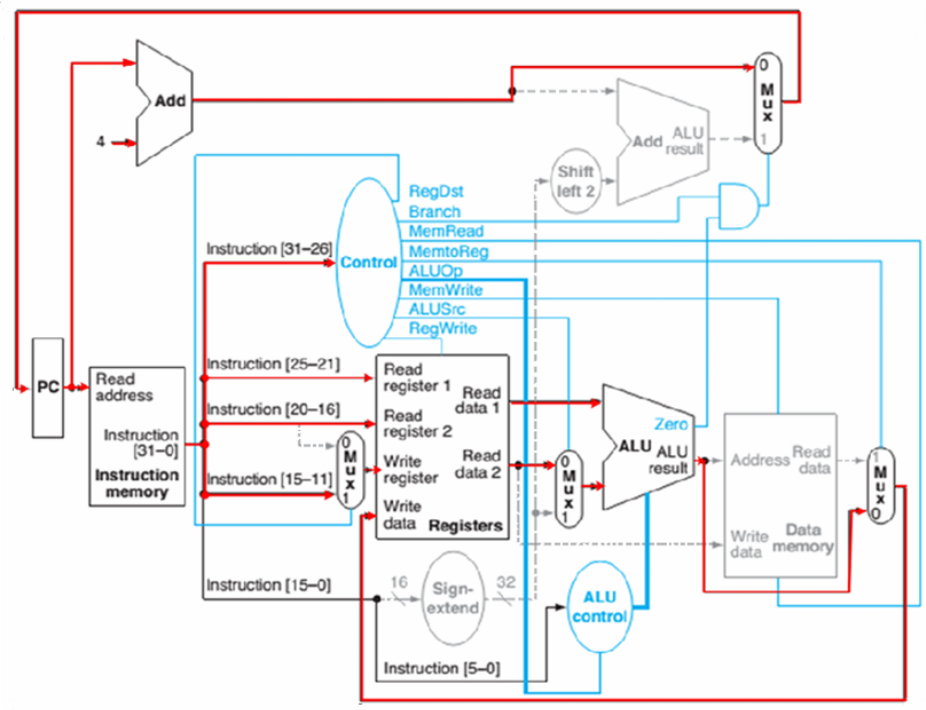
In-struction	Reg Dst	ALU Src	Mem to Reg	Reg Write	Mem Read	Mem Write	Branch	ALU Op	Jump
R Type	1	0	0	1	0	0	0	10	X
lw	0	1	1	1	1	0	0	00	X
sw	X	1	X	0	0	1	0	00	X
beq	X	0	X	0	0	0	1	01	X
Jump	X	X	X	0	0	0	0	XX	1



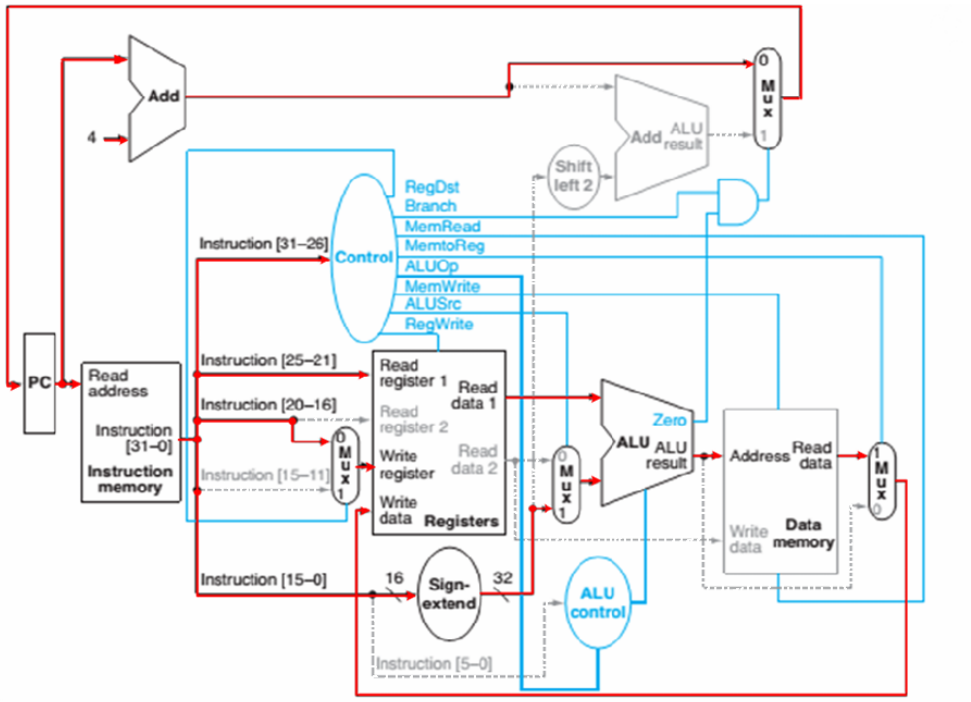
Instruction opcode	ALUop	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-Type	10	add	100000	add	0010
R-Type	10	subtract	100010	subtract	0110
R-Type	10	AND	100010	AND	0000
R-Type	10	OR	100101	OR	0001
R-Type	10	set on less	101010	set on less	0111



