Name ____

ISU ID # ____

Lab Section (circle one—your exam will be given a 0 if you do this incorrectly):

A (W 10-Noon), **B** (R 10-Noon), **C** (F 10-Noon), **D** (W 4-6)

CprE 381
Computer Organization and Assembly Level Programming

Exam #2

4/1/2019 9:00-9:50AM

Directions: There are 4 questions in this exam. Each question is worth points indicated. You should roughly spend 1 minute for every two points—plan accordingly. If a problem appears to be hard, move on and come back. Please read the questions carefully. Show your work, including any assumptions you need to use to solve the problems.

Calculators should NOT be used.

Problem	**Score**
1	**______ / 15 points**
2	**______ / 25 points**
3	**______ / 35 points**
4	**______ / 25 points**
Total	**______ / 100 points**

1. **FUNctional Unit Design (15 points).**
Currently the rotate left (**`rol`**) instruction (see the below excerpt from an ISA manual) is a pseudoinstruction. You will consider implementing the hardware needed to support left rotates for a ***4-bit data path*** (this means that data elements such as general-purpose registers and ALU components are only four bits wide).

Rotate left
```
rol rdest, rsrc1, rsrc2        pseudoinstruction
```

Rotate value in register rsrc1 left by the distance indicated by rsrc2 and put the result in register rdest.

Example:
```
rol     $t1, $t0, 1
```
is assembled as
```
sll     $at,$t0,1
srl     $t1,$t0,3
or      $t1,$t1,$at
```

Implement a functional unit that does both rotate left.

(a) How many levels/stages will be needed? Why? (5 points)

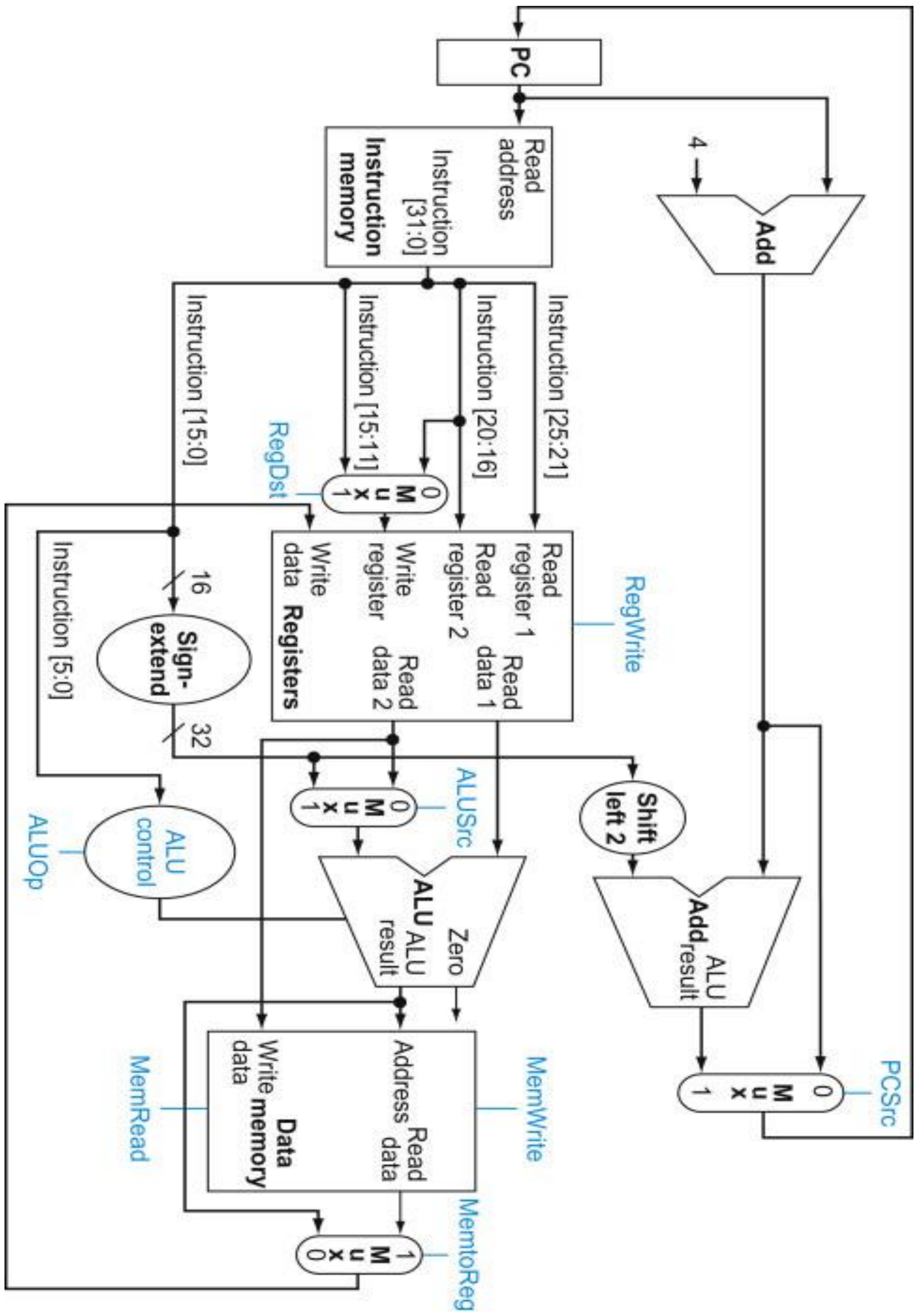
(b) Draw a schematic of the rotate unit below. First, label the inputs and outputs including their widths. Second, draw the required levels of MUXs. Third, hook up the components—make sure the signals are clearly labeled. (10 points)

