CS202 Computer Organization

Midterm Examination, Spring 2021

Date: April 17, 2021 **Time: 14:00 – 16:00**

Student ID: **Name:______________________________**

1. (10 points). You are designing an embedded processor for a pacemaker. Based on an analysis of the monitoring software that it will run, you find the following mix of instructions, which have the specified execution time in your current design:

Instructions	Frequency	Time
Load	10%	7 cycles
Store	15%	10 cycles
Branch	15%	4 cycles
Add	55%	4 cycles
Multiply	5%	5 cycles

(a) (2 points) What is the CPI of your processor on this mix of instructions?

(b) (2 points) If the clock rate of your processor is 1GHz, and the total number of instructions of the software is 5000. Calculate the execution time for the software to run in your processor.

(c) (2 points) Based on your design analysis, you figure out that you can halve the cycle latency of any single category of instruction, although you will need to increase the cycle time by 20%. Should you make this change, and if so, what category of instruction should you speed up?

(d) (2 points) What is the CPI of your new design?

(e) (2 points) What is the speedup of your revised design over the original one?

2. (10 points) List the addressing modes of MIPS assembly instruction. For each addressing mode, please give one instruction as an example.

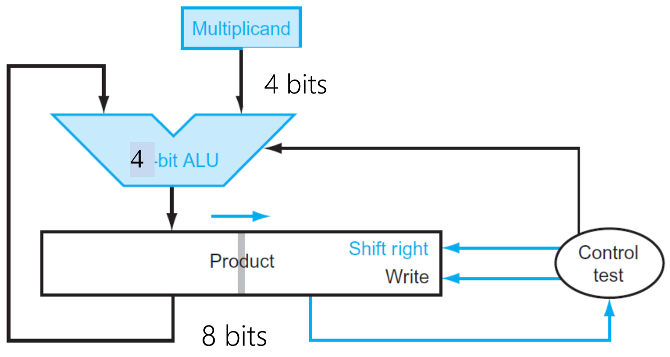
3. (10 points) For the following code:

```
Loop: sll $s1, $s1, 2
add $s2, $s2, $s1
bne $s2, $s3, Loop
```

Please give the 32-bit machine code for the three instructions above.

4. (10 points) Assuming that there are four integers, each of them is in an 8-bit register in format of 2's complement. r1=0xFE, r2=0xF1, r3=0x90, r4=0xF8. If the result is also in an 8-bit register, the multiply of which two registers will generate overflow? the addition of which two registers will generate overflow?

5. (10 points) Calculate the product of 6 × 5 using the hardware described below, with the multiplicand equals to 6. Both multiplicand and multiplier are unsigned 4-bit integers. Please show the contents of each register on each step and the final result.