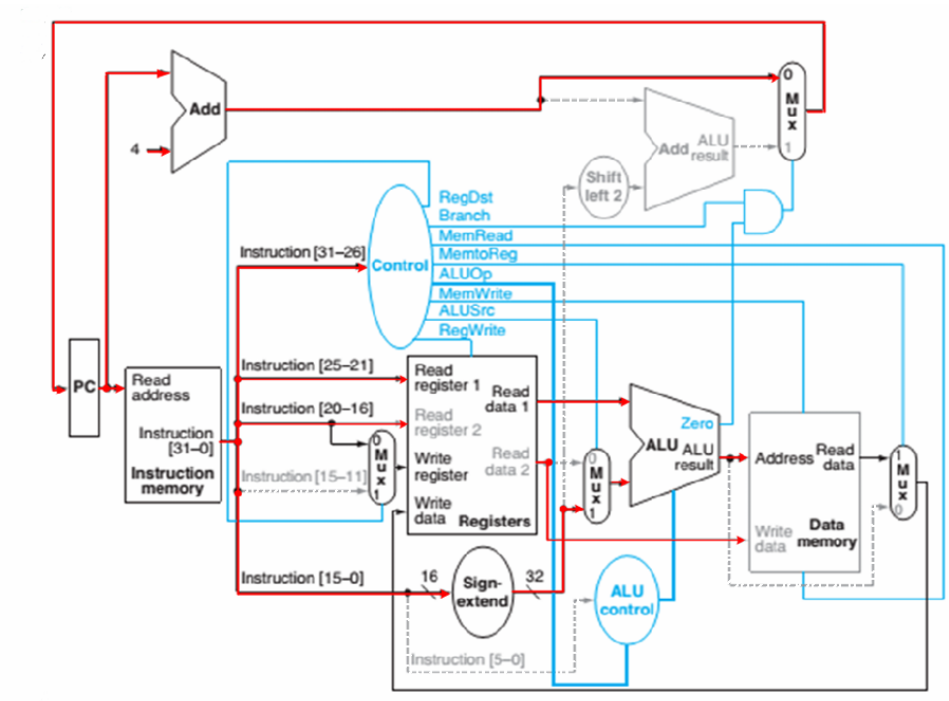
Jump



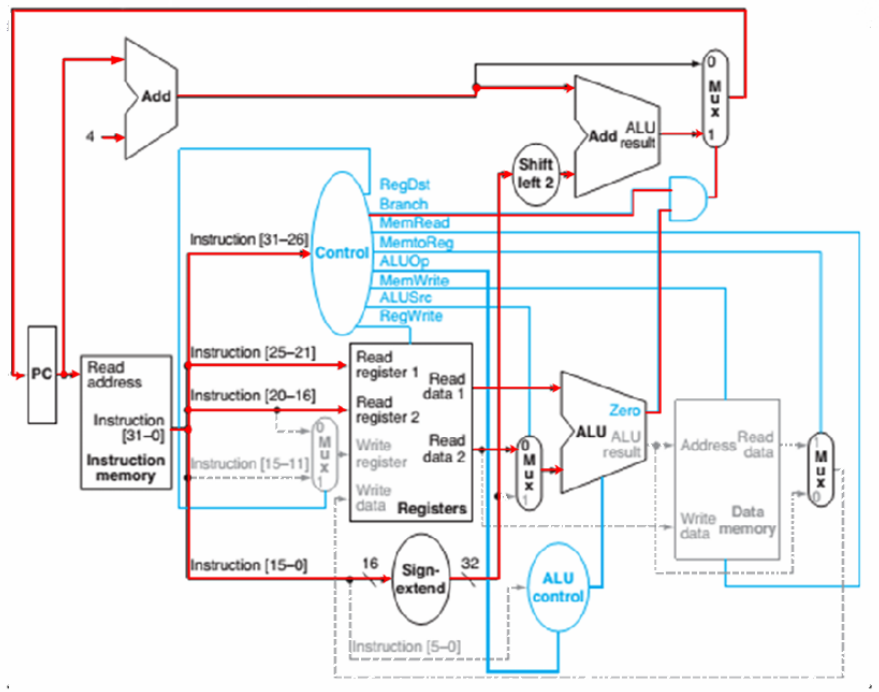
*Các đường đỏ là các đường hoạt động khi lệnh thuộc **nhóm logic và số học** thực thi*



*Các đường đỏ là các đường hoạt động khi lệnh **lw** thực thi*



*Các đường đậm nét (đỏ) là các đường hoạt động khi lệnh **sw** thực thi*



*Các đường đậm nét (đỏ) là các đường hoạt động khi lệnh **beq** thực thi*

PINELINE:

Thời gian giữa 2 lệnh liên tiếp pineline (lý tưởng) = $\frac{thời\ gian\ giữa\ 2\ lệnh\ liên\ tiếp\ không\ pineline}{số\ tầng\ pineline}$

Thực tế thì thời gian giữa 2 lệnh liên tiếp tùy thuộc vào phải chọn công đoạn dài nhất để làm chu kì pineline

Speed up $\approx \frac{\text{thời gian giữa 2 lần liên tiếp không pineline}}{\text{thời gian giữa hai lệnh liên tiếp pineline}}$ tăng tốc ≤ số tầng pineline

Speed up = $\frac{n \times k \times t_c}{(n+k-1)\times t_c} = \frac{n.T_{single}}{(k+n-1).T_{pineline}}$ Với k và số tầng, n là số lượng lệnh

Số chu kì khi sử dụng pineline = (k + n – 1)

Lệnh add: + Chờ 2 chu kì xung clock (nếu đã có kĩ thuật ghi nửa chu kì đầu và đọc cuối chu kì)