2. **Processor Design (25 Points).**
Modify the single-cycle datapath on the following page to include the `minc` instruction. A `minc` instruction reads the word pointed to by the src register, increments it by one, and stores it back into the src register. Specifically, it performs the following rtl:

```
R[rs] ← M[R[rs]] + 1
```

Assume that `minc` uses the I-format. Your modifications can include: severing wires, assigning wires values, adding mux inputs, adding wires, and duplicating any component in the below diagram (and adding corresponding control signals). Assign values to each control signal (both old and new).

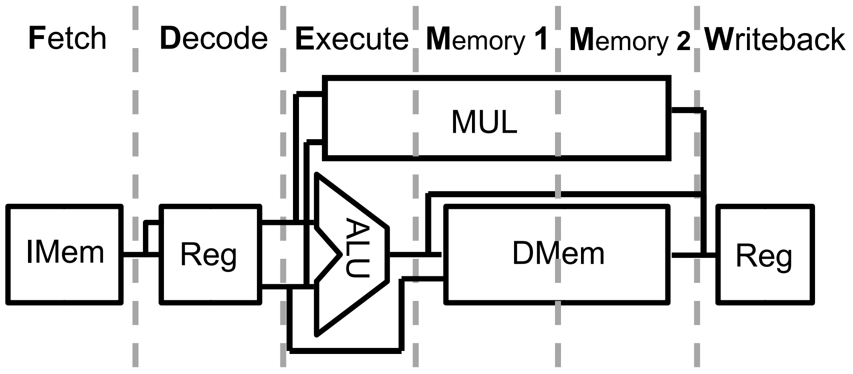
Control Signal	**Value**
RegDst	
RegWrite	
ALUSrc	
ALUOp	
PCSrc	
MemWrite	
MemRead	
MemtoReg	



PC
4
Add
Read address
Instruction [31:0]
Instruction memory
Instruction [25:21]
Instruction [20:16]
Instruction [15:11]
Instruction [15:0]
RegDst
0 M u x 1
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
RegWrite
16
Sign-extend
32
Instruction [5:0]
Shift left 2
Add ALU result
PCSrc
0 M u x 1
ALUSrc
0 M u x 1
ALU control
ALUOp
ALU
Zero
ALU result
Address
Write data
Read data
Data memory
MemWrite
MemRead
MemtoReg
1 M u x 0

3. **Pipelining (35 points).**
We have developed a new 381 pipeline that includes a larger (and thus longer-latency) data memory component and a tightly-integrated multiplier functional unit. The latencies of each of the functional units are tabulated below. Because of the long latencies of the new functional units, the pipeline has been redesigned as shown below (note this is a high-level representation and doesn't include all of the control signals and MUXs).