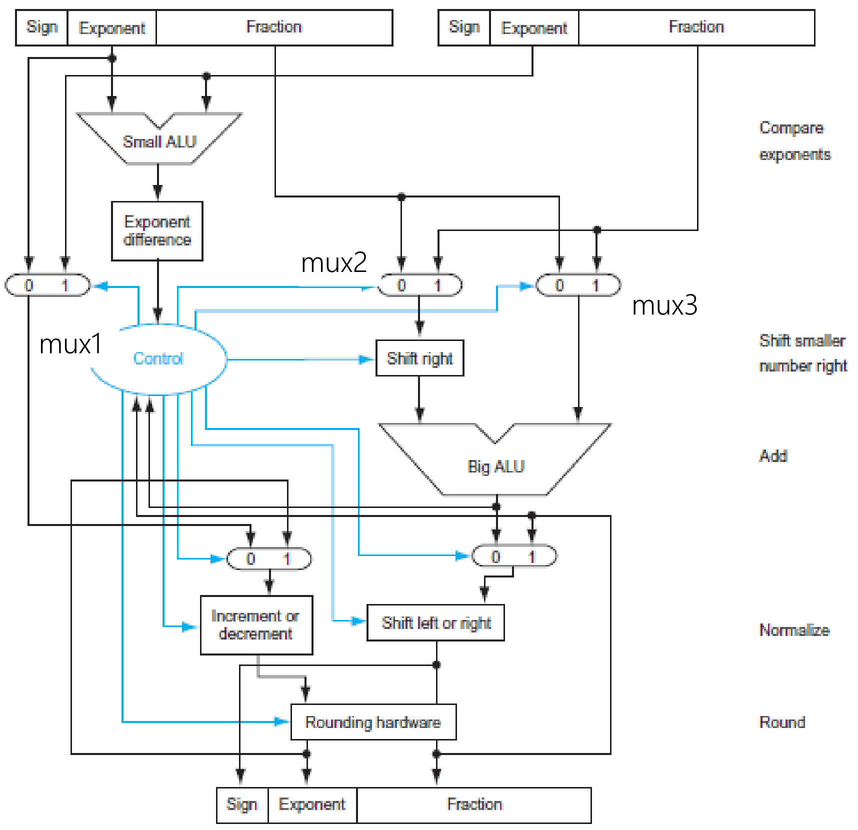


6. (10 points) Floating point number representation.

   a) Express negative floating point decimal number, -26.625, in the 14-bit floating point model which consists of 1 sign bit and 5 bits of exponent and 8 bits of fraction part. Exponent field uses 16 biased exponent. A hidden 1 is assumed in the fraction part.

   b) For the above mentioned 14-bit floating point number format, calculate the range and relative precision, assuming the all-one bit stream and all-zero bit stream in the exponent domain are reserved.

7. (10 points) In the following adder for floating points, assume the left floating-point number is -0.875, the right floating-point number is 1.5. Both numbers follow the 14-bit floating point number format defined in question 4. 1) write down the control input for mux1, mux2 and mux3. Please show the calculation procedure explaining how you get the results. 2) In the normalize stage, should the operation in the module "Increment or decrement" be increment or decrement? Should the operation in the module "shift left or right" be shifting left or right? If shift, by how many bits should it shift? Why? 3) How to determine whether the result is overflowed or underflowed or not?



8. (20 points) These questions refer to the simplified single-cycle MIPS32 datapath (full diagram supplied in the appendix of the test). Recall that this datapath supports the following instructions: add, sub, and, or, slt, lw, sw, beq and j.

a) (3 points) Consider the multiplexor, labelled 8a on the datapath diagram, that is controlled by the RegDst signal. Which of the supported instructions could not be executed correctly if RegDst was stuck at 0?

b) (3 points) Consider the Shift left 2 unit, labelled 8b on the datapath diagram. Which of the supported instructions depend on this unit for their correct operation?

c) (3 points) Consider the multiplexor labelled 8c, that is controlled by the ALUSrc signal. Which of the supported instructions could not be executed correctly if ALUSrc was stuck at 0?

d) (3 points) Suppose the RegWrite signal was stuck at 0. Which of the supported instructions could not be executed correctly?

e) (4 points) For some instruction(s), it does not matter whether the MemtoReg signal (in the right most multiplexor labelled 8e) is set to 0 or 1. For which instruction(s) is that true?

f) (4 points) For the modules PC， Instruction memory, Registers, ALU, MUX, Sign-extend, Data memory, which modules are combinatorial elements? which modules are state/sequential elements?

9. (10 points) Suppose a C programmer writes a C program with three functions foo(), bar() and zoo(); and a prototype MIPS compiler generates the following assembly code for each function.

C code (pseudo-code): Assembly code (with addresses and instructions)

```
                                        . . .
int foo() {                                  foo:
   . . .                                . . .
   x = bar (a, b);                      0x4000  9a  move  $a0, $t0
   printf("x: %d\n", x);                0x4004      move  $a1, $t1
   . . .                                0x4008      jal   bar
}                                       0x400c      move  $a0, $v0
                                        0x4010      li    $v0, 1
                                        0x4014      syscall
```

```
                                        . . .
                                        0x5000 bar: move  $t0, $a0
int bar(int a, int b) {                 0x5004      move  $a0, $a1
   x = zoo(b, a);                       0x5008      move  $a1, $t0
     // note params                     0x500c  9b  jal   zoo
   return x;                            0x5010      jr    $ra
}
```