+ Dùng kĩ thuật nhìn trước: không cần ngưng

Lệnh lw: + Chờ 2 chu kì xung clock (nếu đã có kĩ thuật ghi nửa chu kì đầu và đọc cuối chu kì)

+ Dùng kĩ thuật nhìn trước: vẫn cần phải chờ 1 chu kì xung clock

Lệnh beq: + Chờ một chu kì clock để chờ điều kiện bằng xảy ra

+ Cải tiến bằng phương pháp dự đoán, chương trình sẽ không lãng phí hoặc lãng phí 1 chu kì xung clock

*** Các loại xung đột:**

- Xung đột cấu trúc: + Không thể truy xuất bộ nhớ lấy dữ liệu (Mem) vừa thực hiện truy xuất bộ nhớ lấy lệnh (IF) (lệnh **lw**)

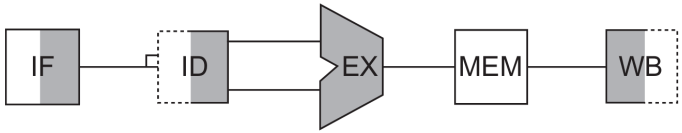
+ Không thể truy cập vào bộ nhớ dữ liệu hoặc ghi dữ liệu cùng một lúc (trong pineline bình thường không thể xảy xa)

+ Sử dụng nhiều ALU trong một chu kì

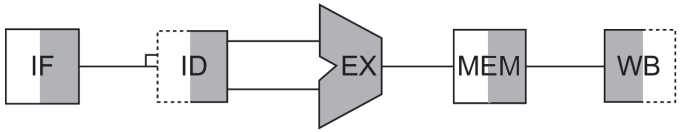
+ Một lệnh đang ghi dữ liệu vào thanh ghi và lệnh khác đang đọc dữ liệu từ thanh ghi đó.

+ Ghi dữ liệu vào cùng một thanh ghi

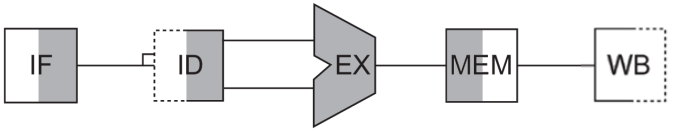
- Xung đột dữ liệu: + RAW: một lệnh sau đọc dữ liệu rồi mà lệnh trước đó chưa hoàn thành việc ghi vào chính thanh ghi đó



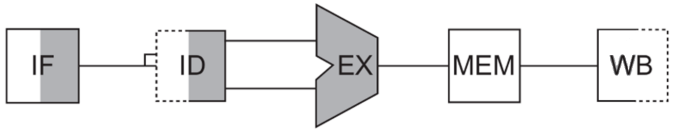
Lệnh add



Lệnh lw



Lệnh sw

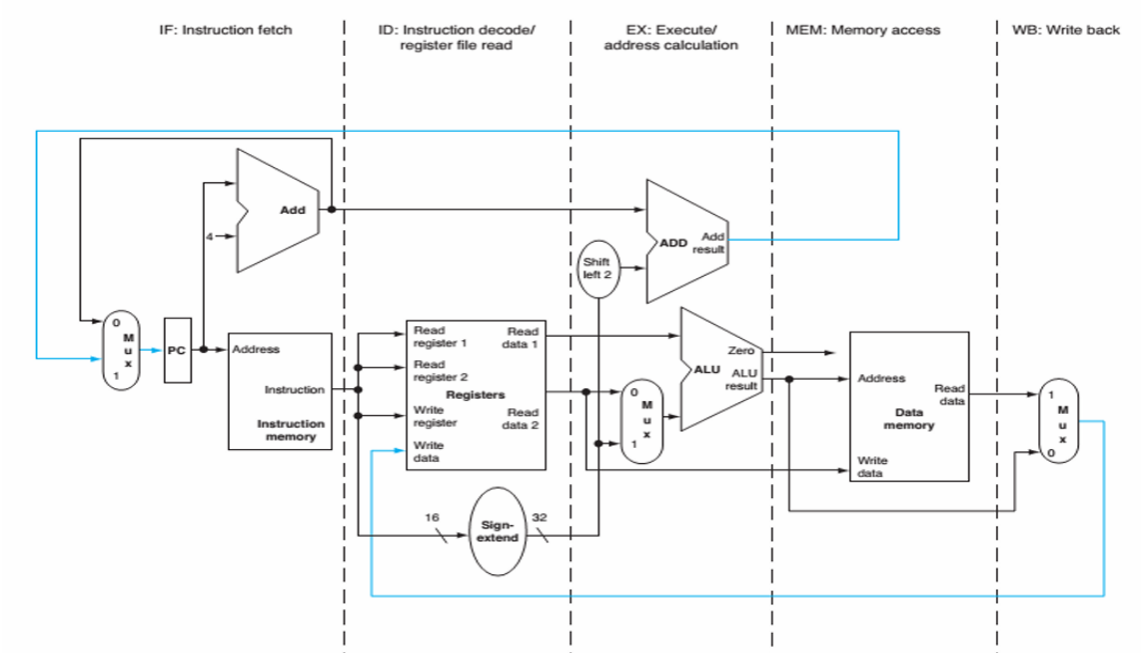


Lệnh beq

+ WAR: ghi ở lệnh sau mà lệnh trước đó chưa đọc xong

+ WAW: cả 2 lệnh đều muốn ghi vào một thanh ghi, ghi cùng lúc có thể dẫn đến xung đột

+ Xung đột điều khiển: Thường gặp ở lệnh rẽ nhanh. Lệnh tiếp theo chưa biết cần phải chạy hay không.



