

CS2100

<http://www.comp.nus.edu.sg/~cs2100/>

COMPUTER ORGANISATION

CS2100 Midterms

Reference Materials



NUS
National University
of Singapore

School of
Computing

9. ASCII Code (2/3)

■ ASCII table

'A': 1000001
(or 65₁₀)

LSBs	MSBs							
	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC ₁	!	1	A	Q	a	q
0010	STX	DC ₂	"	2	B	R	b	r
0011	ETX	DC ₃	#	3	C	S	c	s
0100	EOT	DC ₄	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	O	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

11.2 IEEE 754 Floating-Point Rep. (2/4)

- 3 components: **sign**, **exponent** and **mantissa (fraction)**



- The base (radix) is assumed to be 2.
- Two formats:
 - **Single-precision (32 bits)**: 1-bit sign, 8-bit exponent with bias 127 (excess-127), 23-bit mantissa
 - **Double-precision (64 bits)**: 1-bit sign, 11-bit exponent with bias 1023 (excess-1023), and 52-bit mantissa
- We will focus on the single-precision format
- Reading
 - DLD pages 32 - 33
 - IEEE standard 754 floating point numbers:
<http://steve.hollasch.net/cgindex/coding/ieeefloat.html>

4. MIPS Registers (Recap)

- For simplicity, register numbers (\$0, \$1, ..., \$31) will be used in examples here instead of register names

Name	Register number	Usage
\$zero	0	Constant value 0
\$v0-\$v1	2-3	Values for results and expression evaluation
\$a0-\$a3	4-7	Arguments
\$t0-\$t7	8-15	Temporaries
\$s0-\$s7	16-23	Program variables

Name	Register number	Usage
\$t8-\$t9	24-25	More temporaries
\$gp	28	Global pointer
\$sp	29	Stack pointer
\$fp	30	Frame pointer
\$ra	31	Return address

\$at (register 1) is reserved for the assembler.

\$k0-\$k1 (registers 26-27) are reserved for the operation system.

5. R-Format (1/2)

- Define fields with the following number of bits each:
 - $6 + 5 + 5 + 5 + 5 + 6 = 32$ bits

6	5	5	5	5	6
---	---	---	---	---	---

- Each field has a name:

opcode	rs	rt	rd	shamt	funct
--------	----	----	----	-------	-------

- Each field is an independent 5- or 6-bit unsigned integer
 - A 5-bit field can represent any number 0 – 31
 - A 6-bit field can represent any number 0 – 63

NOTE:

There are only 32 registers, so 5 bits for each register is enough.

Similarly, it is enough for **shamt** field since shifting 32 bits clears the entire register.

6. I-format (2/4)

- Define fields with the following number of bits each:
 - $6 + 5 + 5 + 16 = 32$ bits



- Again, each field has a name:



- Only one field is inconsistent with R-format.
 - `opcode`, `rs`, and `rt` are still in the same locations.

opcode	rs	rt	rd	shamt	funct
0	9	10	8	0	32

NOTE:

We merge `rd`, `shamt` and `funct` to form a 16-bit field used for a constant value (`immediate`). That is, $5 + 5 + 6 = 16$ bits. Just nice!

7. J-Format (2/5)

- Define fields of the following number of bits each:

6 bits	26 bits
--------	---------

- As usual, each field has a name:

opcode	target address
--------	----------------

- Keep **opcode** field identical to R-format and I-format for consistency
- Combine all other fields to make room for larger target address

3.2 Addressing Modes

- Addressing Mode:
 - Ways to specify an operand in an assembly language
- In MIPS, there are only 3 addressing modes:
 - **Register:**
 - Operand is in a register (eg: `add $t1, $t2, $t3`)
 - **Immediate:**
 - Operand is specified in the instruction directly (eg: `addi $t1, $t2, 98`)
 - **Displacement:**
 - Operand is in memory with address calculated as `Base + Offset` (eg: `lw $t1, 20($t2)`)