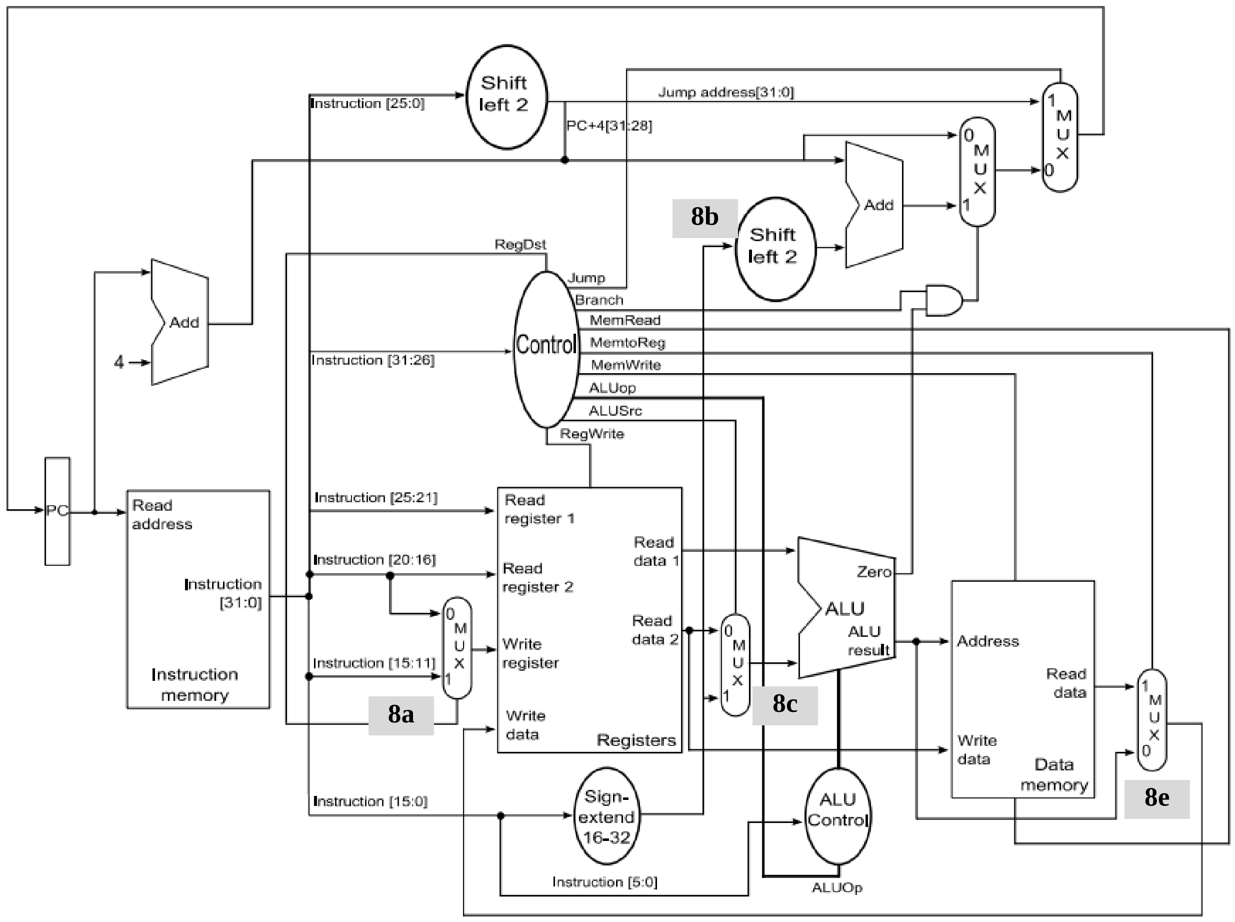
```
                                        . . .
                                        0x6000 zoo: add   $v0, $a0, $a1
int zoo(int a, int b) {                 . . .
   x = a + b;                           0x600c      jr    $ra
   return x;
}
```

a) (6 points) Suppose the initial register states were as follows (the second column) before the program executed the instruction 3a (move $a0, $t0) in foo(). Then, the program makes progress, and now it is about to execute the instruction 3b (jal zoo) in bar(). Please write down the register states before the instruction 3b executes.