Imem	**Reg Read**	**ALU**	**MUL**	**DMem**
10ns	5ns	9ns	28ns	22ns



(a) What is the cycle time of this processor? You may assume that pipeline registers and MUXs have negligible latencies. Why? (10 points)

(b) What is the ~~maximum~~ ideal CPI of this new processor? Why? (5 points)

(c) Complete the table of read after write *data dependencies* in the following MIPS assembly code. You only need to include data dependencies through registers. (10 points)

```
1: mul     $t0, $s0, $s1    # Assume this to be a hardware
                            implementation of the MIPS mul
                            pseudoinstruction
2: sw      $t1, 0($t2)
3: lw      $t1, 4($t2)
4: addiu   $t2, $t0, 4
5: xor     $t0, $t1, $t2
6: sllv    $t3, $t0, $t1
```

	Reading Instruction Mnemonic	**Register**	**Writing Instruction Mnemonic**
1	**xor**	$t2	**addiu**
2			
3			
4			
5			

(d) For the sequence of instructions in (c), identify and ***list*** any data hazards for the pipeline described in (a). Demonstrate these hazards with a pipeline diagram (you must show where the hazard is). Assume no forwarding or stalling is implemented yet, but the register file will read the new value from a register that is written in the same cycle. (10 points)

Mnemonic	**Cyc 1**	**Cyc 2**	**Cyc 3**	**Cyc 4**	**Cyc 5**	**Cyc 6**	**Cyc 7**	**Cyc 8**	**Cyc 9**	**Cyc 10**	**Cyc 11**	**Cyc 12**	**Cyc 13**
`mul`	F	D	E	M1	M2	W							

4. **Performance (25 points).**
You are designing an embedded system that takes pictures of your cat and automatically generates a cat meme. You are considering two different processors. The first processor is a traditional MIPS processor, while the second one also has SIMD (single instruction, multiple data) support. The processors have the following frequencies and CPIs:

Machine	**CPU Speed**	**ALU**	**SIMD ALU**	**Load/Store**	**Branch/Jump**
Original	2 GHz	2	-	3	1
w/ SIMD	???	1	1	2	2

Consider two familiar software implementations for summing all of the elements of a byte array (this operation, known as a "sum reduction," is common in matrix-vector applications such as neural network inference). Assume that register $a1contains the value N.

Implementation 1 (no SIMD instructions):

```
    add   $t0, $zero, $zero
    add   $t2, $zero, $zero
    j     cond
loop:
    addu  $t1, $a0, $t0
    lb    $t1, 0($t1)
    addu  $t2, $t2, $t1
    addiu $t0, $t0, 1
cond:
    slt   $t1, $t0, $a1
    bne   $t1, $zero, loop
exit:
```

Implementation 2 (SIMD):

```
    add   $t0, $zero, $zero
    add   $t2, $zero, $zero
    j     cond
loop:
    addu      $t1, $a0, $t0
    lw        $t1, 0($t1)
    raddu.qb  $t1, $t1
    addu      $t2, $t2, $t1
    addu      $t0, $t0, 4
cond:
    slt   $t1, $t0, $a1
    bne   $t1, $zero, loop
exit:
```

(a) What is the CPI of the two processors on the above implementations (you should choose the appropriate implementation for each processor)? Since N may be very large, we can assume that the CPI is roughly the same as that of one iteration of the loop. SHOW YOUR WORK. (15 points)

(b) (continuation of 4) What is the performance of ***the SIMD processor*** assuming that both processors operate at the same frequency? Hint: Remember why SIMD is beneficial to the number of instructions executed. SHOW YOUR WORK. (10 points)

(END OF EXAM)

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
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	`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