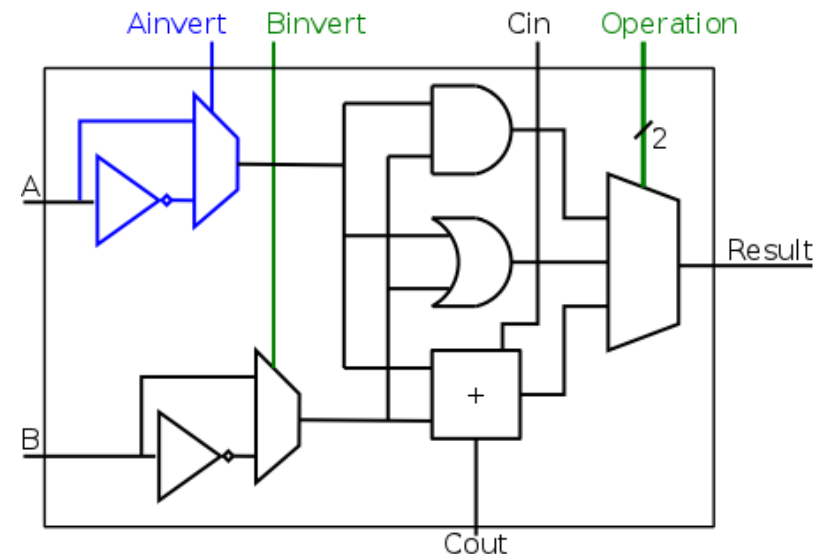
3.2 Addressing Modes: Others

<u>Addressing mode</u>	<u>Example</u>	<u>Meaning</u>
Register	Add R4,R3	$R4 \leftarrow R4 + R3$
Immediate	Add R4,#3	$R4 \leftarrow R4 + 3$
Displacement	Add R4,100(R1)	$R4 \leftarrow R4 + \text{Mem}[100 + R1]$
Register indirect	Add R4,(R1)	$R4 \leftarrow R4 + \text{Mem}[R1]$
Indexed / Base	Add R3,(R1+R2)	$R3 \leftarrow R3 + \text{Mem}[R1 + R2]$
Direct or absolute	Add R1,(1001)	$R1 \leftarrow R1 + \text{Mem}[1001]$
Memory indirect	Add R1,@(R3)	$R1 \leftarrow R1 + \text{Mem}[\text{Mem}[R3]]$
Auto-increment	Add R1,(R2)+	$R1 \leftarrow R1 + \text{Mem}[R2]; R2 \leftarrow R2 + d$
Auto-decrement	Add R1,-(R2)	$R2 \leftarrow R2 - d; R1 \leftarrow R1 + \text{Mem}[R2]$
Scaled	Add R1,100(R2)[R3]	$R1 \leftarrow R1 + \text{Mem}[100 + R2 + R3 * d]$

5. One Bit At A Time (Aha!)

- Can you see how the **ALUcontrol** (4-bit) signal controls the ALU?
 - Note: implementation for **slt** not shown

ALUcontrol			Function
Ainvert	Binvert	Operation	
0	0	00	AND
0	0	01	OR
0	0	10	add
0	1	10	subtract
0	1	11	slt
1	1	00	NOR



5. Generating ALUcontrol Signal

Opcode	ALUop	Instruction Operation	Funct field	ALU action	ALU control
lw	00	load word	xxxxxx	add	0010
sw	00	store word	xxxxxx	add	0010
beq	01	branch equal	xxxxxx	subtract	0110
R-type	10	add	10 0000	add	0010
R-type	10	subtract	10 0010	subtract	0110
R-type	10	AND	10 0100	AND	0000
R-type	10	OR	10 0101	OR	0001
R-type	10	set on less than	10 1010	set on less than	0111