	Before 9a	Before 9b
$t0	0x05	
$t1	0x03	
$a0	0x00	
$a1	0x00	
$v0	0x00	
$ra	0x2600	

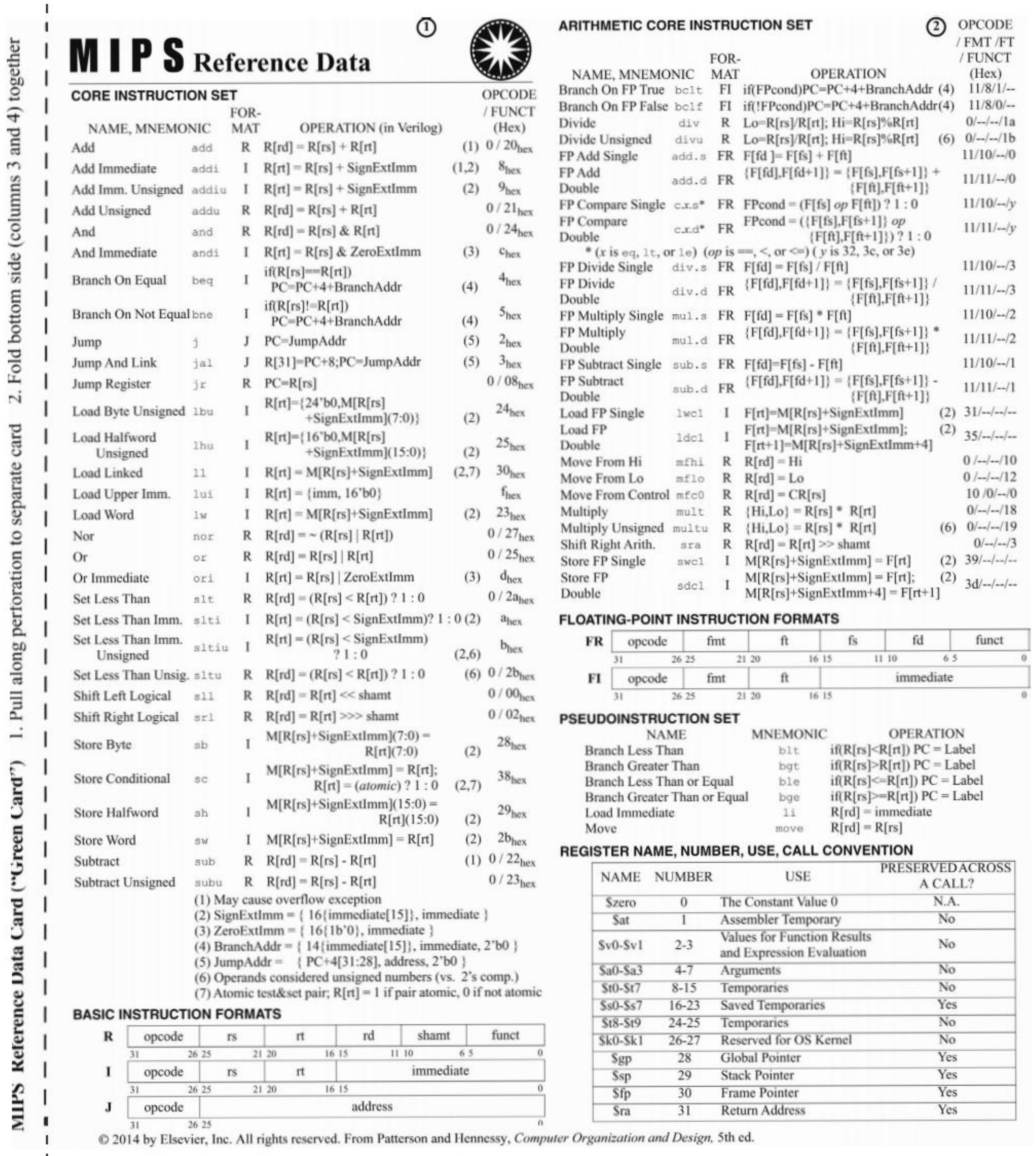
b) (4 points) After running the assembly code, a C programmer found that the program did not write anything. Please find what was wrong in the assembly code and how to fix it. (You do not need to write down new assembly code. A detailed explanation of how to fix it is sufficient).

Figure for question 8:



(3 points)

Appendix:



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] + 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

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

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*
* (*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]+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/--/--/--

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	PRESERVEDACROSS 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