OPCODES, BASE CONVERSION, ASCII SYMBOLS

MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary	Decimal	Hexa-decimal	ASCII Char-acter	Decimal	Hexa-decimal	ASCII Char-acter
(1)	sll	add.f	00 0000	0	0	NUL	64	40	@
		sub.f	00 0001	1	1	SOH	65	41	A
j	srl	mul.f	00 0010	2	2	STX	66	42	B
jal	sra	div.f	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt.f	00 0100	4	4	EOT	68	44	D
bne		abs.f	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov.f	00 0110	6	6	ACK	70	46	F
bgtz	srav	neg.f	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w.f	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w.f	00 1101	13	d	CR	77	4d	M
xori		ceil.w.f	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w.f	00 1111	15	f	SI	79	4f	O
	mfhi		01 0000	16	10	DLE	80	50	P
(2)	mthi		01 0001	17	11	DC1	81	51	Q
	mflo	movz.f	01 0010	18	12	DC2	82	52	R
	mtlo	movn.f	01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
	mult		01 1000	24	18	CAN	88	58	X
	multu		01 1001	25	19	EM	89	59	Y
	div		01 1010	26	1a	SUB	90	5a	Z
	divu		01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	_
lb	add	cvt.s.f	10 0000	32	20	Space	96	60	`
lh	addu	cvt.d.f	10 0001	33	21	!	97	61	a
lwl	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w.f	10 0100	36	24	$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	'	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l
			10 1101	45	2d	-	109	6d	m
swr			10 1110	46	2e	.	110	6e	n
cache			10 1111	47	2f	/	111	6f	o
ll	tge	c.f.f	11 0000	48	30	0	112	70	p
lwc1	tgeu	c.un.f	11 0001	49	31	1	113	71	q
lwc2	tlt	c.eq.f	11 0010	50	32	2	114	72	r
pref	tltu	c.ueq.f	11 0011	51	33	3	115	73	s
	teq	c.olt.f	11 0100	52	34	4	116	74	t
ldc1		c.ult.f	11 0101	53	35	5	117	75	u
ldc2	tne	c.ole.f	11 0110	54	36	6	118	76	v
		c.ule.f	11 0111	55	37	7	119	77	w
sc		c.sf.f	11 1000	56	38	8	120	78	x
swc1		c.ngle.f	11 1001	57	39	9	121	79	y
swc2		c.seq.f	11 1010	58	3a	:	122	7a	z
		c.ngl.f	11 1011	59	3b	;	123	7b	{
		c.lt.f	11 1100	60	3c	<	124	7c	|
sdc1		c.nge.f	11 1101	61	3d	=	125	7d	}
sdc2		c.le.f	11 1110	62	3e	>	126	7e	~
		c.ngt.f	11 1111	63	3f	?	127	7f	DEL

(1) opcode(31:26) == 0

(2) opcode(31:26) == 17_{ten} (11_{hex}); if fmt(25:21)==16_{ten} (10_{hex}) *f* = s (single); if fmt(25:21)==17_{ten} (11_{hex}) *f* = d (double)

IEEE 754 FLOATING-POINT STANDARD

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent - Bias})}$$

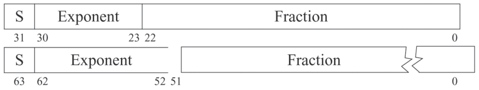
where Single Precision Bias = 127,
Double Precision Bias = 1023.

IEEE 754 Symbols

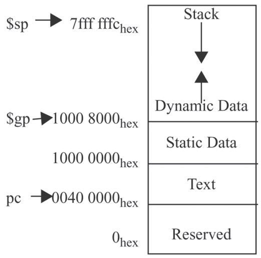
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

S.P. MAX = 255, D.P. MAX = 2047

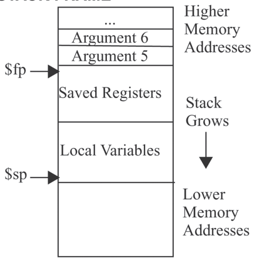
IEEE Single Precision and Double Precision Formats:



MEMORY ALLOCATION



STACK FRAME

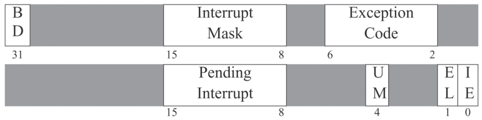


DATA ALIGNMENT

Double Word							
Word				Word			
Halfword		Halfword		Halfword		Halfword	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0	1	2	3	4	5	6	7

Value of three least significant bits of byte address (Big Endian)