1. Nạp lệnh từ bộ nhớ – **IF**
2. Giải mã lệnh và đọc các thanh ghi – **ID**
3. Thực thi – **EX**
4. Truy xuất bộ nhớ – **MEM**
5. Ghi kết quả vào thanh ghi – **WB**

$T_{Single\text{-}cycle}$ = Tất cả các giai đoạn của **một lệnh** công lại

(đường đi tốn nhiều thời gian nhất)

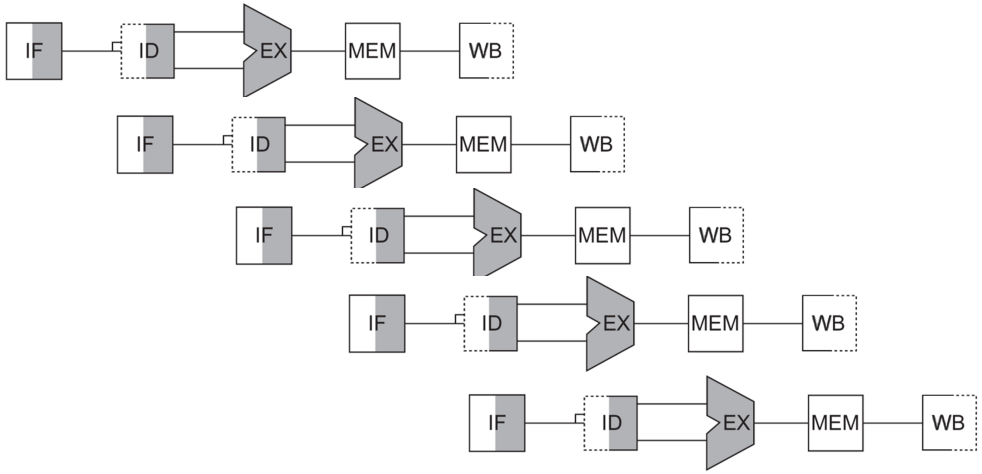
➞ Nếu chương trình có nhiều lệnh thì T phụ thuộc vào **lệnh chậm nhất** trong hệ thống

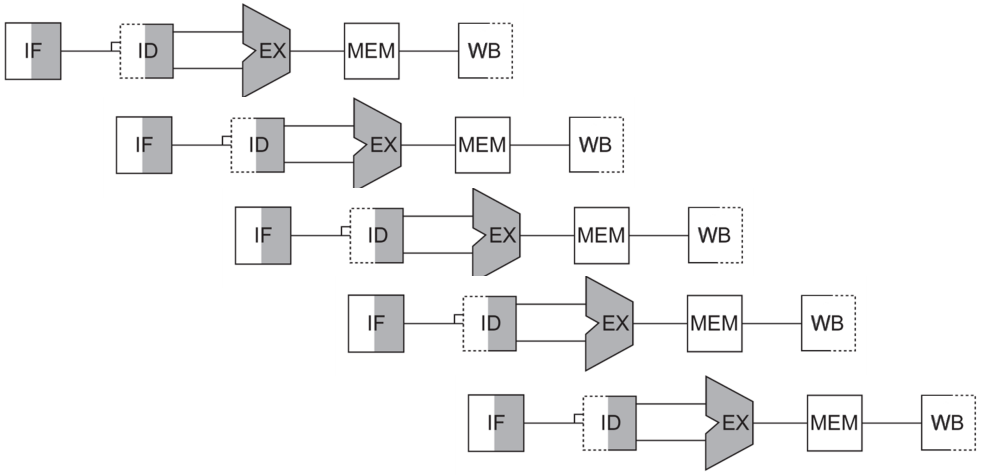
$T_{Pineline\text{-}cycle}$ = Công đoạn chậm chất = $\max(T_{Fetch}, T_{Decode}, T_{Execute}, T_{Memory}, T_{Write\text{-}back})$

Thời gian hoàn thành lệnh đầu tiên: EX = Số công đoạn x T

Lưu ý: Nếu đề chỉ ghi thời gian của Reg (Registers) thì nó là thời gian ID (Instruction decode) và WB (Write back)