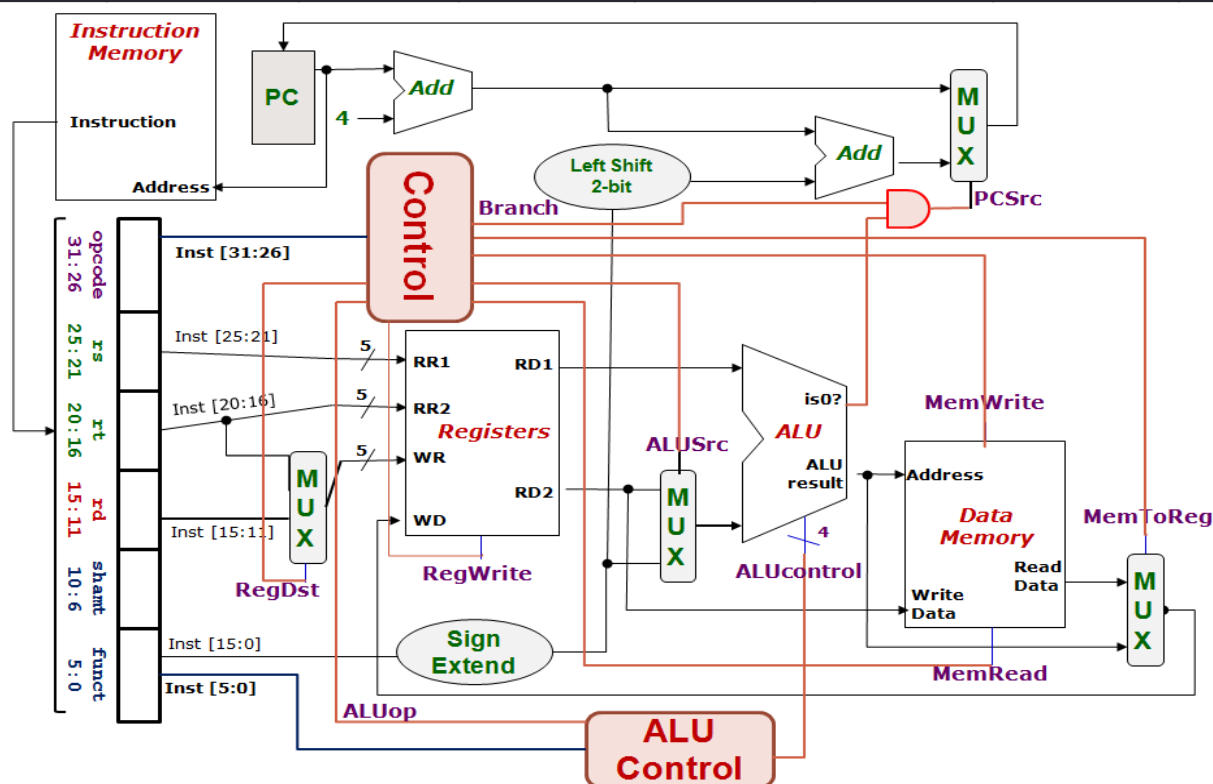
Instruction Type	ALUop
lw / sw	00
beq	01
R-type	10