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS



BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =Interrupt Enable

EXCEPTION CODES

Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception

SIZE PREFIXES

SIZE	PREFIX	SYMBOL	SIZE	PREFIX	SYMBOL	SIZE	PREFIX	SYMBOL	SIZE	PREFIX	SYMBOL
10^{3}	Kilo-	K	2^{10}	Kibi-	Ki	10^{15}	Peta-	P	2^{50}	Pebi-	Pi
10^{6}	Mega-	M	2^{20}	Mebi-	Mi	10^{18}	Exa-	E	2^{60}	Exbi-	Ei
10^{9}	Giga-	G	2^{30}	Gibi-	Gi	10^{21}	Zetta-	Z	2^{70}	Zebi-	Zi
10^{12}	Tera-	T	2^{40}	Tebi-	Ti	10^{24}	Yotta-	Y	2^{80}	Yobi-	Yi

MIPS® DSP ASE Instruction Set Quick Reference

Rd, Rs, Rt	—	Destination (Rd) and source (Rs, Rt) registers
Ac	—	Accumulator register (Ac0 – Ac3)
C, CC	—	Carry (bit 13) and condition code flags (bits 24-27) in DSPControl register
POS, SIZE	—	Position and size (scount) fields in DSPControl register
±	—	Signed operand/operation or sign extension
∅	—	Unsigned operand/operation or zero extension
×	—	Integer multiplication
⦿ / •	—	Fractional multiplication (implied shift left by 1 bit) with / without rounding
® / [ ]	—	Rounding and saturation operations
L / R	—	Left / right 16-bit part of a result or a register
LL, LR, RL, RR	—	The four bytes in a 32-bit register, from left (MSB) to right (LSB)
||	—	Boundary between two or four SIMD elements in a register
::	—	Concatenation of bit fields
R2	—	DSP ASE Revision 2 instruction

Arithmetic Operations: 8-bit Data			
ABSQ_S.QB[R2]	Rd, Rs	$Rd_{XY} = [\|Rs_{XY}^{\pm}\|]$	xy ∈ { LL, LR, RL, RR }
ADDU.QB	Rd, Rs, Rt	$Rd_{XY} = Rs_{XY}^{\varnothing} + Rt_{XY}^{\varnothing}$	xy ∈ { LL, LR, RL, RR }
ADDU_S.QB	Rd, Rs, Rt	$Rd_{XY} = [Rs_{XY}^{\varnothing} + Rt_{XY}^{\varnothing}]$	xy ∈ { LL, LR, RL, RR }
ADDUH.QB[R2]	Rd, Rs, Rt	$Rd_{XY} = (Rs_{XY}^{\varnothing} + Rt_{XY}^{\varnothing}) >> 1$	xy ∈ { LL, LR, RL, RR }
ADDUH_R.QB[R2]	Rd, Rs, Rt	$Rd_{XY} = (Rs_{XY}^{\varnothing} + Rt_{XY}^{\varnothing} + 1^{\varnothing}) >> 1$	xy ∈ { LL, LR, RL, RR }
RADDU.W.QB	Rd, Rs	$Rd = Rs_{LL}^{\varnothing} + Rs_{LR}^{\varnothing} + Rs_{RL}^{\varnothing} + Rs_{RR}^{\varnothing}$	
REPL.QB	Rd, const8	Rd = const8 || const8 || const8 || const8	
REPLV.QB	Rd, Rs	$Rd = Rs_{7:0} \| Rs_{7:0} \| Rs_{7:0} \| Rs_{7:0}$	
SUBU.QB	Rd, Rs, Rt	$Rd_{XY} = Rs_{XY}^{\varnothing} - Rt_{XY}^{\varnothing}$	xy ∈ { LL, LR, RL, RR }
SUBU_S.QB	Rd, Rs, Rt	$Rd_{XY} = [Rs_{XY}^{\varnothing} - Rt_{XY}^{\varnothing}]$	xy ∈ { LL, LR, RL, RR }
SUBUH.QB[R2]	Rd, Rs, Rt	$Rd_{XY} = (Rs_{XY}^{\varnothing} - Rt_{XY}^{\varnothing}) >> 1$	xy ∈ { LL, LR, RL, RR }
SUBUH_R.QB[R2]	Rd, Rs, Rt	$Rd_{XY} = (Rs_{XY}^{\varnothing} - Rt_{XY}^{\varnothing} + 1^{\varnothing}) >> 1$	xy ∈ { LL, LR, RL, RR }