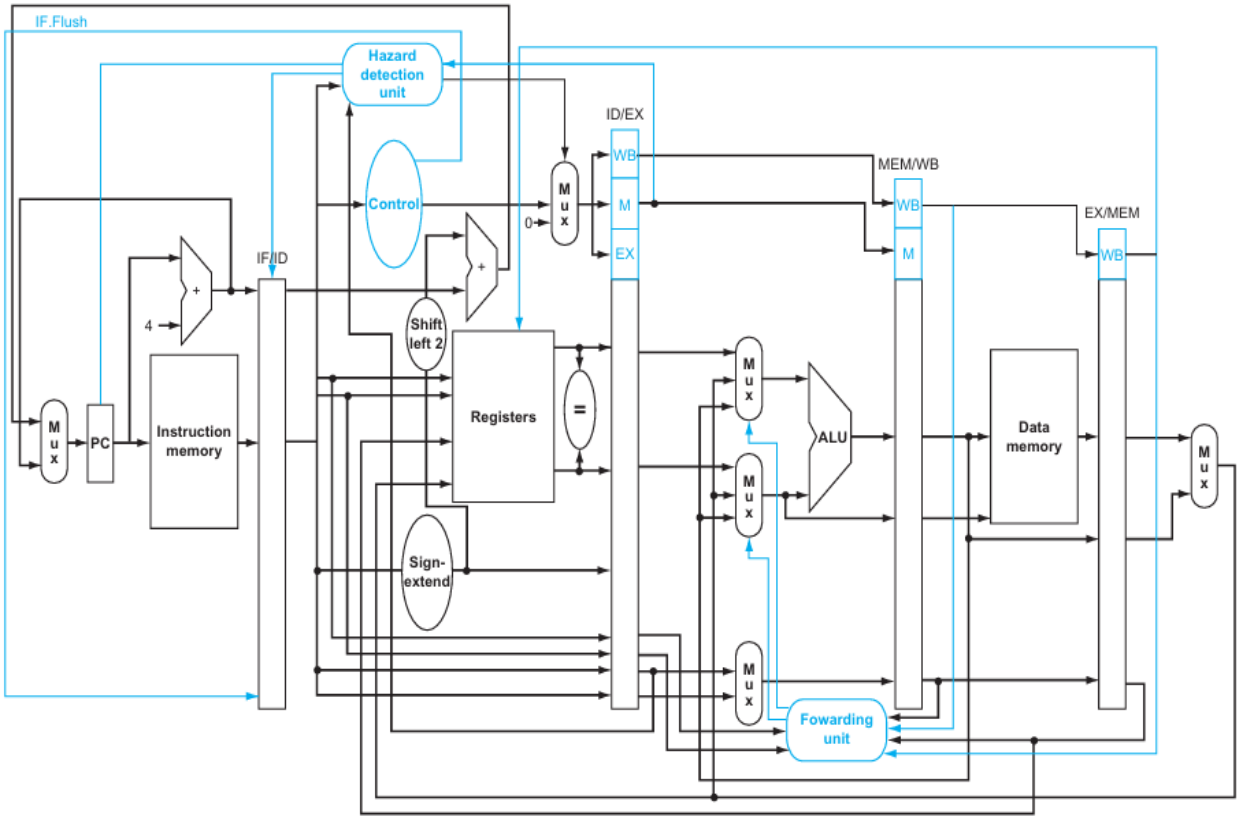
VD: Nếu Reg 100ps thì ID 100ps và WB 100ps





Sơ đồ để nháp khi thi pineline

ĐƯỜNG DÂY DATAPATH ĐÃ PINELINE



Phân biệt liên kết toàn phần, ánh xạ trực tiếp và liên kết tập hợp

Đặc điểm	Liên kết Toàn phần	Ánh xạ Trực tiếp	Liên kết Tập hợp
Vị trí lưu trữ	Bất kỳ dòng nào trong bộ nhớ đệm	Mỗi khối bộ nhớ ánh xạ đến một dòng duy nhất	Mỗi khối ánh xạ đến một tập, có thể vào nhiều dòng trong tập
Tìm kiếm	Tìm tất cả các dòng để tìm thẻ khớp	Chỉ cần kiểm tra một dòng duy nhất	Chỉ kiểm tra trong một tập nhất định
Khả năng xung đột	***Ánh xạ mềm, ít xung đột*** vì khối có thể lưu ở bất kỳ đâu	***Ánh xạ cứng, dễ bị xung đột*** nếu nhiều khối ánh xạ đến cùng dòng	Ánh xạ mềm, trung bình, ít xung đột hơn ánh xạ trực tiếp
Hiệu suất	Tỷ lệ ***lỗi bộ nhớ đệm thấp*** nhưng thời gian truy cập có thể ***chậm*** hơn	Tỷ lệ ***lỗi bộ nhớ đệm cao*** hơn nhưng truy cập ***nhanh*** hơn	Hiệu suất tốt, giảm xung đột với chi phí hợp lý
Chi phí triển khai	Cao, yêu cầu phần cứng ***phức tạp*** để tìm kiếm toàn bộ bộ nhớ đệm	Thấp, phần cứng ***đơn giản*** hơn	***Trung bình***, chi phí hợp lý giữa hai loại trên

Nguyên tắc khi sử dụng hàm đệ quy hoặc hàm con trong hàm chính

- Trước khi gọi hàm con thì lưu địa chỉ cũ trước, sau khi gọi hàm con xong thì lấy địa chỉ về lại thanh ghi $ra

Phải lưu địa chỉ khi trong hàm con có một hàm con khác

addi $sp, $sp, -4 lw $ra, 0($sp)