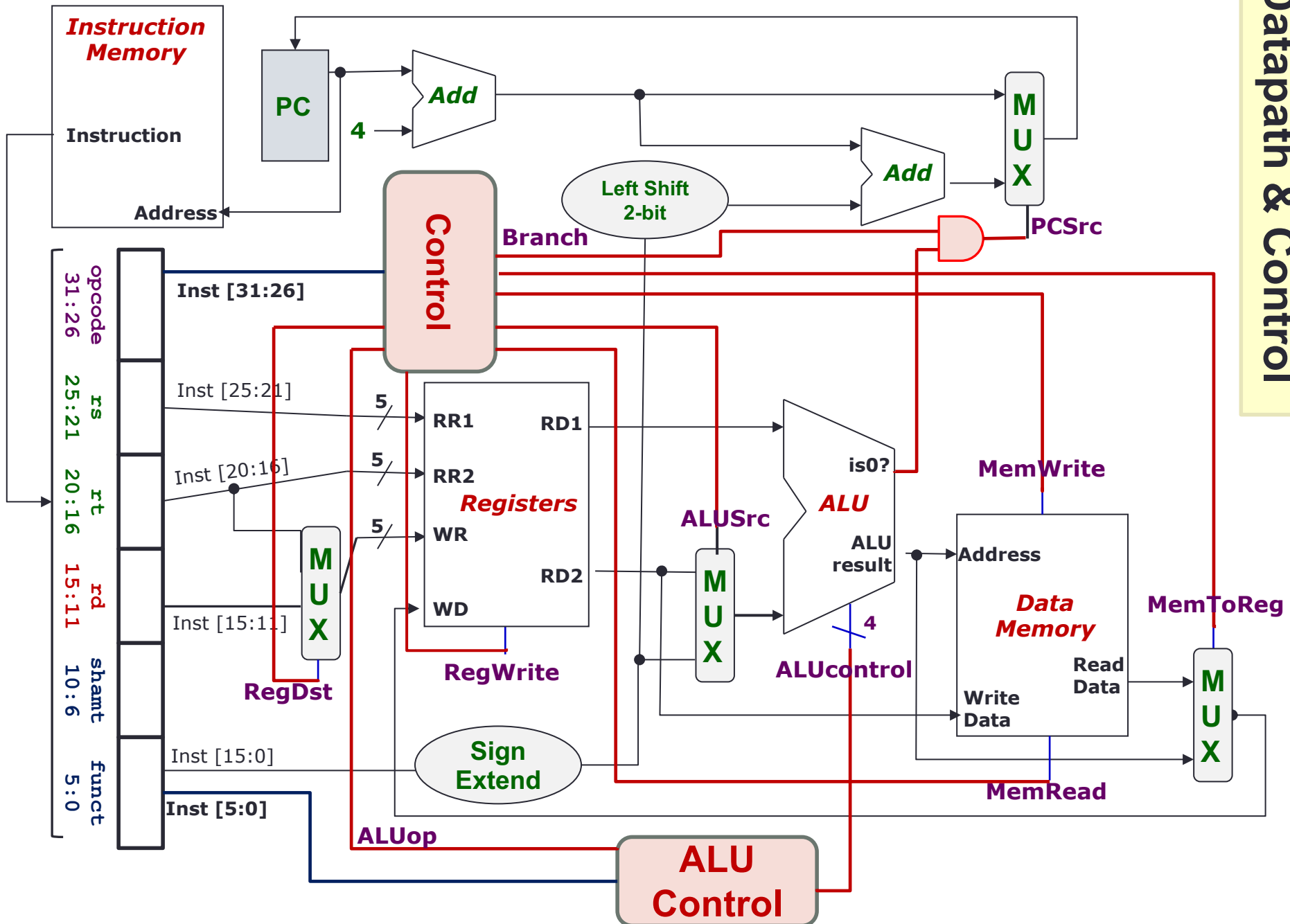
ALUcontrol	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR

Generation of 2-bit **ALUop** signal will be discussed later

5. Control Design: Outputs

	RegDst	ALUSrc	MemToReg	Reg Write	Mem Read	Mem Write	Branch	ALUop	
								op1	op0
R-type	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1





MIPS Reference Data

①



ARITHMETIC CORE INSTRUCTION SET

② OPCODE

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+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	$R[rt] = \{24'b0, M[R[rs]](7:0) + \text{SignExtImm}(7:0)\}$	(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		
	0					
J	opcode	address				
	31	26 25				0

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FUNCT (Hex)
Branch On FP True	bclt FI	if(FPcond)PC=PC+4+BranchAddr	(4) 11/8/1/--
Branch On FP False	bclfi 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
* (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/--/--/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	mghi R	$R[rd] = Hi$	0/--/--/10
Move From Lo	mflr 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	swdc1 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 = 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