sw $ra, 0($sp) addi $sp, $sp, 4

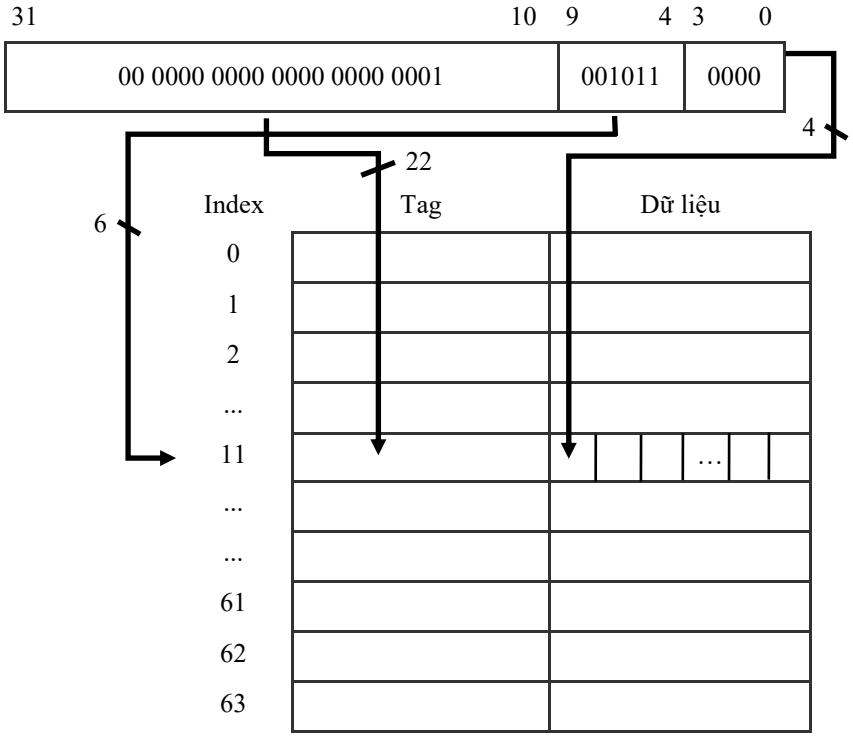
(hàm con) jr $ra

Ví dụ:

```
factorial:  # Lưu $ra lên stack
            addi $sp, $sp, -4       # Đẩy stack xuống
            sw $ra, 0($sp)          # Lưu giá trị của $ra
            # Điều kiện dừng: nếu n == 0, trả về 1
            beq $a0, $zero, base_case
            # Giảm n và gọi đệ quy factorial(n-1)
            addi $a0, $a0, -1
            jal factorial
            # Lấy kết quả factorial(n-1) từ $v0, nhân với n
            lw $ra, 0($sp)          # Khôi phục $ra
            addi $sp, $sp, 4        # Tăng stack lên
            #thực hiện phép return
            mul $v0, $v0, $a0       # n * factorial(n-1)
            jr $ra                  # Quay lại
base_case:  li $v0, 1               # Trả về 1
            lw $ra, 0($sp)          # Khôi phục $ra
            addi $sp, $sp, 4        # Tăng stack lên
            jr $ra                  # Quay lại
```

ÁNH XẠ TRỰC TIẾP

Ví dụ đề bài: Ánh xạ trực tiếp, 64 dòng cache (6 bits), kích thước một dòng 16 byte (4 bits) (do mỗi ô 1 byte) cho biết vị trí cần tham chiếu, bộ nhớ 1200 được ánh xạ vào cache



Chú ý: 1 word (32 bit) = 4 byte, cho nên dễ bị nhầm lẫn

nếu Offset 4 bits = 32 ô nhớ (tức là 32 bytes) = 8 words

Phân biệt miss/hit:

Chỉ **hit** khi mà đã có dữ liệu đã được nạp vào cache rồi, tức là khi tới đọc địa chỉ tiếp theo mà thấy cùng index với cái trước đó gần nhất, ta tiếp tục xem nó có cùng tag hay không, nếu cùng tag là **hit**.

Cách trình bày dạng bài tập:

Caches are important to providing a high-performance memory hierarchy to processors. Below is a list of 32-bit memory address references, given as word addresses. 3, 180, 43, 2, 191, 88, 190, 14, 181, 44, 186, 253

For each of these references, identify the binary address, the tag, and the index given a direct-mapped cache with two-word blocks (1 khối có 2 word) and a total size of 8 blocks (8 khối). Also list if each reference is a hit or a miss, assuming the cache is initially empty.

Giải

2 words/block ➞ offset 3 bits (do mỗi word là 4 bytes) $(2^3 = 8)$

8 blocks ➞ index 3 bits $(2^3 = 8)$

32 - 3 - 3 = 26 bits tag

Word Address	Binary Address	Tag	Index	Hit/Miss
3	0000 0011	0	1	M
180	1011 0100	11	2	M
43	0010 1011	2	5	M
2	0000 0010	0	1	H
191	1011 1111	11	7	M
88	0101 1000	5	4	M
190	1011 1110	11	7	H
14	0000 1110	0	7	M
181	1011 0101	11	2	H
44	0010 1100	2	6	M
186	1011 1010	11	5	M
253	1111 1101	15	6	M

LIÊN KẾT TẬP HỢP

Nếu có n byte kích thước trong một dòng, có 16 dòng, nếu chia thành 2 tập hợp (2 đường). Thì tức là trong một set sẽ có 2 dòng. Ta lấy số dòng cache/số đường là ra 8 sets. Nếu có một địa chỉ ánh xạ vào thì có thể vào đường 1 hoặc đường 2 (nếu set đó đầy) tùy vào thuật toán như:

+ LRU (Least Recently Used) thay thế dòng cache ít sử dụng,

+ LFU (Least Frequently Used) giữ lại khối có tần số sử dụng cao

+ FIFO (First In First Out): Vào trước ra trước, kĩ thuật hàng đợi

Set 0	0	Mem[8]
	1	
Set 1	0	Mem[9]
	1	
Set 2	0	Mem[2]
	1	Mem[18]
Set 3	0	Mem[19]
	1	
Set 4	0	Mem[12]
	1	
Set 5	0	Mem[5]
	1	Mem[29]
Set 6	0	Mem[6]
	1	
Set 7	0	Mem[7]
	1	

*** Cách phân biệt miss/hit dành cho cả liên kết tập hợp và liên kết toàn phần:**

Khi khối đó vẫn còn trong set và chưa bị thay thế bởi khối khác thì là hit

*** Lưu ý:** Phải xem xét kĩ là loại thuật toán nào và thay thế hợp lý

- Bên liên kết toàn phần có thể trình bày theo kiểu khác, Mem[6] có thể vào line 2, line 3 tùy ý ...

LIÊN KẾT TOÀN PHẦN

Liên kết toàn phần cho phép các bạn bỏ vào ô block nào tùy thích, nếu chưa đầy thì bỏ vào chỗ trống, không được chồng vào cái đã có, nếu đầy thì xóa những block cũ theo như các thuật toán như LRU, LFU, FIFO

Ví dụ có 4 dòng cache, mỗi dòng có 2 bytes. Lần lượt cho các khối 3, 6, 5, 7, 8, 9, 7, 8, 3, ta được như sau. Nếu dùng thuật toán LRU ta được như sau

Giải

Mỗi dòng có 2 byte ➞ offset 1 bit Có 4 dòng cache ➞ index 2 bits

32 - 2 - 1 = 29 bits chứa tag

Block	Line 0	Line 1	Line 2	Line 3	Miss/hit
3	Mem[3]				M
6	Mem[3]	Mem[6]			M
5	Mem[3]	Mem[6]	Mem[5]		M
7	Mem[3]	Mem[6]	Mem[5]	Mem[7]	M
8	Mem[8]	Mem[6]	Mem[5]	Mem[7]	M
9	Mem[8]	Mem[9]	Mem[5]	Mem[7]	M
7	Mem[8]	Mem[9]	Mem[5]	Mem[7]	H
8	Mem[8]	Mem[9]	Mem[5]	Mem[7]	H
3	Mem[8]	Mem[9]	Mem[3]	Mem[7]	M

Tham gia classroom của mình (sẽ có đưa bài tập trong tương lai)

Chú ý: Chỉ vào lớp được bằng tài khoản cá nhân, không thể dùng tài khoản .edu

Bảng tra nhanh

Thập phân	Nhị phân	Thập phân	Nhị phân	Thập phân	Nhị phân	Thập phân	Nhị phân
-128	11 1000 0000	-96	11 1010 0000	-64	1100 0000	-32	1110 0000
-127	11 1000 0001	-95	11 1010 0001	-63	1100 0001	-31	1110 0001
-126	11 1000 0010	-94	11 1010 0010	-62	1100 0010	-30	1110 0010
-125	11 1000 0011	-93	11 1010 0011	-61	1100 0011	-29	1110 0011
-124	11 1000 0100	-92	11 1010 0100	-60	1100 0100	-28	1110 0100
-123	11 1000 0101	-91	11 1010 0101	-59	1100 0101	-27	1110 0101
-122	11 1000 0110	-90	11 1010 0110	-58	1100 0110	-26	1110 0110
-121	11 1000 0111	-89	11 1010 0111	-57	1100 0111	-25	1110 0111
-120	11 1000 1000	-88	11 1010 1000	-56	1100 1000	-24	1110 1000
-119	11 1000 1001	-87	11 1010 1001	-55	1100 1001	-23	1110 1001
-118	11 1000 1010	-86	11 1010 1010	-54	1100 1010	-22	1110 1010
-117	11 1000 1011	-85	11 1010 1011	-53	1100 1011	-21	1110 1011
-116	11 1000 1100	-84	11 1010 1100	-52	1100 1100	-20	1110 1100
-115	11 1000 1101	-83	11 1010 1101	-51	1100 1101	-19	1110 1101
-114	11 1000 1110	-82	11 1010 1110	-50	1100 1110	-18	1110 1110
-113	11 1000 1111	-81	11 1010 1111	-49	1100 1111	-17	1110 1111
-112	11 1001 0000	-80	11 1011 0000	-48	1101 0000	-16	1111 0000
-111	11 1001 0001	-79	11 1011 0001	-47	1101 0001	-15	1111 0001
-110	11 1001 0010	-78	11 1011 0010	-46	1101 0010	-14	1111 0010
-109	11 1001 0011	-77	11 1011 0011	-45	1101 0011	-13	1111 0011
-108	11 1001 0100	-76	11 1011 0100	-44	1101 0100	-12	1111 0100
-107	11 1001 0101	-75	11 1011 0101	-43	1101 0101	-11	1111 0101
-106	11 1001 0110	-74	11 1011 0110	-42	1101 0110	-10	1111 0110
-105	11 1001 0111	-73	11 1011 0111	-41	1101 0111	-9	1111 0111
-104	11 1001 1000	-72	11 1011 1000	-40	1101 1000	-8	1111 1000
-103	11 1001 1001	-71	11 1011 1001	-39	1101 1001	-7	1111 1001
-102	11 1001 1010	-70	11 1011 1010	-38	1101 1010	-6	1111 1010
-101	11 1001 1011	-69	11 1011 1011	-37	1101 1011	-5	1111 1011
-100	11 1001 1100	-68	11 1011 1100	-36	1101 1100	-4	1111 1100
-99	11 1001 1101	-67	11 1011 1101	-35	1101 1101	-3	1111 1101
-98	11 1001 1110	-66	11 1011 1110	-34	1101 1110	-2	1111 1110
-97	11 1001 1111	-65	11 1011 1111	-33	1101 1111	-1	1111 1111

Thập phân	Nhị phân	Thập phân	Nhị phân	Thập phân	Nhị phân	Thập phân	Nhị phân
0	0000 0000	32	0010 0000	64	0100 0000	96	0110 0000
1	0000 0001	33	0010 0001	65	0100 0001	97	0110 0001
2	0000 0010	34	0010 0010	66	0100 0010	98	0110 0010
3	0000 0011	35	0010 0011	67	0100 0011	99	0110 0011
4	0000 0100	36	0010 0100	68	0100 0100	100	0110 0100
5	0000 0101	37	0010 0101	69	0100 0101	101	0110 0101
6	0000 0110	38	0010 0110	70	0100 0110	102	0110 0110
7	0000 0111	39	0010 0111	71	0100 0111	103	0110 0111
8	0000 1000	40	0010 1000	72	0100 1000	104	0110 1000
9	0000 1001	41	0010 1001	73	0100 1001	105	0110 1001
10	0000 1010	42	0010 1010	74	0100 1010	106	0110 1010
11	0000 1011	43	0010 1011	75	0100 1011	107	0110 1011
12	0000 1100	44	0010 1100	76	0100 1100	108	0110 1100
13	0000 1101	45	0010 1101	77	0100 1101	109	0110 1101
14	0000 1110	46	0010 1110	78	0100 1110	110	0110 1110
15	0000 1111	47	0010 1111	79	0100 1111	111	0110 1111
16	0001 0000	48	0011 0000	80	0101 0000	112	0111 0000
17	0001 0001	49	0011 0001	81	0101 0001	113	0111 0001
18	0001 0010	50	0011 0010	82	0101 0010	114	0111 0010
19	0001 0011	51	0011 0011	83	0101 0011	115	0111 0011
20	0001 0100	52	0011 0100	84	0101 0100	116	0111 0100
21	0001 0101	53	0011 0101	85	0101 0101	117	0111 0101
22	0001 0110	54	0011 0110	86	0101 0110	118	0111 0110
23	0001 0111	55	0011 0111	87	0101 0111	119	0111 0111
24	0001 1000	56	0011 1000	88	0101 1000	120	0111 1000
25	0001 1001	57	0011 1001	89	0101 1001	121	0111 1001
26	0001 1010	58	0011 1010	90	0101 1010	122	0111 1010
27	0001 1011	59	0011 1011	91	0101 1011	123	0111 1011
28	0001 1100	60	0011 1100	92	0101 1100	124	0111 1100
29	0001 1101	61	0011 1101	93	0101 1101	125	0111 1101
30	0001 1110	62	0011 1110	94	0101 1110	126	0111 1110
31	0001 1111	63	0011 1111	95	0101 1111	127	0111 1111

Thập phân	Nhị phân	Thập phân	Nhị phân	Thập phân	Nhị phân	Thập phân	Nhị phân
128	1000 0000	160	1010 0000	192	1100 0000	224	1110 0000
129	1000 0001	161	1010 0001	193	1100 0001	225	1110 0001
130	1000 0010	162	1010 0010	194	1100 0010	226	1110 0010
131	1000 0011	163	1010 0011	195	1100 0011	227	1110 0011
132	1000 0100	164	1010 0100	196	1100 0100	228	1110 0100
133	1000 0101	165	1010 0101	197	1100 0101	229	1110 0101
134	1000 0110	166	1010 0110	198	1100 0110	230	1110 0110
135	1000 0111	167	1010 0111	199	1100 0111	231	1110 0111
136	1000 1000	168	1010 1000	200	1100 1000	232	1110 1000
137	1000 1001	169	1010 1001	201	1100 1001	233	1110 1001
138	1000 1010	170	1010 1010	202	1100 1010	234	1110 1010
139	1000 1011	171	1010 1011	203	1100 1011	235	1110 1011
140	1000 1100	172	1010 1100	204	1100 1100	236	1110 1100
141	1000 1101	173	1010 1101	205	1100 1101	237	1110 1101
142	1000 1110	174	1010 1110	206	1100 1110	238	1110 1110
143	1000 1111	175	1010 1111	207	1100 1111	239	1110 1111
144	1001 0000	176	1011 0000	208	1101 0000	240	1111 0000
145	1001 0001	177	1011 0001	209	1101 0001	241	1111 0001
146	1001 0010	178	1011 0010	210	1101 0010	242	1111 0010
147	1001 0011	179	1011 0011	211	1101 0011	243	1111 0011
148	1001 0100	180	1011 0100	212	1101 0100	244	1111 0100
149	1001 0101	181	1011 0101	213	1101 0101	245	1111 0101
150	1001 0110	182	1011 0110	214	1101 0110	246	1111 0110
151	1001 0111	183	1011 0111	215	1101 0111	247	1111 0111
152	1001 1000	184	1011 1000	216	1101 1000	248	1111 1000
153	1001 1001	185	1011 1001	217	1101 1001	249	1111 1001
154	1001 1010	186	1011 1010	218	1101 1010	250	1111 1010
155	1001 1011	187	1011 1011	219	1101 1011	251	1111 1011
156	1001 1100	188	1011 1100	220	1101 1100	252	1111 1100
157	1001 1101	189	1011 1101	221	1101 1101	253	1111 1101
158	1001 1110	190	1011 1110	222	1101 1110	254	1111 1110
159	1001 1111	191	1011 1111	223	1101 1111	255	1111 1111